Atomic Transaction Graphs: A Framework for Secure-by-design Protocols for Heterogeneous Blockchain Environments

Anonymous Author(s)

***Abstract*—Limitations of current blockchains, such as interoperability and scalability, have motivated the design of cryptographic protocols over existing blockchains, extending their capabilities (e.g., enabling cross-currency swaps) and improving their performance (e.g., through fast off-chain payment protocols). Due to the heterogeneity of the blockchain landscape, these blockchain protocols are tailored to specific blockchains and applications and hence require custom security proofs. We observe that many blockchain protocols share common security and functionality goals, which can be captured by an *atomic transaction graph* (ATG) describing the structure of desired transfers across different consensus objects. Based on this observation, we contribute a framework for generating secure-by-design protocols that realize these goals. The resulting protocols build upon *Conditional Timelock Contracts* (CTLCs), a novel minimal smart contract functionality that can be implemented in a large variety of consensus objects, including cryptocurrencies with a restricted scripting language (e.g., Bitcoin), and payment channels. We show how ATGs, in addition to enabling novel applications, capture the security and functionality goals of existing protocols, including many examples from payment channel networks, and complex multi-party cross-currency swaps among Ethereum-style cryptocurrencies. Our framework provides generic and provably secure CTLC-based protocols for these use cases, which we can show to still outperform state-of-the-art protocols designed for more restricted settings.**

1. Introduction

Cryptographic protocols are developed on top of blockchain-based cryptocurrencies like Bitcoin or Ethereum to overcome the pressing interoperability [1] and scalability [2] challenges of the underlying blockchain technology. Examples of such protocols, henceforth dubbed *blockchain protocols*, are atomic swap protocols, which enable the trustless exchange of funds between different cryptocurrencies, or off-chain protocols, which reduce the transaction load on the blockchain consensus by securely delegating the processing of certain transactions to smaller user committees.

The diverse and heterogeneous landscape of the targeted cryptocurrencies makes the process of designing blockchain protocols particularly challenging and cumbersome: Cryptocurrencies feature different capabilities, e.g., Ethereum supports the execution of complex stateful programs (so-called smart contracts) while in Bitcoin, payments can only

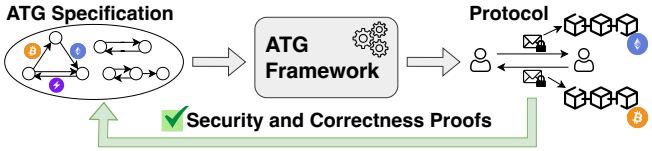

Figure 1: Overview of the Framework

be restricted by conditions expressible in a simple scripting language. Blockchain protocols carefully integrate these features with cryptographic building blocks (such as digital signatures or zero-knowledge proofs) or trusted hardware to achieve use-case-specific functionality and security goals. Hence, the current landscape consists of custom protocols catering to individual use cases and cryptocurrencies. For example, existing atomic swap protocols and off-chain protocols, are tailored to the features of Ethereum [3], [4], [5], [6], [7], Bitcoin [8], [9], [10], [11], [12], utilize properties of the cryptocurrencies' built-in signature checks [13], [14], [15] or leverage trusted hardware [16], [17].

Since blockchain protocols are highly security-critical and design flaws can result in the immediate financial loss of honest parties, all these custom protocols mandate a careful security analysis, usually in the form of complex manual cryptographic security proofs [18]. This is a non-trivial task as demonstrated by the security flaws found so far in blockchain protocols proposed by both academia [19], [20] and industry [14], [21].

In this work, we aim to lift this burden on protocol designers by providing a framework for automatically creating a large class of blockchain protocols that are secure by design. To this end, we make two key observations:

1) Many blockchain protocols, at their core, share common functionality and security goals. More specifically, they aim at the atomic execution of a set of bilateral transfers on different *consensus objects*. A consensus object, in this context, can be a whole cryptocurrency (i.e., Bitcoin or Ethereum) or a consensus structure created on top of a cryptocurrency in the context of an off-chain protocol (i.e., a payment channel, state channel, or roll-up).

2) While many blockchain protocols leverage specific features of the underlying consensus object (e.g., stateful smart contracts), by careful restructuring of the protocol flow, the usage of these features can often be replaced with a simple core mechanism.

Building upon these two observations, our framework (illustrated in Figure 1), on input a specification of the

intended functionality and security goals, generically constructs a concrete blockchain protocol that provably meets these goals and that is based on a core mechanism supported by many consensus objects.

More concretely, we specify the protocol goals in terms of a directed graph, that we call atomic transaction graph (short ATG) and whose nodes represent a set of users while arcs indicate transfers between these users on (potentially different) consensus objects. An ATG specifies the desired protocol functionality in that an honest protocol run shall result in the execution of all transfers in the ATG. Further, the ATG implies a local *atomicity* guarantee for an honest user H in the protocol even in the presence of arbitrary collusions of malicious users: If a transfer corresponding to an outgoing arc of H in the ATG is executed (taking funds *from* H), H can execute transfers for all their ingoing arcs in the ATG (transferring funds *to* H).

We show that the core mechanism that our generic protocol requires from the consensus objects realizing the transfers can be instantiated by a large range of systems, including cryptocurrencies like Ethereum and Bitcoin, as well as payment channels or state channels employed on top of such currencies. Consequently, the generated protocols are ready to be deployed on these platforms.

Thanks to this generality, our framework comprises many existing applications, including arbitrary multi-party atomic swaps between cryptocurrencies that support stateful smart contracts [22], [23], multi-hop payments in payment channel networks [14], [24], [25], [26], or rebalancing of the funds in payment channels [4], [5], [6], [7], [27].

Beyond that, our framework enables the secure design of a large set of new applications for which no custom protocols (or security proofs) exist yet. This, most prominently, includes protocols for complex swap scenarios (involving multiple users and funds transfers) whose transfers are (partly) conducted in cryptocurrencies with limited smart contract support, such as Bitcoin, or in payment channels.

Technically, we make the following contributions:

- For defining provably secure protocols for arbitrary ATGs, we introduce the concept of *transaction tree* (short ttree, c.f. Section 3) as a general intermediate representation for blockchain protocols across different consensus objects. Ttrees capture the flow of such blockchain protocols in terms of a multi-stage, multi-player funds redistribution game and, as such, constitute a contribution of independent interest. We show how to realize an ATG specification with a ttree and provide corresponding security and correctness proofs.
- We realize the mechanics of ttrees with a generic protocol based on a novel building block called *Conditional Timelock Contract (CTLC)* (c.f. Section 4). We formally prove the security and correctness of our CTLC-based protocol w.r.t. the original ttree. We carry out this security analysis in a realistic model of heterogenous blockchain environments featuring a malicious miner with delaying and reordering capabilities and considering restricted participation and data availability of the different consensus objects.
- We demonstrate how our framework efficiently captures a plenitude of existing cross-chain and off-chain applications. To this end, we illustrate how the functionality and security requirements of these applications are expressible in terms of ATGs (c.f. Section 7). Further, we illustrate how the CTLC-based protocols generated from our framework can be instantiated to concrete protocol instances for real-world consensus objects (c.f. Section 6). Most notably, CTLCs can be implemented in virtually every blockchain existing today as they only require the underlying consensus object to support timelocks and transaction authorization based on digital signatures.

To showcase the performance of the resulting CTLC-based protocols, we pick a protocol class from the literature ([22] and [23]), which is subsumed by our framework and show that CTLC-based protocols largely outperform [22] both in storage and computation cost and favorably compare to [23] in these dimensions.

- Our framework significantly extends the scope of existing works by supporting general and flexible specifications in terms of ATGs and by generating protocols for most cryptocurrencies and off-chain (also called layer-two) consensus objects such as payment channels. This, in particular, makes our framework the first one to support secure-by-design applications for protocols based on Bitcoin and payment-channel networks (e.g., Bitcoin's currently deployed Lightning Network).

2. Key Ideas

In this section, we overview our approach to systematically construct protocols that realize the functionality and security as specified by an atomic transaction graph (ATG).

A Simple Atomic Swap Protocol. As a basis for motivating and explaining the core concepts of our generic protocol construction from ATG specifications, we first revisit simple two-party atomic swap protocols [13], [22].

In a two-party atomic swap protocol, two users, Alice (A) and Bob (B), want to exchange funds that they own on different blockchains (called chain $\mathbb{A}$ and chain $\mathbb{B}$ here). To this end, as illustrated in Figure 2, they first lock the corresponding funds in both $\mathbb{A}$ and $\mathbb{B}$ by transferring them into a simple *hashed timelock contract* (HTLC). A hashed timelock contract $HTLC(\mathsf{S}, \mathsf{R}, y, t)$ is a smart contract whose funds can only be withdrawn in two ways: 1) The funds are transferred to the receiver R given that a secret value x for condition y is provided such that $H(x) = y$ (for some fixed hash function H); we say the HTLC got *claimed* in this case. 2) The funds are transferred (back) to the sender S after the time reaches timelock t; we say the HTLC got *refunded* in this case. To initiate an atomic

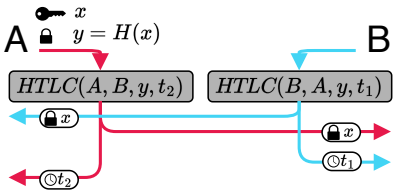

Figure 2: Two-party atomic swap protocol.

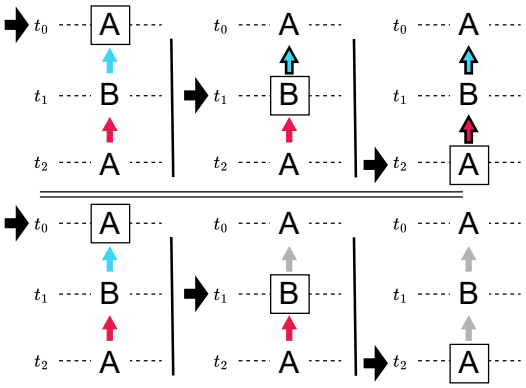

Figure 3: A two-party atomic swap as a round-based game. The black arrow indicates the current round. Players eligible to make a move are indicated by a square. Arrows with a black border indicate pulled edges; arrows without borders indicate available edges, and grayed-out arrows indicate disabled edges. The top picture depicts the game during an honest execution, and the lower picture depicts the game during an execution involving malicious user A.

swap, A chooses a secret x and transfers their funds into a contract $c_{AB} = \mathit{HTLC}(A, B, y, t_2)$ (with $y = H(x)$) on $\mathbb{A}$. Based on this, B transfers their funds into a contract $c_{BA} = \mathit{HTLC}(B, A, y, t_1)$ on $\mathbb{B}$ such that $t_1 < t_2$. To initiate the swap, A withdraws B's funds from c_{BA} before t_1, publishing secret x. Bob, learning x, can claim the funds from c_{AB} before t_2. The timelocks t_1 and t_2 ensure that A's and B's funds will not be indefinitely locked in the respective contracts. It is crucial that there is time between t_1 and t_2 for Bob to safely refund their funds before Alice can do so. Otherwise, a malicious A could potentially withdraw the funds from both c_{AB} and c_{BA} at time t_2.

Fund Redistribution Games. While the two-party atomic swap protocol above is general in that it relies on the HTLC primitive, which is known to be realizable in many cryptocurrencies (including Ethereum and Bitcoin), it cannot be easily generalized to more complicated scenarios involving additional users or transfers. To overcome this limitation, we take a more systematic view by observing that the protocol relies on two key elements: First, claiming c_{AB} is dependent on the claiming of c_{BA}, ensuring that they can only be claimed in a predefined order (first c_{BA}, then c_{AB}). Second, both HTLCs can be refunded in the very same order.

Based on this observation, we can see the atomic swap as a round-based multi-player game represented as a simple linear *transaction tree* (short ttree) shown in Figure 3. Intuitively, the game starts on the top level of the ttree (where Alice is located at the root). It is hence Alice's turn to make a move, where the only available move is pulling the ingoing edge (indicating the claiming of c_{BA}). In the second round, it is Bob's turn. Given that Bob's outgoing edge got pulled before, Bob can now choose to pull its ingoing edge (the claiming of c_{AB}). Alternatively, if Alice did choose in the first round not to pull their ingoing edge, Bob can choose in the second round to *disable* this edge (corresponding to refunding c_{BA}). Correspondingly, Alice could disable their outgoing edge in round three (by refunding c_{AB}).

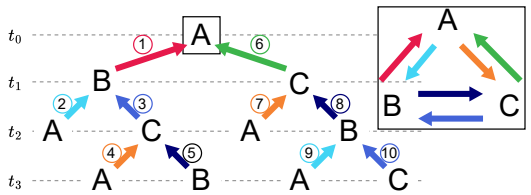

Figure 4: Tree unfolding of the three-party graph in the top right. Edge numbers are given in pre-order and just serve to identify the edges. Duplicated edges are depicted in the same color. Player A is enclosed in a square to indicate that they will be the first player to make a move in the game.

Blockchain Protocols as Games. Introducing the concept of ttrees, we can study how the goals of an ATG specification can be met by users playing fund redistribution games and only later consider how the mechanics of these games can be realized with cryptographic protocols. More precisely, we will show how to directly transform an ATG specification into such a game (represented as a ttree), which satisfies the ATG's functionality and security goals.

We hereby model an ATG as a directed graph $\mathcal{D}$, consisting of a non-empty set $\mathcal{N}$ of nodes (representing users) and a finite set $\mathcal{A}$ of arcs (representing transfers between users). To illustrate the transformation procedure, we use the example of a three-party swap graph (c.f Figure 4).

As in the two-player case, we choose an arbitrary user as the tree's root (hereby called *leader*). The leader will be in the position to initiate the swap by pulling all their ingoing arcs from the graph. In the further construction of the ttree, it needs to be ensured that whenever the outgoing arcs of a user is pulled, they can also pull all ingoing arcs. Correspondingly, the ttree is created: Starting from the leader, all ingoing arcs from the graph are added as edges to the tree. On the next level, for all users present at this level, all ingoing arcs will be appended as edges, constituting the next tree level. This graph unfolding stops whenever a user on a path is encountered for the second time (since otherwise a user could claim an arc from the graph twice).

We show the unfolding of an example graph in Figure 4. Note that due to the original graph's structure, several arcs appear multiple times in the ttree, for example, edges (2) and (9) correspond to the same arc. This is as in the graph in Figure 4, there are multiple paths from C to the leader A. Such duplicate edges (as indicated by the same color) should be considered representatives of the same arc. Consequently, only one of these duplicate edges can be executed. In the ttree execution, this is reflected by duplicate edges getting disabled (so becoming unavailable) once another representative has been executed. Intuitively, this is because the funds to be transferred in this case have already been claimed otherwise (when executing the duplicate edge).

An honest execution of the ttree in Figure 4 is depicted in Figure 5: Suppose every user pulls all their ingoing edges in every round. Then, the pulled edges of the ttree execution cover the arcs of the original graph already after three levels

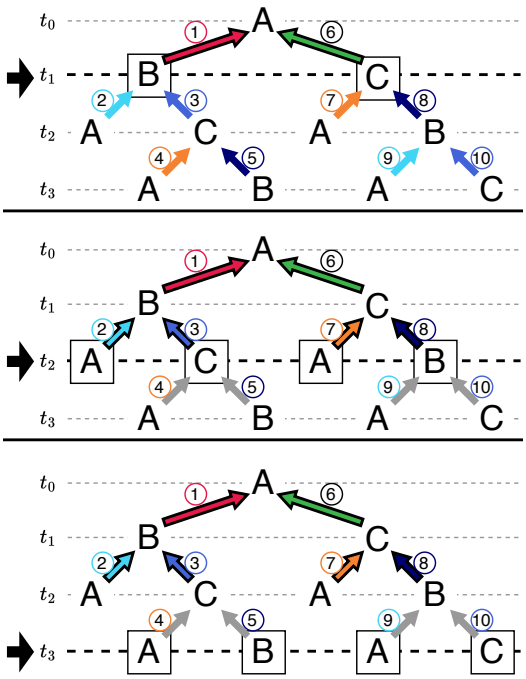

Figure 5: Honest execution of the game tree in Figure 4. Execution states are depicted with the symbols in Figure 3.

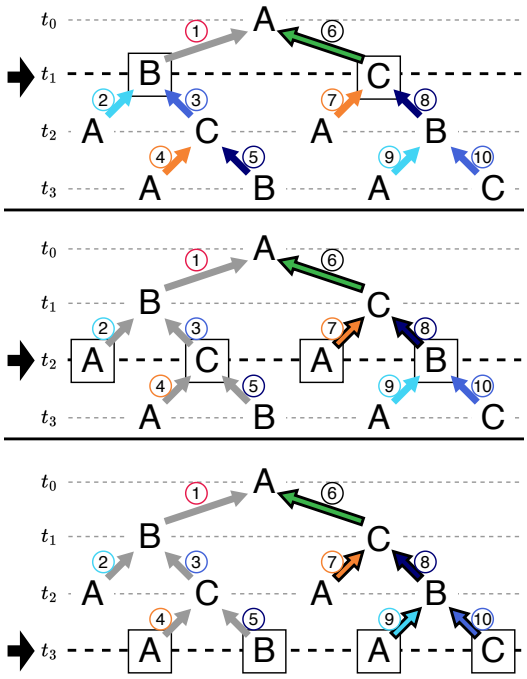

Figure 6: Dishonest execution of the game tree in Figure 4. Execution states are depicted with the symbols in Figure 3.

and all other levels (due to the existence of duplicate edges) got disabled (indicated by grayed-out edges).

The appearance of these duplicate edges is crucial for the security of the game since it is not ensured that malicious users will always pull all their ingoing edges. For example, consider an execution of the ttree (depicted in Figure 6) where A only pulls edge (6) and C pulls edge (8). Without the ingoing edges (9) and (10) appearing in the right subtree, B would lose funds in such a scenario since they could not claim their funds from A and C.

Game Security. We will formally prove in Section 3 that ttrees constructed from an ATG specification meet the ATG goals. To this end, we characterize the possible outcomes that can result from an honest user B playing their honest strategy on a ttree $\mathcal{T}$. The honest strategy of B consists of the user eagerly pulling all possible ingoing edges (only possible when corresponding outgoing edges have been pulled before) and disabling all possible outgoing edges. An outcome of a ttree execution will be represented by all edges that got pulled during the ttree execution. We will denote with $\mathcal{O}_B^{\mathcal{T}}$ the set of outcomes that can result from B following the honest strategy (while other users may behave arbitrarily). Based on this notion, we will show that honest users when playing the game always enjoy the following local atomicity guarantee:

Theorem 2.1 (Ttree Security (informal))**.** *Let $\mathcal{T}$ be a ttree resulting from unfolding graph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$. All outcomes $\omega \in \mathcal{O}_B^{\mathcal{T}}$ of $\mathcal{T}$ for honest user $B \in \mathcal{N}$ satisfy that if they contain an arc $(B, X) \in \mathcal{A}$ (corresponding to an outgoing arc of B in $\mathcal{D}$), then also all ingoing arcs $(Y, B) \in \mathcal{A}$ are contained in ω.*

This result ensures that an honest user never loses funds during the ttree execution: If funds are pulled from them, then they could also claim all funds that they should get according to $\mathcal{D}$.[1]

The described procedure of transforming graphs into ttrees not only applies to strongly connected graphs as the one given in the example but to all graphs that are *in-semiconnected*. A graph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ is in-semiconnected if it contains a node $n \in \mathcal{N}$ that can be reached from every other node $n' \in \mathcal{N}$. Each such node $n \in \mathcal{N}$ is a possible leader in the ttree construction. As we show in Section 7, supporting in-semiconnected graphs paves the way to cover many applications beyond atomic swaps.

Protocols for Ttrees. Ttrees constitute a powerful intermediate representation to capture the essence of complex protocols realizing ATGs. We now show how to enforce the mechanics of ttrees with the help of cryptographic protocols operating across heterogenous blockchain environments. The challenge here lies in finding a building block that is powerful enough to support arbitrary ttrees and at the same time sufficiently simple so that it can be realized on consensus objects with limited capabilities (such as Bitcoin, which supports only checking simple payment conditions). To meet these requirements, we introduce *Conditional Timelock Contract (CTLC)*, a generalization of the HTLC primitive used in the two-party atomic swap protocol (c.f. Figure 2). Note that the HTLC there serves

1. Note that an honest user may end up better off than specified by $\mathcal{D}$: A user can receive all funds corresponding to their ingoing arcs in $\mathcal{D}$ without spending all their funds corresponding to their outgoing arcs. E.g., in the execution in Figure 6, edge (1) is not pulled, indicating that user B does not need to spend their funds intended for A.

two purposes: 1) The condition of the HTLC establishes a dependency between the two ttree edges, ensuring that the second edge can be pulled if and only if the first one was pulled, and, thus, enabling round-based pulling of edges. 2) The consecutive timeouts enable a round-based disabling of edges, ensuring that a user can disable an outgoing edge before their ingoing edges can get disabled.

However, HTLCs do not suffice to enable the same properties for general ttrees, since in ttrees 1) the execution of an edge depends on the prior execution of a whole path of edges (and not only a single edge) and 2) ttrees contain duplicate edges, indicating alternative forms of spending the same funds to the receiver in different phases of the protocol.

CTLCs address these limitations. Firstly, they support flexible conditions that can be composed of several secrets. This is to reflect that each edge of the ttree $\mathcal{T}$ representing arc (X, Y) corresponds to a walk $\vec{a}$ from (X, Y) to the leader (the root) of $\mathcal{T}$ and hence should only be pulled once all other edges along $\vec{a}$ have been pulled. To realize this, we identify each edge e of $\mathcal{T}$ with this walk $\vec{a}$ (so $e = (X, Y)_{\vec{a}}$) and assign it a unique secret $s^{\vec{a}}$ owned by the receiver Y of the edge (similar to how A owns x in Figure 2). The condition for pulling edge $(X, Y)_{\vec{a}}$ will then require the knowledge of all secrets for the edges on the path $\vec{a}$ to the root in $\mathcal{T}$. Second, CTLCs generalize HTLCs in that they allow for nesting multiple CTLCs, meaning that refunding a CTLC can result in the funds being transferred to a follow-up CTLC instead of returning them to the receiver. More precisely, we will consider a CTLC $\mathit{CTLC}(\mathsf{S}, \mathsf{R}, \mathit{SCs})$ to contain a non-empty list SCs of subcontracts of the form $\mathit{sCTLC}(\Phi, t)$ where Φ represents a set of composed spending conditions and t represents the timelock of the contract. For the sake of generality, we will not require the individual conditions Y_i within a composed condition $\vec{Y} \in \Phi$ to be fixed to hash values for a specific hash function H but will simply consider them to be witnesses for some hard relation R (meaning that given Y, it is computationally hard to find x such that $(x, Y) \in R$). Further, we will require that the timelocks of all subcontracts in SCs are strictly increasing. Then $\mathit{CTLC}(\mathsf{S}, \mathsf{R}, \mathit{SCs})$ can evolve in the following ways: If $\mathit{SCs} = [(\mathit{sCTLC}(\Phi_1, t_1)), \dots]$ then the contract funds can be claimed (transferred to R) when providing $\vec{x}$ such that there is a $\vec{Y} \in \Phi_1$ and for all $Y^i \in \vec{Y}$ it holds $(x^i, Y^i) \in R$. Alternatively, after time t_1 the contract funds can be refunded to S (if $\mathit{sCTLC}(\Phi_1, t_1)$ was the last element in SCs), or spent to $\mathit{CTLC}(\mathsf{S}, \mathsf{R}, [(\mathit{sCTLC}(\Phi_2, t_2)), \dots])$.

Duplicate edges can be realized through a CTLC that contains subcontracts for all edges representing the same arc in the graph in ascending order of their appearance in the ttree and increasing timeouts according to their ttree level[2]. For example, the duplicate edges ② and ⑨ from Figure 4 would be realized by the following CTLC $c_{(A,B)}$:

$$\mathit{CTLC}(A, B, [\overbrace{\mathit{sCTLC}(\{(s^{①}, s^{②})\}, t_2)}^{sc^2}, \overbrace{\mathit{sCTLC}(\{(s^{⑥}, s^{⑧}, s^{⑨})\}, t_3)}^{sc^3}])$$

2. Duplicate edges on the same level are modeled by different conditions $\vec{Y} \in \Phi$.

Before time t_2, B can claim the funds by providing secrets $s^{①}$ and $s^{②}$. While $s^{②}$ is chosen by B, B would obtain secret $s^{①}$ if A claims the funds from $c_{(B,A)}$ in edge ①. Consequently, providing these secrets to claim the funds of $c_{(A,B)}$ using subcontract sc^2 would correspond to B pulling the edge ② given that edge ① was pulled.

If edge ① was not pulled then at time t_2, B already disabled ① and, hence, also sc^2 (representing edge ②) can be safely discarded so that only sc^3 representing edge ⑨ is left. Subcontract sc^3 enables B to claim the funds when providing the secrets $s^{⑥}$, $s^{⑧}$, and $s^{⑨}$. Again $s^{⑥}$ and $s^{⑧}$ will be learned from C pulling edge ⑧, while $s^{⑨}$ was chosen by B.

Following these ideas, a ttree $\mathcal{T}$ can be translated into a set of CTLCs representing its edges. Pulling edges in $\mathcal{T}$ corresponds to claiming the CTLC modeling the edge, and disabling an edge is reflected by removing a subcontract for this edge from the corresponding CTLC. Based on this correspondence, we can characterize a general protocol $\Sigma_B^{\mathcal{T}}$ that implements the ttree strategy of honest user B on the ttree $\mathcal{T}$ for the CTLCs resulting from the translation of $\mathcal{T}$.

Protocol Security. To formally prove that the mechanics of a ttree $\mathcal{T}$ are faithfully captured by the protocol $\Sigma_B^{\mathcal{T}}$, we provide a formal model for the execution of CTLC-based protocols: We define how users can interact with an environment of different consensus objects that support CTLCs. More precisely, we characterize all possible protocol runs R that can incur when executing a specific honest user protocol Σ_B in such an environment and write $\Sigma_B \vdash R$ if R results from such an execution. The formal execution model thereby takes security-relevant blockchain-specific characteristics into account, e.g., that the interactions of honest users with the CTLCs on the different consensus objects get known to the attacker before execution and may be maliciously delayed or reordered.

We show now the result for the honest user protocol $\Sigma_B^{\mathcal{T}}$:

Theorem 2.2 (Protocol Security (Informal))**.** *Let $\mathcal{T}$ be a ttree and R be a run stemming from honest user B executing $\Sigma_B^{\mathcal{T}}$ ($\Sigma_B^{\mathcal{T}} \vdash R$) such that the timelocks of all subcontracts sc_e for edges in $\mathcal{T}$ have passed. Then there exists an outcome $\omega \in \mathcal{O}_B^{\mathcal{T}}$ such that the subcontracts sc_e involving B claimed in R correspond to the edges e of B in ω.*

This result ensures that whenever an execution of the protocol $\Sigma_B^{\mathcal{T}}$ advanced enough (namely reached the timelocks of all subcontracts constructed for the ttree $\mathcal{T}$), then the executed subcontracts correspond to the edges of a valid outcome for the honest user B in $\mathcal{T}$. Hence, the outcomes of B in $\mathcal{T}$ as given by $\mathcal{O}_B^{\mathcal{T}}$ soundly reflect how funds of B will be claimed in the protocol.

