Strategy-aware Liquidity for Account-based Blockchains

Ximeng Li[1,2], Sensen Chen[1], Yong Guan[1,3], Qianying Zhang[2], Guohui Wang[2], and Zhiping Shi✉[4,2]

[1] College of Information Engineering, Capital Normal University, Beijing, China
[2] Beijing Key Laboratory of Electronic System Reliability and Prognostics, Capital Normal University, Beijing, China
[3] Beijing Advanced Innovation Center for Imaging Theory and Technology, Capital Normal University, Beijing, China
[4] Beijing Academy of Science and Technology, Beijing, China

Abstract. A prominent functionality of smart contracts is regulating financial transactions on the blockchain. Flaws in a smart contract could cause funds to be frozen in the contract, resulting in financial losses of honest users. The issue of frozen funds is often considered a violation of liquidity requirements. In this work, we propose a liquidity property for account-based blockchain platforms such as Ethereum. The property captures the situation where the process of claiming funds by the honest users can be disrupted by the strategy of the environment. We formally establish the connection between this property and a strategy-less notion of liquidity under particular conditions, devise sound proof methods for the property, mechanize the theories and proof methods in the Rocq prover, and apply the proof methods to verify or refute the property for example smart contracts. Informative verification and refutation results are obtained for the examples — strategies help reveal how a user may act to ensure the transfer of funds out of a contract or how an attacker may exploit the inherent vulnerabilities of a contract to freeze funds.

1 Introduction

Blockchain technology has seen a growing range of applications since the introduction of Bitcoin [26]. A major type of applications is related to finance [38]. The realization of financial contracts on blockchains is facilitated with the advent of Ethereum [39] — an account-based blockchain platform that greatly eased the programming of smart contracts [31].

As smart contracts have become increasingly widespread, their security vulnerabilities have led to significant economic losses [11]. One major type of security vulnerability is that programming errors in a smart contract could cause digital assets to become locked within the contract. In 2017, digital assets worth over 150 million USD were locked in the Parity wallet [8]. More recently, approximately 34 million USD was frozen in the Akutar NFT contract [5]. Many similar incidents with severe consequences have been observed.

Research in formal methods addresses the critical issue of locked funds in smart contracts via the development of *liquidity properties* and techniques for analysis and verification [37, 33, 18, 15, 13, 12, 19]. A clearly defined liquidity property articulates the target requirement of formal verification. A general liquidity requirement is: a group of users should be able to transfer an amount of funds out of a smart contract, despite the potential interference of the environment.

The potential behaviors of the protected users and their environment can be captured by strategies. A strategy of the environment that prevents the desired transfer of funds by the users provides a witness that a smart contract could suffer from the problem of locked funds. A strategy of the protected users that enables the desired transfer elucidates how this problem can be avoided. Hence, strategies offer deeper insights into the liquidity characteristics of a smart contract beyond a simple yes-or-no determination. However, existing definitions of liquidity properties for smart contracts are either strategy-agnostic (e.g., [37, 23, 12, 19]) or tailored to UTxO-based blockchain models such as Bitcoin (e.g., [15, 13]). A foundational criterion for the formal analysis and verification of strategy-aware liquidity requirements in account-based blockchain models is still lacking.

A potential definition of a strategy-aware liquidity property for account-based blockchains would characterize the joint effects of the users and the environment on the amount of funds in an account. It would be possible to verify or refute such a property for a smart contract by directly constructing a proof that the defined property or its negation holds. However, such a direct proof would involve intertwined reasoning about the behaviors of the user and the environment. Moreover, the application-independent part of the direct proof must be replicated for different smart contracts to be examined. Hence, the proposal of a strategy-aware liquidity property for account-based blockchains should be accompanied by proof methods to simplify the verification and refutation of the property.

In this work, we provide a formal foundation for expressing and verifying strategy-aware liquidity requirements for smart contracts in account-based blockchains. Our main technical contributions include:

- the definition of a strategy-aware liquidity property (Section 4),
- a formal result on the relationship between the strength of the property and the strength of the environmental strategy (Section 5),
- a formal result on the relationship between the property and a strategy-less notion of liquidity under particular conditions (Section 5),
- sound proof methods relying on well-founded orders and invariants, for the verification and refutation of the property (Section 6),
- mechanization of the theory and proof methods for the property in the Rocq prover[5] [9, 16] (Section 7),
- proofs and refutations of liquidity for example smart contracts (Section 7).

Compared with the existing definitions of liquidity properties, a distinguishing feature of our property is in allowing the environment to exert influence on the users not only before the process as they attempt to transfer funds out of the smart contract, but also during this process. This feature reflects the potential

[5] formerly known as the Coq proof assistant

ability of a realistic attacker. Our mechanization [6] of theories and liquidity proofs are based on the ConCert framework [28, 10] for modeling and reasoning about smart contracts. Our newly developed definitions total approximately 3.2k LOC and newly constructed proofs approximately 6.2k LOC.

2 Background and Motivation

In an account-based blockchain (e.g., Ethereum [39]), an account is an entity at an address that contains a certain amount of funds. An account may optionally contain a smart contract. The smart contract is an object that maintains data and offers methods that implement the logic of a business contract (e.g., starting shipment of goods upon receipt of digital money). A user with an account may execute transactions — a transaction could perform a transfer of funds to some account, deploy a smart contract at some address, or invoke a method in an existing smart contract.

Consider a smart contract c_{fm} where a user (whose account is at the address *usr*) attempting to withdraw funds has to request for the withdrawal and wait for approval by the administrator (whose account is at the address *adm*). This administrator role is common in lending protocols, insurance contracts, multi-signature wallets, etc (e.g., [20]). The allowed changes of the user's status are illustrated in Fig. 1. Here, a transition arrow represents a transaction. Below each arrow, the action that triggers the transaction is annotated. Above the arrow, the business logic for the transaction is expressed via pseudo-code. In the pseudo-code, *from* refers to the address of the account that performs the action.

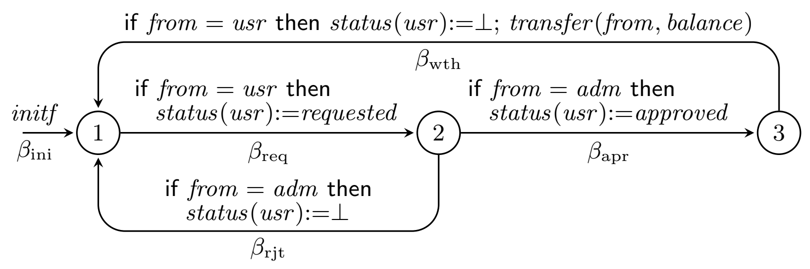
if $from = usr$ then $status(usr) := \bot$; $transfer(from, balance)$
β_{wth}
if $from = usr$ then $status(usr) := requested$
if $from = adm$ then $status(usr) := approved$
$initf$
β_{ini}
1
2
3
β_{req}
β_{apr}
if $from = adm$ then $status(usr) := \bot$
β_{rjt}

Fig. 1. Illustration of the user's status changes in c_{fm}

Intuitively, the smart contract c_{fm} fulfills the liquidity requirements that are sensible in this funds management scenario as long as the administrator is honest. This is because a benign user is able to achieve their intended withdrawal of funds under the approval of the collaborative administrator.

Consider a smart contract $c^{\times}_{\text{fm}}$ that is like c_{fm}, except that the function *initf* used to initialize the contract $c^{\times}_{\text{fm}}$ can be invoked at any time (as opposed to contract creation only) to set the caller as the administrator. This flawed mechanism has been located in real-world financial contracts like Rubixi [4] and 88mph [1]. It enables a malicious user to become the administrator and reject all requests for the withdrawal of funds by the benign user. If Fig. 1 is re-used to help understand how $c^{\times}_{\text{fm}}$ differs from c_{fm}, what the malicious user could do in $c^{\times}_{\text{fm}}$ is the execution of the transition $2 \rightarrow 1$ in the administrator role. Intuitively, the flawed contract $c_{\text{fm}}^{\times}$ violates liquidity requirements because the adversary is able to freeze funds in the contract.

In the literature, a basic notion of liquidity requires the existence of an execution trace that decreases the balance of the given smart contract [37, 19]. A more fine-grained notion of liquidity requires that in any state resulting from the execution of a smart contract potentially driven by a group of users and an adversary, the users should have a way of withdrawing the funds in the contract in a sequence of moves [15, 23]. Since the adversary is non-collaborative, this fine-grained liquidity notion can be employed to express stronger liquidity concerns than expressible by the basic notion. If strategies are used for the adversary, then the fine-grained notion also enables the reasoning about the liquidity guarantees of a smart contract under assumptions on what the adversary could do.

The smart contract c_{fm} fulfills the requirements of both liquidity notions mentioned above. At any state, the balance of the contract can be reduced to zero by following the suffix of the trace $1 \to 2 \to 3 \to 1$ starting from that state. Moreover, each transition in this trace is taken by a member in the group of honest users (consisting of the benign user and the honest administrator). The state of these honest users cannot be influenced by the adversary. The discussion above supports the conformance of the existing liquidity notions with the informal liquidity requirements for the contract c_{fm}.

However, the requirements of both liquidity notions are also fulfilled by the flawed contract $c_{\text{fm}}^{\times}$. Firstly, the trace $1 \to 2 \to 3 \to 1$ is also permitted by this flawed contract, reducing the balance of the contract to zero. Secondly, although the adversary may trigger the transition $2 \to 1$ in the administrator role, the honest administrator can re-gain the administrator role via the function *initf* afterward. Then, the group of honest users can again achieve the intended decrease of $c_{\text{fm}}^{\times}$'s balance through the trace $1 \to 2 \to 3 \to 1$. This discussion reveals the discrepancies between the existing liquidity notions and the informal liquidity requirements for the contract $c_{\text{fm}}^{\times}$.

The discussion above is based on an informal understanding of the liquidity property of [15] in our setting of an account-based blockchain. This liquidity property was formulated in the model of Bitcoin, which is a UTxO-based blockchain platform. To our knowledge, none of the existing liquidity properties for account-based blockchain models distinguishes between the strategies of honest and malicious users. In this article, we propose such a liquidity property. The proposed property captures the fulfillment of the intuitive liquidity requirements by the smart contract c_{fm}, as well as the violation of the intuitive liquidity requirements by the smart contract $c_{\text{fm}}^{\times}$. Furthermore, we devise methods for the sound verification and refutation of the proposed liquidity property, and demonstrate the use of these methods on example smart contracts.

3 Model of Smart Contract Execution

We introduce formal concepts concerning the execution of blockchain transactions according to strategies. These formal concepts provide the basis for our definition of the strategy-aware liquidity property.

3.1 Blockchain States and Transaction Execution

We denote the set of all account addresses as Adr. An account that does not contain a smart contract is called an *externally owned account*. We denote the set of account addresses for externally owned accounts by Adr_{eoa}.

We model the status of the blockchain at a specific point in the execution of transactions by a *blockchain state* $\sigma \in \Sigma$. This blockchain state reflects the status of each existing account — including the balance of the account, as well as the code and the storage state of the smart contract (if any) in the account.

We represent the balance of the account at the address a in the blockchain state σ as $balance(\sigma, a)$. We represent the state of the smart contract at address a in the blockchain state σ as $ctr\text{-}st(\sigma, a)$. If there is no smart contract at a, then $ctr\text{-}st(\sigma, a) = \bot$. We use $s.v$ to represent the value of the variable v in the smart contract state s (where $s \neq \bot$).