Intuitively, this result gives us end-to-end security guarantees: A (sufficiently advanced) execution of the honest user protocol $\Sigma_B^{\mathcal{T}}$ corresponds to an outcome $\omega \in \mathcal{O}_B^{\mathcal{T}}$ and such an outcome ω for a tree $\mathcal{T}$ resulting from unfolding a graph $\mathcal{D}$ was shown to ensure that B does not lose funds (w.r.t. the ATG specification $\mathcal{D}$).

In summary, we have shown how to synthesize an ATG specification $\mathcal{D}$ into a game represented as a ttree $\mathcal{T}$, and how to transform this ttree $\mathcal{T}$ into a CTLC-based protocol. From our security results for both of these transformations, we can show that for an honest user B, the resulting CTLC-based protocol is guaranteed to execute transfers that correspond to a beneficial trade of B w.r.t. $\mathcal{D}$.

3. Ttree Unfolding

In the following, we will give a more formal description of how to represent atomic transaction graphs and how to transform them into ttrees with the intended behavior.

Atomic Transaction Graphs. An atomic transaction graph (ATG) is a directed graph (short digraph) $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N}$ denotes the set of nodes (representing users) and $\mathcal{A}$ denotes the set of arcs (representing transactions between users). Arcs a are given as tuples $a = (A, B)$ with $A, B \in \mathcal{N}$, and we call A the sender (written *sender*(a)) and B the receiver of a (written *receiver*(a)). We call *walk* a sequence of arcs $\vec{a} = [a_{|\vec{a}|-1}, a_{|\vec{a}|-2}, ..., a_0] \in \mathcal{A}^{|\vec{a}|}$ where the receiver of each arc coincides with the sender of its predecessor. We use $\vec{a}_1 \cdot \vec{a}_2$ to denote the concatenation of two walks $\vec{a}_1$ and $\vec{a}_2$. Further, we use $\vec{a}_2 \succcurlyeq \vec{a}_1$ to denote that $\vec{a}_1$ is a suffix of $\vec{a}_2$ (so that $\exists \vec{a} : \ \vec{a}_2 = \vec{a} \cdot \vec{a}_1$) and $\vec{a}_2 \succ^+ \vec{a}_1$ to say that $\vec{a}_2$ extends $\vec{a}_1$ by one arc (so $\exists a : \ \vec{a}_2 = [a] \cdot \vec{a}_1$).

In this work, we are interested in the class of digraphs that we call *in-semiconnected*. A graph is in-semiconnected if there is a node $A \in \mathcal{N}$ (which we also call the *leader*), which can be reached (with a walk) from every other node in $\mathcal{N}$. Every strongly connected graph is also in-semiconnected while the contrary does not hold. A formal proof and a more detailed discussion of the underlying graph theory can be found in Appendix C.

Ttrees Unfolding. We next define how to transform an ATG given as a digraph $\mathcal{D}$ into a ttree. To this end, we will represent ttrees as sets of *edges*, where an edge $e = (A, B)_{\vec{a}}$ is an arc indexed with its walk $\vec{a}$ to the root of the ttree.

Definition 3.1 (Tree unfolding)**.** *Let $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ be a digraph that is in-semiconnected in $A \in \mathcal{N}$. The tree unfolding unfold$(\mathcal{D}, A)$ of $\mathcal{D}$ with leader A is defined as follows:*

$$
\begin{aligned}
&\textit{unfold}(\mathcal{D}, A) := \\
&\{a_{\vec{a}^s} \mid \exists \vec{a} : \vec{a} \succcurlyeq \vec{a}^s \ \wedge \ a \in \mathcal{A} \ \wedge \ a = a^s_{|\vec{a}^s|-1} \\
&\wedge \ \exists B \in \mathcal{N} : \textit{noDupWalk}(\mathcal{D}, \vec{a}, B, A) \\
&\wedge \ (\exists j \leq |\vec{a}| - 2 : B = \textit{receiver}(a_j) \\
&\qquad \vee \ \nexists a' \in \mathcal{A} : B = \textit{receiver}(a'))\}
\end{aligned}
$$

where noDupWalk$(\mathcal{D}, \vec{a}, B, A)$ denotes that $\vec{a}$ is a walk from B to A in $\mathcal{D}$ that does not contain the same arc twice.

The definition states that the unfolding contains all edges $a_{\vec{a}^s}$ on walks $\vec{a}$ from nodes $B \in \mathcal{N}$ to the leader A, which satisfy the following properties (1) $\vec{a}$ does not contain repeated arcs and either (2a) $\vec{a}$ already contains an arc with B in the position of a receiver or (2b) $\vec{a}$ could not be extended beyond B because there is no $a' \in \mathcal{A}$ with B as receiver. This unfolding ensures for every user B that if it occurs as a sender in a path of the resulting ttree then it also occurs as a receiver (if it has receiving arcs in $\mathcal{D}$).

Outcome Sets. We provide a form of game semantics for ttrees by introducing the outcome sets (written $\mathcal{O}^{\mathcal{T}}_B$) of a user B when interacting with a ttree $\mathcal{T}$. Intuitively, an outcome $\omega \in \mathcal{O}^{\mathcal{T}}_B$ corresponds to a partial execution of $\mathcal{T}$ that may result from such an interaction. In particular, the outcomes reflect that the user B can enforce certain minimal guarantees on the ttree execution. More precisely:

1) If $\mathcal{T}$ contains duplicate edges $(X, Y)_{\vec{a}}$, $(X, Y)_{\vec{a}'}$ involving B (so $B \in \{X, Y\}$), at most one of them may be executed.
2) If B is the root user of $\mathcal{T}$ then all ingoing edges $(Y, B)_{\vec{a}}$ (with $|\vec{a}| = 1$) are executed.
3) If an outgoing edge $(B, X)_{\vec{a}}$ of B is executed then also all corresponding ingoing edges $(Y, B)_{\vec{a}'}$ with $\vec{a}' \succ^+ \vec{a}$ are executed.

Definition 3.2 (Outcome Set)**.** *Let $\mathcal{T}$ be a ttree. Then the outcome set $\mathcal{O}^{\mathcal{T}}_B$ of user B in $\mathcal{T}$ is given as*

$$
\begin{aligned}
\mathcal{O}^{\mathcal{T}}_B := \{\omega \in \mathcal{O}_{\textit{full}}(\mathcal{T}) \mid \ & \textit{NoDup}(\mathcal{T}, B, \omega) \\
& \wedge \ \textit{HonestRoot}(\mathcal{T}, B, \omega) \ \wedge \ \textit{EagerPull}(\mathcal{T}, B, \omega)\}
\end{aligned}
$$

where $\mathcal{O}_{\textit{full}}(\mathcal{T})$ denotes the set of all ttrees whose paths (from the leaves to the root) are suffixes of paths in $\mathcal{T}$ and

$$
\begin{aligned}
&\textit{NoDup}(\mathcal{T}, B, \omega) :\Leftrightarrow \\
&(X, Y)_{\vec{a}}, (X, Y)_{\vec{a}'} \in \omega \wedge B \in \{X, Y\} \Rightarrow \vec{a} = \vec{a}', \\
&\textit{HonestRoot}(\mathcal{T}, B, \omega) :\Leftrightarrow \\
&(X, B)_{\vec{a}} \in \mathcal{T} \wedge |\vec{a}| = 1 \Rightarrow (X, B)_{\vec{a}} \in \omega, \\
&\textit{EagerPull}(\mathcal{T}, B, \omega) :\Leftrightarrow \\
&(X, B)_{\vec{a}_1} \in \mathcal{T} \wedge (B, Y)_{\vec{a}_2} \in \omega \wedge \vec{a}_1 \succ^+ \vec{a}_2, \\
&\qquad \Rightarrow \exists \vec{a}_3 : (X, B)_{\vec{a}_3} \in \omega \wedge |\vec{a}_3| \leq |\vec{a}_1|.
\end{aligned}
$$

The predicates *NoDup*, *HonestRoot*, *EagerPull* capture exactly the three requirements on the partial ttree executions.

Security and Correctness of Ttree Unfolding. To show the security of the ttree unfolding, we show that each outcome $\omega \in \mathcal{O}^{\mathcal{T}}_B$ of a ttree $\mathcal{T} = \textit{unfold}(\mathcal{D}, A)$ constitutes a safe outcome for user B w.r.t. the ATG $\mathcal{D}$. An outcome is considered safe if B does not end up *underwater*, meaning that some outgoing arcs of user B in $\mathcal{D}$ are triggered but not all their ingoing arcs in $\mathcal{D}$.

Theorem 3.3 (Security of tree unfolding)**.** *Let $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ be a digraph that is in-semiconnected in $A \in \mathcal{N}$ and $\mathcal{T} = \textit{unfold}(\mathcal{D}, A)$ and $B \in \mathcal{N}$. Then it holds that*

$$
\begin{aligned}
&\forall \omega \in \mathcal{O}^{\mathcal{T}}_B : \\
&\forall \vec{a}, \vec{a}' : \big((X, Y)_{\vec{a}} \in \omega \wedge (X, Y)_{\vec{a}'} \in \omega \wedge B \in \{X, Y\} \\
&\qquad \Rightarrow \vec{a} = \vec{a}' \wedge (X, Y) \in \mathcal{A}\big) \\
&\wedge \forall (B, Y)_{\vec{a}} \in \omega : \ \big((X, B) \in \mathcal{A} \Rightarrow \exists \vec{a}' : (X, B)_{\vec{a}'} \in \omega\big)
\end{aligned}
$$

Intuitively, the statement says that all outcomes ω (so partial ttree executions) that a user B may incur during

interaction with the ttree $\mathcal{T}$ resulting from unfolding the graph $\mathcal{D}$ have the following properties: (1) Edges involving user B correspond to a unique arc in the graph, and (2) if $(B,Y)_{\vec{a}} \in \omega$ (corresponding to an outgoing arc (B,Y) in $\mathcal{D}$ being executed) then for each ingoing arc (X,B) in $\mathcal{D}$, ω also contains a corresponding edge $(X,B)_{\vec{a}'}$ for this arc. The formal proof of this theorem is given in Appendix D.

In addition to security for an honest party, we show that if all parties are honest, all arcs from the original graph get executed, so the intersection of the outcome sets of all users only contains outcomes that cover the whole graph. The proof of Theorem 3.4 can be found in Appendix D.

Theorem 3.4 (Correctness of Tree Unfolding)**.** *Let $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ be a digraph that is in-semiconnected in $A \in \mathcal{N}$ and $\mathcal{T} = unfold(\mathcal{D}, A)$ the tree unfolding of that graph.*

$$\forall \omega^* \in \bigcap_{B_i \in \mathcal{N}} \mathcal{O}^{\mathcal{T}}_{B_i} : (X,Y) \in \mathcal{A} \Leftrightarrow \exists \vec{a} : (X,Y)_{\vec{a}} \in \omega^*$$

4. Protocols for Ttrees

Ttrees provide a general abstraction layer for describing interactive protocols that involve the orchestrated execution of transactions in different heterogenous blockchain environments (HBE). In this section, we show how ttrees can be securely realized by cryptographic protocols over HBE. This will allow us to prove end-to-end security and correctness for general blockchain protocols specified as an ATG.

4.1. CTLCs

Our protocols rely on Conditional Timelock Contracts (CTLCs), the core building block that needs to be provided from the consensus object (e.g., the underlying blockchain). We provide here a formal model of the execution of CTLCs and describe in Section 6 how CTLCs can be realized in practice.

We represent a *CTLC* contract by a list $c^x := [sc^x_1, ..., sc^x_s]$ of subcontracts $sc^x_i := (X, Y, f^\zeta, \lambda_i, \Phi_i)$ where X denotes the contract's sender, Y the contract's receiver, f^ζ the contract fund (with identifier ζ), λ_i the *timelock* of the subcontract, and Φ_i the subcontract's *condition*. While the sender, receiver, and fund need to match for all $sc^x_i \in c^x$ (we hence also write $sender(c^x)$, $receiver(c^x)$, and $fund(c^x)$), timelock λ_i and condition Φ_i are specific to sc^x_i.

The timelock λ_i denotes the time starting from which subcontract $sc^x_i \in c^x$ may be removed from c^x given that it is the first element of c^x (we say that sc^x_i gets *timed out* in this case). Since subcontracts can only be timed out in order, we require that the timelocks of the subcontracts must be strictly increasing. The condition Φ_i is a set of sets of secrets describing different options for claiming sc^x_i: A member $\mathcal{S} \in \Phi_i$ describes a set of secrets whose knowledge is sufficient for claiming the contract.

Semantics. We formally describe the execution of CTLCs using a small-step semantics, so a relation $\vec{\Gamma} \xrightarrow{\alpha} \vec{\Gamma}'$ characterizing how a HBE supporting CTLCs evolves from one state (denoted by $\vec{\Gamma}$) to another ($\vec{\Gamma}'$) when executing an action α. The state $\vec{\Gamma}$ thereby is given as a vector of individual object environments Γ_{ch} per consensus object *ch*, keeping track of the individual stages of the CTLC execution. E.g., the components $\Gamma_{ch}.F_{av}$ and $\Gamma_{ch}.F_{res}$ track the funds currently available in *ch* (e.g., owned by a user), or reserved by a CTLC, respectively. The component $\Gamma_{ch}.C_{en}$ contains CTLCs that have been set up (enabled) for execution in *ch*. More concretely, the semantics considers the following stages of CTLC execution:

$$\frac{\begin{array}{c}\hat{c}^x \in \Gamma_{ch}.C_{adv}, sc^x \in c^x \in \Gamma_{ch}.C_{en},\\ \mathcal{S} \in cond(sc^x), \mathcal{S} \subseteq \Gamma_{ch}.S_{rev},\\ \nexists\, \dot{sc}^x \in \hat{c}^x : position(\dot{sc}^x) < position(sc^x),\\ C'_{adv} := \Gamma_{ch}.C_{adv} \backslash \{c^x\}, C'_{en} := \Gamma_{ch}.C_{en} \backslash \{c^x\},\\ F'_{av} := \Gamma_{ch}.F_{av} \cup \{(receiver(c^x) : fund(c^x))\}\\ F'_{res} := \Gamma_{ch}.F_{av} \backslash \{fund(c^x)\}\\ \Gamma'_{ch} := \Gamma_{ch}[C_{adv} \to C'_{adv}, C_{en} \to C'_{en}, F_{av} \to F'_{av}, F_{res} \to F'_{res}]\end{array}}{\vec{\Gamma} \xrightarrow{claim(c^x, sc^x, \mathcal{S})} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]}$$

Figure 7: Inference rule for claiming a CTLC subcontract (slightly simplified).

(1) Since CTLCs are used as parts of protocols involving multiple CTLC instances, which reside in different object environments, users will agree on the execution of CTLCs in batches. For initiating protocol execution, users will broadcast their intention to execute the protocol and its specification consisting of a set Ψ^{id} of CTLCs across the different HBEs. (2) Based on the announcement of a CTLC batch Ψ^{id}, the protocol participants will commit to the secrets used in the CTLCs $c^x \in \Psi^{id}$. (3) Once all participants committed their secrets, the users in the different object environments can initiate the pair-wise setup of the individual CTLCs c^x. Importantly, starting from this point, CTLCs c^x are considered local objects residing in a single object environment *ch* such that only users of *ch* may interact with them. (4) Contracts c^x previously advertised can be authorized by $sender(c^x)$ and $receiver(c^x)$, effectively marking the c^x as authorized in the corresponding object environments *ch*. (5) Once authorized by both parties a CTLC c^x can be enabled, effectively marking the funds $fund(c^x)$ as reserved. (6) An enabled contract c^x can either be claimed by $receiver(c^x)$ (by claiming its top-level subcontract), or c^x's subcontracts can be successively timed out until c^x can finally be refunded to $sender(c^x)$. In both cases, the funds $fund(c^x)$ are unmarked as reserved and assigned to the corresponding user, either $receiver(c^x)$ or $sender(c^x)$.

The small-step relation $\to$ that formally captures these execution steps, is defined by a set of inference rules. An example of such a rule for the *claim* case is provided in Figure 7: The rule first checks whether all preconditions for claiming a subcontract sc^x of CTLC c^x with conditions $\mathcal{S}$ are met, namely that (1) sc^x was enabled in object environment Γ_{ch} ($sc^x \in c^x \in \Gamma_{ch}.C_{en}$) (2) all secrets as specified by condition $\mathcal{S} \in cond(sc^x)$ have been revealed in object environment Γ_{ch} ($\mathcal{S} \subseteq \Gamma_{ch}.S_{rev}$) and (3) sc^x is the top-level contract of c^x (ensured by checking that there

is no other contract $\dot{sc}^x$ according to the original CTLC advertisement $\hat{c}^x \in \Gamma_{ch}.C_{adv}$ occurring in a position before sc^x, so $position(\dot{sc}^x) < position(sc^x)$). Note that the last condition accesses the original advertisement $\hat{c}^x \in \Gamma_{ch}.C_{adv}$ since this contains subcontracts $\dot{sc}^x$ of c^x that might not have been enabled (and hence not in $\Gamma_{ch}.C_{en}$) but still need to be timed out before a lower-level subcontract can be claimed.

If all conditions are met, the object environment Γ_{ch} is updated to remove the claimed c^x from both $\Gamma_{ch}.C_{en}$ and $\Gamma_{ch}.C_{adv}$ (reflecting that those contracts have been resolved) and to assign the contract funds to $receiver(c^x)$ (indicated by moving $fund(c^x)$ from the set of reserved funds F_{res} to the set of F_{av}, annotating the new ownership $(receiver(c^x) : fund(c^x))$. The full specification of the semantics is in Appendix E.

4.2. Blockchain Execution Model

The small-step semantics describes the (concurrent) execution of CTLCs in HBE. However, when defining cryptographic protocols that leverage CTLCs, the peculiar execution environment of the consensus objects executing CTLCs needs to be considered. In particular, in such consensus objects, user actions (e.g., the execution of a transaction in the blockchain) do not happen instantaneously but are subject to interference with a (potentially malicious) scheduler. Such a scheduler (e.g., a miner in a blockchain) learns about the intended actions of honest users (e.g., when transactions are submitted to the mempool) and can, based on this knowledge, decide on the execution order of actions or insert own actions. The capabilities of the scheduler are limited by the consensus guarantees, ensuring that honest user actions will be scheduled eventually.

To reflect this in our formal model, we adopt the approach taken in [28] and model protocols of honest users (given by a set *Hon*) as *strategies*. A strategy Σ is a function operating on sequences $R = \vec{\Gamma}_0 \xrightarrow{\alpha_0} \vec{\Gamma}_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} \vec{\Gamma}_n$ of valid transitions according to the small-step semantics, which represent the execution history of the HBE. We will call such sequences *runs*. On input of a run R, a strategy Σ outputs a set of actions $\{\alpha'_1, \alpha'_2, \ldots, \alpha'_m\}$ that the user aims to execute and hence should be appended to R.

In addition to the strategies Σ_{B_i} of honest users $B_i \in$ *Hon* (modeling the behavior of honest protocol participants), we assume the existence of an adversarial strategy Σ_{Adv} that models the behavior of malicious protocol participants and the malicious scheduler. Such a strategy Σ_{Adv}, in addition to the run R, gets as input the set $\mathcal{P} = \bigcup_{B_i \in Hon} \Sigma_B(R)$ of all outputs of honest user strategies and based on that outputs the next action α to append to R. Σ_{Adv} is limited to only output actions $\alpha = \Sigma_{Adv}(R, \mathcal{P})$ that are valid extensions of R according to the small-step semantics. Further, Σ_{Adv} may not schedule any actions α for which honest users have the privilege unless those are included in $\mathcal{P}$ (e.g., actions revealing the secrets of honest users). Finally, Σ_{Adv} may only advance the time t of the HBE by an offset δ (indicated by the execution of a dedicated action *elapse* δ) if all honest users agreed to this by scheduling actions *elapse* $\delta_i \in \Sigma_{B_i}(R)$ with $\delta_i \geq \delta$. This requirement reflects the inclusion guarantees of the consensus objects that enable honest users to meet deadlines[3].

We say that a run $R = \vec{\Gamma}_0 \xrightarrow{\alpha_0} \vec{\Gamma}_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} \vec{\Gamma}_n$ conforms to $(\Sigma_{Hon}, \Sigma_{Adv})$ (written $(\Sigma_{Hon}, \Sigma_{Adv}) \vdash R$) if R results from the interactions of the honest user strategies $\Sigma_{B_j} \in \Sigma_{Hon}$ with the adversarial strategy Σ_{Adv}, meaning that $\alpha_i = \Sigma_{Adv}(R_i, \mathcal{P}_i)$ for $R_i = \vec{\Gamma}_0 \xrightarrow{\alpha_0} \ldots \xrightarrow{\alpha_{i-1}} \vec{\Gamma}_i$ and $\mathcal{P}_i = \bigcup_{B_j \in Hon} \Sigma_{B_j}(R_i)$ for all $i = 0, \ldots, n-1$. So, for proving (security) properties for a protocol specified by a set of Σ_{Hon} of honest user strategies, one needs to consider all runs R such that $(\Sigma_{Hon}, \Sigma_{Adv}) \vdash R$ for any adversarial strategy Σ_{Adv}.

4.3. Ttree Protocols

We formally define the strategy $\Sigma_B^{\mathcal{T}}$ of an honest user B that sets up and executes a ttree $\mathcal{T}$. To this end, we first specify how to translate a ttree $\mathcal{T}$ to a batch Ψ^{id} of CTLCs whose subcontracts represent the edges $e \in \mathcal{T}$. $\Sigma_B^{\mathcal{T}}$ will then try to advertise and enable Ψ^{id}, and, if successful, interact with the CTLCs $c^x \in \Psi^{id}$ according to the way that user B would interact with the ttree $\mathcal{T}$.

From Ttrees to CTLCs. Defining a ttree protocol requires additional information on how a given ttree $\mathcal{T}$ should integrate with the HBE, e.g., which edges should use which funds of which consensus object. For this reason, we consider tuples of the form $(id, \mathcal{T}, t_0, spec)$ where *id* denotes a unique ttree identifier, $\mathcal{T}$ the ttree to be executed, $spec$ a function mapping edges $e \in \mathcal{T}$ to pairs (ch, f^{ζ}) of a consensus object *ch* and funds f^{ζ}, and t_0 the time when the ttree execution should start. Note that for $spec$ to be valid we need to assign the same (ch, f^{ζ}) to all edges e, e' with the same sender and receiver (since those constitute duplicate edges that should be represented by the same CTLC).

We define the batch Ψ^{id} of CTLCs for ttree $\mathcal{T}$ as follows:

Definition 4.1 (Ttree to CTLC conversion)**.** *Let $\mathcal{T}$ be a ttree,*

3. It may at first seem like a restriction that honest user actions will be included at the same time as scheduled. However, the attacker can still schedule arbitrarily many actions before the inclusion of an honest user action. This models the effects of an attacker appending several blocks before including the honest user action.

id an identifier, $t_0 \in \mathbb{R}$*, and spec a valid specification.*

$$
\begin{aligned}
&\textit{treetoCTLCBadge}(id, \mathcal{T}, t_0, \textit{spec}) := \\
&\{c^x_{(X,Y)} \mid c^x_{(X,Y)} = [\textit{sc}^x_{\ell_1}, \ldots, \textit{sc}^x_{\ell_n}] \ \wedge \ \ell_1 < \cdots < \ell_n \\
&\quad \wedge \ E_{(X,Y)} = \{(X,Y)_{\vec{a}} \in \mathcal{T}\} \\
&\quad \wedge \ \mathcal{L}_{(X,Y)} = \{|\vec{a}| \mid (X,Y)_{\vec{a}} \in E_{(X,Y)}\} \\
&\qquad\qquad = \{\ell_1, \ldots, \ell_n\} \\
&\quad \wedge \ x = (\textit{id}, X, Y) \ \wedge \ \ell \in \mathcal{L}_{(X,Y)} \ \wedge e^* \in E_{(X,Y)} \\
&\quad \wedge \ \textit{spec}(e^*) = (\textit{ch}, f^\zeta) \\
&\quad \wedge \ \textit{sc}^x_\ell = (X, Y, f^\zeta, t_0 + \ell\Delta, \Phi_\ell) \\
&\quad \wedge \ \Phi_\ell = \{\psi_e \mid e \in E_{(X,Y)} \ \wedge \ e = (X,Y)_{\vec{a}} \\
&\qquad \wedge \ |\vec{a}| = \ell \\
&\qquad \wedge \ \psi_e = \{\textit{s}^{\textit{id}}_{e'} \mid e' \in \textit{onPathToRoot}(\mathcal{T}, e)\}\}\}
\end{aligned}
$$

This definition states that there is a CTLC $[\textit{sc}^x_{\ell_1}, \ldots, \textit{sc}^x_{\ell_n}]$ in Ψ^{id} for each unique sender-receiver pair (X,Y) for which there is an edge $e^* = (X,Y)_{\vec{a}}$ in $\mathcal{T}$. The subcontracts $\textit{sc}^x_\ell$ of this CTLC then correspond to the levels $\ell \in \mathcal{L}_{(X,Y)}$ where edges $(X,Y)_{\vec{a}'}$ occur in $\mathcal{T}$ and are ordered correspondingly in increasing order (placing the subcontract corresponding to the lowest ttree level first). The subcontract $\textit{sc}^x_\ell$ for level ℓ has timeout $t_0 + \ell\Delta$ and the condition Φ_ℓ contains an element ψ_e for each edge $e = (X,Y)_{\vec{a}'}$ at level ℓ. One such element ψ_e contains secrets $\textit{s}^{\textit{id}}_{e'}$ for all edges e' on the path from e to the root of $\mathcal{T}$ (denoted by $\textit{onPathToRoot}(\mathcal{T}, e)$). According to this construction, executing an edge $e = (X,Y)_{\vec{a}} \in \mathcal{T}$ corresponds to claiming a subcontract $\textit{sc}^x_{|\vec{a}|}$ with secrets ψ_e (so all secrets from the e to the root). This ensures that the subcontract can only be claimed (by Y) if (1) all subcontracts corresponding to edges $e' = (X,Y)_{\vec{a}'}$ on a higher level ($|\vec{a}'| < |\vec{a}|$) have been timed out before and (2) the secrets for all edges e' on the path to the root have been revealed (indicating that these edges have been claimed). This realizes the intended game semantics described in Section 2. In particular, this construction gives us a mapping from edges $(X,Y)_{\vec{a}}$ in a ttree $\mathcal{T}$ specified by $(id, \mathcal{T}, t_0, \textit{spec}) = \mathcal{T}^{\textit{id}}_{\textit{spec}}$ to a corresponding contract $c^x_{(X,Y)}$, subcontract $\textit{sc}^x_{|\vec{a}|} \in c^x_{(X,Y)}$ and claiming condition $\psi_{(X,Y)_{\vec{a}}} \in \textit{sc}^x_{|\vec{a}|}.\Phi$ modeling that edge. We formally capture this correspondence by a function $h(\mathcal{T}^{\textit{id}}_{\textit{spec}}, (X,Y)_{\vec{a}}) = (c^x_{(X,Y)}, \textit{sc}^x_{|\vec{a}|}, \psi_{(X,Y)_{\vec{a}}})$.

Honest User Protocol. The honest user protocol (given by a strategy $\Sigma^{\mathcal{T}}_B$) for executing a ttree as given by $(id, \mathcal{T}, t_0, \textit{spec})$ specifies how the user behaves in the different phases of the setup and execution of a CTLC batch $\Psi^{id} = \textit{treetoCTLCBadge}(id, \mathcal{T}, t_0, \textit{spec})$.

The strategy, in the setup phase, advertises the contract batch and eagerly tries to set up the CTLCs corresponding to edges $e \in \mathcal{T}$. This means that it advertises, authorizes, and enables those CTLCs of Ψ^{id} containing subcontracts representing e. Here, it is taken into account that a subcontract $\textit{sc}^x$ representing e may only be enabled once all subcontracts representing its ingoing edges have been enabled before (to avoid the loss of funds). After the setup phase, $\Sigma^{\mathcal{T}}_B$ times out and refunds subcontracts representing edges $e \in \mathcal{T}$ of B as soon as the corresponding timeouts are reached. Further, $\Sigma^{\mathcal{T}}_B$ claims subcontracts $\textit{sc}^x$ representing ingoing edges $e \in \mathcal{T}$ if their outgoing edge has been pulled before (or there is no such edge). To this end, $\Sigma^{\mathcal{T}}_B$ shares secrets that were revealed in other consensus objects (for executing outgoing edges there) and reveals the remaining secret of B to claim $\textit{sc}^x$, and, once this was successful, claims $\textit{sc}^x$. Finally, whenever all possible actions have been executed, $\Sigma^{\mathcal{T}}_B$ schedules a *elapse* δ to proceed to the next protocol round.

Note that we can easily lift user strategies to operate on sets $\mathbb{T}$ of trees of the form $(id, \mathcal{T}, t_0, \textit{spec})$, given that their identifiers are unique and their funds are disjoint. This enables the secure concurrent execution of multiple ttree protocols. A full specification of such generalized strategies $\Sigma^{\mathbb{T}}_B$ can be found in Appendix F.

4.4. Security and Correctness

We prove that the protocol given by $\Sigma^{\mathbb{T}}_B$ is secure and correct. Intuitively, security in this context means that funds transfers observable in the protocol execution can be mapped to a valid ttree behavior. More formally, this is captured by the following theorem:

Theorem 4.2 (Protocol Security)**.** *Let* B *be an honest user,* $\mathbb{T}$ *be a set of tuples of the form* $(id, \mathcal{T}, t_0, \textit{spec})$*, which is well-formed, and* $\Sigma^{\mathbb{T}}_B$ *the honest user strategy for* B *executing* $\mathbb{T}$*. Let* $\Sigma^{\mathbb{T}}_{Adv}$ *be an arbitrary adversarial strategy. Then for all final runs* R *with* $(\Sigma^{\mathbb{T}}_B, \Sigma^{\mathbb{T}}_{Adv}) \vdash R$*, starting from an initial environment, and for all* $\mathcal{T}^{\textit{id}}_{\textit{spec}} = (id, \mathcal{T}, t_0, \textit{spec}) \in \mathbb{T}$ *there exists* $\widetilde{\omega}_{\textit{id}} \in \mathcal{O}^{\mathcal{T}}_B \cup \{\emptyset\}$ *such that*

$$
\begin{aligned}
&\forall \textit{id}, e \in \widetilde{\omega}_{\textit{id}} : B \in \textit{users}(e) \\
&\qquad \Rightarrow \textit{claim}(h(\mathcal{T}^{\textit{id}}_{\textit{spec}}, e)) \in \textit{actions}(R) \textit{ and}
\end{aligned}
$$

$$
\begin{aligned}
&\forall \textit{claim}(c^x, \textit{sc}^x, \psi) \in \textit{actions}(R) : B \in \textit{users}(c^x) \\
&\qquad \Rightarrow \exists \textit{id}, e \in \widetilde{\omega}_{\textit{id}} : h(\mathcal{T}^{\textit{id}}_{\textit{spec}}, e) = (c^x, \textit{sc}^x, \psi).
\end{aligned}
$$

where a run R *is considered* final *if it passed time* $t_0 + \textit{depth}(\mathcal{T})$*, an* initial *environment is a vector* $\vec{\Gamma}^0$ *where the components of all elements but* $\Gamma^0_{\textit{ch}}.F_{\textit{av}}$ *are empty and* $\textit{actions}(R)$ *denote the actions appearing in run* R*.*

The theorem states that for every protocol run R of $\Sigma^{\mathbb{T}}_B$ passing time $t_0 + \textit{depth}(\mathcal{T})$ and interacting with an arbitrary adversary, there is an outcome $\widetilde{\omega}_{\textit{id}} \in \mathcal{O}^{\mathcal{T}}_B \cup \{\emptyset\}$ for every ttree $\mathcal{T}$ in $\mathbb{T}$ such that these $\widetilde{\omega}_{\textit{id}}$ correspond exactly to all CTLCs that have been claimed during R. In particular, this excludes that there are funds claimed by the attacker in a way that is not covered by $\mathcal{O}^{\mathcal{T}}_B$ for the ttrees $\mathcal{T}$ in $\mathbb{T}$. The case $\widetilde{\omega}_{\textit{id}} = \emptyset$ covers the case where B may be the root node of $\mathcal{T}$ but still their ingoing edges are not executed because the adversary caused the ttree set up to fail. However, in this case, the security statement ensures that no funds were transferred between users, and hence B did not lose money. In particular, together with Theorem 3.3, we immediately obtain end-to-end security for ttrees $\mathcal{T} = \textit{unfold}(\mathcal{D}, A)$

resulting from the unfolding of an in-semiconnected ATG $\mathcal{D}$ because we know that a user B, in this case, cannot be underwater in any of its outcomes $\omega \in \mathcal{O}_B^{\mathcal{T}}$. The formal proof of Theorem 4.2 and the one of an end-to-end security statement can be found in Appendix H.3.

Similar to security, we can show that the protocol $\Sigma_{B_i}^{\mathbb{T}}$ faithfully executes the tree given that all users B_i involved in $\mathbb{T}$ follow the protocol. Formally, this is captured by the following theorem:

Theorem 4.3 (Protocol Correctness)**.** *Let Hon* $= \{B_1, \ldots, B_k\}$ *be a set of honest users. Let* $\mathbb{T}$ *be a well-formed set of tuples of the form* $(id, \mathcal{T}, t_0, spec)$ *and users*$(\mathbb{T}) \subseteq$ *Hon. Let* $\Sigma_{Hon}^{\mathbb{T}} = \{\Sigma_{B_1}^{\mathbb{T}}, \ldots, \Sigma_{B_k}^{\mathbb{T}}\}$ *a set of honest user strategies executing* $\mathbb{T}$*. Let* $\Sigma_{Adv}^{\mathbb{T}}$ *be an arbitrary adversarial strategy (for Hon). Let* R *with* $(\Sigma_{Hon}^{\mathbb{T}}, \Sigma_{Adv}^{\mathbb{T}}) \vdash R$ *be a final run starting from an initial configuration* $\vec{\Gamma}_0$ *that is liquid w.r.t.* $\mathbb{T}$*. Further, let* $\vec{\Gamma}_0.t < t_0 - depth(\mathcal{T})\Delta$ *then for all* $\mathcal{T}_{spec}^{id} = (id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ *there exists a* $\omega_{id}^* \in \bigcap_{B \in Hon} \mathcal{O}_B^{\mathcal{T}}$ *such that*

$$\forall id, e \in \omega_{id}^* \Rightarrow claim(h(\mathcal{T}_{spec}^{id}, e)) \in actions(R) \text{ and}$$

$$\forall claim(c^x, sc^x, \psi) \in actions(R) : users(c^x) \cap Hon \neq \emptyset \Rightarrow \exists id, e \in \omega_{id}^* : h(\mathcal{T}_{spec}^{id}, e) = (c^x, sc^x, \psi).$$

Where $\vec{\Gamma}_0$ *is liquid w.r.t.* $\mathbb{T}$ *requires that all funds as specified in* $\mathbb{T}$ *are present in* $\vec{\Gamma}_0$*.*

The theorem states that for every final protocol run R with users behaving honestly (following their corresponding $\Sigma_{B_i}^{\mathbb{T}}$ strategies) and that started sufficiently early to complete the setup (so before $t_0 - depth(\mathcal{T})\Delta$) and in a state were all funds to be used by the protocol are available, there is an outcome $\omega_{id}^* \in \bigcap_{B \in Hon} \mathcal{O}_B^{\mathcal{T}}$ for every tree $\mathcal{T}$ in $\mathbb{T}$ such that these ω_{id}^* correspond exactly to all CTLCs that have been claimed during R. Intuitively, such outcomes ω_{id}^* represent the protocol outcomes for tree $\mathcal{T}$ with identifier *id* that all honest users agree upon. In the case of $\mathcal{T} = unfold(\mathcal{D}, A)$, ω_{id}^* corresponds to the execution of the whole graph $\mathcal{D}$ (as proven in Theorem 3.4), which gives us the end-to-end correctness guarantee that if all preconditions of Theorem 4.3 are met, the protocol will execute all transactions described by $\mathcal{D}$. A formal proof of this statement is given in Appendix H.3.

Challenges. While realizing the game-playing logic of the ttree with the help of secrets and timeouts may seem intuitive at first sight, proving the exact security and correctness guarantees that an honest user can achieve in a highly adversarial HBE comes with distinct challenges: An honest user B only has control over those CTLCs in which they are involved, meaning where they act as sender or receiver. The execution of other CTLCs is not only out of the control of B, but their execution may even not be observable by B (since they might happen in a consensus object, such as a payment channel, that B does not have access to). Consequently, B needs to be able to enforce local security guarantees based on their local view and capabilities. This implies, e.g., that a ttree outcome for B needs to consider partial ttree executions that contain duplicates of edges not involving B. Similar to wormhole attacks known from payment channels [14], B cannot be sure that the protocol has been faithfully executed for pulling these edges, or whether these executions have only been simulated towards the honest user B. This is reflected correspondingly in the ttree semantics as given by the outcome sets only excluding duplicate edges for such edges including B.

Further, during the protocol execution, the execution order as dictated by the ttree cannot always be upheld: While intuitively, at least those edges involving B should only be executed in order (an edge e_1 on the path from edge e_2 to the root in $\mathcal{T}$ should be executed before e_2), even this invariant can be violated during protocol execution. After submitting a transaction pulling an ingoing edge $e_1 = (A_1, B)_{\vec{a}_1}$ (corresponding to $\Sigma_B^{\mathbb{T}}$ outputting a *claim* action α_1) an adversary controlling the miner and users $A_1, \ldots, A_n$ on the path between $e_2 = (B, A_n)_{\vec{a}_2}$ and e_1 could publish the transaction pulling e_2 (corresponding to Σ_{Adv} outputting a corresponding *claim* action α_2) before α_1. It hence needs to be proven that a secure state (including both α_1 and α_2) will eventually be reestablished in such a case. This complexity is reflected in the intricacy and size of the proof of the main security invariant proven in Theorem H.14.

5. Related Work

Custom Blockchain Protocols. Due to the heterogeneity of the blockchain landscape, existing blockchain protocols are tailored to specific applications (like swaps or payments) and consensus objects (like different cryptocurrencies or payment channels). We present a (possibly incomplete) list of these protocols in Table 1, where we focus on the applications that are supported by our framework, as we further discuss in Section 7.

These protocols do not generalize beyond the intended application and consensus object that they were designed for. Consequently, they require individual yet careful security analysis, which at best comes in the form of complex manual cryptographic proofs [18]. Designing and proving these highly specific blockchain protocols secure is an error-prone task as demonstrated by the security flaws found so far in blockchain protocols proposed by both academia [19], [20] and industry [14], [21].

In contrast to these works on specialized blockchain protocols, our framework synthesizes from a high-level specification of the protocol goals (provided as ATG) a concrete protocol compatible with a variety of different consensus objects. This process is proven correct, resulting in a generic correctness and security proof of the synthesized protocol. As we show in Section 7, our framework supports many of the applications existing today. Furthermore, our framework generalizes beyond those applications, enabling new applications for which not even custom blockchain protocols exist today.

While still being specific to cryptocurrencies with support for stateful smart contracts, the results from [22], [23]

Applications	Consensus Objects
2-party Atomic Swaps	Turing Complete [3], HTLC [8], [9], Digital Signatures [13], Trusted Hardware [16]
n-party Atomic Swaps	Turing Complete [22], [23]
Multi-hop Payments	Turing Complete [24], HTLC [25], [26], Digital Signatures [14], Trusted Hardware [17]
Atomic Multi-Path Payments	HTLC [10], [11], [12], Digital Signatures [15]
Rebalancing	Turing Complete [4], [5], [6], [7], Digital Signatures [27]
Loop-in	HTLC [29]
Crowdfunding	Turing Complete [30], Digital Signatures [31]

TABLE 1: Summary of the existing blockchain protocols for the different applications and consensus objects.

are the closest to our work in that they provide protocols for a full class of applications, namely cross-currency swaps for different user and swap topologies. This application class is still more narrow than the one expressible with the help of ATGs, and the security of the protocols presented in [22], [23] is not analyzed in a realistic blockchain execution model. Still, due to their more general nature, we consider these works most suitable to conduct a performance comparison, demonstrating that even when applying our framework to this restricted application class, the resulting protocols come with performance benefits.

Frameworks for n-party Atomic Swap Protocols. [22] presents a general protocol for cross-currency swaps among several users as can be represented by strongly connected graphs. While starting from a similar (though more restricted) specification format, the presented protocol substantially differs from ours, preventing further generalization. More precisely, the protocol relies on locking funds by transferring them into a stateful smart contract and making their release to the receiver subject to a complex unlocking procedure. This unlocking procedure requires the smart contract to store a copy of the whole graph. For unlocking, the receiver of the transfer then needs to perform operations for all its paths in the graph to all nodes in a dedicated set of leaders, which (depending on the graph structure) may encompass all graph nodes.

This is reflected in the asymptotic complexity of the protocol shown in Table 2. Here, as in [23] we consider *local time*: an upper bound on the computation cost to process the unlocking of an arc; and *local space*: the total amount of bits that are stored in the blockchain per arc.

The authors of [22] point out that inherent performance limitations stem from their protocol design relying on a set of leaders and pose it as an open challenge to develop a protocol based on a single leader. Our work, as a side product, addresses this open research problem, resulting in protocols with immediate asymptotic performance benefits as compared to the protocol presented in [22]. This is as our protocol does not require storing the whole graph structure, and additionally, the number of operations for unlocking funds is bounded by the number of subcontracts in a CTLC, hence, scaling at most linearly with the number of graph nodes.

Interestingly, [23] propose a protocol to improve on the asymptotic performance drawbacks of [22] (see Table 2) and matches our asymptotic performance without relying on a single leader. This is achieved by making the unlocking procedure independent of individual paths in the graph but only dependent on the length of these paths. However, the corresponding check still requires checking a complex condition that can only be realized with expressive smart contracts. For a more detailed comparison with [22], [23], we refer the reader to Section B.

	Local Time	Local Space
Herlihy [22]	$O(\mathcal{N} \cdot \mathcal{L})$	$O(\mathcal{A})$
Imoto et al. [23]	$O(\mathcal{N})$	$O(\mathcal{N})$
Ours	$O(\mathcal{N})$	$O(\mathcal{N})$

TABLE 2: Comparison of asymptotic complexity for [22], [23] and ours. Here $\mathcal{L}$ denotes the set of leaders.

6. *CTLC* Implementation

Here, we compare the concrete performance of our work to that of [22], [23] in order to show that CTLC-based protocols compare favorably to protocols that are tailored to more restricted settings. We then describe how to implement *CTLC*s in cryptocurrencies with restricted scripting languages (e.g., Bitcoin) and in payment channels. We thereby demonstrate that our work subsumes previous works in practice by covering a wider range of consensus objects while being at least as efficient.

6.1. Performance Evaluation and Comparison

Since the smart contracts required in [22], [23] are only known to be realizable in Ethereum-like cryptocurrencies, we compare them with ethCTLC, our Ethereum-based implementation of *CTLC* described in Appendix A. The source code is made available at [32]. We have tested all of them in an Ethereum development blockchain as spawned by Ganache [33]. We have used the Truffle framework [34] to develop and test the three smart contracts, using as input the arc $B \to C$ in the running example of Figure 4. Our evaluation results are included in Table 3. We show the gas cost to deploy the contract, claim an arc and refund an arc.

We observe that the gas costs reflect the asymptotic performance of the compared protocols. For instance: the contract in [22] needs to store a complete copy of the graph and consequently has the highest deployment cost. Similarly, the contract in [22] also has the worst gas cost for the claim operation, reflecting the worst asymptotic performance of this approach. Perhaps more interestingly, we additionally observe that in the best case, both [22] and [23] have better gas cost for refund than ethCTLC. This gap comes from the different approaches to handling the refund of an arc.

	Deploy	Claim		Refund	
		Best	Worst	Best	Worst
Herlihy [22]	2080742	385027		34809	315129
Imoto et al. [23]	1278108	57433	92213	34417	
Ours	865120	45478	96427	80666	

TABLE 3: Comparison of gas cost for [22], [23] and ours.

In [22] and [23], the refund is implemented as a *timeout*, that is, funds can be claimed until a certain time t, after which refunding the arc is the only option allowed. In ethCTLC, the refund is implemented as a *timelock*, that is, the enabling of the i-th subcontract is not allowed until time t_i and doing so effectively disables subcontract $i-1$. Consequently, the refund of the arc requires to have previously disabled all subcontracts. In a nutshell, using timeouts makes it possible to implement refund as a constant operation, while timelocks require a number of operations linear in the number of subcontracts. While we could have adopted the refund approach based on timeouts, we decided otherwise because timeouts are only realizable in cryptocurrencies with expressive smart contracts. By using timelocks instead, *CTLC* can be realized in cryptocurrencies with restricted smart contracts (e.g., Bitcoin) and support many more applications (c.f. Section 7).

A More Generic *CTLC* Implementation. Contrary to the smart contracts in [22], [23] which require a complex smart contract language, implementing *CTLC*s only requires support for (i) authorization of transactions based on a digital signature; (ii) timelock functionality; and (iii) validation of several (condition, secret) pairs. Many cryptocurrencies existing today, including Bitcoin, offer (i), (ii), and (iii), enabling to implement *CTLC*s in them (c.f. Appendix A). Going beyond, next, we describe a *CTLC* implementation based on adaptor signatures [35], compatible with restricted consensus objects that do not offer (iii) (e.g., Monero).

An adaptor signature enables the creation of a digital signature on a transaction conditioned on the knowledge of a cryptographic secret. Imagine that users A and B have a shared account vk_{AB} (i.e., a shared public key) with α coins. Then, A and B can jointly *pre-sign* a transaction m spending from vk_{AB} with respect to a condition Y. Afterward, each user on their own can *adapt* the pre-signature into a valid signature using the secret s corresponding to Y.

An adaptor signature-based *CTLC* is shown in Figure 8. Rounded boxes denote transactions, and squared boxes within represent the funds. The diamond symbol represents the choices for transferring the funds, each of which is denoted by an arrow. The text over the arrow denotes the required conditions to take this choice, namely (i) $vk_{AB}, \vec{Y}_i$ requires a signature from both A and B and the secrets for conditions $\vec{Y}_i$; (ii) t_i requires that time t_i has passed. Intuitively, each transaction corresponds to a subcontract where (i) encodes claiming it and (ii) encodes disabling it.[4]

More concretely, to setup such a *CTLC*, for each transaction tx_i both parties (i) create a pre-signature $\hat{\sigma}_i$ with respect to conditions $\vec{Y}_i$ to spend the asset to B; and (ii) sign the transfer of the asset to tx_{i+1}. One technical subtlety here is that adaptor signatures support pre-signatures with respect to a single condition Y whereas *CTLC*s require several conditions $\vec{Y}_i$ per pre-signature. Fortunately, as shown in [18], one can securely merge $\vec{Y}_i$ conditions into Y^* and several secrets $\vec{s}_i$ into s^* such that s^* is a valid secret for Y^* iff secrets $\vec{s}_i$ are the valid ones for the conditions $\vec{Y}_i$.

4. For simplicity, we show a single claim option per subcontract. If needed (e.g., when there are duplicated edges on the same level in a ttree), additional claim operations can be encoded by adding spending conditions of the type $(vk_{AB}, \vec{Y}_i)$ to the corresponding transaction.

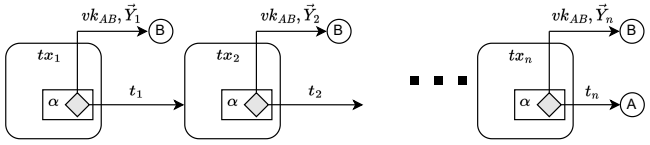

Figure 8: CTLC implementation in UTXO cryptocurrencies.

After all pre-signatures and signatures are created and verified by both parties, the *CTLC* is set up. To execute it, B can *adapt* the pre-signature on tx_1 into a valid signature with secrets $\vec{s}_1$, thereby getting the funds and finalizing the *CTLC* contract. Otherwise, after t_1 expires, A can use the signature on tx_1 to transfer the funds to tx_2, effectively enabling the next subcontract. This process is repeated until (1) at any step i, B *adapts* the pre-signature on tx_i using the secrets $\vec{s}_i$; or (2) A gets the funds back after t_n expires.

6.2. *CTLC* Implementation in Payment Channels

A payment channel (PC) allows two users to securely perform an arbitrary amount of instantaneous transactions between each other while including only two transactions on the blockchain: one for opening the PC and one to close it. To open a PC, two users publish a transaction that locks their funds into a shared account (e.g., vk_{AB}). Afterward, both users have the guarantee that funds in vk_{AB} can only be transferred through (possibly many) transactions that they jointly sign and invalidate. Such set of transactions is called a PC state. The PC lifetime ends when one of the users decides to publish the last state on the blockchain. Moreover, if one of the users publishes a previously invalidated state, the counterparty can obtain all funds in the PC.

***CTLC* Implementation in PCs.** Assume that users A and B have already opened a PC with α coins in it. Then, it is possible to lift the implementation of a *CTLC* based on adaptor signatures (c.f. Figure 8) to the PC setting. For that, both A and B can compute a new PC state containing all transactions shown in Figure 8 along with set of pre-signatures and signatures for each transaction. At that point, both users have the guarantee that they can execute the *CTLC* in the PC. For instance, assume that timeout t_1 is expired, then A can either (i) agree with B to create another PC state that considers only transactions from tx_2 to tx_n; or (ii) publish tx_2 in the blockchain, effectively closing the PC at the current state. In the latter case, the execution of the remaining *CTLC* stays as described earlier.

7. Applications

Here, we overview blockchain applications that can be specified as an ATG and thus benefit from this work in that we provide a generic protocol that realizes them that comes with security and correctness proofs. Note that as shown in Figure 9, the ATGs specifying some of these applications are in-semiconnected but not strongly connected, and transfers are conducted on multiple different consensus objects (e.g., Bitcoin blockchain and different PCs). This demonstrates that our work subsumes previous works in that it supports a larger class of applications across HBE.

Multi-hop Payments in Payment-Channel Networks [14]. In a payment-channel network (PCN), a payment between two users that do not share a PC is routed through a path of intermediaries between such two users, hence called multi-hop payment. In the illustrative example in Figure 9, A intends to pay 3 coins to E through the path B, C, and D. Each intermediary receives a payment of value 3 from its predecessor and forwards it to its successor in the path. The protocol in [14] corresponds to our *CTLC*s-based protocol where the *CTLC* is implemented with adaptor signatures. Therefore, the protocol in [14] (and the rest we discuss in this section) can be seen as instances of our framework.

Rebalancing in PCN [36]. The value that can be transferred in a PC between A and B is limited by the funds they include when creating the PC. Assume that A decides to lock 10 coins in the PC with Bob. Then, A can use the PC for a value up to 10. After that, the PC is depleted and A can no longer use it to pay B. To overcome this situation, A can perform a cyclic payment to themself to receive back coins from B in the PC shared with them. In Figure 9, A performs a cyclic payment where they pay 10 to F and receive back 10 from B, effectively rebalancing 10 coins from the channel with B into the channel with F.

Loop-in [29]. Rebalancing requires a cycle in the PCN with enough capacity. An alternative when such a cycle does not exist is to combine a PCN multi-hop payment with an on-chain transaction. In the example shown in Figure 9, A can pay 10 coins, here BTC, on-chain to I, which will then forward it to A by a multi-hop payment in the PCN using B and C as intermediaries. This protocol is called loop-in since on-chain BTC are later used inside the PCN. This is an example of an HBE where users are involved in totally different consensus objects, namely a blockchain and PCs.

Atomic Multi-path Payments [10]. In a PCN, the amount of coins that can be transferred in a multi-hop payment is restricted by the PC with the lowest capacity. To overcome this restriction, atomic multi-path payments consider several paths between sender and receiver so that the sum of capacities available at each of the paths is enough for the intended payment amount. In Figure 9, A can pay E an amount larger than 3 (i.e., 5 in this case) by forwarding the remaining 2 coins by the additional path F, G, and H.

Crowdfunding [31]. A similar application to atomic multi-path payments is crowdfunding. Instead of having a single sender, an atomic crowdfunding allows multiple users to jointly pay a single user. Atomicity here denotes the intuitive property that either all users contribute the preagreed amount to the crowdfunded user, or all the funders get their coins back. In Figure 9, A contributes 3 coins and I contributes 10 coins to fund E with 13 coins in total.

Composing Applications. In practice, the applicability of our framework is not restricted to applications that can be characterized by ATGs. That is because such applications can often be transformed into a structure that fits our framework by adding cyclic transfers that do not incur any financial harm on users.

Consider the scenario where two applications expressed as independent in-semiconnected graphs should be executed atomically. For example, in Figure 9, assume a user A is interested in atomically executing a loop-in and PCN rebalancing where they are involved. The union of the ATGs describing those applications would not be in-semiconnected and hence out of scope of our framework. This issue could easily be solved by adding a cycle payment between the nodes of A in these two graphs. Note that by adding such a cycle payment, A does not incur any loss as long as the funds that are sent and received are the same. Moreover, such an amount can be arbitrarily small (e.g., the smallest amount supported by the underlying consensus object) since it is independent of the other applications.

We can formally capture this observation as a general composition result: If $\mathcal{D}_1$ is an ATG in-semiconnected in node A_1 and $\mathcal{D}_2$ is an ATG in-semiconnected in node A_2. Then the union of $\mathcal{D}_1$ and $\mathcal{D}_2$ is in-semiconnected in A_1 and A_2 when adding the edges (A_1, A_2) and (A_2, A_1) to it. A formal proof for this statement is given in Theorem C.8.

8. Conclusion

We present a framework for secure-by-design protocols for heterogenous blockchain environments. The framework encompasses (i) the provably correct and secure translation from atomic transaction graph specifications into transaction trees, an intermediate layer representing protocols as interactive funds redistribution games among users in different consensus objects; and (ii) a generic protocol that realizes transaction trees from a simple smart contract building block that we call Conditional Timelock Contract (CTLC).