We capture the *execution of a transaction* by $\sigma \xrightarrow{\beta} \sigma'$. Here, σ and σ' are the blockchain states before and after the transaction, respectively. In addition, β is the *action* triggering the transaction. This action could be the deployment of a contract, a call to a contract, or the transfer of funds to an account. We write $orig(\beta)$ for the address of the account that originally initiated the transaction. We assume that transactions are always initiated by externally owned accounts, as is the case for Ethereum [39]. Hence, we have $orig(\beta) \in Adr_{\text{eoa}}$.

The blockchain users send transactions in a parallel fashion. However, the transactions are uniquely ordered when recorded in the blocks, and are executed in this order. We write $\sigma \to^* \sigma'$ to express that σ and σ' are related by the reflexive, transitive closure of the binary relation induced by $\exists \beta : \sigma \xrightarrow{\beta} \sigma'$. Hence, $\sigma \to^* \sigma'$ captures the effects of executing a sequence of blockchain transactions.

We capture the history of transaction execution at a specific point in the evolvement of the blockchain state by a *trace* π. It is an alternating sequence of blockchain states and actions, starting and ending with blockchain states. We denote the set of all traces by $Trace$. We use $last(\pi)$ to represent the last blockchain state of the trace π.

We say that the smart contract c is *initial* in the blockchain state σ, as denoted by $init\text{-}ctr(a, c, \sigma)$, if the blockchain state σ is reachable from the initial state of the blockchain, and the contract has just been deployed at a and has not received any transfers or invocations in the blockchain state σ.

3.2 Strategy-driven Transaction Execution

We use strategies to capture the intended ways in which the users attempt to interact with the blockchain.

Definition 1 (Strategies). *A* strategy *for the set as of addresses is a function* $\delta^{as} \in Trace \to 2^{\{\beta \in Act \mid orig(\beta) \in as\}}$.

Intuitively, after each possible history $\pi \in Trace$, the strategy gives the set of possible actions that could be performed by the accounts at the addresses in as. The strategies we consider are non-deterministic (as has been employed in the security literature [30, 25, 40]). A strategy could suggest no action to be performed, or multiple candidate actions to be picked from for execution, after an execution history of the blockchain.

Example 1. Based on the understanding of the business logic of the smart contract c_{fm} (Section 2), the following strategy $\delta^{\{usr,adm\}}_{\text{fm-u}}$ captures a way in which the honest users may ensure the withdrawal of funds from the contract.

$$\delta^{\{usr,adm\}}_{\text{fm-u}}(\pi) := \begin{cases} \{\beta_{\text{req}}\} & \text{if } \mathit{ctr\text{-}st}(\mathit{last}(\pi), a).\mathit{status}(\mathit{usr}) = \bot \\ \{\beta_{\text{apr}}\} & \text{if } \mathit{ctr\text{-}st}(\mathit{last}(\pi), a).\mathit{status}(\mathit{usr}) = \mathit{requested} \\ \{\beta_{\text{wth}}\} & \text{if } \mathit{ctr\text{-}st}(\mathit{last}(\pi), a).\mathit{status}(\mathit{usr}) = \mathit{approved} \\ \emptyset & \text{otherwise} \end{cases}$$

Following this strategy, the benign user executes the action β_{req} if the status of this user is $\bot$ in the last blockchain state of the history. Furthermore, the administrator executes the action β_{apr} if the status of the benign user is *requested* in the last blockchain state of the history. Moreover, the benign user executes the action β_{wth} if the status of this user is *approved* in the last blockchain state of the history. Finally, if none of the above conditions holds for the current history, then the benign user and the administrator do not perform any action. □

We capture the execution of a transaction according to the strategy δ^{as} with $\delta^{as} \vdash \pi \to \pi'$, which is given by

$$\delta^{as} \vdash \pi \to \pi' := \exists \beta \in \delta^{as}(\pi) : \exists \sigma' : \mathit{last}(\pi) \xrightarrow{\beta} \sigma' \wedge \pi' = \pi \,\hat{}\, [\beta] \,\hat{}\, [\sigma']$$

That is, the execution of a transaction according to the strategy δ^{as} extends the trace π to the trace π', if and only if there is an action β allowed by δ^{as} for the history π, such that the execution of β from the last blockchain state of π results in the blockchain state σ', and the trace π' is obtained by appending β and σ' to the trace π in order. In the definition, an expression of the form $[\nabla]$ represents a singleton sequence containing the element ∇. The operator $\hat{}$ gives the concatenation of two sequences. Moreover, we write $\delta^{as} \vdash \pi \to^* \pi'$ to express that π and π' are related by the reflexive, transitive closure of the binary relation induced by $\delta^{as} \vdash \cdot \to \cdot$.

4 Definition of Strategy-aware Liquidity

In this section, we define the strategy-aware liquidity property. The basic requirement of this property is: from an arbitrary blockchain state reached after the target smart contract is initialized, the honest users may interact with the contract to decrease the balance of the contract. In realistic scenarios, the interaction of the honest users with the smart contract is performed under potential interference by other users. This interference has potentially adversarial effects on the honest users. Hence, we model these other users collectively as an adversary. We suppose the honest users and the adversary always interact with the smart contract following their strategies.

The requirement imposed by our liquidity property is illustrated in Fig. 2.

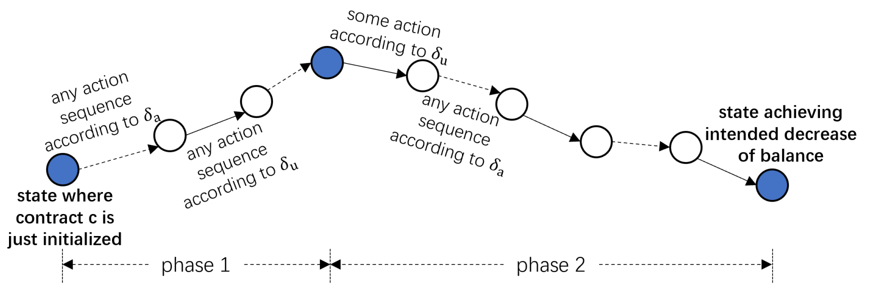
some action according to δ_u
any action sequence according to δ_a
any action sequence according to δ_u
any action sequence according to δ_a
state where contract c is just initialized
state achieving intended decrease of balance
phase 1
phase 2

Fig. 2. Illustration of the requirements of strategy-aware liquidity

The part annotated with "phase 1" in Fig. 2 reflects the process where the adversary and the honest users alternatingly perform actions to update the blockchain state. This process is captured with the help of the judgment

$$(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash \pi \rightsquigarrow \pi' : \mathit{nxt}$$

Here, π and π' are the traces before and after the alternating execution of the actions from the honest users (following the strategy $\delta_{\mathrm{u}}^{as_1}$) and the adversary (following the strategy $\delta_{\mathrm{a}}^{as_2}$). Moreover, $\mathit{nxt} \in \{\mathrm{u\_nxt}, \mathrm{a\_nxt}\}$ indicates whether the honest user or the adversary will execute next. The instances of this judgment are established inductively, according to the rules in Fig. 3.

$$[\text{REFL}] \quad \frac{}{(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash \pi \rightsquigarrow \pi : \mathit{nxt}} \quad \text{if } \mathit{nxt} \in \{\mathrm{u\_nxt}, \mathrm{a\_nxt}\}$$

$$[\text{U-EXE}] \quad \frac{(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash \pi \rightsquigarrow \pi'' : \mathrm{u\_nxt} \qquad \delta_{\mathrm{u}}^{as_1} \vdash \pi'' \rightarrow \pi'}{(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash \pi \rightsquigarrow \pi' : \mathrm{a\_nxt}}$$

$$[\text{A-EXE}] \quad \frac{(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash \pi \rightsquigarrow \pi'' : \mathrm{a\_nxt} \qquad \delta_{\mathrm{a}}^{as_2} \vdash \pi'' \rightarrow^* \pi'}{(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash \pi \rightsquigarrow \pi' : \mathrm{u\_nxt}}$$

Fig. 3. Alternating execution of actions from honest users and adversary

The rule [REFL] captures the effect of zero execution step. The rule [A-EXE] expresses that if the adversary should execute next, and they take zero or more steps, then the alternating execution is extended by these steps, switching the next turn back to the honest users. The rule [U-EXE] expresses that if the honest users should execute next, and they take a further step, then the alternating execution is extended by this step, switching the next turn to the adversary. Multiple consecutive steps by honest users are derived by jointly using the rules [U-EXE] and [A-EXE], since [A-EXE] allows the adversary to take zero step.

The part annotated with "phase 2" in Fig. 2 reflects that the honest users are able to decrease the balance of the smart contract c at address a, even though the adversary may interfere with the process after each transaction executed by the honest users. We capture this requirement with the predicates *usl* and *asl*. These two predicates are defined in a mutual inductive fashion, as shown in Fig. 4. In the definition, we use the predicate ϕ_{bal} to express that the intended change of balance for the target contract is achieved by the honest users.

$$\text{[U-BASE]} \quad \frac{}{\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma, a, c)} \quad \text{if } \phi_{\mathrm{bal}}(\sigma, \mathit{last}(\pi), a, c)$$

$$\text{[U-TRAN]} \quad \frac{\delta_{\mathrm{u}}^{as_1} \vdash \pi \rightarrow \pi' \quad \mathit{asl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi', \sigma, a, c)}{\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma, a, c)}$$

$$\text{[A-BASE]} \quad \frac{}{\mathit{asl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma, a, c)} \quad \text{if } \phi_{\mathrm{bal}}(\sigma, \mathit{last}(\pi), a, c)$$

$$\text{[A-INTF]} \quad \frac{\forall \pi' : \delta_{\mathrm{a}}^{as_2} \vdash \pi \rightarrow^* \pi' \Rightarrow \mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi', \sigma, a, c)}{\mathit{asl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma, a, c)} \quad \text{if } \neg\phi_{\mathrm{bal}}(\sigma, \mathit{last}(\pi), a, c)$$

Fig. 4. The rules defining the predicates *usl* and *asl*

In Fig. 4, the rules [U-BASE] and [A-BASE] express that if $\phi_{\mathrm{bal}}(\sigma, \mathit{last}(\pi), a, c)$ holds, then both $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma, a, c)$ and $\mathit{asl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{u}}^{as_2}, \pi, \sigma, a, c)$ can be derived. The rule [U-TRAN] expresses that if the honest users execute a transaction to extend the trace π to the trace π', and $\mathit{asl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{u}}^{as_2}, \pi', \sigma, a, c)$ can be derived, then $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma, a, c)$ can be derived. The rule [A-INTF] expresses that if $\phi_{\mathrm{bal}}(\sigma, \mathit{last}(\pi), a, c)$ does not hold, but for any trace π' resulting from zero or more transactions by the adversary from the trace π, $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi', \sigma, a, c)$ can be derived, then $\mathit{asl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma, a, c)$ can be derived.

Hence, the expression $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma, a, c)$ can be interpreted as: if the honest users transact next (when the current trace is π), then there exists a way to achieve the intended decrease of balance for the target contract, relative to the blockchain state σ. The expression $\mathit{asl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{u}}^{as_2}, \pi, \sigma, a, c)$ can be interpreted as: if the adversary may transact next (when the current trace is π), then no matter how it behaves (in accordance to its strategy), there exists a way for the honest users to achieve the intended decrease of balance for the target contract, relative to the blockchain state σ.

We give the definition of our strategy-aware liquidity property below.