Our framework subsumes previous work in theory (i.e., supporting a larger class of applications with end-to-end security and correctness proofs in a realistic blockchain model) and in practice (providing instantiations for CTLCs in virtually any cryptocurrency and payment channels).

References

[1] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, "Sok: Communication across distributed ledgers," in *Financial Cryptography and Data Security (FC)*, 2021.

[2] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *Financial Cryptography and Data Security (FC)*, 2020.

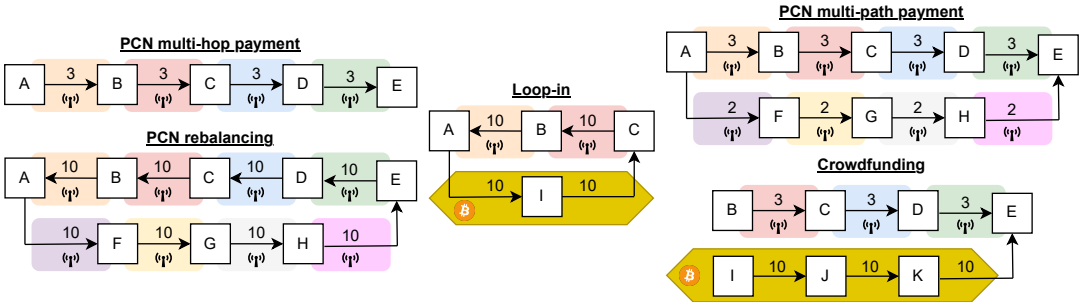

Figure 9: Example of applications covered with our work. Squared boxes represent users. Different consensus objects are represented in differently colored boxes: payment channels (rounded) and Bitcoin blockchain (hexagon).

[3] C. Baum, B. David, and T. Frederiksen, "P2dex: Privacy-preserving decentralized cryptocurrency exchange," Cryptology ePrint Archive, Paper 2021/283, 2021, https://eprint.iacr.org/2021/283. [Online]. Available: https://eprint.iacr.org/2021/283

[4] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," Cryptology ePrint Archive, Paper 2017/823, 2017, https://eprint.iacr.org/2017/823. [Online]. Available: https://eprint.iacr.org/2017/823

[5] Z. Avarikioti, K. Pietrzak, I. Salem, S. Schmid, S. Tiwari, and M. Yeo, "Hide & seek: Privacy-preserving rebalancing on payment channel networks," Cryptology ePrint Archive, Paper 2021/1401, 2021, https://eprint.iacr.org/2021/1401. [Online]. Available: https://eprint.iacr.org/2021/1401

[6] Z. Hong, S. Guo, R. Zhang, P. Li, Y. Zhan, and W. Chen, "Cycle: Sustainable off-chain payment channel network with asynchronous rebalancing," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022, pp. 41–53.

[7] Z. Ge, Y. Zhang, Y. Long, and D. Gu, "Shaduf: Non-cycle payment channel rebalancing," in *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/auto-draft-254/

[8] B. Community, "Atomic swap," https://en.bitcoin.it/wiki/Atomic_swap.

[9] S. Bowe and D. Hopwood, "Hashed time-locked contract transactions," https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki.

[10] C. Fromknecht and O. Osuntokun, "Bolt 21: Atomic multi-path payments," 2021, https://github.com/cfromknecht/lightning-rfc/blob/bolt-amp/21-atomic-multi-path-payments.md.

[11] D. Piatkivskyi and M. Nowostawski, "Split payments in payment networks," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings*, ser. Lecture Notes in Computer Science, J. García-Alfaro, J. Herrera-Joancomartí, G. Livraga, and R. Rios, Eds., vol. 11025. Springer, 2018, pp. 67–75. [Online]. Available: https://doi.org/10.1007/978-3-030-00305-0_5

[12] V. K. Bagaria, J. Neu, and D. Tse, "Boomerang: Redundancy improves latency and throughput in payment networks," *CoRR*, vol. abs/1910.01834, 2019. [Online]. Available: http://arxiv.org/abs/1910.01834

[13] S. A. Thyagarajan, G. Malavolta, and P. Moreno-Sanchez, "Universal atomic swaps: Secure exchange of coins across all blockchains," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1299–1316.

[14] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," 2018. [Online]. Available: https://eprint.iacr.org/2018/472

[15] S. Mazumdar and S. Ruj, "Cryptomaze: Privacy-preserving splitting of off-chain payments," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1060–1073, 2023.

[16] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Conference on Computer and Communications Security, (CCS)*, 2019.

[17] J. Lind, I. Eyal, F. Kelbert, O. Naor, P. R. Pietzuch, and E. G. Sirer, "Teechain: Scalable blockchain payments using trusted execution environments," *CoRR*, vol. abs/1707.05454, 2017. [Online]. Available: http://arxiv.org/abs/1707.05454

[18] E. Tairi, P. Moreno-Sanchez, and C. Schneidewind, "Ledgerlocks: A security framework for blockchain protocols based on adaptor signatures," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds. ACM, 2023, pp. 859–873. [Online]. Available: https://doi.org/10.1145/3576915.3623149

[19] N. Glaeser, M. Maffei, G. Malavolta, P. Moreno-Sanchez, E. Tairi, and S. A. K. Thyagarajan, "Foundations of coin mixing services," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 1259–1273. [Online]. Available: https://doi.org/10.1145/3548606.3560637

[20] P. Gerhart, D. Schröder, P. Soni, and S. A. K. Thyagarajan, "Foundations of adaptor signatures," in *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. Joye and G. Leander, Eds., vol. 14652. Springer, 2024, pp. 161–189. [Online]. Available: https://doi.org/10.1007/978-3-031-58723-8_6

[21] J. Harris and A. Zohar, "Flood & loot: A systemic attack on the lightning network," *CoRR*, vol. abs/2006.08513, 2020. [Online]. Available: https://arxiv.org/abs/2006.08513

[22] M. Herlihy, "Atomic cross-chain swaps," 2018. [Online]. Available: https://doi.org/10.48550/arxiv.1801.09515

[23] S. Imoto, Y. Sudo, H. Kakugawa, and T. Masuzawa, "Atomic cross-chain swaps with improved space, time and local time complexities," *Information and Computation*, vol. 292, p. 105039, 2023.

[24] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," Cryptology ePrint Archive, Paper 2018/320, 2018, https://eprint.iacr.org/2018/320. [Online]. Available: https://eprint.iacr.org/2018/320

[25] J. Poon and T. Dryja, "Lightning network," 2016, https://lightning.network.

[26] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," Cryptology ePrint Archive, Paper 2017/820, 2017, https://eprint.iacr.org/2017/820. [Online]. Available: https://eprint.iacr.org/2017/820

[27] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," Cryptology ePrint Archive, Paper 2019/583, 2019, https://eprint.iacr.org/2019/583. [Online]. Available: https://eprint.iacr.org/2019/583

[28] M. Bartoletti and R. Zunino, "Bitml: A calculus for bitcoin smart contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 83–100. [Online]. Available: https://doi.org/10.1145/3243734.3243795

[29] L. Labs, "Loop," 2024, https://lightning.engineering/loop/.

[30] P. Moreno-Sanchez, T. Ruffing, and A. Kate, "Pathshuffle: Credit mixing and anonymous payments for ripple," *Proc. Priv. Enhancing Technol.*, vol. 2017, no. 3, p. 110, 2017. [Online]. Available: https://doi.org/10.1515/popets-2017-0031

[31] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM, 2019, pp. 801–815. [Online]. Available: https://doi.org/10.1145/3319535.3345666

[32] Anonymous, "Extended version of this work and ctlc-implementation in anonymized github repository," 2024, https://anonymous.4open.science/r/Extended-Version-and-CTLC-Implementation-E6D1/README.md.

[33] T. S. Community, "What is ganache?" 2022, https://trufflesuite.com/docs/ganache/.

[34] ——, "Truffle: The most comprehensive suite of tools for smart contract development," 2022, https://trufflesuite.com.

[35] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostáková, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Generalized channels from limited blockchain scripts and adaptor signatures," in *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. Tibouchi and H. Wang, Eds., vol. 13091. Springer, 2021, pp. 635–664. [Online]. Available: https://doi.org/10.1007/978-3-030-92075-3_22

[36] Z. Avarikioti, S. Schmid, and S. Tiwari, "Musketeer: Incentive-compatible rebalancing for payment channel networks," Cryptology ePrint Archive, Paper 2023/938, 2023, https://eprint.iacr.org/2023/938. [Online]. Available: https://eprint.iacr.org/2023/938

[37] J. Bang-Jensen and G. Gutin, *Classes of Directed Graphs*, ser. Springer Monographs in Mathematics. Springer Cham, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-71840-8

Appendix A. *CTLC* Implementations

In this section, we present more details on the concrete implementations of the *CTLC* for the different settings considered in this work.

A.1. ethCTLC: Implementation Details

We provide an excerpt of ethCTLC, the Ethereum-based implementation of *CTLC*, in Figure 10.

We show an excerpt of the *CTLC* smart contract between a party and a counterparty in Figure 10. When a *CTLC* is created, the first subcontract is enabled (i.e., by having the variable *height* pointing to it). The contract permits the counterparty to claim the current subcontract by providing the secrets for the corresponding conditions (i.e., claim function). Alternatively, the party can disable the current subcontract when the corresponding timeout expires, thereby enabling the next subcontract (i.e., disableSubcontract function). Finally, the party can get refunded if the last subcontract is enabled and its corresponding timeout is expired (i.e., refund function).

```
contract ethCTLC {

address payable party; // sender
address payable counterparty; // receiver
uint[] timelock; // One per subcontract
bytes32[][][] conditions; // One per (
    subcontract, path)
uint height = 0; // pointer to current
    subcontract

function claim(uint i, uint j, bytes [] memory
     secrets) {
    require(msg.sender == counterparty);
    require (i == height);
    require (secrets.length == conditions[i][j
        ].length);
    for (uint k = 0; k < secrets.length; k++)
        require (H(secrets[k]) == conditions[i
            ][j][k]);
    counterparty.transfer(address(this).
        balance);}

function disableSubcontract (uint i) {
    require(msg.sender == party);
    require (i == height);
    require(block.number >= timelock[i]);
    height ++; }

function refund () {
    require(msg.sender == party);
    require(height == timelock.length -1);
    require (block.number >= timelock[height])
        ;
    party.transfer(address(this).balance);}}
```

Figure 10: Pseudocode for *CTLC* smart contract.

A.2. *CTLC* Implementation in Bitcoin

To show the implementation of *CTLC*s in Bitcoin, we leverage BitML [28]. BitML is a domain-specific language for specifying contracts that describe transfers of bitcoins among a set of users without relying on a trusted intermediary. A compiler is then provided to translate BitML contracts into Bitcoin transactions. Participants can execute the contract by appending these transactions to the Bitcoin blockchain according to compiled strategies for each user.

Therefore, to implement *CTLC*s in Bitcoin we can express *CTLC*s in BitML (c.f. Figure 11) and use the existing compiler to extract the corresponding Bitcoin transactions.

$CTLC(B, C) := Pay1 + \text{after } t_1 : Pay2 + \text{after } t_2 : \text{withdraw } B$
$Pay1 := \text{reveal } s_1 \wedge s_3 \wedge s_5 \text{ then withdraw } C$
$Pay2 := \text{reveal } s_6 \wedge s_8 \text{ then withdraw } C$

Figure 11: BitML contract implementing a *CTLC* for the arc $B \to C$ as required in the running example atomic swap graph.

Appendix B. Extended Related Work

In this section, we describe and compare in detail with the works in [22], [23], where the authors present a framework to design protocols for the concrete application of atomic swaps.

Detailed Comparison with [22]. Herlihy presents in [22] a protocol for realizing strongly connected swap graphs. The protocol relies on a set of leaders forming a minimal feedback vertex set, meaning that removing the leaders from the graph results in an acyclic graph, and thus there is a unique path from each node to all leaders. The funds for each arc are getting locked in a complex (Ethereum-style) smart contract which enforces the protocol execution along the paths to the leaders, each of which holds an individual secret. The swap execution starts when leaders partially unlock ingoing arcs in the first round by providing their secret and a signature on it. In the next round, every user whose outgoing arc was partially unlocked does the same for their ingoing arcs by providing the learned secret and their signature on the signature obtained in the previous round (as proof for the path to the leader through which the secret was obtained). This procedure continues until all arcs have been reached via all their paths to all leaders. Consequently, for claiming a fund, the corresponding arc must have been fully unlocked meaning that for all paths to all leaders, the corresponding leader secret and a nested signature on that secret of all users on the path to the leader must have been provided in the adequate round. A partial lock is implemented using a primitive called *hash key* which checks that before a predefined timeout, the preimage for some hash value and a nested signature on that preimage for a specific user path is provided. If in any round, the expected partial unlocking of an arc did not happen, the funds of that arc can be refunded to the owner.

The work proves that this protocol is uniform, meaning that 1) if all parties follow the protocol, then all arcs are executed (corresponding to our ttree correctness result); 2) a party following the protocol can never end up *underwater*, meaning that at least one outgoing arc is triggered, whereas at least one ingoing arc is not triggered (roughly corresponding to our ttree security result).

They further prove an impossibility result showing that no such uniform swap protocol can exist for graphs that are not strongly connected. At first, this seems contradictory to our result (that proves the security of unfolding for in-semiconnected graphs). However, we opt for a slightly relaxed security notion: Instead of requiring for a user to be underwater to end up with at least one outgoing arc triggered and at least one ingoing arc missing, we only require that if an outgoing arc has been triggered also *all* ingoing arcs (which could potentially be none) must have been triggered. This means in particular that if a graph contains a user that functions as a sender only, the security notion still applies in a meaningful way. We show in Section 7 that such scenarios, indeed, find applicability in practice. For strongly connected graphs, our security notion coincides with the one from [22].

A main advantage of our protocol is that it operates using a single leader. This advantage is two-fold. First, having a single leader enhances the liveness of the protocol: Leaders are in the position to block the execution (similar to how user A can force the protocol to timeout by not pulling their ingoing edge in the example presented in Section 2) without encountering financial harm. Second, in [22], it is observed that in single-leader scenarios no digital signatures and hashkeys are needed but only timeouts, and finding such a protocol for the general case is posed as an open challenge. Our work solves this challenge and, with that, brings multiple practical benefits (as demonstrated in Section 6): 1) The logic of *CTLC*s, our smart contract for realizing edges, is much simpler (since no hash keys need to be checked), resulting in improved on-chain performance, even when implementing *CTLC*s on Ethereum. 2) The on-chain computation cost for executing a swap is asymptotically lower: In the protocol [22], in the worst case, users need to perform as many transactions unlocking the contract for an arc before claiming it as there are paths to all leaders in $\mathcal{D}$. In our protocol, the worst case occurs if all *CTLC* edges time out, requiring disabling as many subcontracts as there are paths to the (single) leader. 3) Our protocol has improved honest on-chain execution cost: If all protocol participants behave honestly, they can claim the assets for their arcs with a single blockchain transaction. In contrast, even in the honest case, the protocol from [22] requires unlocking all hash keys; 4) The smart contract based on hash keys is only known to be realizable in blockchains with expressive smart contract languages, while *CTLC*s can be realized in blockchains with limited or even no scripting capabilities, enabling the implementation of our protocol among all existing cryptocurrencies and even on layer-two solutions, further reducing the on-chain cost.

Detailed Comparison with [23]. Imoto et al. [23] build upon the work in [22] to propose a protocol that improves its performance drawbacks. As in [22], the funds for each arc in the graph are locked in a complex (Ethereum-style) smart contract. Different to [22], the funds are guarded by one secret per user. After one such contract per arc has been set up, the proposed protocol proceeds in rounds. In the i-th round, the funds at one arc can be released providing i-many signatures on the secrets of all users. In the first round, any user can claim the funds on one arc with only their own signature on all secrets. But if they do so, those users whose funds where taken away have learnt a signature (in addition to their own) and can claim their ingoing arcs in the next round. Such protocol can take as many rounds

as the diameter of the graph, after which arcs are timed out.

When compared to [22], this work achieves the same security guarantees. However, the security analysis is in the same model of [22], thereby sharing the same limitations. Instead, as we detail in Section 6, this work improves upon [22] in storage requirements and computation cost. When compared to our work, [23] and ours share similar storage requirements and computation cost, an improvement over [22]. Yet, [23] inherits several disadvantages from [22], namely 1) it only considers strongly connected graphs and is restricted to the specific application of atomic cross-chain swaps; and 2) the required smart contract is only known to be realizable in cryptocurrencies with complex (Ethereum-style) smart contracts.

Appendix C. Graph Theory

This section will briefly introduce the needed definitions and properties of directed graphs for this work. Further details and a rigorous treatment of directed graphs can be found in [37].

Definition C.1. *A* directed graph *(or* digraph*)* $\mathcal{D} := (\mathcal{N}, \mathcal{A})$ *consists of a non-empty finite set* $\mathcal{N}$ *of* vertices *or* nodes *and a finite set* $\mathcal{A}$ *of ordered pairs of distinct vertices called* arcs*. For an arc* $a = (A, B) \in \mathcal{A}$ *we call* A *its* sender*,* B *its* receiver *and define the functions*

$$\textit{sender}(a) := pr_1(a) = A$$
$$\textit{receiver}(a) := pr_2(a) = B,$$

where $pr_i(.)$ *is the projection onto the* i*-th component of a tuple. Two nodes* $A, B \in \mathcal{N}$ *are* connected *if there exists an arc* $(A, B) \in \mathcal{A}$ *or* $(B, A) \in \mathcal{A}$*, which will be visualized with* $A \to B$ *or* $B \to A$ *respectively. We require the two vertices forming an arc to be distinct, which removes loops from one node back to itself from the set of arcs [37, p.3].*

Definition C.2. *A* walk *in* $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ *is a finite tuple of arcs*

$$\vec{a} := [a_{k-1}, a_{k-2}, ..., a_1, a_0]$$

for some $|\vec{a}| = k \in \mathbb{N}$ *and*

$$\forall 0 \leq j \leq k-1 : a_j \in \mathcal{A} \textit{ as well as}$$
$$\textit{if } k \geq 2 : \forall 0 \leq i \leq k-2 : \textit{sender}(a_{i+1}) = \textit{receiver}(a_i).$$

We say that $\vec{a}$ *is a* walk from $\textit{sender}(a_{k-1})$ to $\textit{receiver}(a_0)$ *or a* $(\textit{sender}(a_{k-1}), \textit{receiver}(a_0))$-walk *[37, p.7]. We use* $\vec{a} \cdot \vec{a}'$ *to denote the concatenation of two walks* $\vec{a}$ *and* $\vec{a}'$*.*

Definition C.3. *A digraph* $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ *is* strongly connected *if*

$$\forall A, B \in \mathcal{N} : \exists\, (A, B)\textit{-walk} \wedge \exists\, (B, A)\textit{-walk [37, p.7].}$$

Definition C.4. *Let* $A \in \mathcal{N}$ *be an arbitrarily chosen node from* $\mathcal{D}$*, then we define the* extended out-neighbourhood *of* A *as*

$$N^+_{\mathcal{D}}(A) := \{B \in \mathcal{N} \setminus \{A\} \mid \exists\, (A, B)\textit{-walk}\},$$

and the extended in-neighbourhood *of* A *as*

$$N^-_{\mathcal{D}}(A) := \{C \in \mathcal{N} \setminus \{A\} \mid \exists\, (C, A)\textit{-walk}\}.$$

The union of these sets is often referred to as the reachable set of a node, e.g., in [37, p.16].

Definition C.5. *Let* $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ *be a digraph and* $A \in \mathcal{N}$*. Then we call* $\mathcal{D}$ in-semiconnected w.r.t. A *if and only if*

$$N^-_{\mathcal{D}}(A) \cup \{A\} = \mathcal{N}.$$

Example C.6. *The digraph*

$$A \longrightarrow B \longrightarrow C,$$

is in-semiconnected w.r.t. C *but not strongly connected. For that a walk from* C *to* B *and from* B *to* A *has to be added. By including these arcs we obtain*

$$A \rightleftarrows B \rightleftarrows C,$$

which is now strongly connected.

Corollary C.7. *It holds for every digraph* $\mathcal{D} = (\mathcal{N}, \mathcal{A})$*:*

$$\mathcal{D} \textit{ is strongly connected}$$
$$\Rightarrow \mathcal{D} \textit{ is in-semiconnected w.r.t. any } A \in \mathcal{N}$$

Proof: Let $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ be a strongly connected digraph and $A \in \mathcal{N}$ chosen arbitrarily. By Definition C.3 we have

$$\forall B \in \mathcal{N} \setminus \{A\}\, \exists (B, A)\text{-walk}.$$

This implies $N^-_{\mathcal{D}}(A) = \mathcal{N} \setminus \{A\}$.

Theorem C.8. *Let* $\mathcal{D}_1 = (\mathcal{N}_1, \mathcal{A}_1)$ *be in-semiconnected w.r.t.* $A \in \mathcal{N}_1$ *and* $\mathcal{D}_2 = (\mathcal{N}_2, \mathcal{A}_2)$ *in-semiconnected w.r.t.* $B \in \mathcal{N}_2$*. We define* $\mathcal{D}_3 := (\mathcal{N}_3, \mathcal{A}_3)$ *with*

$$\mathcal{N}_3 := \mathcal{N}_1 \cup \mathcal{N}_2,$$
$$\mathcal{A}_3 := \begin{cases} \mathcal{A}_1 \cup \mathcal{A}_2 \cup \{(A, B), (B, A)\} & \textit{,if } A \neq B \\ \mathcal{A}_1 \cup \mathcal{A}_2 & \textit{,if } A = B. \end{cases}$$

Then $\mathcal{D}_3$ *is in-semiconnected w.r.t.* A *and in-semiconnected w.r.t.* B*.*

Proof: By Definition C.5 we have to show that

$$N^-_{\mathcal{D}_3}(A) \cup \{A\} = \mathcal{N}_3$$

holds. Unfolding this statement using Definition C.4 results in:

$$\forall C \in \mathcal{N}_3 \setminus \{A\}\, \exists (C, A)\text{-walk}$$

Firstly, we assume $A = B$. Let $C \in \mathcal{N}_3 \setminus \{A\}$ be arbitrary. If $C \in \mathcal{N}_1$ there is an (C, A)-walk consisting of arcs in $\mathcal{A}_1$. If $C \in \mathcal{N}_2$, the same is true for $\mathcal{A}_2$. Secondly, we assume $A \neq B$. We show that $\mathcal{D}_3$ is in-semiconnected w.r.t. A. For all $C \in \mathcal{N}_1 \setminus \{A\}$ there again is a (C, A)-walk consisting of arcs from $\mathcal{A}_1$. For all $C \in \mathcal{N}_2 \setminus \{B\}$, by definition, a (C, B)-walk $\vec{a}$ exists with arcs from $\mathcal{A}_2$. Additionally, $(B, A) \in \mathcal{A}_3$ and so we get a (C, A)-walk $\vec{a} \cdot [(B, A)]$. And finally, for B there is a (B, A)-walk, more specifically $[(B, A)]$. Analogously we can show that $\mathcal{D}_3$ is in-semiconnected w.r.t. B.

Appendix D. Graph to Tree Conversion

As in the previous Section we notate walks consisting of arcs in $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ with

$$\vec{a} = [a_{|\vec{a}|-1}, a_{|\vec{a}|-2}, ..., a_0] \in \mathcal{A}^{|\vec{a}|}.$$

We will represent trees by their paths from the leaves to the root of the tree.

Tree Unfolding. We first formally define the unfolding of a graph $\mathcal{D}$ into a *game tree*. This unfolding is given as follows:

Definition D.1 (Tree Unfolding)**.** *For a given digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ and leader $A \in \mathcal{N}$ we define the unfolding*

$$\begin{aligned}
&\mathit{unfold}(\mathcal{D}, A) := \{\vec{a} \mid \mathit{walk}(\mathcal{D}, A, \vec{a})\} \text{ with} && (1)\\
&\mathit{walk}(\mathcal{D}, A, \vec{a}) := \exists X \in \mathcal{N} : a_0 = (X, A)\\
&\wedge \forall j \in \{|\vec{a}| - 2, ..., 0\}\, \exists X, Y, Z \in \mathcal{N} : && (2)\\
&\quad a_{j+1} = (Z, Y), a_j = (Y, X)\\
&\wedge \forall i, j \in \{|\vec{a}| - 1, ..., 0\} \forall X, Y, Z,\\
&\quad a_i = (X, Y), a_j = (Z, Y) \Rightarrow X = Z, i = j\\
&\wedge \forall X, Y, a_{|\vec{a}|-1} = (X, Y) \Rightarrow \exists j \in \{|\vec{a}| - 2, ..., 0\}, Z :\\
&\quad (Z, X) = a_j \vee \nexists Z : (Z, X) \in \mathcal{A}.
\end{aligned}$$

The condition walk checks that a path is a walk in $\mathcal{D}$ ending in node A (first two conditions), in particular, whenever an edge a_{j+1} ends in a node Y the next edge a_j should start in Y again. The third condition says that no arc can be revisited. This means whenever there is a node Y with two different edges in the same path, both ending in Y, they should already be the same. The fourth and last condition ensures that the path always ends with a repeated node or because there is no successor. Thus, $\mathit{unfold}(\mathcal{D}, A) =: \mathcal{T}$ is a set of walks. To uniquely identify the edges in $\mathcal{T}$, we index them with their partial walk, starting with the edge itself and ending with the leader, the root of the tree. For example $(X, Y)_{\vec{a}}$ is indexed with $\vec{a} = [(X, Y), ..., (Z, A)]$ for a leader A and some party Z. Note that $\vec{a}$ is a partial path starting with (X, Y) and following one of the paths in $\mathcal{T}$ upwards to the leader.

In the following we will use $\vec{a}_1 \cdot \vec{a}_2$ to denote the concatenation of two paths $\vec{a}_1$ and $\vec{a}_2$. In addition, we will use $\vec{a}_2 \succcurlyeq \vec{a}_1$ to denote that $\vec{a}_1$ is a suffix of $\vec{a}_2$ (so that $\exists \vec{a} : \ \vec{a}_2 = \vec{a} \cdot \vec{a}_1$).

Using this, we define what it means for an edge to be an element in the tree:

$$(X, Y)_{\vec{a}} \mathrel{\widehat{\in}} \mathcal{T} :\Leftrightarrow \exists \vec{a}_1 \in \mathcal{T}, \vec{a}_2 : \vec{a}_1 = \vec{a}_2 \cdot \vec{a}$$

In other words, there is a full walk $\vec{a}_1$ in $\mathcal{T}$ such that $\vec{a}$ is part of it. For edges $(X, Y)_{\vec{a}}, (X', Y')_{\vec{a}'} \mathrel{\widehat{\in}} \mathcal{T}$, we define what it means to be on the same path:

$$\begin{aligned}
&\mathit{onPath}((X, Y)_{\vec{a}}, (X', Y')_{\vec{a}'})\\
&:\Leftrightarrow \exists \vec{a}'' \in \mathcal{T} : \vec{a}'' \succcurlyeq \vec{a} \ \wedge \ \vec{a}'' \succcurlyeq \vec{a}'
\end{aligned}$$

We define the depth of $(X, Y)_{\vec{a}}$ by its tree level or equivalently the length of $\vec{a}$:

$$\mathit{depth}((X, Y)_{\vec{a}}) := |\vec{a}|$$

The set of edges in the tree that are on the same path to the root is then given as:

$$\begin{aligned}
&\mathit{onPathToRoot}(\mathcal{T}, e) :=\\
&\{e' \mathrel{\widehat{\in}} \mathcal{T} \mid \mathit{onPath}(e, e') \wedge \mathit{depth}(e') \leq \mathit{depth}(e)\}
\end{aligned}$$

This implies $e \in \mathit{onPathToRoot}(\mathcal{T}, e)$.

The rest of this section assumes a digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, which has been unfolded into a tree $\mathcal{T}$. With J, we denote all levels of the tree and index it with $j \in J$. The parties in a given tree level j are notated as the set N^j.

Definition D.2. *We define a game tree as a finite set of walks*

$$\begin{aligned}
\mathcal{T} :&= \{\vec{a}_1, \vec{a}_2, ..., \vec{a}_n\},\\
\forall 1 \leq i \leq n : \vec{a}_i &= [a_{|\vec{a}|-1}, a_{|\vec{a}|-2}, ..., a_0]
\end{aligned}$$

were each element of a $\vec{a}_i$ is defined as a tuple $a_j = (X, Y)_{\vec{a}'}$ where X, Y, with $X \neq Y$, come from an underlying set of nodes $\mathcal{N}$ and $\vec{a}' = [a_j, a_{j-1}, ..., a_0]$ with $\vec{a}_i \succcurlyeq \vec{a}'$. As all $\vec{a}_i$ are walks, Def. C.2 holds. For $\mathcal{T}$ to be a game tree, the following two conditions need to hold:

$$\begin{aligned}
&(i)\ \exists! \, Y \in \mathcal{N} \forall \vec{a}_i \in \mathcal{T} \exists X \in \mathcal{N} :\\
&\quad a_0 = (X, Y)_{[(X,Y)]} \textit{ (Leader)}\\
&(ii)\ \forall \vec{a} = [a_{|\vec{a}|-1}, a_{|\vec{a}|-2}, ..., a_0],\\
&\quad \vec{a}' = [a'_{|\vec{a}'|-1}, a'_{|\vec{a}'|-2}, ..., a'_0] \in \mathcal{T} :\\
&\quad \exists i, j \in \mathbb{N} : a_i = a'_j \Rightarrow [a_i, ..., a_0] = [a'_j, ..., a'_0]\\
&\quad \textit{with } \vec{a} \succcurlyeq [a_i, ..., a_0], \vec{a}' \succcurlyeq [a'_j, ..., a'_0] \textit{ (Crossings)}
\end{aligned}$$

Outcome sets. Based on a game tree $\mathcal{T}$, we next define the set $\mathcal{O}_B^{\mathcal{T}}$ of an honest user B, a participant in $\mathcal{T}$. Intuitively, the outcome set contains all possible tree executions that an honest user may observe when eagerly pulling all their ingoing tree edges whenever possible.

Definition D.3 (Outcome Set)**.** *Let $\mathcal{T}$ be a game tree. Assume the following definitions:*

$$\begin{aligned}
\textit{p-walks}(\mathcal{T}) &:= \{\vec{a}' \mid \exists \vec{a} \in \mathcal{T} : \vec{a} \succcurlyeq \vec{a}'\} && (3)\\
\textit{p-trees}(\mathcal{T}) &:= \{\mathcal{T}_p \mid \mathcal{T}_p \subseteq \textit{p-walks}(\mathcal{T})\} && (4)\\
\mathcal{O}_{\textit{full}}(\mathcal{T}) &:= \{\{(X, Y)_{\vec{a}} \mathrel{\widehat{\in}} \mathcal{T}_p\} \mid \mathcal{T}_p \in \textit{p-trees}(\mathcal{T})\} && (5)
\end{aligned}$$

Further, assume the following predicates:

$$\begin{aligned}
&\mathit{NoDup}(\mathcal{T}, B, \omega) :\Leftrightarrow (X, Y)_{\vec{a}} \in \omega \wedge (X, Y)_{\vec{a}'} \in \omega && (6)\\
&\qquad \wedge \ (X = B \vee Y = B) \Rightarrow \vec{a} = \vec{a}'\\
&\mathit{HonestRoot}(\mathcal{T}, B, \omega) :\Leftrightarrow && (7)\\
&(X, B)_{\vec{a}} \mathrel{\widehat{\in}} \mathcal{T} \ \wedge \ \mathit{depth}((X, B)_{\vec{a}}) = 1 \Rightarrow (X, B)_{\vec{a}} \in \omega
\end{aligned}$$

$$\begin{aligned}&\textit{EagerPull}(\mathcal{T}, B, \omega) :\Leftrightarrow \qquad (8)\\ &(X,B)_{\vec{a}_1} \,\widehat{\in}\, \mathcal{T} \;\wedge\; (B,Y)_{\vec{a}_2} \in \omega \wedge \vec{a}_1 = [(X,B)] \cdot \vec{a}_2\\ &\quad \Rightarrow \exists \vec{a}_3 : (X,B)_{\vec{a}_3} \in \omega\\ &\qquad \wedge\; \textit{depth}((X,B)_{\vec{a}_3}) \leq \textit{depth}((X,B)_{\vec{a}_1})\end{aligned}$$

Then the outcome set $\mathcal{O}_B^{\mathcal{T}}$ of user B in $\mathcal{T}$ is given as

$$\begin{aligned}\mathcal{O}_B^{\mathcal{T}} := \{\omega \in \mathcal{O}_{\textit{full}}(\mathcal{T}) \mid \quad & \textit{NoDup}(\mathcal{T}, B, \omega) \qquad (9)\\ \wedge\; & \textit{HonestRoot}(\mathcal{T}, B, \omega)\\ \wedge\; & \textit{EagerPull}(\mathcal{T}, B, \omega)\}\end{aligned}$$

The definition incrementally constructs $\mathcal{O}_B^{\mathcal{T}}$ by first defining the set $\mathcal{O}_{\textit{full}}(\mathcal{T})$ that contains a set of all its edges for each partial tree of $\mathcal{T}$ (Equation (5)). Partial trees (defined in Equation (4)) are given by arbitrary subsets of partial walks (defined in Equation (3)) of the tree $\mathcal{T}$. Finally, $\mathcal{O}_B^{\mathcal{T}}$ restricts $\mathcal{O}_{\textit{full}}(\mathcal{T})$ further to only those outcome sets that do not contain any duplicate edges involving honest user B (described by *NoDup*) and that are compliant with an honest strategy of user B (so satisfying the predicates *HonestRoot* and *EagerPull*): More precisely, Equation (7) requires that if B is the root, so has an ingoing edge $(X,B)_{\vec{a}} \in \mathcal{T}$, then this edge is also included in an outcome set ω. This corresponds to B always pulling all ingoing edges when being the root user. Next, Equation (8) requires that if B has an ingoing edge $(X,B)_{\vec{a}_1} \in \mathcal{T}$ whose outgoing edge $(B,Y)_{\vec{a}_2}$ is included in the outcome set ω then also $(X,B)_{\vec{a}_1}$ (or a duplicate thereof on the same or higher tree level) must be contained in ω. This corresponds to B eagerly pulling all ingoing edges whenever possible.

To show the security of the tree-unfolding, we show that each outcome set $\omega \in \mathcal{O}_B^{\mathcal{T}}$ constitutes a good outcome for user B, meaning that user B does not end up *underwater*. A user is considered underwater if the swap triggers some outgoing arcs of $\mathcal{D}$ for user B but not all their ingoing arcs of $\mathcal{D}$. We capture this notion formally with the following theorem:

Theorem D.4 (Security of tree unfolding)**.** *Let $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ be a digraph that is in-semiconnected w.r.t. $A \in \mathcal{N}$ and $\mathcal{T} = \textit{unfold}(\mathcal{D}, A)$ the tree unfolding of that graph and $B \in \mathcal{N}$ be a node representing a user. Then, it holds that*

$$\begin{aligned}&\forall \omega \in \mathcal{O}_B^{\mathcal{T}} :\\ &\forall \vec{a}, \vec{a}' : \big((X,Y)_{\vec{a}} \in \omega \wedge (X,Y)_{\vec{a}'} \in \omega \wedge B \in \{X,Y\}\\ &\qquad \Rightarrow \vec{a} = \vec{a}' \wedge (X,Y) \in \mathcal{D}\big) \qquad (10)\\ &\wedge \forall (B,Y)_{\vec{a}} \in \omega : \big((X,B) \in \mathcal{D} \Rightarrow \exists \vec{a}' : (X,B)_{\vec{a}'} \in \omega\big) \qquad (11)\end{aligned}$$

Proof: The proof will be carried out by first showing Equation (10) and then showing Equation (11).

Let $\omega \in \mathcal{O}_B^{\mathcal{T}}$ then by definition of $\mathcal{O}$ we have that for all $(X,Y)_{\vec{a}} \in \omega$ that $(X,Y)_{\vec{a}} \,\widehat{\in}\, \mathcal{T}$. Since $\mathcal{T} = \textit{unfold}(\mathcal{D}, A)$ only contains walks consisting of arcs from $\mathcal{D}$. Also, all elements from ω are indexed edges from $\mathcal{D}$.

Further, let $(X,Y)_{\vec{a}}, (X,Y)_{\vec{a}'} \in \omega$ and $B \in \{X,Y\}$. Assume towards contradiction that $\vec{a} \neq \vec{a}'$. By Definition D.3, it holds that $\textit{NoDup}(\mathcal{T}, B, \omega)$ and consequently, by the definition of *NoDup* also $\vec{a} = \vec{a}'$ immediately giving a contradiction.

Let $\omega \in \mathcal{O}_B^{\mathcal{T}}$ and $(B,Y)_{\vec{a}} \in \omega, (X,B) \in \mathcal{D}$. We make a case distinction on whether there exists some edge $(Z,B) \in \vec{a}$.

- Assume that $\vec{a} = \vec{a}_1 \cdot [(Z,B)] \cdot \vec{a}_2$. Since $(B,Y)_{\vec{a}} \in \omega$, by definition of *unfold* there exists $\vec{a}_t \in \mathcal{T}$ such that $\vec{a}_t \succcurlyeq \vec{a}$. In particular, this means that (same as $\vec{a}_t$) also $\vec{a}$ is a walk in $\mathcal{D}$ not visiting any arc twice. Consequently, $\vec{a}_2$ does not contain any other edge (Z',B). And so, also $\vec{a}' = [(X,B)] \cdot \vec{a}_2$ is a walk in $\mathcal{D}$ not visiting any node twice. Consequently, by definition of *unfold*, there must be some $\vec{a}'_t \in \mathcal{T}$ such that $\vec{a}'_t \succcurlyeq \vec{a}'$ and so also $(X,B)_{\vec{a}'} \,\widehat{\in}\, \mathcal{T}$. We do another case distinction on $\vec{a}_2$:
  - If $\vec{a}_2$ is empty, then $\textit{depth}(\vec{a}') = 1$ and hence by $\textit{HonestRoot}(\mathcal{T}, B, \omega)$ it follows from $(X,B)_{\vec{a}'} \,\widehat{\in}\, \mathcal{T}$ that also $(X,B)_{\vec{a}'} \in \omega$.
  - If $\vec{a}_2 = [(B,U)_{\vec{a}_2}] \cdot \vec{a}_3$ then also $(B,U)_{\vec{a}_2} \in \omega$ (because $(B,Y)_{\vec{a}} \in \omega$ by definition of $\mathcal{O}_{\textit{full}}$ implies that also all $(V,W) \in \vec{a}$ are included in ω). By definition of $\textit{EagerPull}(\mathcal{T}, B, \omega)$, it follows from $(X,B)_{\vec{a}'} \,\widehat{\in}\, \mathcal{T}$ and $(B,U)_{\vec{a}_2} \in \omega$ that there must be some $(X,B)_{\vec{a}_4} \in \omega$, which concludes the case.
- Assume that there is no $(Z,B) \in \vec{a}$. Since $(B,Y)_{\vec{a}} \in \omega$, by definition of *unfold* there exists $\vec{a}_t \in \mathcal{T}$ such that $\vec{a}_t \succcurlyeq \vec{a}$. In particular, this means that (as $\vec{a}_t$) also $\vec{a}$ is a walk in $\mathcal{D}$ not visiting any node twice. Then also $\vec{a}' = [(X,B)] \cdot \vec{a}$ is a walk in $\mathcal{D}$ not visiting any node twice. Consequently, by definition, there must be some $\vec{a}'_t \in \mathcal{T}$ such that $\vec{a}'_t \succcurlyeq \vec{a}'$ and so also $(X,B)_{\vec{a}'} \,\widehat{\in}\, \mathcal{T}$. By definition of $\textit{EagerPull}(\mathcal{T}, B, \omega)$, it follows from this and $(B,Y)_{\vec{a}} \in \omega$ that there must be some $(X,B)_{\vec{a}_3} \in \omega$, which concludes the case.

Theorem D.5 (Correctness of Tree Unfolding)**.** *Let $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ be a digraph that is in-semiconnected in $A \in \mathcal{N}$ and $\mathcal{T} = \textit{unfold}(\mathcal{D}, A)$ the tree unfolding of that graph. If all parties $B_i \in \mathcal{N}$ are honest, a representative of every edge in $\mathcal{T}$ will be executed, which means*

$$\begin{aligned}\mathcal{H} := \bigcap_{B_i \in \mathcal{N}} \mathcal{O}_{B_i}^{\mathcal{T}} \cong \mathcal{T} \textit{ i.e.}\\ \mathcal{H}/\sim \;= \{e \,\widehat{\in}\, \mathcal{T}\}/\sim\end{aligned}$$

with $(X,Y)_{\vec{a}} \sim (X,Y)_{\vec{a}'}$ for all $\vec{a}$, $\vec{a}'$.

With $\mathcal{H}/\sim$ and $\{e \,\widehat{\in}\, \mathcal{T}\}/\sim$ the creation of equivalence classes according to '$\sim$' is meant.

Proof:

Let $\omega \in \mathcal{H}$. Then we know that for all $B_i \in \mathcal{N}$, we have $\omega \in \mathcal{O}_{B_i}^{\mathcal{T}}$.

To prove the claim, we show that for all $(X,Y)_{\vec{a}} \in \omega$ it holds that $(X,Y) \in \mathcal{D}$ and that for all $(X,Y) \in \mathcal{D}$ there exists $\vec{a}$ such that $(X,Y)_{\vec{a}} \in \omega$. The first claim follows directly from Theorem D.4. To show the second claim, assume that $(X,Y) \in \mathcal{D}$. We proceed by case distinction:

- If $Y = A$ then by the definition of *unfold* it holds that $(X, A)_{[(X,A)]} \widehat{\in} \mathcal{T}$. Since $\omega \in \mathcal{O}(A, \mathcal{T})$, we also know that *HonestRoot*$(\mathcal{T}, A, \omega)$. Consequently, it immediately follows from the definition of *HonestRoot* that

$$(X, A)_{[(X,A)]} \in \omega.$$

- If $Y \neq A$ then since $\mathcal{D}$ is in-semiconnected, there must be walks from Y to the leader A. These walks can be ranked based on their lengths. With this, we end up with (potentially multiple) shortest walks $\vec{a}_s$ (with $|\vec{a}_s| > 1$) from Y to A. Hence, they visit every node only once. Because there is the possibility of Y appearing multiple times at the same tree level but on different walks, there can be multiple shortest walks with the same length. By definition of *unfold* it holds that for all these walks $\vec{a}_s$ we have that $(X, Y)_{[(X,Y)]\cdot\vec{a}_s} \widehat{\in} \mathcal{T}$. We assume toward contradiction that there is no $\vec{a}_s$ such that $(X, Y)_{[(X,Y)]\cdot\vec{a}_s} \in \omega$.
Then either for all $\vec{a}_s$ it holds that for all $(V, W)_{[(V,W)]\cdot\vec{a}_2}$ with

$$\vec{a}_s = \vec{a}_1 \cdot [(V, W)] \cdot \vec{a}_2$$

for some $\vec{a}_1, \vec{a}_2$ that $(V, W)_{[(V,W)]\cdot\vec{a}_2} \notin \omega$ or there is a walk $\vec{a}_j$ such that $(V, W)_{[(V,W),(W,Z)]\cdot\vec{a}_2} \notin \omega$ with

$$\vec{a}'_s = \vec{a}_1 \cdot [(V, W), (W, Z)] \cdot \vec{a}_2$$

for some $\vec{a}_1, \vec{a}_2$ and $(W, Z)_{[(W,Z)]\cdot\vec{a}_2} \in \omega$ and for all $\vec{a}_s$ it holds that for all $(V', W')_{[(V',W')]\cdot\vec{a}'_2}$ with

$$\vec{a}_s = \vec{a}'_1 \cdot [(V', W')] \cdot \vec{a}'_2$$

for some $\vec{a}'_1, \vec{a}'_2$ and $|\vec{a}'_1| \leq |\vec{a}_1|$ that $(V', W')_{[(V',W')]\cdot\vec{a}'_2} \notin \omega$. So either all elements of shortest walks $\vec{a}_s$ are not contained in ω or there must be a highest level where the shortest walk $\vec{a}'_s$ contains an edge (V, W) not in ω followed by an edge (W, Z) in ω. The first case immediately leads to a contradiction because it implies that also $(V, W)_{[(V,W)]} \notin \omega$ with $\vec{a}_s = \vec{a}_1 \cdot [(V, W)]$. However, since $\omega \in \mathcal{O}(W, \mathcal{T})$, we also know that *HonestRoot*$(\mathcal{T}, W, \omega)$. Consequently, it immediately follows from the definition of *HonestRoot* that $(V, W)_{[(V,W)]} \in \omega$.
We, hence, assume the existence of $\vec{a}'_s$ as described above. Since $\omega \in \mathcal{O}(W, \mathcal{T})$, we also know that *EagerPull*$(\mathcal{T}, W, \omega)$ and hence that there must be a $\vec{a}_3$ such that $(V, W)_{\vec{a}_3} \in \omega$ and

$$|\vec{a}_3| \leq |[(V, W), (W, Z)] \cdot \vec{a}_2|.$$

If $|\vec{a}_3| < |[(V, W), (W, Z)] \cdot \vec{a}_2|$ then $\vec{a}_1 \cdot \vec{a}_3$ would be a shorter walk from Y to A in $\mathcal{D}$ than $\vec{a}'_s$, which contradicts that $\vec{a}'_s$ is a shortest path.
If $|\vec{a}_3| = |[(V, W), (W, Z)] \cdot \vec{a}_2|$ then $(V, W)_{\vec{a}_3} \in \omega$ with $\vec{a}_s = \vec{a}'_1 \cdot \vec{a}_3$ since

$$\begin{aligned} |\vec{a}'_s| &= |\vec{a}_1 \cdot [(V, W), (W, Z)] \cdot \vec{a}_2| \\ &= |\vec{a}_1| + |\vec{a}_3| = |\vec{a}_s| = |\vec{a}'_1| + |\vec{a}_3|. \end{aligned}$$

So there is a shortest walk $\vec{a}_s$ where an edge (V', W') already occurs in ω on a higher level than in $\vec{a}'_s$, which contradicts the assumption that $\vec{a}'_s$ is a shortest walk with the highest level where such edge occurs in ω.

Appendix E. *CTLC* Semantics

An environment is given as

$$\Gamma := [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t]$$

$$\begin{aligned} \Gamma.S_{com} &:= \{\text{committed secrets}\} \\ \Gamma.S_{rev} &:= \{\text{revealed secrets}\} \\ \Gamma.B &:= \{\text{Set of advertised Batches}\} \\ \Gamma.C_{adv} &:= \{\text{advertised } \textit{CTLC}\text{s}\} \\ \Gamma.C_{en} &:= \{\text{enabled } \textit{CTLC}\text{s}\} \\ \Gamma.C_{aut} &:= \{\text{authorizations for } \textit{CTLC}\text{s}\} \\ \Gamma.C_{cla} &:= \{\text{claimed } \textit{CTLC}\text{s}\} \\ \Gamma.F_{av} &:= \{\text{available funds}\} \\ \Gamma.F_{res} &:= \{\text{reserved funds}\} \\ \Gamma.t &:= t \end{aligned}$$

To realize multiple environments at once, we are defining

$$\vec{\Gamma} := [\Gamma_1, ..., \Gamma_{|\vec{\Gamma}|}],$$

where every Γ_{ch} is defined as Γ above. The index ch stands for "channel" and identifies one of the elements in $\vec{\Gamma}$.

To identify who is a participant in a given environment, we define the assignment function $users$, which assigns users from $\vec{\Gamma}$ to every $\Gamma_{ch} \in \vec{\Gamma}$ with

$$users(\Gamma_{ch}) := \{\text{users participating in } \Gamma_{ch}\}.$$

This function $users$ is given together with $\vec{\Gamma}$ and the notation $users \vdash \vec{\Gamma}$ means that $users$ is defined for all $\Gamma_{ch} \in \vec{\Gamma}$. Additionally, $users$ cannot be altered and stays constant throughout all changes to $\vec{\Gamma}$. In the following, we will always assume that $\vec{\Gamma}$ is given together with $users$, and whenever we write $\vec{\Gamma}$, we mean $users \vdash \vec{\Gamma}$ implicitly. Further, we define

$$\begin{aligned} \vec{\Gamma}.S_{com} &:= \bigcup_{1 \leq ch \leq |\vec{\Gamma}|} \Gamma_{ch}.S_{com}, \\ \vec{\Gamma}.S_{rev} &:= \bigcup_{1 \leq ch \leq |\vec{\Gamma}|} \Gamma_{ch}.S_{rev}, \\ \vec{\Gamma}.t &:= \Gamma_1.t = ... = \Gamma_{|\vec{\Gamma}|}.t, \\ \vec{\Gamma}.B &:= \Gamma_1.B = ... = \Gamma_{|\vec{\Gamma}|}.B \end{aligned}$$

We also define the accumulation analogously for the other components of $\Gamma_{ch} \in \vec{\Gamma}$. For changing only an element Γ_{ch} to Γ'_{ch} within $\vec{\Gamma}$ we use the update notation

$$\vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}].$$

A *CTLC* contract is given as a set of subcontracts $c^x := \{sc_1^x, ..., sc_s^x\}$ with $\forall 1 \leq i \leq s$:

$$sc_i^x := \left[X, Y, f_X^\zeta, \lambda_i, secret - set(sc_i^x)\right]$$

where $X, Y \in \mathcal{N}$, the set of participants in $\vec{\Gamma}$, and

$$secret - set(sc_i^x) := \big\{\{s^{id,1}_{\vec{a}_1}, ..., s^{id,1}_{\vec{a}_{n_i}}\}, \{s^{id,2}_{\vec{a}_1}, ..., s^{id,2}_{\vec{a}_{n_i}}\}, ..., \{s^{id,m}_{\vec{a}_1}, ..., s^{id,m}_{\vec{a}_{n_i}}\}\big\}$$

is a set of secret sets. The used x is a unique identifier of c^x where every subcontract sc^x with the same identifier x is an element of c^x. The f_X^ζ is a token for a fund of user X with a unique identifier ζ. λ_i is a timelock with $\lambda_i \in \mathbb{R}^+$ and if $i < s : \lambda_i \leq \lambda_{i+1}$. The identifier id of the secrets is explained in the following paragraph. For each of those $sc_i^x \in c^x$ we define

$$\mathit{position}(sc_i^x) := i.$$

Advertised but not yet enabled contracts are notated with $\hat{c}^x$. Once they are advertised, they are notated with c^x. For sc^x, we don't use this notation as they are always considered as an element of a contract. The enabling and advertisement of contracts and sub-contracts is defined in the following semantics. A batch is given as a set of *CTLC* contracts $\Psi^{id} := \{\hat{c}_1^x, \hat{c}_2^{x'}, ...\}$. With $\mathit{Hon}(\Psi^{id})$, we denote the set of honest users in a given Ψ^{id}.

The remaining element is a set, which, again, consists of sets containing secrets. Each secret $s^{id,j}_{\vec{a}_\iota}$ is specific to an id, the identifier of Ψ^{id}, and walk $\vec{a}_\iota$. Here, in the standalone definition of *CTLC*s, walks do not have a meaning yet, but this notation will be helpful in the following Appendix F where we bring together *CTLC*s with game trees. For now, view $\vec{a}_\iota$ just as a differentiating index. Furthermore, each secret is owned by $\mathit{owner}(s^{id,j}_{\vec{a}_\iota}) \in users(\Gamma_{ch})$ for some $\Gamma_{ch} \in \vec{\Gamma}$.

We define $s := |c^x|$. To access the individual components, we also define the following functions:

$$\begin{aligned}
\mathit{sender}(c^x) &= \mathit{sender}(sc_1^x) = ... = \mathit{sender}(sc_s^x) = X\\
\mathit{receiver}(c^x) &= \mathit{receiver}(sc_1^x) = ... = \mathit{receiver}(sc_s^x) = Y\\
\mathit{users}(c^x) &= \mathit{users}(sc_1^x) = ... = \mathit{users}(sc_s^x)\\
&= \{\mathit{sender}(c^x), \mathit{receiver}(c^x)\}\\
\mathit{users}(\hat{c}^x) &= \mathit{users}(c^x)\\
\mathit{users}(\Psi^{id}) &= \bigcup_{\hat{c}^x \in \Psi^{id}} \mathit{users}(\hat{c}^x)\\
\mathit{fund}(c^x) &= \mathit{fund}(sc_1^x) = ... = \mathit{fund}(sc_s^x) = f_X^\zeta\\
\mathit{timeout}(sc^x) &= t_0 + \lambda_i \Delta
\end{aligned}$$

$$\begin{aligned}
\mathit{secret}_1(sc_i^x) &= \{s^{id,1}_{\vec{a}_1}, ..., s^{id,1}_{\vec{a}_{n_i}}\}\\
\mathit{secret}_2(sc_i^x) &= \{s^{id,2}_{\vec{a}_1}, ..., s^{id,2}_{\vec{a}_{n_i}}\}\\
&...\\
\mathit{secret}_m(sc_i^x) &= \{s^{id,m}_{\vec{a}_1}, ..., s^{id,m}_{\vec{a}_{n_i}}\}\\
\mathit{secret}(sc_i^x) &= \{\mathit{secret}_1(sc_i^x)\} \cup \{\mathit{secret}_2(sc_i^x)\} \cup ...
\end{aligned}$$

Definition E.1. *$\vec{\Gamma} := [\Gamma_1, ..., \Gamma_{|\vec{\Gamma}|}]$ is called initial if*

$$\begin{aligned}
\forall \Gamma_{ch} \in \vec{\Gamma} : \Gamma_{ch}.S_{com} &= \Gamma_{ch}.S_{rev} = \Gamma_{ch}.B = \Gamma_{ch}.C_{adv}\\
&= \Gamma_{ch}.C_{en} = \Gamma_{ch}.C_{aut} = \Gamma_{ch}.C_{cla}\\
&= \Gamma_{ch}.F_{res} = \emptyset.
\end{aligned}$$

For all $\Gamma_{ch} \in \vec{\Gamma}$ all elements/sets in Γ_{ch} are empty besides $\Gamma_{ch}.F_{av}$. In $\Gamma_{ch}.F_{av}$ there can be funds f_X^ζ for an $X \in users(\Gamma_{ch})$. Each fund $f_X^\zeta \in \vec{\Gamma}.F_{av}$ is unique with an unique identifier ζ. For any contract used in the following semantics, a fund needs to be predefined in this initial environment because funds cannot be created afterward, and every contract needs a unique fund. The following will start with an initial environment.