Definition 2 (Strategy-aware liquidity). *The smart contract c at address a satisfies* strategy-aware liquidity *with user strategy $\delta_{\mathrm{u}}^{as_1}$ and adversary strategy $\delta_{\mathrm{a}}^{as_2}$, denoted by $SL(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, a, c)$, if and only if*

$$\forall \sigma, \pi : \mathit{init\text{-}ctr}(a, c, \sigma) \wedge (\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash [\sigma] \rightsquigarrow \pi : \mathrm{u\_nxt} \Rightarrow \mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \mathit{last}(\pi), a, c)$$

Hence, the formal requirement of strategy-aware liquidity is: for any blockchain state σ that is reachable from the initial state of the blockchain and where the contract c is in its initial state, if trace π results from arbitrary execution of transactions driven jointly by the adversary and the honest users, then the honest users must be able to decrease the balance of the contract c through further transactions (following $\delta_{\mathrm{u}}^{as_1}$) that extend π, no matter how the adversary continues to interfere with the process following $\delta_{\mathrm{a}}^{as_2}$.

Compared with existing notions of liquidity [37, 15, 13, 23, 12, 19], a salient feature of strategy-aware liquidity is that the adversary may keep interfering with the honest users' process of withdrawing funds. Intuitively, this feature is illustrated in Fig. 2 where any finite sequence of actions from the adversary may be inserted between two actions of the honest users in phase 2. Technically, the admittance of the interfering actions is captured by the rule [A-INTF] in Fig. 4.

The aforementioned feature of strategy-aware liquidity captures strengthened ability of the adversary. This feature reflects that in reality, it can be difficult to guarantee that all the transactions the honest users should execute to achieve the intended decrease in the balance of a smart contract are packaged consecutively in a block. Hence, the adversary can exert influence between two consecutive transactions executed by the honest users.

Remark 1. It may seem that the second phase in the interacting process of the honest users and the adversary as described by strategy-aware liquidity (more precisely, by *usl*) does not permit any interfering actions of the adversary before the honest users' first transaction. This is not a problem because the first phase ends with arbitrary interference according to the adversary strategy.

Remark 2. The strategy-aware liquidity property (Definition 2) can potentially be understood in the setting of two-player games (e.g., [32]). However, game theory does not directly offer tools that help with the design of the proof and refutation methods for our liquidity property (see Section 6). Hence, we present our concepts and results without relying on game-theoretic notions.

5 Theoretical Aspects of Strategy-aware Liquidity

We formally establish the relationship between the strength of the adversary strategy and the strength of strategy-aware liquidity. Moreover, we establish the connection between strategy-aware liquidity and a strategy-less notion of liquidity, under specific strategies for the adversary and the honest users.

5.1 Monotonicity

Our adversary strategies specify the set of actions that can be performed by the adversary, after it observes each history. Intuitively, given the same history, the more actions allowed by the adversary strategy, the stronger liquidity requirement that is imposed by the strategy-aware liquidity property. Below, we establish this result formally.

We start by defining the relative strength of strategies.

Definition 3 (Relative strength of strategies). *For two strategies δ_1^{as} and δ_2^{as} over the same set of addresses, δ_2^{as} is* at least as strong as *δ_1^{as}, as denoted by $\delta_1^{as} \preceq_{\mathrm{stg}} \delta_2^{as}$, if and only if for each trace π, it holds that $\delta_1^{as}(\pi) \subseteq \delta_2^{as}(\pi)$.*

The relation $\preceq_{\mathrm{stg}}$ is a partial order on the set of strategies for each as.

We establish the following result about how the relative strength of strategies affects the strength of the predicate *usl*.

Lemma 1. *If $\delta_1^{as} \preceq_{\mathrm{stg}} \delta_2^{as}$ holds, and $usl(\delta_{\mathrm{u}}^{as'}, \delta_2^{as}, \pi, \sigma, a, c)$ can be established, then $usl(\delta_{\mathrm{u}}^{as'}, \delta_1^{as}, \pi, \sigma, a, c)$ can be established.*

This lemma expresses that if $\delta_1^{as} \preceq_{\text{stg}} \delta_2^{as}$, and the honest users are able to reduce the balance of the target contract under the interference from the adversary following the strategy δ_2^{as}, then the honest users are able to achieve the same goal under the interference from the adversary following the strategy δ_1^{as}.

We establish the following monotonicity result of strategy-aware liquidity based on Lemma 1.

Proposition 1 (Monotonicity). *If $\delta_1^{as} \preceq_{\text{stg}} \delta_2^{as}$ holds, and $SL(\delta_{\text{u}}^{as'}, \delta_2^{as}, a, c)$ can be established, then $SL(\delta_{\text{u}}^{as'}, \delta_1^{as}, a, c)$ can be established.*

Firstly, this monotonicity result is a sanity check on the definition of strategy-aware liquidity. The result reflects the intuition that the more actions can be attempted by the attacker, the more easily a system could violate liquidity requirements. Secondly, it supports the refutation of a smart contract's liquidity property – if the contract is shown to violate strategy-aware liquidity once the adversary is permitted to perform particular actions, the contract must also violate strategy-aware liquidity if further actions could be performed by the adversary.

There are usable smart contracts that do not satisfy liquidity requirements if the adversary may act arbitrarily. However, if an action by the adversary is unlikely to occur in reality, we can consider a weakened strategy of the adversary that does not permit the performance of this action. It can be sensible to examine the satisfaction of strategy-aware liquidity under this weakened strategy. An example is with an insurance smart contract (e.g., Nexus Mutual [7]), where the adversary may in principle vote to reject a legitimate insurance claim, but it will not actually do so for fear of losing its staked crypto-assets.

5.2 Connection with Strategy-less Liquidity

In the literature, there are notions of liquidity (e.g., considered in [37, 23, 19]) that disregard the distinction between the honest users and the adversary, as well as their strategies. We define strategy-less liquidity in our technical setting. We show that strategy-aware liquidity and strategy-less liquidity express the same requirements under specific strategies for the adversary and the honest users.

We refer to our strategy-less notion of liquidity as basic liquidity.

Definition 4 (Basic liquidity). *The smart contract c at address a satisfies* basic liquidity*, denoted by $BL(a, c)$, if and only if*

$$\forall \sigma, \sigma' : \textit{init-ctr}(a, c, \sigma) \wedge \sigma \rightarrow^* \sigma' \Rightarrow \exists \sigma'' : \sigma' \rightarrow^* \sigma'' \wedge \phi_{\text{bal}}(\sigma', \sigma'', a, c)$$

Hence, the requirement of basic liquidity is: For any blockchain state σ that is reachable from the initial state of the blockchain and where the contract c is in its initial state, if an arbitrary blockchain state σ' can be reached from σ through the execution of zero or multiple transactions, then there should exist a further series of transactions that update the blockchain state to σ'', where the intended decrease of the balance for the smart contract c is achieved.

We introduce the notion of a *complete strategy*. A strategy δ^{as} is complete, denoted by $\textit{is-complete}(\delta^{as})$, if and only if

$$\forall \beta, \pi : (\exists \sigma' : \textit{last}(\pi) \xrightarrow{\beta} \sigma') \wedge \textit{orig}(\beta) \in \textit{as} \Rightarrow \beta \in \delta^{as}(\pi)$$

We establish the following result about the connection between strategy-aware liquidity and basic liquidity.

Proposition 2 (Connection between liquidity notions).
If is-complete$(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}})$ *holds, then for all addresses a and contracts c, we have* $SL(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, a, c) \iff BL(a, c)$.

This proposition expresses that if the strategy of the honest users is complete and there are no adversarial users (i.e., all users are corporative), then strategy-aware liquidity is equivalent to basic liquidity. If there exist adversarial users, however, then $usl(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, last(\pi), a, c)$ in the definition of strategy-aware liquidity is typically stronger than $\exists \sigma'' : \sigma' \to^{*} \sigma'' \wedge \phi_{\mathrm{bal}}(\sigma', \sigma'', a, c)$ in the definition of basic liquidity, assuming $last(\pi)$ is the blockchain state σ'.

6 Proof Methods for Strategy-aware Liquidity

We devise two proof methods that facilitate the verification and refutation of strategy-aware liquidity, respectively. We establish the soundness of both proof methods. The application of the proof methods will be demonstrated in Section 7.

6.1 Method for Verification

The verification of strategy-aware liquidity can be performed by a direct proof of $SL(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, a, c)$. However, this will involve intertwined reasoning about the behaviors of the honest users and the behaviors of the adversary. We propose sound verification conditions for strategy-aware liquidity. These verification conditions disentangle the reasoning about the behaviors of the honest users and the reasoning about the behaviors of the adversary, thereby reducing the conceptual complexity in the verification of strategy-aware liquidity.

For the design of the verification conditions, we use an *invariant condition* $\mathcal{I} \in \Sigma \times \Sigma \to Prop$. For two blockchain states σ and σ', $\mathcal{I}(\sigma, \sigma')$ captures the condition that is satisfied by σ' relative to σ. Intuitively, the invariant condition describes the condition that the current state σ' keeps satisfying relative to some state σ as transactions execute on the blockchain. In addition, we consider a *transitive, well-founded order* $\prec_{\mathrm{vc}}$ on a set X. Recall that all descending chains are finite in a well-founded order. We also use a *variant function* [17] $\mathcal{V} \in \Sigma \times \Sigma \to X \cup \{\bot\}$. For two blockchain states σ and σ', if $\mathcal{V}(\sigma, \sigma') = x \in X$, then x captures the changes from σ to σ'. Intuitively, such changes should contribute to the satisfaction of the target condition $\phi_{\mathrm{bal}}(\sigma, \sigma', a, c)$. The variant function $\mathcal{V}$ need not be defined for all pairs of blockchain states. This feature increases the level of flexibility in providing the formal definition of $\mathcal{V}$ in verification tasks.

In Fig. 5, we present the verification conditions for the strategy-aware liquidity of the smart contract c at the address a. The parameters c and a are implicit in these verification conditions. The condition *VC_B* says that for two blockchain states σ and σ', if the invariant holds and the variant gives a minimal element in the well-founded order, then the predicate ϕ_{bal} should hold on σ and σ'. Intuitively, this means if the result of the variant cannot decrease any further, then the honest users have achieved the intended decrease of balance for the target contract. The condition *VC_R* says that for any blockchain state that is

$$\begin{aligned}
\mathit{VC\_B} &:= \forall\sigma,\sigma',x\in X: \mathcal{I}(\sigma,\sigma')\wedge\mathcal{V}(\sigma,\sigma')=x\wedge \mathit{min}(x) \;\Rightarrow\; \phi_{\mathrm{bal}}(\sigma,\sigma',a,c)\\
\mathit{VC\_R} &:= \forall\sigma_0,\sigma: \mathit{init\text{-}ctr}(a,c,\sigma_0)\wedge\sigma_0\to^*\sigma \;\Rightarrow\; \mathcal{I}(\sigma,\sigma)\wedge\mathcal{V}(\sigma,\sigma)\neq\bot\\
\mathit{VC\_A}(\delta_{\mathrm{a}}^{as}) &:= \forall\sigma,\pi,\pi',x\in X:\\
&\qquad \mathcal{I}(\sigma,\mathit{last}(\pi))\wedge\mathcal{V}(\sigma,\mathit{last}(\pi))=x\wedge\delta_{\mathrm{a}}^{as}\vdash\pi\to^*\pi' \;\Rightarrow\\
&\qquad\quad \mathcal{I}(\sigma,\mathit{last}(\pi'))\wedge\\
&\qquad\quad \exists x'\in X:\mathcal{V}(\sigma,\mathit{last}(\pi'))=x'\wedge(\neg\mathit{min}(x)\Rightarrow x'\preceq_{\mathrm{vc}}x)\\
\mathit{VC\_U}(\delta_{\mathrm{u}}^{as}) &:= \forall\sigma,\pi,x\in X:\\
&\qquad \mathcal{I}(\sigma,\mathit{last}(\pi))\wedge\mathcal{V}(\sigma,\mathit{last}(\pi))=x\wedge\neg\mathit{min}(x) \;\Rightarrow\\
&\qquad\quad \exists\pi':\delta_{\mathrm{u}}^{as}\vdash\pi\to\pi'\wedge\mathcal{I}(\sigma,\mathit{last}(\pi'))\wedge\\
&\qquad\qquad\quad \exists x'\in X:\mathcal{V}(\sigma,\mathit{last}(\pi'))=x'\wedge x'\prec_{\mathrm{vc}}x
\end{aligned}$$