To shorten the following rule, we define the well-formedness of a batch as a standalone function:

$$\begin{aligned}
&well - formed(\Psi^{id}) :\Leftrightarrow \Psi^{id} = \{\hat{c}_1^x, \hat{c}_2^{x'}, ...\}\\
&\wedge \mathit{Hon}(\Psi^{id}) \neq \emptyset\\
&\wedge \forall \hat{c}^x \in \Psi^{id} : s, \lambda_1, ..., \lambda_s \in \mathbb{N} \wedge \hat{c}^x := \{sc_1^x, ..., sc_s^x\} \text{ with}\\
&\forall 1 \leq i \leq s : sc_i^x := \left[X, Y, f_X^\zeta, \mathit{secret}(sc_i^x), t_0 + \lambda_i \Delta\right],\\
&\text{and if } i < s : \lambda_i \leq \lambda_{i+1}.
\end{aligned} \tag{12}$$

$$\frac{\begin{array}{c}\Psi^{id} \text{ with } well - formed(\Psi^{id}) \wedge \forall \hat{c}^x \in \Psi^{id} :\\ \exists ch : \mathit{fund}(\hat{c}^x) \in \Gamma_{ch}.F_{av}\\ \wedge \forall A \in \mathit{users}(\hat{c}^x) : A \in users(\Gamma_{ch})\\ \wedge \forall sc^x \in \hat{c}^x : \mathit{secret}(sc^x) \cap (\Gamma.S_{com} \cup \Gamma.S_{rev}) = \emptyset,\\ \vec{\Gamma}' := [\Gamma'_1, ..., \Gamma'_{|\vec{\Gamma}|}], \forall 1 \leq ch \leq |\vec{\Gamma}| \text{ set}\\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t],\\ B' := \Gamma_{ch}.B \cup \Psi^{id}\\ \Gamma'_{ch} := [S_{com}, S_{rev}, B', C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t]\end{array}}{\vec{\Gamma} \xrightarrow{\mathit{advBatch}, \Psi^{id}} \vec{\Gamma}'}\ [\mathit{advBatch}] \tag{13}$$

The *advBatch* rule checks that the proposed batch is well formed and that for every included contract the fund is available in a channel, which includes the users included in this contract. Additionally none of the secrets used in this batch should have been used before. Batches are global objects, and so they get copied to all environments. For talking about their secrets we define for $A \in \mathit{users}(\Psi^{id})$

$$\begin{aligned}
S_A(\Psi^{id}) &:= \{s^{id}_{\vec{a}} \mid \exists sc^x \in \hat{c}^x \in \Psi^{id} : \ s^{id}_{\vec{a}} \in \mathit{secret}(sc^x)\},\\
S(\Psi^{id}) &:= \bigcup_{A \in \mathit{users}(\Psi^{id})} S_A(\Psi^{id}).
\end{aligned}$$

$$[commitBatch]\frac{\begin{array}{c}\Psi^{id} \in \Gamma_{ch}.B, A \in users(\Psi^{id})\\ S_A(\Psi^{id}) \cap (\Gamma_{ch}.S_{com} \cup \Gamma_{ch}.S_{rev}) = \emptyset\\ S'_{com} := S_{com} \cup S_A(\Psi^{id})\\ \vec{\Gamma}' := [\Gamma'_1, ..., \Gamma'_{|\vec{\Gamma}|}], \forall 1 \leq ch \leq |\vec{\Gamma}| \text{ set}\\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t],\\ \Gamma'_{ch} := [S'_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t]\end{array}}{\vec{\Gamma} \xrightarrow{A:\, commitBatch\ \Psi^{id}} \vec{\Gamma}'} \tag{14}$$

With the *commitBatch* action a user commits to all secrets appearing in a given batch. It is a global action.

$$[advCTLC]\frac{\begin{array}{c}\Gamma_{ch} \in \vec{\Gamma}, \hat{c}^x \in \Psi^{id} \in \Gamma_{ch}.B, \hat{c}^x \notin \Gamma_{ch}.C_{adv}\\ \forall sc^x \in \hat{c}^x : secret(sc^x) \subseteq \Gamma_{ch}.S_{com},\\ fund(\hat{c}^x) \in \Gamma_{ch}.F_{av},\\ users(\hat{c}^x) \cap Hon(\Psi^{id}) \neq \emptyset,\\ users(\hat{c}^x) \subseteq users(\Gamma_{ch}),\\ C'_{adv} := \Gamma_{ch}.C_{adv} \cup \{\hat{c}^x\},\\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t],\\ \Gamma'_{ch} := [S_{com}, S_{rev}, B, C'_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t],\end{array}}{\vec{\Gamma} \xrightarrow{advCTLC_{ch}\ \hat{c}^x} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]} \tag{15}$$

*CTLC*s are local objects, so advertising one is a local operation.

$$[authCTLC]\frac{\begin{array}{c}\Gamma_{ch} \in \vec{\Gamma}, \hat{c}^x \in \Gamma_{ch}.C_{adv}, (A, \hat{c}^x) \notin \Gamma_{ch}.C_{aut}\\ \big(A = receiver(\hat{c}^x)\\ \vee (A = sender(\hat{c}^x) \wedge (receiver(\hat{c}^x), \hat{c}^x) \in \Gamma_{ch}.C_{aut})\big),\\ fund(\hat{c}^x) \in \Gamma_{ch}.F_{av},\\ C'_{aut} = \Gamma_{ch}.C_{aut} \cup \{(A, \hat{c}^x)\},\\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t],\\ \Gamma'_{ch} := [S_{com}, S_{rev}, B, C_{adv}, C'_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t]\end{array}}{\vec{\Gamma} \xrightarrow{A:\, authCTLC\ \hat{c}^x} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]} \tag{16}$$

Note that $sender(\hat{c}^x) = owner(fund(\hat{c}^x))$. For authorizing a *CTLC*, all its secrets and the fund need to be available. To make sure that no contract can get executed without the consent of all included participants, both sender and receiver need to authorize them before it can proceed with enabling them in the next step.

$$[enableCTLC]\frac{\begin{array}{c}\Gamma_{ch} \in \vec{\Gamma}, \hat{c}^x \in \Gamma_{ch}.C_{adv}, c^x \notin \Gamma_{ch}.C_{en},\\ c^x := \{sc^x\}, \text{ for } sc^x \in \hat{c}^x, s = |\hat{c}^x|,\\ position(sc^x) = s,\\ Aut := \{(sender(\hat{c}^x), \hat{c}^x), (receiver(\hat{c}^x), \hat{c}^x)\},\\ Aut \subseteq \Gamma_{ch}.C_{aut}, fund(\hat{c}^x) \in \Gamma_{ch}.F_{av},\\ A = sender(c^x), C'_{en} := \Gamma_{ch}.C_{en} \cup \{c^x\}\\ C'_{aut} := \Gamma_{ch}.C_{aut} \backslash Aut,\ F'_{av} := \Gamma_{ch}.F_{av} \backslash \{fund(\hat{c}^x)\},\\ F'_{res} := \Gamma_{ch}.F_{res} \cup \{fund(\hat{c}^x)\},\\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t]\\ \Gamma'_{ch} := [S_{com}, S_{rev}, B, C_{adv}, C'_{aut}, C'_{en}, C_{cla}, F'_{av}, F'_{res}, t]\end{array}}{\vec{\Gamma} \xrightarrow{enableCTLC_{ch}\ c^x} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]} \tag{17}$$

In case authorizations have been given and the fund for a *CTLC* is available it can be enabled, which is a local action happening in one channel. With this the last of its subcontracts is made available in $\Gamma_{ch}.C_{en}$. When a *CTLC* has been enabled the remaining subcontracts can be enabled one by one by the sender.

$$[enableSubC]\frac{\begin{array}{c}\Gamma_{ch} \in \vec{\Gamma}, \hat{c}^x \in \Gamma_{ch}.C_{adv}, c^x \in \Gamma_{ch}.C_{en},\\ sc^x \in \hat{c}^x \backslash c^x, A = sender(c^x)\\ C'_{en} := \Gamma_{ch}.C_{en} \backslash \{c^x\} \cup \{c^x \cup \{sc^x\}\}\\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t],\\ \Gamma'_{ch} := [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C'_{en}, C_{cla}, F_{av}, F_{res}, t]\end{array}}{\vec{\Gamma} \xrightarrow{A:\, enableSubC\ sc^x} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]} \tag{18}$$

Since *CTLC*s are local objects, subcontracts are also local. Now that contracts and their subcontracts can be enabled the next step towards executing them is revealing their secrets.

$$[revealSecret]\frac{\begin{array}{c}\Gamma_{ch} \in \vec{\Gamma}, s^{id}_{\vec{a}} \in \Gamma_{ch}.S_{com} \backslash \Gamma_{ch}.S_{rev},\\ A = owner(s^{id}_{\vec{a}}) \in users(\Gamma_{ch}),\\ S'_{com} := \Gamma_{ch}.S_{com} \backslash \{s^{id}_{\vec{a}}\},\\ S'_{rev} := \Gamma_{ch}.S_{rev} \cup \{s^{id}_{\vec{a}}\},\\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t],\\ \Gamma'_{ch} := [S'_{com}, S'_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t]\end{array}}{\vec{\Gamma} \xrightarrow{A:\, revealSecret_{ch}\ s^{id}_{\vec{a}}} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]} \tag{19}$$

Where committing to secrets was a global operation, revealing them is local.

$$[shareSecret]\frac{\begin{array}{c}\Gamma_{ch}, \Gamma_{ch'} \in \vec{\Gamma}, s^{id}_{\vec{a}} \in \Gamma_{ch'}.S_{rev} \backslash \Gamma_{ch}.S_{rev},\\ A \in users(\Gamma_{ch}) \cap users(\Gamma_{ch'}),\\ S'_{rev} := \Gamma_{ch}.S_{rev} \cup \{s^{id}_{\vec{a}}\},\\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t],\\ \Gamma'_{ch} := [S_{com}, S'_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t]\end{array}}{\vec{\Gamma} \xrightarrow{A:\, shareSecret^{ch'}_{ch}\ s^{id}_{\vec{a}}} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]} \tag{20}$$

With 'shareSecret', a secret already known in $\Gamma_{ch'}$ gets shared with an environment Γ_{ch}. As participants in $\Gamma_{ch'}$ do not unlearn a secret it also stays in $\Gamma_{ch'}.S_{rev}$. Any user can execute this operation if they are part of Γ_{ch} and $\Gamma_{ch'}$.

$$[timeout]\frac{\begin{array}{c}\hat{c}^x \in \Gamma_{ch}.C_{adv}, c^x \in \Gamma_{ch}.C_{en},\\ |\hat{c}^x| > 1, sc^x \in \hat{c}^x,\\ \nexists \dot{sc}^x \in \hat{c}^x : position(\dot{sc}^x) < position(sc^x),\\ timelock(sc^x) \leq \Gamma_{ch}.t,\\ C'_{en} := \Gamma_{ch}.C_{en} \backslash \{c^x\} \cup \{c^x \backslash \{sc^x\}\},\\ C'_{adv} := \Gamma_{ch}.C_{adv} \backslash \{\hat{c}^x\} \cup \{\hat{c}^x \backslash \{sc^x\}\},\\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t],\\ \Gamma'_{ch} := [S_{com}, S_{rev}, B, C'_{adv}, C_{aut}, C'_{en}, C_{cla}, F_{av}, F_{res}, t]\end{array}}{\vec{\Gamma} \xrightarrow{timeout\ (c^x, sc^x)} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]} \tag{21}$$

With *timeout*, one of the subcontracts in a *CTLC* can get disabled after its timelock has run out. If only one subcontract is left, the whole contract can be refunded instead. These actions remove them from the advertised and enabled contracts and as *advCTLC*, *enableCTLC*, and *enableSubC* where local actions this also happens locally in one Γ_{ch}.

$$\frac{\begin{array}{c}\hat{c}^x \in \Gamma_{ch}.C_{adv}, c^x \in \Gamma_{ch}.C_{en}, |\hat{c}^x| = 1, sc^x \in \hat{c}^x, \\ \Gamma_{ch}.t \geq \mathit{timelock}(sc^x), \\ C'_{en} := \Gamma_{ch}.C_{en} \backslash \{c^x\}, C'_{adv} := \Gamma_{ch}.C_{adv} \backslash \{c^x\}, \\ F'_{av} := \Gamma_{ch}.F_{av} \cup \{\mathit{fund}(c^x)\}, F'_{res} := \Gamma_{ch}.F_{res} \backslash \{\mathit{fund}(c^x)\}, \\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t], \\ \Gamma'_{ch} := [S_{com}, S_{rev}, B, C'_{adv}, C_{aut}, C'_{en}, C_{cla}, F'_{av}, F'_{res}, t]\end{array}}{\vec{\Gamma} \xrightarrow{\mathit{refund}\, c^x} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]}\ [\mathit{refund}] \tag{22}$$

If the timelock for a subcontract has not been reached, its fund is reserved and all its secrets are known locally it can get claimed.

$$\frac{\begin{array}{c}\hat{c}^x \in \Gamma_{ch}.C_{adv}, sc^x \in c^x \in \Gamma_{ch}.C_{en}, \\ \exists i : \mathit{secret}_i(sc^x_\iota) \subseteq \Gamma_{ch}.S_{rev}, \mathit{fund}(c^x) \in \Gamma_{ch}.F_{res}, \\ \nexists \dot{sc}^x \in \hat{c}^x : \mathit{position}(\dot{sc}^x) < \mathit{position}(sc^x), \\ C'_{adv} := \Gamma_{ch}.C_{adv} \backslash \{c^x\}, \\ C'_{en} := \Gamma_{ch}.C_{en} \backslash \{c^x\}, C'_{cla} := \Gamma_{ch}.C_{cla} \cup \{c^x \cap \{sc^x\}\}, \\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t], \\ \Gamma'_{ch} := [S_{com}, S_{rev}, B, C'_{adv}, C_{aut}, C'_{en}, C'_{cla}, F_{av}, F_{res}, t]\end{array}}{\vec{\Gamma} \xrightarrow{\mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x))} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]}\ [\mathit{claim}] \tag{23}$$

Note that the *claim* action needs all secrets of one of the secret sets $\mathit{secret}_i(sc^x_\iota)$ to be revealed in one Γ_{ch}, predefined by the location of $\mathit{fund}(c^x)$. This means that it is possible to *claim* sc^x_ι based on the secrets from $\mathit{secret}_1(sc^x_\iota)$ **or** $\mathit{secret}_2(sc^x_\iota)$ **or** ... . Also, note that sc^x_ι can only be claimed when being the *top-level contract*, meaning that the contract below (and hence all contracts below) has been timed out before. One subtlety here is that also non-enabled subcontracts must have timed out (as long as the main CTLC has been enabled). This is achieved by checking for the appearance in the set $\hat{c}^x \in \Gamma_{ch}.C_{adv}$ of advertised subcontracts (belonging to the enabled contract sc^x_ι). The reason for this modeling is the static nature of CTLCs: Subcontracts are pre-defined spending options that can be dynamically enabled by the contract senders but whose execution follows a strict hierarchical order. To ensure that low-hierarchy spending options are made available as expected, it needs to be ensured that high-priority spending options cannot be enabled at a later point in time, messing with the execution order. For this reason, the spending whole spending option (independent of whether being enabled or not) can time out.

Finally, a claimed contract can be withdrawn. Here, in the second line of the rule, the owner of the fund belonging to c^x is set to the receiver of this contract. Before this action, the owner of this fund was the sender of c^x.

$$\frac{\begin{array}{c}\hat{c}^x \in \Gamma_{ch}.C_{adv}, sc^x \in c^x \in \Gamma_{ch}.C_{cla}, \\ \mathit{owner}(\mathit{fund}(c^x)) := \mathit{receiver}(c^x), \mathit{fund}(c^x) \in \Gamma_{ch}.F_{res}, \\ C'_{cla} := \Gamma_{ch}.C_{cla} \backslash \{c^x\}, \\ F'_{av} := \Gamma_{ch}.F_{av} \cup \{\mathit{fund}(c^x)\} \\ F'_{res} := \Gamma_{ch}.F_{res} \backslash \mathit{fund}(c^x) \\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t], \\ \Gamma'_{ch} := [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C'_{cla}, F'_{av}, F'_{res}, t]\end{array}}{\Gamma_{ch} \xrightarrow{\mathit{withdraw}(c^x, sc^x)} \vec{\Gamma}[\Gamma_{ch} \to \Gamma'_{ch}]}\ [\mathit{withdraw}] \tag{24}$$

$$\frac{\begin{array}{c}\vec{\Gamma}' := [\Gamma'_1, ..., \Gamma'_{|\vec{\Gamma}|}], \forall 1 \leq ch \leq |\vec{\Gamma}| \text{ set} \\ t' = t + \delta \\ \Gamma_{ch} = [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t], \\ \Gamma'_{ch} := [S_{com}, S_{rev}, B, C_{adv}, C_{aut}, C_{en}, C_{cla}, F_{av}, F_{res}, t']\end{array}}{\vec{\Gamma} \xrightarrow{elapse\, \delta} \vec{\Gamma}'}\ [\text{elapse } \delta] \tag{25}$$

Time can only be changed in all $\Gamma_{ch} \in \vec{\Gamma}$ at once an thus we write $\Gamma.t := \Gamma_1.t = ... = \Gamma_{|\vec{\Gamma}|}.t$.

Appendix F. Honest User Strategy

To model execution on consensus objects, we will adopt a symbolic execution model similar to the one presented in [28]. We will call sequences $\vec{\Gamma}_0 \xrightarrow{\alpha_0} \vec{\Gamma}_1 \xrightarrow{\alpha_1} \cdots$ runs where α_i are transition labels and $\vec{\Gamma}_0$ is an initial environment. We will refer to these transition labels as *moves* or *actions* in the following. Given a run

$$R := \vec{\Gamma}_0 \xrightarrow{\alpha_0} \cdots \xrightarrow{\alpha_{n-1}} \vec{\Gamma}_n$$

of length $n \in \mathbb{N}$ we set $\vec{\Gamma}_n =: \mathit{lastEnv}(R)$ as the last environment of R.

A participant strategy is a function Σ_B taking as input a run R and outputting a set of transition labels, indicating the actions that the user wants to schedule.

Definition F.1 (Participant strategies)**.** *The strategy of an (honest) user B is a function Σ_B, taking as input a run R. The output is a set of α-moves such that the following conditions hold:*

1) *Participant strategies can only output actions that are valid with respect to the semantics: $\forall \alpha \in \Sigma_B(R) : R \xrightarrow{\alpha}$;*
2) *Users can only schedule restricted actions if they are the ones to whom the action is restricted:*

$$\forall \alpha \in \Sigma_B(R) : \alpha = U : \alpha' \Rightarrow B = U;$$

3) *Participant strategies must be persistent, meaning that a participant strategy needs to keep scheduling an action as long as it is valid:*

$$\forall \alpha \in \Sigma_B(R) : R \xrightarrow{\alpha'} R' \xrightarrow{\alpha} \ \Rightarrow \alpha \in \Sigma_B(R \xrightarrow{\alpha'} R').$$

To model the power of miners in the execution of honest user actions in a blockchain environment, we define the adversary strategy to be a function that given a run and the outputs of the honest user strategies can produce the next action to extend the run. firefox on This models both, the attacker's capability to order honest user actions arbitrarily and the adversary's power to include own transactions based on the knowledge gathered from the scheduled actions of honest users. To give basic guarantees to the honest user, the attacker strategy is restricted to only be able to make time pass once all honest users either agree to do so or have no more actions scheduled. This ensures that honest users can always meet deadlines and that the protocol execution

cannot advance (in time) without their scheduled actions being taken into account.

Definition F.2 (Adversary strategy)**.** *Let Hon $= \{B_i, \dots, B_k\}$ be a set of honest users. An adversary strategy $\Sigma_{\mathcal{A}}$ (for Hon) is a function taking as input a run R and a list $\vec{\Lambda} = [\Lambda_1, \dots \Lambda_k]$ of sets of moves for each $B_i \in$ Hon. The output is a single adversary action α such that the following conditions hold:*

1) *The adversary strategy can only output actions that are valid with respect to the semantics: $\forall \alpha \in \Sigma_{\mathcal{A}}(R, \vec{\Lambda}) : R \xrightarrow{\alpha}$*
2) *Restricted actions of honest users can only be chosen by the adversary strategy if scheduled by the corresponding honest user strategy $\Sigma_{\mathcal{A}}(R, \vec{\Lambda}) = U : \alpha' \wedge U \in$ Hon $\Rightarrow \exists i : B_i = U \wedge U : \alpha' \in \Lambda_i$*
3) *The adversary can only output a time elapse action if all users agree to do so: $\Sigma_{\mathcal{A}}(R, \vec{\Lambda}) =$ elapse $\delta \Rightarrow \forall B_i \in$ Hon $: \Lambda_i = \emptyset \vee \exists \delta_i :$ elapse $\delta_i \in \Lambda_i \wedge \delta_i \geq \delta \geq \epsilon > 0$*

We assume an $\epsilon > 0$ as a constant minimum size for all elapse δ actions throughout the run.

Based on these notions, we can define when a run R is conformant with a given set of strategies:

Definition F.3 (Conformant runs)**.** *Let Hon $= \{B_1, \dots, B_k\}$ be a set of honest users and $\Sigma_{Hon} = \{\Sigma_{B_1}, \dots, \Sigma_{B_k}\}$ a corresponding set of user strategies. Let $\Sigma_{\mathcal{A}}$ be an adversary strategy (for Hon). We say that a run R* conforms to *$(\Sigma_{Hon}, \Sigma_{\mathcal{A}})$ (written $(\Sigma_{Hon}, \Sigma_{\mathcal{A}}) \vdash R$) if one of the following holds*

1) *$R = \vec{\Gamma}_0$ is an initial environment*
2) *$R = R' \xrightarrow{\alpha} \vec{\Gamma}$ where $(\Sigma_{Hon}, \Sigma_{\mathcal{A}}) \vdash R'$ and $\alpha = \Sigma_{\mathcal{A}}(R', [\Sigma_{B_1}(R'), \dots, \Sigma_{B_k}(R')])$*

We will also write $\Sigma_{Hon} \vdash R$ as shorthand for

$$\exists \Sigma_{\mathcal{A}} : (\Sigma_{Hon}, \Sigma_{\mathcal{A}}) \vdash R.$$

Note that for the sake of simplicity, as opposed to [28], we assume here strategies to be functions (instead of PPTIME algorithms). To achieve computational soundness results, one could require these functions to be PPTIME algorithms and give them access to a (user-specific) source of randomness. It will be evident to see that the honest user strategies presented in this work run in PPTIME.

Another major adaption with respect to the model from [28] is that we do not explicitly model the values of secrets. This is because our semantics does not model any computations on secret values which allows us to only refer to characterize the function of these secrets purely in terms of user access to these secrets. This simplification substantially simplifies the theoretical model because we are not required to explicitly strip secret values from runs and user actions in order to model that those are not accessible to users and the attacker.

Preparation Phase. This section establishes the formal connection between trees and *CTLC*s. We will denote $\vec{\Gamma} =$ *lastEnv*(R) in the following. We define $\mathbb{T}$ as a set of *tree specifications* consisting of elements $(id, \mathcal{T}, t_0, spec)$ where id is a unique identifier for this tree, which will be used to establish the connection to a batch. In the following, we assume a game tree $\mathcal{T}$ (Def. D.2) to be given. As defined in Equation (1), it can result from unfolding a digraph $\mathcal{D}$ with $\mathcal{T} :=$ *unfold*$(\mathcal{D}, A)$. With $t_0 \in \mathbb{R}^+$, the beginning of the Execution Phase is determined. All *timelock*s will be set after t_0, and for the honest user, the setup of contracts will be done before t_0, and executions only happen after t_0. The last element, $spec$ is defined as a function on all $e \mathrel{\widehat{\in}} \mathcal{T}$:

$$spec(e) := (fund(e), channel(e)) = (f_X^\zeta, ch)$$

Here, $fund(e)$ defines the fund as a token in $\Gamma_{channel(e)}$ for the subcontract that will resemble this edge. X is the sender of e and hence owner of f_X^ζ. The *specification* $spec(e)$ defines the preconditions for the initial environment for it to be able to resemble this edge.

Definition F.4. *A specification spec on $\mathcal{T}$ is valid if for all $e, e' \mathrel{\widehat{\in}} \mathcal{T}$ it holds*

$$spec(e) = spec(e') :\Leftrightarrow$$
$$sender(e) = sender(e') \wedge receiver(e) = receiver(e')$$

This condition ensures that edges in different locations in the tree resulting from one arc in $\mathcal{D}$ through the unfolding process feature the same output of $spec$. We call these *duplicated edges*, and in the following, the equality of their specifications will be used to identify them. In the following, $spec$ will be a valid specification.

We assign a unique secret

$$secret((A, B)_{\vec{a}}, id) =: s_{\vec{a}}^{id} \tag{26}$$

with $owner(s_{\vec{a}}^{id}) = B$ to any edge $(A, B)_{\vec{a}} \in \mathcal{T}$. For defining the *CTLC* contracts resulting from a given $\mathbb{T}$, we first look at a single $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ and group together duplicated edges on the same level:

$$edges_{dupl}(\mathcal{T}) := \bigcup_{e \mathrel{\widehat{\in}} \mathcal{T}} \{e' \mathrel{\widehat{\in}} \mathcal{T} \mid depth(e) = depth(e') \wedge sender(e) = sender(e') \wedge receiver(e) = receiver(e')\} \tag{27}$$

For a fixed $\Omega \in edges_{dupl}(\mathcal{T})$ we look at edges $e \in \Omega$, and for these, we set

$$h_{sec}(e, id) := \bigcup_{e' \in onPathToRoot(\mathcal{T}, e)} \{secret(e', id)\}. \tag{28}$$

For all $\Omega \in edges_{dupl}(\mathcal{T})$, we define secret sets:

$$secret - set(\Omega, id, \mathcal{T}) := \bigcup_{e \in \Omega} \{h_{sec}(e, id)\}$$

Note that if there is no additional edge on the same tree level with the same sender and receiver and disjoint path to the root, only the secret set of e itself is included. By construction, for any fixed Ω there exists $X, Y \in \mathcal{N}$, $j \in \mathbb{N}$

and f_X^ζ with:

$$\begin{aligned} \mathit{sender}(e) &= X && ,\forall e \in \Omega \\ \mathit{receiver}(e) &= Y && ,\forall e \in \Omega \\ \mathit{depth}(e) &= j && ,\forall e \in \Omega \\ \mathit{fund}(e) &= f_X^\zeta && ,\forall e \in \Omega \end{aligned}$$

This means that all edges from Ω are between the same parties, their *spec* features the same fund and they are located on the same tree level. For every Ω we demand f_X^ζ to be a fresh, unused fund with a unique identifier ζ. Based on the X, Y, j and f_X^ζ from above we map a set Ω to a subcontract sc^x with $x = (id, X, Y)$ and

$$\begin{aligned} &H(\Omega, (id, \mathcal{T}, t_0, spec)) := sc^x \\ &= [X, Y, f_X^\zeta, t_0 + j\Delta, secret - set(\Omega, id, \mathcal{T})]. \end{aligned}$$

For all $e \in \Omega$ we define

$$h(e, id) := sc^x. \tag{29}$$

Since $edges_{dupl}(\mathcal{T})$ is a partition of $\mathcal{T}$, this definition and also the one of h_{sec} can be extended to all $e \mathrel{\widehat{\in}} \mathcal{T}$. Later in this section (see (34)), we will see that an honest user only commits to batches that are built according to this construction. Hence, h gives us a surjective mapping from all edges in $\mathcal{T}$ to the sub-contracts in a batch. We will now define this batch:

$$\begin{aligned} &\mathit{treetoCTLCBadge}((id, \mathcal{T}, t_0, spec), \mathcal{S}) := \Psi^{id} = \\ &\bigcup_{e \mathrel{\widehat{\in}} \mathcal{T}} \{h(e', id) \mid e' \mathrel{\widehat{\in}} \mathcal{T} \wedge spec(e') = spec(e)\} \end{aligned} \tag{30}$$

By definition, this groups sub-contracts of duplicated edges together and gives a batch of *CTLC*s that represents $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$, hence the same id is used for Ψ^{id}. Further, we define the mapping

$$\begin{aligned} &\mathit{treeSettoCTLCBadges}(\mathbb{T}, \mathcal{S}) := \\ &\bigcup_{(id, \mathcal{T}, t_0, spec) \in \mathbb{T}} \mathit{treetoCTLCBadge}((id, \mathcal{T}, t_0, spec), \mathcal{S}), \end{aligned} \tag{31}$$

which outputs a set of batches, one for each tree.

For later use, we also define the following objects. Given a $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ and let $A, B \in \mathcal{N}$ be fixed. Then the number of edges (A, B) in $\mathcal{T}$ is given as

$$\begin{aligned} E^{\mathcal{T}}_{(A,B)} &:= \{(X, Y)_{\vec{a}} \mathrel{\widehat{\in}} \mathcal{T} \mid X = A \wedge Y = B\}, \\ s^{\mathcal{T}}_{(A,B)} &:= |E^{\mathcal{T}}_{(A,B)}|. \end{aligned} \tag{32}$$

For advertising and committing to batches we require the initial environment to be liquid with respect to a set of tree specifications $(id, \mathcal{T}, t_0, spec)$, defined as follows.

Definition F.5 (Liquid Environment)**.** *Let $\mathbb{T}$ be a set of tuples of the form $(id, \mathcal{T}, t_0, spec)$. We say that a environment $\vec{\Gamma}$ is liquid w.r.t. $\mathbb{T}$ if the following condition holds:*

$$\begin{aligned} &\forall (id, \mathcal{T}, t_0, spec) \in \mathbb{T} : \forall e \mathrel{\widehat{\in}} \mathcal{T} : \forall (f_X^\zeta, ch) : \\ &spec(e) = (f_X^\zeta, ch) \Rightarrow f_X^\zeta \in \Gamma_{ch}.F_{av} \end{aligned}$$

We also define when we consider a $\mathbb{T}$ to be well-formed.

Definition F.6 (Well-Formed Treeobject)**.** *Let $\mathbb{T}$ be a set of tuples of the form $(id, \mathcal{T}, t_0, spec)$. We say that $\mathbb{T}$ is well-formed if all spec functions are valid, all ids are unique, and the spec functions do not intersect, formalized with the following condition:*

$$\begin{aligned} &specInter(\mathbb{T}) :\Leftrightarrow \\ &\forall (id, \mathcal{T}, t_0, spec), (id', \mathcal{T}', t_0', spec') \in \mathbb{T} \ \forall e \mathrel{\widehat{\in}} \mathcal{T} \ \forall e' \mathrel{\widehat{\in}} \mathcal{T} : \\ &(\mathit{sender}(e), \mathit{receiver}(e), id) \neq (\mathit{sender}(e'), \mathit{receiver}(e'), id') \\ &\Rightarrow \forall (f_X^\zeta, ch), (f_{X'}^{\prime\zeta}, ch') : \\ &spec(e) = (f_X^\zeta, ch) \ \wedge \ spec(e') = (f_{X'}^{\prime\zeta}, ch') \Rightarrow X \neq X' \end{aligned}$$

Firstly, for every $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$, a Batch needs to be advertised. Anybody can advertise batches. Thus, it is in the honest strategy to advertise all not yet advertised batches. In the initial environment, only funds are given. Batches are only advertised at or before $t_0 - \mathit{depth}(\mathcal{T})\Delta$ with

$$\mathit{depth}(\mathcal{T}) := max\{\mathit{depth}(e) \mid e \mathrel{\widehat{\in}} \mathcal{T}\}, \tag{33}$$

because this ensures that the setup process can go through as desired. The environment also needs to be liquid w.r.t. the current $\mathbb{T}$. Additionally, honest users only accept well-formed $\mathbb{T}$. This ensures that the same fund cannot be used for two different contracts. After *advBatch*, B commits to the batch. We denote $\vec{\Gamma} = \mathit{lastEnv}(R)$. The following sub-strategy, where B is the currently operating party, formalizes this process for a given $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$:

$$\begin{aligned} &newBatch(\mathbb{T}, R) \\ &:= \begin{cases} 2 & \text{,if } \forall \Gamma_{ch} \in \vec{\Gamma} : \Psi^{id} \notin \Gamma_{ch}.B, \\ & \mathbb{T} \text{ well-formed, } \vec{\Gamma} \text{ liquid w.r.t. } \mathbb{T}, \\ & \exists (id, \mathcal{T}, t_0, spec) \in \mathbb{T} : \\ & \Psi^{id} = \mathit{treetoCTLCBadge}((id, \mathcal{T}, t_0, spec), \mathcal{S}), \\ & \vec{\Gamma}.t \leq t_0 - \mathit{depth}(\mathcal{T})\Delta \\ 1 & \text{,if } \exists \Gamma_{ch} \in \vec{\Gamma} : \Psi^{id} \in \Gamma_{ch}.B, \\ & \mathbb{T} \text{ well-formed, } \vec{\Gamma} \text{ liquid w.r.t. } \mathbb{T}, \\ & S_B(\Psi^{id}) \neq \emptyset, \Gamma_{ch}.t < t_0, \\ & S_B(\Psi^{id}) \not\subseteq \Gamma_{ch}.S_{com}, \exists (id, \mathcal{T}, t_0, spec) \in \mathbb{T} : \\ & \Psi^{id} = \mathit{treetoCTLCBadge}((id, \mathcal{T}, t_0, spec), \mathcal{S}) \\ 0 & \text{,else.} \end{cases} \end{aligned}$$

$$\begin{aligned} &\widehat{\Sigma}^{\mathbb{T}}_B(R) \\ &:= \begin{cases} \{\mathit{advBatch}\, \Psi^{id}\} & \text{,if } newBatch(\mathbb{T}, R) = 2, \\ \{B : \mathit{commitBatch}\, \Psi^{id}\} & \text{,if } newBatch(\mathbb{T}, R) = 1, \\ \emptyset & \text{,if } newBatch(\mathbb{T}, R) = 0. \end{cases} \end{aligned} \tag{34}$$

In the first case, a corresponding Ψ^{id} for $(id, \mathcal{T}, t_0, spec)$ gets advertised. If the batch is well-formed, meaning it is aligned with the said mapping, B also commits to it. If none of these options is available, B schedules nothing.

Enabling Phase. Given $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ and $e \mathrel{\widehat{\in}} \mathcal{T}$. We first define 2 helper functions that evaluate whether all ingoing edges have been enabled (1) and, if so, if an entirely new *CTLC* should be advertised, an available *CTLC* should be authorized or enabled or just an additional sub-contract

(2) enabled. The condition (1) is checked in the function

$$ingoing(e,R) \tag{35}$$
$$:= \begin{cases} 1 & \text{,if } e \,\widehat{\in}\, \mathcal{T}, \exists X, \vec{a} : e = (X,B)_{\vec{a}} \textbf{ or} \\ & \forall e' \,\widehat{\in}\, \mathcal{T} \text{ with } \textit{onPath}(e,e'), \textit{depth}(e') = \textit{depth}(e)+1 \\ & \exists \widetilde{c^x} \in \Gamma_{channel(e')}.C_{en} : h(e',id) \in \widetilde{c^x} \\ & \wedge \vec{\Gamma}.t < t_0, \\ & \exists \Psi^{id} := \textit{treetoCTLCBadge}((id,\mathcal{T},t_0,spec),\mathcal{S}) \in \vec{\Gamma}.B: \\ & h(e,id) \in c^x \in \Psi^{id} \wedge h(e',id) \in \widetilde{c^x} \in \Psi^{id} \\ & \wedge \nexists ch : h(e,id) \in c^x \in \Gamma_{ch}.C_{en} \\ 0 & \text{, else.} \end{cases}$$

The function $ingoing$ checks for a given edge e and current run R whether all edges below it have been enabled for this appearance of B in the tree $\mathcal{T}$. Additionally, it is checked that all these edges correspond to a tree.

This information gets used in the mapping *newC*, which checks for condition (2) from above. For this we define

$$Aut(\hat{c}^x) := \{(\textit{sender}(\hat{c}^x), \hat{c}^x), (\textit{receiver}(\hat{c}^x, \hat{c}^x))\},$$

$$newC(e,R) \tag{36}$$
$$:= \begin{cases} 4 & \text{,if } ingoing(e,R) = 1 \wedge \exists X, \vec{a} : e = (B,X)_{\vec{a}} \,\widehat{\in}\, \mathcal{T} \\ & \wedge \exists h(e,id) \in \hat{c}^x \in \vec{\Gamma}.C_{adv} \\ & \wedge c^x \in \vec{\Gamma}.C_{en} \wedge h(e,id) \notin c^x, \\ 3 & \text{,if } ingoing(e,R) = 1 \wedge \exists X, \vec{a} : e = (B,X)_{\vec{a}} \,\widehat{\in}\, \mathcal{T} \\ & \wedge \exists h(e,id) \in \hat{c}^x \in \vec{\Gamma}.C_{adv} \\ & \wedge c^x \notin \vec{\Gamma}.C_{en} \wedge Aut(\hat{c}^x) \subseteq \vec{\Gamma}.C_{aut}, \\ 2 & \text{,if } ingoing(e,R) = 1 \\ & \wedge \exists h(e,id) \in \hat{c}^x \in \vec{\Gamma}.C_{adv} \\ & \wedge c^x \notin \vec{\Gamma}.C_{en} \wedge \textit{fund}(\hat{c}^x) \in \vec{\Gamma}.F_{av} \\ & \wedge if\ B = \textit{sender}(\hat{c}^x) : (\textit{receiver}(\hat{c}^x), \hat{c}^x) \in \vec{\Gamma}.C_{aut} \\ 1 & \text{,if } ingoing(e,R) = 1 \wedge \exists X, \vec{a} : e = (B,X)_{\vec{a}} \,\widehat{\in}\, \mathcal{T} \\ & \wedge \nexists h(e,id) \in \hat{c}^x \in \vec{\Gamma}.C_{adv} \\ & \wedge \nexists h(e,id) \in c^x \in \vec{\Gamma}.C_{en} \\ & \wedge \forall sc^x \in \hat{c}^x : \textit{secret}(sc^x) \subseteq \vec{\Gamma}.S_{com}, \\ 0 & \text{, } ingoing(e,R) = 0. \end{cases}$$

In the first case the *CTLC* of the subcontract for the given edge e has been advertised and enabled, but $h(e,id)$ has not been enabled yet. Therefore, B will enable it in the following substrategy. In the second case, c^x has not been enabled but advertised and authorized by the sender and receiver. Thus, B will enable it. The third case includes c^x not to be enabled but advertised. Additionally no other contract $\hat{c}^x$ with the same identifier x should have been authorized by B before. If this true B will authorize it. In the fourth case, the *CTLC* has neither been enabled nor advertised, so B will advertise it under given well-formedness conditions. In the fifth and last case, $ingoing(e,R) = 0$, and no action of B is required.

Then, the substrategy for these actions is defined as

$$\overline{\Sigma}^e_B(R)$$
$$:= \begin{cases} \left\{B : \textit{enableSubC}\, h(e,id)\right\} & \text{,if } newC(e,R) = 4, \vec{\Gamma}.t < t_0, \\ \left\{\textit{enableCTLC}_{ch}\, c^x\right\} & \text{,if } newC(e,R) = 3, h(e,id) \in c^x, \\ & ch = channel(e), \vec{\Gamma}.t < t_0, \\ \left\{B : \textit{authCTLC}\, \hat{c}^x\right\} & \text{,if } newC(e,R) = 2, h(e,id) \in \hat{c}^x, \\ & \nexists \hat{c}^x : (B : \textit{authCTLC}\, \hat{c}^x) \in \textit{actions}(R), \\ & \vec{\Gamma}.t < t_0, \\ \left\{\textit{advCTLC}_{ch}\, \hat{c}^x\right\} & \text{,if } newC(e,R) = 1, h(e,id) \in \hat{c}^x, \\ & ch = channel(e), \vec{\Gamma}.t < t_0, \\ \emptyset & \text{,if } newC(e,R) = 0. \end{cases} \tag{37}$$

Execution Phase. Before discussing the decision and execution of contracts, we want to ensure that contracts run into timeout or refund when possible. As any party can timeout or refund any contract as soon as their timeout has been reached, we make it part of the honest user strategy. For this, let $e \,\widehat{\in}\, \mathcal{T}$ be given and we define

$$\widetilde{\Sigma}^e_B(R) \tag{38}$$
$$:= \begin{cases} \left\{\textit{timeout}(c^x, h(e,id))\right\} & \text{,if } \exists c^x \in \Gamma_{channel(e)}.C_{en} : \\ & B \in \textit{users}(c^x) \wedge |c^x| > 1 \\ & \wedge sc^x_j = h(e,id) \in c^x \\ & \wedge \textit{timelock}(h(e,id)) \leq \vec{\Gamma}.t \\ & \wedge sc^x_{j-1} \notin c^x \\ \left\{\textit{refund}\, c^x\right\} & \text{,if } \exists c^x \in \Gamma_{channel(e)}.C_{en} : \\ & B \in \textit{users}(c^x) \wedge |c^x| = 1 \\ & \wedge \exists h(e,id) \in c^x : \\ & \textit{timelock}(h(e,id)) \leq \vec{\Gamma}.t \\ & \wedge \textit{refund}\,(c^x) \notin \textit{actions}(R) \\ \emptyset & \text{,else.} \end{cases}$$

In the first case, the subcontract corresponding to the given edge has run into timeout, but there is still another subcontract in the *CTLC* with a larger timeout. In the second case, the current sub-contract is the last one in the *CTLC* so the whole *CTLC* gets refunded.

Given an edge $e \,\widehat{\in}\, \mathcal{T}$. For the corresponding subcontract to be claimable 3 things need to be given:

- It needs to be enabled, meaning

$$\begin{aligned} enabled(e,\Gamma) :\Leftrightarrow \exists c^x \in \Gamma_{channel(e)}.C_{en} : \big(& h(e,id) \in c^x \\ & \wedge \nexists sc^x \in \hat{c}^x \in \Gamma_{channel(e)}.C_{adv} : \\ & \textit{timelock}(sc^x) < \textit{timelock}(h(e,id))\big). \end{aligned}$$

- The current time should be before its timeout, meaning

$$t_0 \leq \vec{\Gamma}.t \leq \textit{timelock}(h(e,id)).$$

- Secrets of other people for this contract should be revealed in the environment of the contract already s.t. only the ones of B are missing. We denote this condition as

$$\begin{aligned} secretsAv(e) :\Leftrightarrow & \exists i\, \forall s^{id}_{\vec{a}} \in \textit{secret}_i(h(e,id)) \\ & \text{with } s^{id}_{\vec{a}} \notin \Gamma_{ch}.S_{rev} : s^{id}_{\vec{a}} = \textit{secret}(e,id). \end{aligned}$$

Additionally, by construction, we want B only to execute ingoing edges, and this only in case an outgoing edge has been executed before, which is formalized with:

$$\begin{aligned} isIngoing(e_{in}, R) :\Leftrightarrow & \exists Y \in \mathcal{N}, \vec{a} \in \mathcal{T} : e_{in} = (Y, B)_{\vec{a}} \\ & \wedge \text{ if } \mathit{depth}(e_{in}) > 1 : \\ & \forall e_{out} \in \vec{a} \text{ with } \mathit{depth}(e_{out}) = \mathit{depth}(e_{in}) - 1 \\ & \exists c^x : \mathit{claim}(c^x, h(e_{out}, id), h_{sec}(e_{out}, id)) \\ & \quad \in \mathit{actions}(R) \\ & \wedge \text{ if } \mathit{depth}(e_{in}) = 1 : \\ & \forall e'_{in} \in \mathcal{T} \text{ with } \mathit{depth}(e'_{in}) = 1 : \\ & \exists c^x, \mathit{ch} : h(e'_{in}, id) \in c^x \in \Gamma_{ch}.C_{en} \\ & \quad \wedge \mathit{ch} = \mathit{channel}(e'_{in}) \\ & \quad \vee \mathit{claim}(c^x, h(e'_{in}, id), h_{sec}(e'_{in}, id)) \\ & \quad \in \mathit{actions}(R) \end{aligned} \tag{39}$$

Note that e_{out} could also be specified using $\mathit{onPath}(.,.)$ or $\mathit{onPathToRoot}(.)$ but by construction of $\mathcal{T}$ they would specify exactly $\vec{a}$ again. The following helper function validates the conditions from above

$$chContract(e_{in}, R) \tag{40}$$
$$:= \begin{cases} 3 & \text{, if } \exists h(e_{in}, id) \in c^x \in \Gamma_{channel(e_{in})}.C_{dec} \\ 2 & \text{, if } enabled(e_{in}) \wedge secretsAv(e_{in}) \wedge isIngoing(e_{in}, R) \\ & \wedge t_0 \leq \vec{\Gamma}.t < \mathit{timelock}(h(e_{in}, id)) \\ 1 & \text{, if } enabled(e_{in}) \wedge isIngoing(e_{in}, R) \\ & \wedge \exists s^{id}_{\vec{a}} \in \mathit{secret}(h(e_{in}, id)) \backslash \Gamma_{channel(e_{in})}.S_{rev} \\ & \quad \exists \mathit{ch}' : s^{id}_{\vec{a}} \in \Gamma_{\mathit{ch}'}.S_{rev} \wedge B \in users(\Gamma_{\mathit{ch}'}) \\ 0 & \text{, else.} \end{cases}$$

In the first case, there is a decided/claimed contract with a subcontract resembling a given edge. It is part of the honest user strategy to execute any contract that is decided upon. In the second case, the contract the subcontract for e_{in} belongs to has been enabled, the remaining secrets are owned by B, e_{in} is an ingoing edge, all previous sub-contracts of the same *CTLC* have timed out, and the timeout has not run out yet. In the third case, there is a secret a^A_ι in another environment $\Gamma_{ch'}$ different from Γ_{ch} which has been revealed there but not in Γ_{ch} and belongs to the subcontract coming from e_{in}. Additionally, B is part of both environments.

To avoid B revealing secrets for two duplicated edges on the same level we define a function that looks up whether B scheduled such an action before. From the partition $edges_{dupl}(\mathcal{T})$ (see 27) we notice for any given $e \widehat{\in} \mathcal{T}$

$$\exists! \Omega \in edges_{dupl}(\mathcal{T}) : e \in \Omega.$$

Based on this specific Ω, we define the function

$$\begin{aligned} no - dupl(e, R) :\Leftrightarrow & \nexists e' \widehat{\in} \mathcal{T} : e' \neq e \\ & \wedge \mathit{sender}(e') = \mathit{sender}(e) \\ & \wedge \mathit{receiver}(e') = \mathit{receiver}(e) \\ & \wedge \mathit{secret}(e', id) \in \vec{\Gamma}.S_{rev}. \end{aligned}$$

This completes the set of functions and conditions needed for the definition of the next sub-strategy:

$$\Sigma^e_B(R)$$
$$:= \begin{cases} \left\{ \mathit{withdraw}(c^x, sc^x) \right\} & \text{,if } chContract(e, R) = 3, \\ & sc^x = h(e, id) \in c^x \\ \left\{ \mathit{claim}(c^x, sc^x, sec) \right\} & \text{,if } chContract(e, R) = 2, h(e, id) \in c^x, \\ & \exists i : \mathit{secret}_i(h(e, id)) \subseteq \Gamma_{channel(e)}.S_{rev}, \\ & sec := \mathit{secret}_i(h(e, id)) \\ \left\{ B : \mathit{revealSecret}_{ch}\, s^{id}_{\vec{a}} \right\} & \text{,if } chContract(e, R) = 2, \\ & h(e, id) \in c^x \in \vec{\Gamma}.C_{en}, \\ & \mathit{ch} := channel(e), s^{id}_{\vec{a}} = \mathit{secret}(e, id), \\ & \mathit{secret}_i(h(e, id)) \nsubseteq \Gamma_{ch}.S_{rev}, \\ & s^{id}_{\vec{a}} \in \mathit{secret}_i(h(e, id)) \backslash \Gamma_{ch}.S_{rev}, \\ & no - dupl(e, R) \\ \left\{ B : \mathit{shareSecret}^{ch'}_{ch}\, s^{id}_{\vec{a}} \right\} & \text{,if } chContract(e, R) = 1, h(e, id) \in c^x, \\ & \mathit{ch} := channel(e), \\ \emptyset & \text{,if } chContract(e, R) = 0 \end{cases} \tag{41}$$

In the first case, the *CTLC* has already met all conditions for execution and, therefore, has been decided by moving it to $\Gamma.C_{cla}$. It can get executed as long as its timeout has not run out yet. In the second case, all conditions are met, and B executes *claim*. In the third case, one secret of B needs to be revealed before the contract can be decided. With $\mathit{secret}_i(h(e, id))$, we mean the same secret set that was found in $secretsAv(e_{in})$, i.e. the same i. If this is true for more than one i it is chosen arbitrarily from the ones fulfilling the condition. The $no - dupl(e, R)$ condition implies that for two duplicated edges on the same level, only for one of them, the revealing of its edges gets scheduled by B. With this it is also ensured that only one of them can get claimed. By Definition (31), this situation cannot occur for $\mathit{shareSecret}^{ch'}_{ch}\, s^{id}_{\vec{a}}$. In the fourth case, a secret needed for claiming this contract has been revealed in another environment B is part of thus, B shares the secret with the environment of the contract $h(e, id) \in c^x$. If none of these 4 cases apply, B cannot act on this contract and schedules no action.

Combining Sub-Strategies. All previously defined sub-strategies are dependent on an edge from a specific tree $\mathcal{T}$, except $\widehat{\Sigma}^{\mathbb{T}}_B(R)$ which is only $\mathbb{T}$-dependent. Hence we first unite over all $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ and then over all edges e from a given $\mathcal{T}$:

$$\begin{aligned} & \Sigma^{\mathbb{T}, temp}_B(R) \\ & := \widehat{\Sigma}^{\mathbb{T}}_B(R) \bigcup_{(id, \mathcal{T}, t_0, spec) \in \mathbb{T}} \bigcup_{e \in \mathcal{T}} \overline{\Sigma}^e_B(R) \cup \widetilde{\Sigma}^e_B(R) \cup \Sigma^e_B(R) \end{aligned} \tag{42}$$

If $\Sigma^{\mathbb{T}, temp}_B(R)$ outputs no action, we want to wait and elapse time which is defined with

$$\Sigma^{\mathbb{T}, local}_B(R) := \begin{cases} \left\{ \mathit{elapse}\, \delta \right\} & \text{,if } \Sigma^{\mathbb{T}, temp}_B(R) = \emptyset \\ \Sigma^{\mathbb{T}, temp}_B(R) & \text{,else.} \end{cases} \tag{43}$$

Here δ is defined as

$$\begin{aligned}\delta_{t_0} &:= t_0 + j\Delta - \vec{\Gamma}.t \\ &\text{with } j = min\{j \in \mathbb{Z} \mid t_0 + j\Delta - \vec{\Gamma}.t > 0\} \\ \delta &:= min\{\delta_{t_0} \mid (id, \mathcal{T}, t_0, spec) \in \mathbb{T}\}\end{aligned}$$

which is exactly the time until the next timestamp at which new sub-contracts potentially become available for being enabled or for execution. Note that allowing j to come from the Integers $\mathbb{Z}$ implies that both the Enabling Phase as well as the Execution Phase are covered. Since the honest user should not change his mind, we want to stick to an output as long as it has not been executed and is still a valid extension of the current run. To formalize this notion, we set

$$\textit{actions}(R) := \{\alpha \mid R = R_1 \xrightarrow{\alpha} R_2\}, \tag{44}$$

$$\textit{s-actions}(R) := \{\alpha \mid \exists \vec{\Gamma} : R \xrightarrow{\alpha} \vec{\Gamma}\}. \tag{45}$$

By including this addition, we recursively define the *honest user strategy for an honest user* B:

$$\Sigma_B^{\mathbb{T}}(R) := \begin{cases} \{\alpha\} & \text{,if } \exists R_1, R_2 : R = R_1 \to R_2 \\ & \exists \alpha \in \Sigma_B^{\mathbb{T}}(R_1) : \\ & \alpha \notin \textit{actions}(R) \\ & \wedge \alpha \in \textit{s-actions}(R) \\ \Sigma_B^{\mathbb{T},local}(R) & \text{,else.} \end{cases} \tag{46}$$

We call the first case the consistency condition of the honest user strategy.

Appendix G. Progression of Time

This section formalizes that in a run including an honest participant time always progresses after a finite number of actions and thus a *final run*, which will be formally defined here, will always be reached at some point.

Definition G.1. *Given a run* $R = R_1 \xrightarrow{\alpha} \vec{\Gamma}$ *we define*

$$\textit{lastEnv}(R) := \vec{\Gamma}.$$

In the following, we assume a run R of arbitrary length to be given and a set of honest users $\textit{Hon} = \{B_1, \ldots, B_k\}$ with strategies $\Sigma_{\textit{Hon}} = \{\Sigma_{B_1}, \ldots, \Sigma_{B_k}\}$ to participate. We recap from Definition F.2 that an adversary can only schedule an *elapse* δ action if all users agree.