Fig. 5. The verification conditions for strategy-aware liquidity

reachable from a blockchain state σ_0 where the contract c has just been created, the invariant holds and the variant is defined. This condition provides a basis for reasoning about the strategy-driven transitions of the adversary and the honest users, as long as these transitions take place from reachable blockchain states. The condition $\mathit{VC\_A}(\delta_{\mathrm{a}}^{as})$ says that if the adversary executes a series of transitions according to the strategy δ_{a}^{as}, then the invariant and the well-definedness of the variant are preserved under these transitions, and the value of the variant does not increase, unless the value x of the variant before these transitions is minimal. If x is already minimal, then the condition $\mathit{VC\_B}$ helps ensure that the intended decrease of balance for the target contract is already achieved. Hence, no further reasoning is required for the verification. The condition $\mathit{VC\_U}(\delta_{\mathrm{u}}^{as})$ says that if the invariant is satisfied and the variant is defined and not minimal in the blockchain state σ, then the honest users can take a further transition from σ. This transition will preserve the invariant and decrease the variant.

The soundness of the verification conditions in Fig. 5 is supported by the following theorem.

Theorem 1 (Soundness of verification). *The following statement holds.*

$$\forall\delta_{\mathrm{u}}^{as_1},\delta_{\mathrm{a}}^{as_2}: \mathit{VC\_B}\wedge\mathit{VC\_R}\wedge\mathit{VC\_U}(\delta_{\mathrm{u}}^{as_1})\wedge\mathit{VC\_A}(\delta_{\mathrm{a}}^{as_2}) \;\Rightarrow\; \mathit{SL}(\delta_{\mathrm{u}}^{as_1},\delta_{\mathrm{a}}^{as_2},a,c)$$

The theorem can be used to show the satisfaction of strategy-aware liquidity for all adversary strategies or for a class C of adversary strategies. To achieve this, we define a user strategy $\delta_{\mathrm{u}}^{as_1}$ that could potentially depend on an arbitrary adversary strategy $\delta_{\mathrm{a}}^{as_2}$ (potentially restricted in the class C) and then establish the verification conditions $\mathit{VC\_B}$, $\mathit{VC\_R}$, $\mathit{VC\_U}$, and $\mathit{VC\_A}$. The validity of these verification conditions implies the satisfaction of strategy-aware liquidity by the smart contract c. Furthermore, the strategy $\delta_{\mathrm{u}}^{as_1}$ reveals how the honest users can act to ensure the intended transfer of funds out of the contract.

We sketch the proof of Theorem 1 below. For this proof, the following lemma is a key intermediate result.

Lemma 2. *The following statement holds.*

$$\forall \sigma_{\mathrm{ref}}, \pi, \delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2} :$$
$$\begin{pmatrix} \mathcal{I}(\sigma_{\mathrm{ref}}, \mathit{last}(\pi)) \wedge \mathcal{V}(\sigma_{\mathrm{ref}}, \mathit{last}(\pi)) \in X \\ \wedge\ \mathit{VC\_B} \wedge \mathit{VC\_U}(\delta_{\mathrm{u}}^{as_1}) \wedge \mathit{VC\_A}(\delta_{\mathrm{a}}^{as_2}) \end{pmatrix} \Rightarrow \mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma_{\mathrm{ref}}, a, c)$$

Proof sketch. The proof of this lemma is by well-founded induction on the value x of $\mathcal{V}(\sigma_{\mathrm{ref}}, \mathit{last}(\pi))$. If $\mathit{min}(x)$ holds, then we can derive $\phi_{\mathrm{bal}}(\sigma_{\mathrm{ref}}, \mathit{last}(\pi), a, c)$ using $\mathit{VC\_B}$. Hence, we derive $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma_{\mathrm{ref}}, a, c)$ using the rule [U-BASE]. Suppose $\neg \mathit{min}(x)$ holds. Then, using the condition $\mathit{VC\_U}(\delta_{\mathrm{u}}^{as_1})$, we derive $\delta_{\mathrm{u}}^{as} \vdash \pi \to \pi'$, $\mathcal{I}(\sigma_{\mathrm{ref}}, \mathit{last}(\pi'))$, $\mathcal{V}(\sigma_{\mathrm{ref}}, \mathit{last}(\pi')) = x'$, and $x' \prec_{\mathrm{vc}} x$ for some π' and x'. Suppose $\phi_{\mathrm{bal}}(\sigma_{\mathrm{ref}}, \mathit{last}(\pi'), a, c)$ holds. Then, using the rule [A-BASE] and then the rule [U-TRAN], we derive $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma_{\mathrm{ref}}, a, c)$. On the other hand, suppose $\neg\phi_{\mathrm{bal}}(\sigma_{\mathrm{ref}}, \mathit{last}(\pi'), a, c)$ holds. Then, we have $\neg \mathit{min}(x')$ using $\mathit{VC\_B}$. Using the condition $\mathit{VC\_A}(\delta_{\mathrm{a}}^{as_2})$, we have that for all π'', if $\delta_{\mathrm{a}}^{as} \vdash \pi' \to^* \pi''$, then $\mathcal{I}(\sigma_{\mathrm{ref}}, \mathit{last}(\pi''))$, $\mathcal{V}(\sigma_{\mathrm{ref}}, \mathit{last}(\pi'')) = x''$ and $x'' \preceq_{\mathrm{vc}} x'$ for some x''. Using the transitivity of $\prec_{\mathrm{vc}}$, we have $x'' \prec_{\mathrm{vc}} x$. Hence, using the induction hypothesis, we have $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi'', \sigma_{\mathrm{ref}}, a, c)$. Using the rule [A-INTF] and then the rule [U-TRAN], we derive $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma_{\mathrm{ref}}, a, c)$. ☐

To establish Theorem 1, we assume $\mathit{init\text{-}ctr}(a, c, \sigma)$ and $(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash [\sigma] \rightsquigarrow \pi : \mathrm{u\_nxt}$. From the latter, we are able to derive $\sigma \to^* \mathit{last}(\pi)$. Hence, using the condition $\mathit{VC\_R}$, we have $\mathcal{I}(\mathit{last}(\pi), \mathit{last}(\pi))$ and $\mathcal{V}(\mathit{last}(\pi), \mathit{last}(\pi)) \in X$. Then, using Lemma 2, we derive $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \mathit{last}(\pi), a, c)$. This reasoning shows $\mathit{SL}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, a, c)$ and establishes Theorem 1.

6.2 Method for Refutation

The refutation of strategy-aware liquidity for the smart contract c at address a can be performed by a direct proof of $\neg \mathit{SL}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, a, c)$. This would involve establishing the negation of the predicate *usl* that supports the definition of strategy-aware liquidity. The predicate *usl* is defined by mutual induction with the predicate *asl*, but it would be cumbersome to set up an inductive proof to refutate strategy-aware liquidity for each new smart contract. We propose sound refutation conditions to simplify the refutation proof.

The refutation conditions are based on two invariants, $\mathcal{I}_1 \in \Sigma \times \Sigma \to \mathit{Prop}$ and $\mathcal{I}_2 \in \Sigma \times \Sigma \to \mathit{Prop}$. These conditions are presented in Fig. 6. The condition $\mathit{RC\_B}$ says for two arbitrary world states σ and σ', if either invariant holds, then the condition $\phi_{\mathrm{bal}}(\sigma, \sigma', a, c)$ does not hold. Hence, in any blockchain state σ' satisfying $\mathcal{I}_1$ or $\mathcal{I}_2$ with a reference state σ, the intended decrease of balance for the target contract is not achieved by the honest users. The condition $\mathit{RC\_R}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2})$ says that there exists a trace consisting of alternating execution by the adversary and the honest users such that the last state of the trace satisfies the invariant $\mathcal{I}_1$. The condition $\mathit{RC\_U}(\delta_{\mathrm{u}}^{as})$ says that starting from a blockchain state satisfying $\mathcal{I}_1$, no matter how the honest users act according to their strategy δ_{u}^{as}, the resulting blockchain state satisfies $\mathcal{I}_2$. The condition $\mathit{RC\_A}(\delta_{\mathrm{a}}^{as})$ says that starting from a blockchain state satisfying $\mathcal{I}_2$, the adversary is able to update the blockchain state such that $\mathcal{I}_1$ is again satisfied.

$$\begin{aligned}
RC\_B &:= \forall \sigma, \sigma' : (\mathcal{I}_1(\sigma,\sigma') \vee \mathcal{I}_2(\sigma,\sigma')) \Rightarrow \neg\phi_{\mathrm{bal}}(\sigma,\sigma',a,c)\\
RC\_R(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) &:= \forall \sigma_0 : \mathit{init\text{-}ctr}(a,c,\sigma_0) \Rightarrow\\
&\qquad \exists \pi : (\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash [\sigma_0] \rightsquigarrow \pi : \mathrm{u\_nxt} \wedge \mathcal{I}_1(\mathit{last}(\pi), \mathit{last}(\pi))\\
RC\_U(\delta_{\mathrm{u}}^{as}) &:= \forall \sigma, \pi, \pi' : \mathcal{I}_1(\sigma, \mathit{last}(\pi)) \wedge \delta_{\mathrm{u}}^{as} \vdash \pi \to \pi' \Rightarrow \mathcal{I}_2(\sigma, \mathit{last}(\pi'))\\
RC\_A(\delta_{\mathrm{a}}^{as}) &:= \forall \sigma, \pi : \mathcal{I}_2(\sigma, \mathit{last}(\pi)) \Rightarrow \exists \pi' : \delta_{\mathrm{a}}^{as} \vdash \pi \to^* \pi' \wedge \mathcal{I}_1(\sigma, \mathit{last}(\pi'))
\end{aligned}$$

Fig. 6. The refutation conditions for strategy-aware liquidity

The soundness of the refutation conditions in Fig. 6 is supported by the following theorem.

Theorem 2 (Soundness of refutation). *The following statement holds.*

$$\forall \delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2} : RC\_B \wedge RC\_R(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \wedge RC\_U(\delta_{\mathrm{u}}^{as_1}) \wedge RC\_A(\delta_{\mathrm{a}}^{as_2}) \Rightarrow \\ \neg SL(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, a, c)$$

This theorem can be used to show that strategy-aware liquidity is violated by a smart contract irrespective of the honest users' strategy. To achieve this, we define an attacker strategy $\delta_{\mathrm{a}}^{as_2}$ that could potentially depend on an arbitrary strategy $\delta_{\mathrm{u}}^{as_1}$ of the honest users, and then establish the refutation conditions RC_B, RC_R, RC_U, and RC_A. The validity of these refutation conditions implies the violation of strategy-aware liquidity. Furthermore, the definition of the strategy $\delta_{\mathrm{a}}^{as_2}$ explains the liquidity vulnerability of the target smart contract.

For the proof of Theorem 2, the lemma below is a key intermediate result.

Lemma 3. *The following statement holds.*

$$\begin{aligned}
&\forall \delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma_{\mathrm{ref}}, a, c :\\
&\left(\begin{array}{l} \mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma_{\mathrm{ref}}, a, c) \Rightarrow \\ (RC\_B \wedge RC\_U(\delta_{\mathrm{u}}^{as_1}) \wedge RC\_A(\delta_{\mathrm{a}}^{as_2}) \wedge \mathcal{I}_1(\sigma_{\mathrm{ref}}, \mathit{last}(\pi)) \Rightarrow \mathsf{false}) \end{array} \right) \wedge\\
&\left(\begin{array}{l} \mathit{asl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \sigma_{\mathrm{ref}}, a, c) \Rightarrow \\ (RC\_B \wedge RC\_U(\delta_{\mathrm{u}}^{as_1}) \wedge RC\_A(\delta_{\mathrm{a}}^{as_2}) \wedge \mathcal{I}_2(\sigma_{\mathrm{ref}}, \mathit{last}(\pi)) \Rightarrow \mathsf{false}) \end{array} \right)
\end{aligned}$$

This lemma can be proven by mutual induction on *usl* and *asl*. We omit the details about this proof.

To establish Theorem 2, we show that under the assumption of $\mathit{init\text{-}ctr}(a, c, \sigma)$, there exists a trace π, such that $(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash [\sigma] \rightsquigarrow \pi : \mathrm{u\_nxt}$ but not $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \mathit{last}(\pi), a, c)$. Using $RC\_R(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2})$, there exists π such that $(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash [\sigma] \rightsquigarrow \pi : \mathrm{u\_nxt}$ and $\mathcal{I}_1(\mathit{last}(\pi), \mathit{last}(\pi))$. Hence, using Lemma 3, we deduce that the condition $\mathit{usl}(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}, \pi, \mathit{last}(\pi), a, c)$ does not hold.

7 Mechanization and Evaluation

We start by discussing a key issue addressed in mechanizing the theories about strategy-aware liquidity based on the ConCert framework [16]. We then describe how the proof methods are applied to verify or refute strategy-aware liquidity for a few smart contract examples in the Rocq prover.

7.1 Mechanization in a Proof Assistant

Our definition of strategy-aware liquidity is based on the relation $\cdot \xrightarrow{\beta} \cdot$ (see Section 3.1) that captures the execution of a complete transaction. This treatment helps reason about the liquidity guarantees enjoyed by the blockchain users initiating these transactions. The definition also articulates that the intended decrease of the target contract's balance should be achieved at the boundary of transactions — as opposed to in the middle of a transaction.

The ConCert framework defines the relation $(\cdot, \beta) \Downarrow \cdot$ that captures the evaluation of a single action. If a transaction is triggered by an action β, further actions may be generated and executed after the evaluation of β to complete the transaction. For instance, if a transaction is triggered by an action β_{wth} that calls the function used to withdraw funds in the contract c_{fm} in Section 2, then the action β_{trsf} that transfers funds from the contract to the caller will be generated and evaluated as the result of evaluating β_{wth}.

In the Rocq prover, we formally define the transaction execution relation $\cdot \xrightarrow{\beta} \cdot$ based on the action evaluation relation $(\cdot, \beta) \Downarrow \cdot$ of ConCert. We omit the details for space reasons. After mechanizing the definitions of strategy-aware liquidity and basic liquidity, we have mechanized the proofs of the propositions and theorems. These mechanized proofs are available in electronic form [6].

7.2 Verification and Refutation Examples

The Motivating Examples. We verify that the smart contract c_{fm} in Section 2 satisfies strategy-aware liquidity under any adversary strategy, as long as the honest users follow the strategy $\delta_{\text{fm-u}}^{\{usr,adm\}}$ in Example 1. This verification is performed using the proof method of Section 6.1.

The contract c_{fm} goes through several status changes for the benign user, until the benign user is able to withdraw funds from the contract. We reflect these status changes by the decrease of the variant. Since the relevant statuses for c_{fm} are simple, we define the variant to give optional natural numbers, and define the well-founded order $\prec_{\text{vc}}$ between the results to be the "less than" order of natural numbers. The definition of the variant is given below.

$$\mathcal{V}_{\text{fm}}(\sigma, \sigma') := \begin{cases} 3 & \text{if } \mathit{ctr\text{-}st}(\sigma', a).\mathit{status}(\mathit{usr}) = \bot \wedge \mathit{balance}(\sigma', a) > 0 \\ 2 & \text{if } \mathit{ctr\text{-}st}(\sigma', a).\mathit{status}(\mathit{usr}) = \mathit{requested} \\ 1 & \text{if } \mathit{ctr\text{-}st}(\sigma', a).\mathit{status}(\mathit{usr}) = \mathit{approved} \\ 0 & \text{if } \mathit{ctr\text{-}st}(\sigma', a).\mathit{status}(\mathit{usr}) = \bot \wedge \mathit{balance}(\sigma', a) \le 0 \\ \bot & \text{otherwise} \end{cases}$$

We define the invariant $\mathcal{I}_{\text{fm}}$ to say that the administrator recorded in the state of the contract c_{fm} is the user designated as the administrator upon the initialization of c_{fm}. This reflects that the adversary cannot usurp the authority of the administrator and cause state changes that increase the variant (e.g., rejecting the benign user's withdrawal request by changing its status from *requested* to $\bot$). A special case of this definition is that $\mathcal{V}_{\text{fm}}(\sigma, \sigma') = \bot$ if $\mathit{ctr\text{-}st}(\sigma', a) = \bot$ — if no smart contract exists yet at the address a in the blockchain state σ'.

The verification result for the smart contract c_{fm} is the following theorem.

Theorem 3 (Verification result for c_{fm}). *For any adversary strategy δ_{a}^{as}, it holds that $SL(\delta_{\mathrm{fm}}^{\{usr,adm\}}, \delta_{\mathrm{a}}^{as}, a, c_{\mathrm{fm}})$, where a is the address of c_{fm}.*

Hence, it is ascertained that the smart contract c_{fm} satisfies liquidity requirements under the potential influence of an arbitrary adversary. Furthermore, the strategy $\delta_{\mathrm{fm}}^{\{usr,adm\}}$ provides guidance for the honest users on how to achieve their intended withdrawal of funds from the contract.

We also show that the flawed contract $c_{\mathrm{fm}}^{\times}$ in Section 2 violates strategy-aware liquidity if the adversary follow the strategy $\delta_{\text{fm-a}}^{as}$ (as given below), irrespective of how the honest users behave.

$$\delta_{\text{fm-a}}^{as}(\pi) := \begin{cases} \{\beta_{\mathrm{ini}}\} & \text{if } \mathit{ctr\text{-}st}(\mathit{last}(\pi), a).\mathit{status}(\mathit{usr}) = \mathit{requested} \vee \\ & \quad \mathit{ctr\text{-}st}(\mathit{last}(\pi), a).\mathit{status}(\mathit{adm}) = \mathit{requested} \\ \emptyset & \text{otherwise} \end{cases}$$

An adversary following this strategy will re-initialize the smart contract $c_{\mathrm{fm}}^{\times}$ if an honest user has just requested for withdrawal of funds. This will clear the recorded status *requested* for the honest users, essentially rejecting their requests.

The refutation of strategy-aware liquidity for $c_{\mathrm{fm}}^{\times}$ is performed using the refutation method of Section 6.2. We instantiate the invariant $\mathcal{I}_1$ to express that the status of any user is $\perp$, and the contract $c_{\mathrm{fm}}^{\times}$ has positive balance. We instantiate the invariant $\mathcal{I}_2$ to express that the status of the benign user and the initially designated administrator could be either $\perp$ or *requested*, the status for any other user is $\perp$, and the contract $c_{\mathrm{fm}}^{\times}$ has positive balance. The refutation result for the smart contract $c_{\mathrm{fm}}^{\times}$ is the following theorem.

Theorem 4 (Refutation result for $c_{\mathrm{fm}}^{\times}$). *For any strategy $\delta_{\mathrm{u}}^{\{usr,adm\}}$ of the honest users, $\neg SL(\delta_{\mathrm{u}}^{\{usr,adm\}}, \delta_{\text{fm-a}}^{as}, a, c_{\mathrm{fm}}^{\times})$ holds, where a is the address of $c_{\mathrm{fm}}^{\times}$.*

The adversary strategy $\delta_{\text{fm-a}}^{as}$ explains the reason that $c_{\mathrm{fm}}^{\times}$ suffers from liquidity problems. According to Proposition 1, $c_{\mathrm{fm}}^{\times}$ also violates strategy-aware liquidity for any adversary strategy that suggests more candidate actions than $\delta_{\text{fm-a}}^{as}$ does for some of the histories. However, attacks following these alternative strategies would be more expensive than attacks following $\delta_{\text{fm-a}}^{as}$ are.

Further Verification and Refutation Examples. We consider two further smart contracts that handle financial transactions. The first is an escrow contract and the second is a honeypot contract.

Escrow Contract. We consider an escrow contract that is similar in spirit to [2]. The escrow contract holds an agreed-upon amount of digital currency for the purchase of a product. If the seller ships the product and the buyer accepts it, the digital currency in the contract will be transfered to the seller. Either the buyer or the seller may trigger a dispute in case of disagreement. The dispute can be resolved by an arbitrator, either completing the payment to the seller, or refunding the cryptocurrency to the buyer.

We prove the satisfaction of strategy-aware liquidity by the escrow contract under any adversarial strategy, in case the adversary is the seller or the buyer. In doing so, we also identify the strategy of the honest users that ensures the

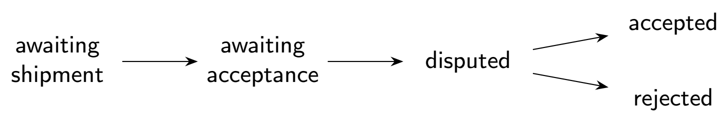

Fig. 7. The Well-founded Order based on the Statuses of the Escrow Contract

transfer of digital currency out of the contract. In applying the proof method of Section 6.1, we directly define the well-founded order $\prec_{\mathrm{vc}}$ on the statuses of the escrow contract instead of mapping the blockchain states to natural numbers first. This definition is illustrated in Fig. 7. In this figure, each node is labeled with the status of the escrow contract. A path from x to x' represents $x' \preceq_{\mathrm{vc}} x$. Hence, the use of a non-linear order as enabled by our verification method provides convenience for the liquidity verification of the escrow contract.

Honeypot Contract. We consider a honeypot contract [3, 36]. The contract allows a user to set a password for manipulating the funds in the contract, under the condition that the password is not locked and the user transfers at least 1 unit of funds in the action to set the password (see the transition labeled *setPass h*). If the password is not locked, and a user provides the hash of the current password, then the user may lock the password (see the transition labeled *lockPass h*). If a user is able to provide the current password, then the user may withdraw all the funds stored in the contract (see the transitions labeled *getGift p*).

A user could attempt to profit from the honeypot contract by setting the password to their own, and withdraw all the funds using this password. However, the owner of the contract can freeze the funds of all users by updating the password and then locking it. We prove that the honeypot contract violates strategy-aware liquidity under an adversary strategy that updates the password and then locks it. In applying our refutation method, we instantiate both invariants, $\mathcal{I}_1$ and $\mathcal{I}_2$, with the condition that the balance of the contract is positive, and the password is the one provided by the adversary and is already locked.

Common Aspects, and Summary of the Proof Efforts. The verification condition VC_R for strategy-aware liquidity asserts what should be satisfied by all blockchain states that are reachable from a blockchain state where a smart contract has just been initialized. The proof of such a condition is supported by the proof tactic of *contract induction* in the ConCert framework. The application of this proof tactic improves the level of automation in establishing VC_R in our verification examples.

Both the verification condition VC_A and the refutation condition RC_U concern the effects of arbitrary transactions. Although our smart contract examples feature relatively simple business logics, the reasoning required to establish VC_A and RC_U can be non-trivial. This is because each transaction may amount to a series of actions. For instance, the execution of the action β_{wth} on the smart contract c_{fm} generates the action β_{trf}, which transfers the amount withdrawn to the account performing β_{wth}. Moreover, each call to a smart contract may be re-entrant. We identify intermediate invariants preserved by the execution of a single action to support the formal proof of VC_A and RC_U.

For the verification or refutation of strategy-aware liquidity for the smart contracts c_{fm}, $c_{\text{fm}}^{\times}$, the escrow contract, and the honeypot contract, the ratio between the LOC for contract formalization and the LOC for proof code is approximately $1 : 8.6$. The proof methods provide clear guidance on how to approach and organize these verification and refutation tasks.

8 Related Work

Since liquidity issues heavily undermine the security of smart contracts, substantial recent research efforts in formal and semi-formal analysis and verification [34] have been dedicated to addressing liquidity concerns.

The Securify tool [37] is capable of verifying liquidity properties by statically checking compliance patterns in internal representations of smart contract code. The Solvent tool [12] verifies a class of liquidity properties through bounded model checking and predicate abstraction. In the target property, the maximal number of transactions that are needed to implement the desired transfer of assets can be specified. Neither work considers strategies as we do.

Bartoletti and Zunino [15] propose a liquidity property that is aware of the strategies of users and an adversary for Bitcoin contracts written in their BitML language [14]. They develop model checking techniques for the verification of the proposed property. They also extend their development to support recursion in BitML [13]. Compared with our present work, their property design does not address situations where an adversary could interfere with the process as honest users attempt to transfer funds out of the target smart contract.

The MAIAN tool [29] detects the bugs in smart contracts with symbolic execution. The detection covers the bugs of locked funds. There are alternative tools that detect the same kind of bugs through static and dynamic analysis [18, 33, 21, 24] or fuzzing [27, 35]. These developments do not consider formal notions of liquidity or offer approaches to formal verification.

Other developments help obtain liquidity guarantees through built-in mechanisms of programming languages. The semantics of the Marlowe language ensures that the unspent money is returned to users by the end of any smart contract's execution [22]. This mechanism helps provide liquidity guarantees for the Cardano blockchain which is UTxO-based (like Bitcoin). The Stipula language is equipped with types that support algorithms to verify a notion of liquidity and a weaker notion of k-separate liquidity [23]. These two developments do not consider user strategies in the liquidity requirement.

Our work differs from existing work in defining a strategy-aware liquidity property for the account-based blockchain model. A second difference is that we formally establish meta-theoretical properties of the proposed liquidity property, including monotonicity and connection with strategy-less liquidity. A third difference is that we support a deductive approach to the verification and refutation of the proposed liquidity property via sound proof methods with mechanization.

9 Conclusion

We consider the problem of verifying that a group of users is not exposed to the risk of locked funds when using a smart contract in an adversarial environment. We address this problem by formulating and enforcing liquidity properties. Technically, we define a liquidity property that is aware of the strategies followed by the users and the environment for account-based blockchains like the Ethereum. We establish theoretical properties of the proposed liquidity property, including its relationship with strategy-less liquidity, and the relationship between its strength and the strength of the environmental strategy. We devise sound proof methods for the verification and refutation of the property. We mechanize the theories and proof methods in the Rocq prover, and apply the proof methods to smart contract examples. Our work addresses a more powerful adversary model than existing work on liquidity analysis and verification does, allowing the adversary to interrupt the honest users' process of transferring funds out of a smart contract. Our deductive approach to liquidity verification complements existing formal approaches based on model checking and static analysis.

Currently, the strategy for the honest users (resp. the adversary) needs to be manually specified in a verification (resp. refutation) of strategy-aware liquidity. A potential direction for future work is the synthesis of these strategies. Another future direction is the further improvement of proof automation to facilitate the verification or refutation of strategy-aware liquidity for complex smart contracts.

Acknowledgments. This work was partially supported by the National Natural Science Foundation of China (62272322, 62272323, 62372312, 62372311).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Critical bug identified in 88mph awarded with $42,069 bounty. https://iosiro.com/blog/88mph-bug-bounty-post-mortem, accessed: 2025-05-31
2. Decentralized escrow. https://github.com/ShivamKumar2002/30days-of-solidity-web3compass/blob/main/day-24-decentralized-escrow/DecentralizedEscrow.sol, accessed: 2025-10-31
3. Etherscan — Gift_1_ETH contract. https://etherscan.io/address/0xd8993f49f372bb014fb088eabec95cfdc795cbf6, accessed: 2025-06-05
4. Etherscan — Rubixi contract. https://etherscan.io/address/0xe82719202e5965Cf5D9B6673B7503a3b92DE20be, accessed: 2025-05-31
5. How Akutar NFT loses 34M USD. https://blocksecteam.medium.com/how-akutar-nft-loses-34m-usd-60d6cb053dff, accessed: 2025-06-10
6. Mechanization of the development in the Rocq prover, https://github.com/lixm/liquidity
7. Nexus Mutual — claim assessment. https://docs.nexusmutual.io/protocol/claims-assessment, accessed: 2025-05-27

8. Parity wallet hack: What, when and how? https://medium.com/@web3author/parity-wallet-hack-demystified-all-you-need-to-know-91b8dcb5b81, accessed: 2025-06-10
9. ROCQ. https://rocq-prover.org/, accessed: 2025-08-25
10. Annenkov, D., Nielsen, J.B., Spitters, B.: ConCert: A smart contract certification framework in Coq. In: Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP). pp. 215–228 (2020)
11. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts (SoK). In: Proceedings of 6th International Conference on Principles of Security and Trust (POST), Held as Part of ETAPS. pp. 164–186 (2017)
12. Bartoletti, M., Ferrando, A., Lipparini, E., Malvone, V.: Solvent: Liquidity verification of smart contracts. In: International Conference on Integrated Formal Methods (IFM). pp. 256–266 (2024)
13. Bartoletti, M., Lande, S., Murgia, M., Zunino, R.: Verifying liquidity of recursive Bitcoin contracts. Logical Methods in Computer Science **18** (2022)
14. Bartoletti, M., Zunino, R.: BitML: a calculus for Bitcoin smart contracts. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 83–100 (2018)
15. Bartoletti, M., Zunino, R.: Verifying liquidity of Bitcoin contracts. In: International Conference on Principles of Security and Trust (POST), Held as Part of ETAPS. pp. 222–247 (2019)
16. Bertot, Y., Castéran, P.: Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions. Springer Science & Business Media (2013)
17. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)
18. Feist, J., Grieco, G., Groce, A.: Slither: a static analysis framework for smart contracts. In: IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). pp. 8–15 (2019)
19. Ferariu, T., Wadler, P., Melkonian, O.: Validity, liquidity, and fidelity: Formal verification for smart contracts in Cardano. In: 6th International Workshop on Formal Methods for Blockchains (FMBC). pp. 1–21 (2025)
20. Kao, H.T., Chitra, T., Chiang, R., Morrow, J.: An analysis of the market risk to participants in the compound protocol. In: Third International Symposium on Foundations and Applications of Blockchain (FAB). pp. 1–10 (2020)
21. Khan, Z.A., Namin, A.S.: Dynamic analysis for the detection of locked ether smart contracts. In: IEEE International Conference on Big Data (BigData). pp. 2466–2472 (2023)
22. Lamela Seijas, P., Nemish, A., Smith, D., Thompson, S.: Marlowe: implementing and analysing financial contracts on blockchain. In: Financial Cryptography and Data Security: FC International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC. pp. 496–511 (2020)
23. Laneve, C.: Liquidity analysis in resource-aware programming. Journal of Logical and Algebraic Methods in Programming **135**, 100889 (2023)
24. Li, X., Chen, T., Luo, X., Wang, C.: CLUE: Towards discovering locked cryptocurrencies in Ethereum. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC). pp. 1584–1587 (2021)
25. Li, X., Nielson, F., Nielson, H.R.: Factorization of behavioral integrity. In: 20th European Symposium on Research in Computer Security (ESORICS). pp. 500–519 (2015)
26. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)

27. Nguyen, T.D., Pham, L.H., Sun, J., Lin, Y., Minh, Q.T.: sFuzz: An efficient adaptive fuzzer for solidity smart contracts. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE). pp. 778–788 (2020)
28. Nielsen, J.B., Spitters, B.: Smart contract interactions in Coq. In: International Symposium on Formal Methods (FM). pp. 380–391. Springer (2019)
29. Nikolić, I., Kolluri, A., Sergey, I., Saxena, P., Hobor, A.: Finding the greedy, prodigal, and suicidal contracts at scale. In: Proceedings of the 34th annual computer security applications conference. pp. 653–663 (2018)
30. Rafnsson, W., Hedin, D., Sabelfeld, A.: Securing interactive programs. In: 25th IEEE Computer Security Foundations Symposium (CSF). pp. 293–307 (2012)
31. Szabo, N.: Smart contracts: Building blocks for digital markets. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html (1996)
32. Tadelis, S.: Game Theory: An Introduction. Princeton University Press (2013)
33. Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E., Alexandrov, Y.: Smartcheck: Static analysis of Ethereum smart contracts. In: Proceedings of the 1st international workshop on emerging trends in software engineering for blockchain. pp. 9–16 (2018)
34. Tolmach, P., Li, Y., Lin, S., Liu, Y., Li, Z.: A survey of smart contract formal specification and verification. ACM Computing Surveys **54**(7), 148:1–148:38 (2022)
35. Torres, C.F., Iannillo, A.K., Gervais, A., State, R.: Confuzzius: A data dependency-aware hybrid fuzzer for smart contracts. In: IEEE European Symposium on Security and Privacy (EuroS&P). pp. 103–119 (2021)
36. Torres, C.F., Steichen, M., State, R.: The art of the scam: Demystifying honeypots in Ethereum smart contracts. In: 28th USENIX Security Symposium. pp. 1591–1607 (2019)
37. Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., Vechev, M.: Securify: Practical security analysis of smart contracts. In: Proceedings of ACM SIGSAC conference on computer and communications security (CCS). pp. 67–82 (2018)
38. Werner, S., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., Knottenbelt, W.J.: SoK: Decentralized finance (DeFi). In: Proceedings of the 4th ACM Conference on Advances in Financial Technologies (AFT). pp. 30–46 (2022)
39. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. https://ethereum.github.io/yellowpaper/paper.pdf
40. Yao, J., Li, S., Yin, X.: Sensor deception attacks against security in supervisory control systems. Automatica **159**, 111330 (2024)

Appendices

We have mechanized [6] the proofs of all the theoretical results and verification results in this paper in the Rocq prover. To improve the accessibility of the proofs, we also provide the formal proofs of Proposition 1 in Appendix A and provide the formal proofs of Proposition 2 in Appendix B.

A Proof of Proposition 1

We establish the following auxiliary result concerning the relative strength of adversary strategies and the existence of an alternating execution sequence by the adversary and the honest users.

Lemma 4. *If* $\delta_1^{as} \preceq_{\text{stg}} \delta_2^{as}$ *holds, and it can be established that* $({\delta_{\text{u}}^{as}}', \delta_1^{as}) \vdash \pi \rightsquigarrow \pi' : nxt$*, then it can be established that* $({\delta_{\text{u}}^{as}}', \delta_2^{as}) \vdash \pi \rightsquigarrow \pi' : nxt$*.*

Proof. We perform an induction on the derivation of

$$({\delta_{\text{u}}^{as}}', \delta_1^{as}) \vdash \pi \rightsquigarrow \pi' : nxt \tag{1}$$

- Suppose (1) is derived using the rule [REFL]. Then $\pi = \pi'$ holds. Hence, $({\delta_{\text{u}}^{as}}', \delta_2^{as}) \vdash \pi \rightsquigarrow \pi' : nxt$ can be derived using the rule [REFL].
- Suppose (1) is derived using the rule [U-EXE]. Then, we have $nxt = \text{a\_nxt}$, and

$$({\delta_{\text{u}}^{as}}', \delta_1^{as}) \vdash \pi \rightsquigarrow \pi'' : \text{u\_nxt} \tag{2}$$

$${\delta_{\text{u}}^{as}}' \vdash \pi'' \rightarrow \pi' \tag{3}$$

  Using (2) and the induction hypothesis, we have $({\delta_{\text{u}}^{as}}', \delta_2^{as}) \vdash \pi \rightsquigarrow \pi'' : \text{u\_nxt}$. Hence, using (3) and the rule [U-EXE], we have $({\delta_{\text{u}}^{as}}', \delta_2^{as}) \vdash \pi \rightsquigarrow \pi' : \text{a\_nxt}$.
- Suppose (1) is derived using the rule [A-EXE]. Then, we have $nxt = \text{u\_nxt}$, and

$$({\delta_{\text{u}}^{as}}', \delta_1^{as}) \vdash \pi \rightsquigarrow \pi'' : \text{a\_nxt} \tag{4}$$

$$\delta_1^{as} \vdash \pi'' \rightarrow^* \pi' \tag{5}$$

  Using (4) and the induction hypothesis, we have

$$({\delta_{\text{u}}^{as}}', \delta_2^{as}) \vdash \pi \rightsquigarrow \pi'' : \text{a\_nxt} \tag{6}$$

  By induction on the derivation of (5) and using $\delta_1^{as} \preceq_{\text{stg}} \delta_2^{as}$, it can be derived that

$$\delta_2^{as} \vdash \pi'' \rightarrow^* \pi' \tag{7}$$

  Using (6), (7) and the rule [A-EXE], we have $({\delta_{\text{u}}^{as}}', \delta_2^{as}) \vdash \pi \rightsquigarrow \pi' : \text{u\_nxt}$.

The inductive reasoning above completes the proof of this lemma. □

We re-state Lemma 1 and prove it below.

Lemma 1 (restated). *If* $\delta_1^{as} \preceq_{\text{stg}} \delta_2^{as}$ *holds, and* $usl({\delta_{\text{u}}^{as}}', \delta_2^{as}, \pi, \sigma, a, c)$ *can be established, then* $usl({\delta_{\text{u}}^{as}}', \delta_1^{as}, \pi, \sigma, a, c)$ *can be established.*

Proof. According to the definition of the predicates *usl* and *asl*, the expression $usl({\delta_{\text{u}}^{as}}', \delta_2^{as}, \pi, \sigma, a, c)$ can be derived in one of the following three ways.

1. using the rule [U-BASE] — based on the condition $\phi_{\mathrm{bal}}(\sigma, \mathit{last}(\pi), a, c)$
2. using the rules [U-TRAN] and [A-BASE] — based on the conditions

$$\delta_{\mathrm{u}}^{as'} \vdash \pi \to \pi' \tag{8}$$

$$\phi_{\mathrm{bal}}(\sigma, \mathit{last}(\pi'), a, c) \tag{9}$$

3. using the rules [U-TRAN] and [A-INTF] — based on the conditions

$$\delta_{\mathrm{u}}^{as'} \vdash \pi \to \pi' \tag{10}$$

$$\forall \pi'' : \delta_{\mathrm{a}}^{as_2} \vdash \pi' \to^* \pi'' \Rightarrow \mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_{\mathrm{a}}^{as_2}, \pi'', \sigma, a, c) \tag{11}$$

$$\neg \phi_{\mathrm{bal}}(\sigma, \mathit{last}(\pi'), a, c) \tag{12}$$

We proceed to show $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_1^{as}, \pi, \sigma, a, c)$ by induction on the derivation of $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_2^{as}, \pi, \sigma, a, c)$.

– Suppose $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_2^{as}, \pi, \sigma, a, c)$ is derived in the first way. Then, using the rule [U-BASE] we can derive $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_1^{as}, \pi, \sigma, a, c)$.
– Suppose $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_2^{as}, \pi, \sigma, a, c)$ is derived in the second way. Then, using the conditions (8) and (9), and applying the rules [U-TRAN] and [A-BASE], we can derive $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_1^{as}, \pi, \sigma, a, c)$.
– Suppose $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_2^{as}, \pi, \sigma, a, c)$ is derived in the third way.
  Pick an arbitrary trace π''' and assume $\delta_{\mathrm{a}}^{as_1} \vdash \pi' \to^* \pi'''$. By induction on the derivation of $\delta_{\mathrm{a}}^{as_1} \vdash \pi' \to^* \pi'''$ and using the condition $\delta_1^{as} \preceq_{\mathrm{stg}} \delta_2^{as}$, we can derive $\delta_{\mathrm{a}}^{as_2} \vdash \pi' \to^* \pi'''$. Hence, using (11), we have $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_{\mathrm{a}}^{as_2}, \pi''', \sigma, a, c)$. Hence, using the induction hypothesis, we have $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_{\mathrm{a}}^{as_1}, \pi''', \sigma, a, c)$. From the above reasoning, we have

$$\forall \pi''' : \delta_{\mathrm{a}}^{as_1} \vdash \pi' \to^* \pi''' \Rightarrow \mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_{\mathrm{a}}^{as_1}, \pi''', \sigma, a, c) \tag{13}$$

  Using the conditions (10), (13), and (12), and applying the rules [U-TRAN] and [A-INTF], we can derive $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_1^{as}, \pi, \sigma, a, c)$.

The induction above completes the proof of this lemma. □

We re-state and prove the monotonicity result (Proposition 1) about strategy-aware liquidity below.

Proposition 1 (restated). *If $\delta_1^{as} \preceq_{\mathrm{stg}} \delta_2^{as}$ holds, and $\mathit{SL}(\delta_{\mathrm{u}}^{as'}, \delta_2^{as}, a, c)$ can be established, then $\mathit{SL}(\delta_{\mathrm{u}}^{as'}, \delta_1^{as}, a, c)$ can be established.*

Proof. We assume the following hypotheses in the theorem.

$$\delta_1^{as} \preceq_{\mathrm{stg}} \delta_2^{as} \tag{14}$$

$$\mathit{SL}(\delta_{\mathrm{u}}^{as'}, \delta_2^{as}, a, c) \tag{15}$$

Furthermore, we assume the following conditions hold and we proceed to show $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_{\mathrm{a}}^{as_1}, \pi, \mathit{last}(\pi), a, c)$ holds.

$$\mathit{init\text{-}ctr}(a, c, \sigma) \tag{16}$$

$$(\delta_{\mathrm{u}}^{as'}, \delta_{\mathrm{a}}^{as_1}) \vdash [\sigma] \rightsquigarrow \pi : \mathrm{u\_nxt} \tag{17}$$

Using (14), (17), and Lemma 4, we have $(\delta_{\mathrm{u}}^{as'}, \delta_{\mathrm{a}}^{as_2}) \vdash [\sigma] \rightsquigarrow \pi : \mathrm{u\_nxt}$. Hence, using (15) and (16), we have $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_{\mathrm{a}}^{as_2}, \pi, \mathit{last}(\pi), a, c)$. Therefore, using (14) and Lemma 1, we have $\mathit{usl}(\delta_{\mathrm{u}}^{as'}, \delta_{\mathrm{a}}^{as_1}, \pi, \mathit{last}(\pi), a, c)$. This completes the proof of Proposition 1. □

B Proof of Proposition 2

We establish two lemmas about the connection between (basic) transaction execution and strategy-driven transaction execution.

Lemma 5. *If* $(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash \pi \rightsquigarrow \pi' : \mathit{nxt}$ *can be derived, then we have* $\mathit{last}(\pi) \to^* \mathit{last}(\pi')$*.*

Proof. We perform an induction on the derivation of

$$(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash \pi \rightsquigarrow \pi' : \mathit{nxt} \tag{18}$$

- Suppose (18) is established using the rule [REFL]. Then we have $\pi = \pi'$. Hence, we have $\mathit{last}(\pi) \to^* \mathit{last}(\pi')$.
- Suppose (18) is established using the rule [U-EXE]. Then, we have the following conditions for some π''.

  $$(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash \pi \rightsquigarrow \pi'' : \mathrm{u\_nxt} \tag{19}$$
  $$\delta_{\mathrm{u}}^{as_1} \vdash \pi'' \to \pi' \tag{20}$$

  Using (19) and the induction hypothesis, we have $\mathit{last}(\pi) \to^* \mathit{last}(\pi'')$. Moreover, from (20) we have $\mathit{last}(\pi'') \xrightarrow{\beta} \mathit{last}(\pi')$ for some β. Ultimately, we have $\mathit{last}(\pi) \to^* \mathit{last}(\pi')$.
- Suppose (18) is established using the rule [A-EXE]. Then, we have the following conditions for some π''.

  $$(\delta_{\mathrm{u}}^{as_1}, \delta_{\mathrm{a}}^{as_2}) \vdash \pi \rightsquigarrow \pi'' : \mathrm{a\_nxt} \tag{21}$$
  $$\delta_{\mathrm{a}}^{as_2} \vdash \pi'' \to^* \pi' \tag{22}$$

  Using (21) and the induction hypothesis, we have $\mathit{last}(\pi) \to^* \mathit{last}(\pi'')$. Moreover, by inductive reasoning over the derivation of (22), we can derive $\mathit{last}(\pi'') \to^* \mathit{last}(\pi')$. Ultimately, we have $\mathit{last}(\pi) \to^* \mathit{last}(\pi')$.

The induction above completes the proof of this lemma. □

Lemma 6. *If is-complete*$(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}})$ *holds,* $\sigma \to^* \sigma'$ *can be derived, and* $\sigma = \mathit{last}(\pi)$*, then it holds that* $\exists \pi' : (\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}) \vdash \pi \rightsquigarrow \pi' : \mathrm{u\_nxt} \wedge \sigma' = \mathit{last}(\pi')$*.*

Proof. It can be shown that $\sigma \to^* \sigma'$ is inductively derivable using the following two rules (we omit the details here).

1. from $\sigma = \sigma'$, derive $\sigma \to^* \sigma'$
2. for some σ'', from $\sigma \to^* \sigma''$ and $\sigma'' \xrightarrow{\beta} \sigma'$, derive $\sigma \to^* \sigma'$

We perform an induction on the above derivation of $\sigma \rightarrow^* \sigma'$.

– Suppose $\sigma \rightarrow^* \sigma'$ is derived in the first way. Then, we have $\sigma = \sigma'$. Using the rule [REFL], we have $(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}) \vdash \pi \rightsquigarrow \pi : \mathrm{u\_nxt}$. From $\sigma = \mathit{last}(\pi)$ and $\sigma = \sigma'$, we have $\sigma' = \mathit{last}(\pi)$. Hence, there exists $\pi' = \pi$ such that $(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}) \vdash \pi \rightsquigarrow \pi' : \mathrm{u\_nxt}$ and $\sigma' = \mathit{last}(\pi')$.
– Suppose $\sigma \rightarrow^* \sigma'$ is derived in the second way. Using $\sigma \rightarrow^* \sigma''$ and the induction hypothesis, there exists π'' such that

$$(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}) \vdash \pi \rightsquigarrow \pi'' : \mathrm{u\_nxt} \tag{23}$$

$$\sigma'' = \mathit{last}(\pi'') \tag{24}$$

From $\sigma'' \xrightarrow{\beta} \sigma'$, we have $\mathit{orig}(\beta) \in Adr_{\mathrm{eoa}}$. Hence, using $\mathit{is\text{-}complete}(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}})$ and (24), we have $\beta \in \delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}(\pi'')$. Hence, using $\sigma'' \xrightarrow{\beta} \sigma'$ and (24), we have

$$\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}} \vdash \pi'' \rightarrow \pi''\hat{}[\beta]\hat{}[\sigma'] \tag{25}$$

Hence, using (23) and applying the rule [U-EXE], we have

$$(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}) \vdash \pi \rightsquigarrow \pi''\hat{}[\beta]\hat{}[\sigma'] : \mathrm{a\_nxt} \tag{26}$$

Moreover, we have $\delta_{\mathrm{a}}^{\emptyset} \vdash \pi''\hat{}[\beta]\hat{}[\sigma'] \rightarrow^* \pi''\hat{}[\beta]\hat{}[\sigma']$. Ultimately, applying the rule [A-EXE] gives $(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}) \vdash \pi \rightsquigarrow \pi''\hat{}[\beta]\hat{}[\sigma'] : \mathrm{u\_nxt}$. Therefore, there exists $\pi' = \pi''\hat{}[\beta]\hat{}[\sigma']$ such that $(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}) \vdash \pi \rightsquigarrow \pi' : \mathrm{u\_nxt}$ and $\sigma' = \mathit{last}(\pi')$.

The induction above completes the proof of this lemma. □

We next establish two lemmas about the connection between (basic) transaction execution and the predicate *usl*.

Lemma 7. *If* $\mathit{is\text{-}complete}(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}})$ *holds,* $\mathit{last}(\pi) \rightarrow^* \sigma'$ *can be derived, and* $\phi_{\mathrm{bal}}(\sigma_0, \sigma', a, c)$ *holds, then* $\mathit{usl}(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \sigma_0, a, c)$ *can be derived.*

Proof. We perform an induction on the derivation of $\mathit{last}(\pi) \rightarrow^* \sigma'$.

– Suppose $\mathit{last}(\pi) = \sigma'$. Then, using $\phi_{\mathrm{bal}}(\sigma_0, \sigma', a, c)$ and applying the rule [U-BASE], we have $\mathit{usl}(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \sigma_0, a, c)$.
– Suppose $\mathit{last}(\pi) \xrightarrow{\beta} \sigma''$ for some β and σ'', and $\sigma'' \rightarrow^* \sigma'$. From $\mathit{last}(\pi) \xrightarrow{\beta} \sigma''$, we have $\mathit{orig}(\beta) \in Adr_{\mathrm{eoa}}$. Hence, using $\mathit{is\text{-}complete}(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}})$, we have $\beta \in \delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}(\pi)$. Hence, using $\mathit{last}(\pi) \xrightarrow{\beta} \sigma''$, we have

$$\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}} \vdash \pi \rightarrow \pi\hat{}[\beta]\hat{}[\sigma''] \tag{27}$$

We proceed with a case analysis on whether $\phi_{\mathrm{bal}}(\sigma_0, \sigma'', a, c)$ or its negation holds.
  • Suppose $\phi_{\mathrm{bal}}(\sigma_0, \sigma'', a, c)$ holds. Then, we have

$$\mathit{asl}(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi\hat{}[\beta]\hat{}[\sigma''], \sigma_0, a, c) \tag{28}$$

  using the rule [A-BASE]. Using (27) and (28) and applying the rule [U-TRAN], we have $\mathit{usl}(\delta_{\mathrm{u}}^{Adr_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \sigma_0, a, c)$.

- Suppose $\neg\phi_{\mathrm{bal}}(\sigma_0, \sigma'', a, c)$ holds. According to the definition of strategies, there cannot be any action $\beta' \in \delta_{\mathrm{a}}^{\emptyset}(\pi \hat{} [\beta] \hat{} [\sigma''])$. Pick an arbitrary π'' and assume $\delta_{\mathrm{a}}^{\emptyset} \vdash \pi \hat{} [\beta] \hat{} [\sigma''] \rightarrow^* \pi''$. We then have $\pi'' = \pi \hat{} [\beta] \hat{} [\sigma'']$. Hence, using $\sigma'' \rightarrow^* \sigma'$, we have $\mathit{last}(\pi'') \rightarrow^* \sigma'$. Hence, using the induction hypothesis and $\phi_{\mathrm{bal}}(\sigma_0, \sigma', a, c)$, we have $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi'', \sigma_0, a, c)$. The above reasoning establishes
$$\forall \pi'' : \delta_{\mathrm{a}}^{\emptyset} \vdash \pi \hat{} [\beta] \hat{} [\sigma''] \rightarrow^* \pi'' \;\Rightarrow\; \mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi'', \sigma_0, a, c) \tag{29}$$
Hence, using $\neg\phi_{\mathrm{bal}}(\sigma_0, \sigma'', a, c)$, we have $\mathit{asl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi \hat{} [\beta] \hat{} [\sigma''], \sigma_0, a, c)$. Hence, using (27) and applying the rule [U-TRAN], we derive the condition $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \sigma_0, a, c)$.

The induction above completes the proof of this lemma. □

Lemma 8. *If $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \sigma_0, a, c)$ can be derived, then it holds that $\exists \sigma' : \mathit{last}(\pi) \rightarrow^* \sigma' \wedge \phi_{\mathrm{bal}}(\sigma_0, \sigma', a, c)$.*

Proof. According to the definition of the predicates usl and asl, the expression $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \sigma_0, a, c)$ can be derived in one of the following three ways.

1. using the rule [U-BASE] — based on the condition $\phi_{\mathrm{bal}}(\sigma_0, \mathit{last}(\pi), a, c)$
2. using the rules [U-TRAN] and [A-BASE] — based on the conditions
$$\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}} \vdash \pi \rightarrow \pi' \tag{30}$$
$$\phi_{\mathrm{bal}}(\sigma_0, \mathit{last}(\pi'), a, c) \tag{31}$$
3. using the rules [U-TRAN] and [A-INTF] — based on the conditions
$$\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}} \vdash \pi \rightarrow \pi' \tag{32}$$
$$\forall \pi'' : \delta_{\mathrm{a}}^{\emptyset} \vdash \pi' \rightarrow^* \pi'' \Rightarrow \mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi'', \sigma, a, c) \tag{33}$$
$$\neg\phi_{\mathrm{bal}}(\sigma, \mathit{last}(\pi'), a, c) \tag{34}$$

We proceed to show $\exists \sigma' : \mathit{last}(\pi) \rightarrow^* \sigma' \wedge \phi_{\mathrm{bal}}(\sigma_0, \sigma', a, c)$ by induction on the derivation of $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \sigma_0, a, c)$.

- Suppose $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \sigma_0, a, c)$ is derived in the first way. Then, we have $\phi_{\mathrm{bal}}(\sigma_0, \mathit{last}(\pi), a, c)$. Hence, there exists $\sigma' = \mathit{last}(\pi)$ such that $\mathit{last}(\pi) \rightarrow^* \sigma'$ and $\phi_{\mathrm{bal}}(\sigma_0, \sigma', a, c)$.
- Suppose $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \sigma_0, a, c)$ is derived in the second way. By (30), we have $\mathit{last}(\pi) \xrightarrow{\beta} \mathit{last}(\pi')$ for some β. Hence, by (31), there exists $\sigma' = \mathit{last}(\pi')$, such that $\mathit{last}(\pi) \rightarrow^* \sigma'$ and $\phi_{\mathrm{bal}}(\sigma_0, \sigma', a, c)$.
- Suppose $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \sigma_0, a, c)$ is derived in the third way. We have $\delta_{\mathrm{a}}^{\emptyset} \vdash \pi' \rightarrow^* \pi'$. Hence, using (33), we have $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi', \sigma, a, c)$. Hence, using the induction hypothesis, we have
$$\exists \sigma' : \mathit{last}(\pi') \rightarrow^* \sigma' \wedge \phi_{\mathrm{bal}}(\sigma_0, \sigma', a, c) \tag{35}$$
By (32), we have $\mathit{last}(\pi) \xrightarrow{\beta} \mathit{last}(\pi')$ for some β. Hence, using (35), we have $\exists \sigma' : \mathit{last}(\pi) \rightarrow^* \sigma' \wedge \phi_{\mathrm{bal}}(\sigma_0, \sigma', a, c)$.

The induction above completes the proof of this lemma. □

We next re-state and prove the result about the connection between strategy-aware liquidity and basic liquidity.

Proposition 2 (restated). *If $\mathit{is\text{-}complete}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}})$ holds, then for all addresses a and contracts c, we have $\mathit{SL}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, a, c) \iff \mathit{BL}(a, c)$.*

Proof. We consider each direction of the equivalence separately.

$\Rightarrow$. We assume

$$\mathit{is\text{-}complete}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}) \tag{36}$$

$$\mathit{SL}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, a, c) \tag{37}$$

Moreover, we assume

$$\mathit{init\text{-}ctr}(a, c, \sigma) \tag{38}$$

$$\sigma \rightarrow^* \sigma' \tag{39}$$

We proceed to show $\exists \sigma'' : \sigma' \rightarrow^* \sigma'' \wedge \phi_{\mathrm{bal}}(\sigma', \sigma'', a, c)$.

Using (36), (39), and applying Lemma 6, we have $(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}) \vdash [\sigma] \rightsquigarrow \pi : \mathrm{u\_nxt}$ and $\sigma' = \mathit{last}(\pi)$ for some π. Hence, using (37) and (38), we have $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \mathit{last}(\pi), a, c)$. Hence, using $\sigma' = \mathit{last}(\pi)$ and applying Lemma 8, we have $\exists \sigma'' : \sigma' \rightarrow^* \sigma'' \wedge \phi_{\mathrm{bal}}(\sigma', \sigma'', a, c)$.

This completes the proof of the $\Rightarrow$ part.

$\Leftarrow$. We assume

$$\mathit{is\text{-}complete}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}) \tag{40}$$

$$\mathit{BL}(a, c) \tag{41}$$

Moreover, we assume

$$\mathit{init\text{-}ctr}(a, c, \sigma) \tag{42}$$

$$(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}) \vdash [\sigma] \rightsquigarrow \pi : \mathrm{u\_nxt} \tag{43}$$

We proceed to show $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \mathit{last}(\pi), a, c)$.

Using (43) and applying Lemma 5, we have $\sigma \rightarrow^* \mathit{last}(\pi)$. Hence, using (41) and (42), we have $\mathit{last}(\pi) \rightarrow^* \sigma''$ and $\phi_{\mathrm{bal}}(\mathit{last}(\pi), \sigma'', a, c)$ for some σ''. Hence, using (40) and applying Lemma 7, we have $\mathit{usl}(\delta_{\mathrm{u}}^{\mathit{Adr}_{\mathrm{eoa}}}, \delta_{\mathrm{a}}^{\emptyset}, \pi, \mathit{last}(\pi), a, c)$.

This completes the proof of the $\Leftarrow$ part. □