Theorem G.2. *Let* $R' := R \xrightarrow{\alpha_0} \vec{\Gamma}_1 \xrightarrow{\alpha_1} \vec{\Gamma}_2 \xrightarrow{\alpha_2} \ldots$ *be an extension of* R *with* $\Sigma_{\textit{Hon}} \vdash R'$. *Then there exists* $\delta \in \mathbb{R}^{>0}$ *s.t.*

$$\exists n \in \mathbb{N} : \alpha_n = \textit{elapse}\,\delta \vee |R'| - |R| \leq n.$$

Proof: As noted above, an adversary can only schedule a finite number of actions between honest user actions. All *CTLC* actions, defined in Appendix E, are relative to a batch Ψ^{id}. Let $B \in \textit{Hon}$. From the honest user strategy, especially (34) and (36), we know that all honest users B only schedule actions regarding a Ψ^{id} with

$$\Psi^{id} = \textit{treetoCTLCBadge}((id, \mathcal{T}, t_0, spec), \mathcal{S})$$

for some $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ and secret set $\mathcal{S}$. We assume $\mathbb{T}$ to be finite, i.e., the number of game trees executed simultaneously is finite. Also, we assume every tree only to include finitely many walks, which again are assumed to be of finite length, i.e.

$$\forall \vec{a} \in \mathcal{T}\, \exists m \in \mathbb{N} : \vec{a} = [a_{m-1}, a_{m-2}, ..., a_0].$$

Therefore, the total number of edges in $\mathbb{T}$, given as the size of

$$\bigcup_{(id,\mathcal{T},t_0,spec)\in\mathbb{T}} \{e \,\hat{\in}\, \mathcal{T}\},$$

is finite. Every such edge gets mapped to a subcontract $sc^x = h(e, id)$, and based on the honest user strategy, only finitely many actions are implied by this.

Assume towards contradiction $|R'| - |R| > n$ and

$$\nexists n \in \mathbb{N} : \alpha_n = \textit{elapse}\,\delta.$$

We show that any honest user B_i only schedules finitely many actions before nothing but *elapse* δ_i will be scheduled, regardless of the actions of other users. Combining this with the fact that an adversary can only schedule a finite number of actions between honest user actions and the assumption that $\mathbb{T}$ is finite, we arrive at the desired contradiction. We proceed with going through the sub-strategies of the honest user strategy for a given $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ one by one.

Let an honest user B be given. For the preparation phase we have $\widehat{\Sigma}_B^{\mathbb{T}}(R)$ (see (34)) which depends on $newBatch(\mathbb{T}, R)$. In case $newBatch(\mathbb{T}, R) = 2$ all preconditions for advertising a new batch are fulfilled. Since it is checked that a batch has not been advertised before, this can only be the case as often as the cardinality of $\mathbb{T}$ allows. In case $newBatch(\mathbb{T}, R) = 1$, party B will commit to a given Batch. With $S_B(\Psi^{id}) \not\subseteq \Gamma_{ch}.S_{com}$, it is checked that this only happens once per batch. In case $newBatch(\mathbb{T}, R) = 0$ an empty set will be outputted, which, in case all other sub-strategies also output $\emptyset$, will be turned into an *elapse* δ_i action (see (46)).

For the enabling phase we have $\overline{\Sigma}_B^e(R)$. In case

$$\overline{\Sigma}_B^e(R) = \left\{\textit{advCTLC}_{ch}\,\hat{c}^x\right\}$$

$newC(e, R) = 1$ holds, which includes the condition $\nexists h(e, id) \in \hat{c}^x \in \Gamma.C_{adv}$. Hence, the corresponding contract for the sub-contract belonging to e has not been advertised, which implies that this action can only happen once. Assume $\left\{\textit{advCTLC}_{ch}\,\hat{c}^x\right\}$ happens twice in a run. Then, it needs to be removed from $\vec{\Gamma}.C_{adv}$ in between. This can only be done with a *refund* c^x action, which by construction of $h(e, id)$ in Equation (29) can only happen after t_0. This leads to a contradiction as it is a condition of

$$\overline{\Sigma}_B^e(R) = \left\{\textit{advCTLC}_{ch}\,\hat{c}^x\right\}$$

that $\vec{\Gamma}.t < t_0$. Analogously, the same is true for

$$\overline{\Sigma}^e_B(R) = \left\{ B : \mathit{authCTLC}\, c^x \right\}$$

as it is checked for $(B, id) \notin \vec{\Gamma}.C_{aut}$. This also holds for

$$\overline{\Sigma}^e_B(R) = \left\{ \mathit{enableCTLC}_{ch}\, c^x \right\}$$

because of the $c^x \notin \vec{\Gamma}.C_{en}$ condition, and for

$$\overline{\Sigma}^e_B(R) = \left\{ B : \mathit{enableSubC}\, h(e, id) \right\}$$

with the

$$\exists h(e, id) \in \hat{c}^x \in \vec{\Gamma}.C_{adv} \wedge c^x \in \vec{\Gamma}.C_{en} \wedge h(e, id) \notin c^x$$

condition.

The honest user strategy for the execution phase consists of two sub-strategies, $\widetilde{\Sigma}^e_B(R)$ and $\Sigma^e_B(R)$. In $\widetilde{\Sigma}^e_B(R)$ the actions *timeout* and *refund* are handled, and for both of them it is part of the conditions that they have not been scheduled before. In $\Sigma^e_B(R)$ we have

$$\Sigma^e_B(R) = \left\{ B : \mathit{revealSecret}_{ch}\, s^{id}_{\vec{a}} \right\}$$

conditionally linked to a sub-contract for which this secret is used. As noted previously, sub-contracts can only be enabled once. Therefore, this action can be scheduled only once per secret. Analogously

$$\Sigma^e_B(R) = \left\{ \mathit{shareSecret}^{ch'}_{ch}\ s^{id}_{\vec{a}} \right\}$$

cannot be done more than once per secret, as it is linked to a sub-contract and its channel. For

$$\Sigma^e_B(R) = \left\{ \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x_\iota)) \right\}$$

it is guaranteed that $t_0 \leq \vec{\Gamma}.t$, and with $isIngoing(e, R)$ that B is the receiver of the contract. Hence, the authorization of B is needed for this contract to be enabled (see (17)). Part of the condition for B to authorize is $\vec{\Gamma}.t < t_0$ according to the honest user strategy (see (37)). Therefore, c^x cannot be enabled again after it was claimed by B. For

$$\Sigma^e_B(R) = \left\{ \mathit{withdraw}(c^x, sc^x) \right\}$$

it is guaranteed in $chContract(e_{in}, R)$ that

$$h(e, id) \in c^x \in \Gamma_{channel(e)}.C_{dec}$$

holds. The action *claim* is the only one that can move contracts into $\Gamma_{channel(e)}.C_{dec}$ and *withdraw* removes them from this set. Hence, *withdraw* cannot be executed more often than *claim* in a given run. As noted previously, *claim* can only happen once per contract, which is, therefore, also the case for *withdraw*

In a setting with multiple $B_i \in \mathit{Hon}$, the above reasoning applies to all B_i. Hence all $\Sigma^{\mathbb{T}}_{B_i}$ output $\{\mathit{elapse}\, \delta_i\}$ after n-many steps for some δ_i. By Definition (see (*F*.2)), an *elapse* δ action with $\delta_i \geq \delta$ is then appended to the run by the adversary.

Corollary G.3. *Let $R' := R \xrightarrow{\alpha_0} \vec{\Gamma}_1 \xrightarrow{\alpha_1} \vec{\Gamma}_2 \xrightarrow{\alpha_2} ...$ be an extension of R with $\Sigma_{Hon} \vdash R'$. Given an arbitrary $t' \in \mathbb{R}^{>0}$ with $t' > \vec{\Gamma}.t$ then there exists an $n \in \mathbb{N}$ s.t. $\vec{\Gamma}_{n+1}.t > t'$ or $|R'| - |R| \leq n$. In other words, extending a run can reach any point in time.*

Proof: The Definition *F*.2 of the adversary strategy sets a minimum size $\epsilon > 0$ for δ. W.l.o.g. assume $\delta = \epsilon$ for all *elapse* δ actions in R'. Thus

$$\left\lceil \frac{t' - \vec{\Gamma}.t}{\epsilon} \right\rceil$$

many *elapse* δ actions are needed to reach t'. From Theorem G.2 we know that after a finite number of steps an *elapse* δ gets appended to the run and thus time t' is reached after finitely many steps.

Definition G.4. *A run R with $\vec{\Gamma} = \mathit{lastEnv}(R)$ is considered final if*

$$\forall (id, \mathcal{T}, t_0, spec) \in \mathbb{T} : \forall e \in \mathcal{T} : \\ \mathit{timelock}(h(e, id)) < \vec{\Gamma}.t.$$

Corollary G.5. *For any run R there exists an $n \in \mathbb{N}$ s.t. for all extensions $R' := R \xrightarrow{\alpha_0} \vec{\Gamma}_1 \xrightarrow{\alpha_1} \vec{\Gamma}_2 \xrightarrow{\alpha_2} ...$, both of finite and infinite length, it holds:*

$$R' \text{ is final} \vee |R'| - |R| \leq n$$

This means that every run will become final at some point.

Proof: As it is explained in the proof of Theorem G.2 the total number of edges in $\mathbb{T}$ is finite. Thus the maximum

$$max\{\mathit{timelock}(h(e, id)) \mid e \in \mathcal{T}, (id, \mathcal{T}, t_0, spec) \in \mathbb{T}\}$$

exists. By setting this value as t' in Corollary G.3 the statement follows from it.

Appendix H. *CTLC* Properties

H.1. Strategy-independent properties

We first define properties that are independent of the concrete, honest user strategies applied.

Lemma H.1. *Let R be a CTLC run with $\vec{\Gamma} = \mathit{lastEnv}(R)$. It holds*

$$\forall \Gamma_{ch} \in \vec{\Gamma} : c^x \in \Gamma_{ch}.C_{en} \Rightarrow |c^x| \geq 1.$$

Proof: This statement directly follows from the inference rules *enableCTLC* (17) and *enableSubC* (18) by induction.

Lemma H.2. *Let R be a CTLC run with $\vec{\Gamma} = \mathit{lastEnv}(R)$. It holds*

$$\forall \Gamma_{ch} \in \vec{\Gamma} : c^x \in \Gamma_{ch}.C_{cla} \Rightarrow \exists!\, sc^x : sc^x \in c^x.$$

Proof: This statement directly follows from the inference rules *claim* (23) and *withdraw* (24) by induction.

Lemma H.3. *Let R be a CTLC run with $\vec{\Gamma} = \mathit{lastEnv}(R)$. It holds*

$$\forall \Gamma_{ch} \in \vec{\Gamma} : c^x \in \Gamma_{ch}.C_{en} \Rightarrow \hat{c}^x \in \Gamma_{ch}.C_{adv} \wedge c^x \sqsubseteq \hat{c}^x.$$

Proof: This statement directly follows from the inference rules *enableCTLC* (17) and *enableSubC* (18) by induction.

Lemma H.4. *Let $Hon = \{B_1, \ldots, B_k\}$ be a set of honest users and $\Sigma_{Hon} = \{\Sigma_{B_1}, \ldots, \Sigma_{B_k}\}$ a corresponding set of user strategies. Let $\Sigma_{\mathcal{A}}$ be an adversary strategy (for Hon) and let R be a run such that $(\Sigma_{Hon}, \Sigma_{\mathcal{A}}) \vdash R$. Further, let $\vec{\Gamma} = \mathit{lastEnv}(R)$. Then it holds*

$$\begin{aligned}&\forall \Gamma_{ch} \in \vec{\Gamma} : c^x \in \Gamma_{ch}.C_{en}\\ &\quad \Rightarrow (\mathit{sender}(\hat{c}^x) : \mathit{authCTLC}\, \hat{c}^x) \in \mathit{actions}(R)\\ &\quad \wedge (\mathit{receiver}(\hat{c}^x) : \mathit{authCTLC}\, \hat{c}^x) \in \mathit{actions}(R).\end{aligned}$$

See Equation (44) *for the definition of $\mathit{actions}(R)$.*

Proof: This statement directly follows from the inference rules *enableCTLC* (17) and *authCTLC* (16) by induction.

Lemma H.5. *Let $Hon = \{B_1, \ldots, B_k\}$ be a set of honest users and $\Sigma_{Hon} = \{\Sigma_{B_1}, \ldots, \Sigma_{B_k}\}$ a corresponding set of user strategies. Let $\Sigma_{\mathcal{A}}$ be an adversary strategy (for Hon) and let R be a run such that $(\Sigma_{Hon}, \Sigma_{\mathcal{A}}) \vdash R$. Further, let $\vec{\Gamma} = \mathit{lastEnv}(R)$. Then it holds*

$$\begin{aligned}&c^x \in \vec{\Gamma}.C_{cla}\\ &\Rightarrow \exists sc^x : \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R)\end{aligned}$$

Proof: This statement directly follows from the inference rule *claim* (23) by induction.

Lemma H.6. *Let $Hon = \{B_1, \ldots, B_k\}$ be a set of honest users and $\Sigma_{Hon} = \{\Sigma_{B_1}, \ldots, \Sigma_{B_k}\}$ a corresponding set of user strategies. Let $\Sigma_{\mathcal{A}}$ be an adversary strategy (for Hon) and let R be a run such that $(\Sigma_{Hon}, \Sigma_{\mathcal{A}}) \vdash R$. Further, let $\vec{\Gamma} = \mathit{lastEnv}(R)$. Then it holds*

$$\begin{aligned}&c^x \in \vec{\Gamma}.C_{en}\\ &\Rightarrow \exists \dot{c}^x : \dot{c}^x \sqsupseteq c^x \mathit{sender}(\dot{c}^x) : \mathit{authCTLC}(\dot{c}^x) \in \mathit{actions}(R)\end{aligned}$$

Proof: This statement directly follows from the inference rule *claim* (23) by induction.

Lemma H.7. *Let $Hon = \{B_1, \ldots, B_k\}$ be a set of honest users and $\Sigma_{Hon} = \{\Sigma_{B_1}, \ldots, \Sigma_{B_k}\}$ a corresponding set of user strategies. Let $\Sigma_{\mathcal{A}}$ be an adversary strategy (for Hon) and let R be a run such that $(\Sigma_{Hon}, \Sigma_{\mathcal{A}}) \vdash R$. Further, let $\vec{\Gamma} = \mathit{lastEnv}(R)$. Then it holds*

$$\begin{aligned}c^x \in \vec{\Gamma}.C_{cla} \Rightarrow \exists \dot{c}^x : \dot{c}^x \sqsupseteq c^x\\ \wedge \mathit{sender}(\dot{c}^x) : \mathit{authCTLC}(\dot{c}^x) \in \mathit{actions}(R).\end{aligned}$$

Proof: This statement directly follows from the inference rule *claim* (23) by induction.

Lemma H.8. *Let $Hon = \{B_1, \ldots, B_k\}$ be a set of honest users and $\Sigma_{Hon} = \{\Sigma_{B_1}, \ldots, \Sigma_{B_k}\}$ a corresponding set of user strategies. Let $\Sigma_{\mathcal{A}}$ be an adversary strategy (for Hon) and let R be a run such that $(\Sigma_{Hon}, \Sigma_{\mathcal{A}}) \vdash R$. Further, let $\vec{\Gamma} = \mathit{lastEnv}(R)$. Then it holds for all $\vec{\Gamma}_0, \vec{\Gamma}'$ with $R = \vec{\Gamma}_0 \longrightarrow \ldots \longrightarrow \vec{\Gamma}' \longrightarrow \ldots \longrightarrow \vec{\Gamma}$:*

$$\forall ch : \Gamma'_{ch}.S_{rev} \sqsubseteq \Gamma_{ch}.S_{rev}$$

Proof: This statement directly follows from the inference rules *revealSecret* (19) and *shareSecret* (20) by induction.

H.2. Properties implied by the honest user strategy

We show general properties of runs that can be enforced by the honest user strategy $\Sigma_B^{\mathbb{T}}$ defined in Appendix F.

Lemma H.9. *Let B be an honest user, $\mathbb{T}$ be a set of tuples of the form $(id, \mathcal{T}, t_0, spec)$, which is well-formed according to Definition F.6, and let $\Sigma_B^{\mathbb{T}}$ be the honest user strategy for B executing $\mathbb{T}$. Let $\Sigma_{Adv}^{\mathbb{T}}$ be an arbitrary adversarial strategy. Let R be a run with $\Sigma_B^{\mathbb{T}}, \Sigma_{Adv}^{\mathbb{T}} \vdash R$, starting from an initial environment. Further, let $\vec{\Gamma} = \mathit{lastEnv}(R)$. Then it holds for all $e \mathrel{\widehat{\in}} \mathcal{T}$ with $B = \mathit{receiver}(e)$:*

$$\begin{aligned}&s = \mathit{secret}(e, id) \in \Gamma_{channel(e)}.S_{rev}\\ &\Rightarrow B : \mathit{revealSecret}_{ch}\, s \in \mathit{actions}(R)\\ &\quad \mathit{with}\ ch = channel(e)\end{aligned}$$

Proof: This statement directly follows from the inference rules *revealSecret* (19) and the fact that

$$\mathit{owner}(\mathit{secret}(e, id)) = \mathit{receiver}(e)$$

from Equation (26).

Additionally, we show that whenever a contract has been claimed, it will not be enabled anymore for a second time.

Lemma H.10. *Let B be an honest user, $\mathbb{T}$ be a well-formed set of tuples of the form $(id, \mathcal{T}, t_0, spec)$ and $\Sigma_B^{\mathbb{T}}$ the honest user strategy for B executing $\mathbb{T}$. Let $\Sigma_{Adv}^{\mathbb{T}}$ be an arbitrary adversarial strategy. Let R be a run with $\Sigma_B^{\mathbb{T}}, \Sigma_{Adv}^{\mathbb{T}} \vdash R$, starting from an initial environment. Further, let $\vec{\Gamma} = \mathit{lastEnv}(R)$. Then it holds for all id:*

$$\begin{aligned}&\mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc_\iota^x)) \in \mathit{actions}(R)\\ &\wedge B \in \mathit{users}(c^x)\\ &\Rightarrow \nexists\, \dot{c}^x \in \vec{\Gamma}.C_{en} \wedge \nexists (B, \dot{c}^x) \in \vec{\Gamma}.C_{aut}\end{aligned}$$

Proof: By induction on the length of R. The only interesting cases to consider are those where contracts are added to $\vec{\Gamma}.C_{en}$ and $\vec{\Gamma}.C_{aut}$ and where a contract is claimed. So we look at the cases where the last action α in $R = R' \xrightarrow{\alpha} \Gamma$ is $\alpha = \mathit{enableCTLC}(c^x)$, $\alpha = A : \mathit{authCTLC}(c^x)$ or $\alpha = \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x))$.

If $\alpha = \mathit{enableCTLC}(c^x)$ we would have $(B, \dot{c}^x) \in \vec{\Gamma}'.C_{aut}$, by the *enableCTLC* rule (17), which is ruled out by I.H..

If $\alpha = A : \mathit{authCTLC}(c^x)$ then we distinguish the cases on whether $A = B$. If $A = B$, then we know that α must

have been scheduled by $\Sigma_B^{\mathbb{T}}$ and hence by definition of $\Sigma_B^{\mathbb{T}}$, we have that $B : \mathit{authCTLC}(\hat{\dot{c}}^x) \notin \mathit{actions}(R')$. Now assume towards contradiction that

$$\mathit{claim}(\dot{c}^x, \dot{sc}^x, \mathit{secret}_i(\dot{sc}_\iota^x)) \in \mathit{actions}(R').$$

Then there must be some R^* with $R' = R^* \xrightarrow{\alpha'} R^{**}$ for some R^{**} and $\mathit{lastEnv}(R^*) =: \vec{\Gamma}^*$ such that $\dot{c}^x \in \vec{\Gamma}^*.C_{cla}$. But then by Lemma H.7, we get that $B : \mathit{authCTLC}(c'^x) \in \mathit{actions}(R^*) \subseteq \mathit{actions}(R')$ for some $c'^x \supseteq \dot{c}^x$ leading to a contradiction. If $A \neq B$, it does not have an effect on the Lemma.

If $\alpha = \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x))$ we have $c^x \in \vec{\Gamma}'.C_{en}$ and $c^x \notin \vec{\Gamma}'.C_{en}$ by the *claim* rule (23). For $c^x \in \vec{\Gamma}'.C_{en}$ there needs to be $B : \mathit{authCTLC}(c^x) \in \mathit{actions}(R')$ by Lemma H.4. This given authorization was then removed when enabling c^x according to the *enableCTLC* rule (17). According to the honest user strategy, see Equation (37), B will not authorize another contract with the same identifier, and hence

$$\nexists(B : \mathit{authCTLC}(\dot{c}^x)) \in \mathit{actions}(R')$$

which implies $\nexists(B, \dot{c}^x) \in \vec{\Gamma}.C_{aut}$ and $\nexists\, \dot{c}^x \in \vec{\Gamma}.C_{en}$. Part of the preconditions of *claim* (23) is $c^x \in \vec{\Gamma}.C_{en}$, and part of its effects is that it removes c^x from $\vec{\Gamma}.C_{en}$. Thus, it needs to be argued that c^x cannot be enabled again.

Assume towards contradiction that in $\vec{\Gamma} = \mathit{lastEnv}(R)$ we have $c^x \in \vec{\Gamma}.C_{en}$. By Lemma H.4, we have that both sender and receiver need to authorize c^x again. Since $\mathit{users}(c^x) \cap Hon \neq \emptyset$, the honest user strategy determines at least one of these authorizations. Now, the honest user strategy, more specifically Equation (37), tells us that honest users will only authorize a contract once.

Lemma H.11. *Let B be an honest user, $\mathbb{T}$ a well-formed set of tuples of the form $(id, \mathcal{T}, t_0, spec)$ and $\Sigma_B^{\mathbb{T}}$ the honest user strategy for B executing $\mathbb{T}$. Let $\Sigma_{Adv}^{\mathbb{T}}$ be an arbitrary adversarial strategy. Let R be a run with $\Sigma_B^{\mathbb{T}}, \Sigma_{Adv}^{\mathbb{T}} \vdash R$, starting from an initial environment. Further, let $\vec{\Gamma} = \mathit{lastEnv}(R)$. Then it holds*

$$\begin{aligned}
&sc^x \in c^x \in \vec{\Gamma}.C_{en} \wedge \mathit{position}(sc^x) = j \\
&\wedge B \in \mathit{users}(c^x) \\
&\wedge \exists R', R'' : R = R' \to R'' \text{ with } \vec{\Gamma}' = \mathit{lastEnv}(R') : \\
&\quad \exists \dot{c}^x, \dot{sc}^x \in \dot{c}^x \in \vec{\Gamma}'.C_{en} \text{ with } \mathit{position}(\dot{sc}^x) > j \\
&\quad \Rightarrow \dot{sc}^x \in c^x \in \vec{\Gamma}.C_{en}.
\end{aligned}$$

Proof: This statement follows from the inference rule *timeout* (21) together with the previous Lemma H.10 by induction. The honest user strategy implies that once a contract has been claimed, it cannot be enabled again.

Lemma H.12. *Let B be an honest user, $\mathbb{T}$ a well-formed set of tuples of the form $(id, \mathcal{T}, t_0, spec)$ and $\Sigma_B^{\mathbb{T}}$ the honest user strategy for B executing $\mathbb{T}$. Let $\Sigma_{Adv}^{\mathbb{T}}$ be an arbitrary adversarial strategy. Let R be a run with $\Sigma_B^{\mathbb{T}}, \Sigma_{Adv}^{\mathbb{T}} \vdash R$, starting from an initial environment $\vec{\Gamma}_0$ which is liquid w.r.t. $\mathbb{T}$, see Definition F.5. Further, let $\vec{\Gamma} = \mathit{lastEnv}(R)$. Then it holds*

$$\begin{aligned}
&\forall (id, \mathcal{T}, t_0, spec) \in \mathbb{T}\ \forall e \mathrel{\widehat{\in}} \mathcal{T} : B = \mathit{sender}(e) \\
&\wedge\ \nexists \Psi^{id} \in \vec{\Gamma}.B : h(e, id) \in c^x \in \Psi^{id} \\
&\Rightarrow fund(e) \in \Gamma_{channel(e)}.F_{av}
\end{aligned}$$

Proof: Recalling from Definition F.5 we know that in $\vec{\Gamma}_0$ we have

$$\forall (id, \mathcal{T}, t_0, spec) \in \mathbb{T}\ \forall e \mathrel{\widehat{\in}} \mathcal{T} : fund(e) \in \Gamma_{0,channel(e)}.F_{av}.$$

Only the *enableCTLC* (17) action removes funds from $\vec{\Gamma}.F_{av}$. This action relies on $\hat{c}^x \in \vec{\Gamma}.C_{adv}$, which can only be added by the *advCTLC* (15) action, and the authorization of B. Hence, if $B = \mathit{owner}(fund(e))$ does not give its authorization, it cannot be removed from $\Gamma_{channel(e)}.F_{av}$. The action *advCTLC* relies on advertised batches $\Psi^{id} \in \vec{\Gamma}.B$ and hence, if a corresponding batch has not been advertised, which cannot be removed from $\vec{\Gamma}.B$, the following steps also have not happened yet.. Hence the fund is still available.

H.3. Protocol Security

In the following, we will use $e \in \mathcal{T}$ instead of $e \mathrel{\widehat{\in}} \mathcal{T}$ for brevity of notation.

Definition H.13. *Given a $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ and a valid specification spec on $\mathcal{T}$ (F.4), we define the set of all tree edges that involve B as*

$$\widehat{\varrho}_{B,id} := \{(X, Y)_{\vec{a}} \mathrel{\widehat{\in}} \mathcal{T} \mid X = B \vee Y = B\}.$$

Subsets $\varrho_{B,id} \subseteq \widehat{\varrho}_{B,id}$ are called consistent by definition if and only if

$$\begin{aligned}
&\forall e \in \varrho_{B,id} : \\
&\big(\forall e' \in \mathit{onPathToRoot}(\mathcal{T}, e) \cap \widehat{\varrho}_{B,id} : e' \in \varrho_{B,id} \\
&\wedge \nexists e'' \in \varrho_{B,id} \backslash \{e\} : spec(e'') = spec(e)\big).
\end{aligned}$$

In other words, for all edges in $\varrho_{B,id}$, the edges appearing on its path to the root that are also in $\widehat{\varrho}_{B,id}$ are in ϱ_B. Therefore, ϱ_B is closed regarding walks upwards in the tree. Also, there are no duplicates in $\varrho_{B,id}$. We define

$$\varrho_{B,id} \widehat{\subseteq} \mathcal{T} :\Leftrightarrow \varrho_{B,id} \subseteq \{e \mid e \mathrel{\widehat{\in}} \mathcal{T}\}.$$

We use $\varrho_{B,id} \subseteq \mathcal{T}$ instead of $\varrho_{B,id} \widehat{\subseteq} \mathcal{T}$ to simplify the notation.

Theorem H.14. *Let B be an honest user, $\mathbb{T}$ be a set of tuples of the form $(id, \mathcal{T}, t_0, spec)$ (with $\mathcal{T}$ being a tree, $id \in \mathbb{N}$ a unique id for the tree, $t_0 \in \mathbb{R}^{>0}$ and $spec(e)$ the specification for e), which is well-formed (Definition F.6), and $\Sigma_B^{\mathbb{T}}$ the honest user strategy for B executing $\mathbb{T}$. Let $\Sigma_{Adv}^{\mathbb{T}}$ be an arbitrary adversarial strategy. Then for all runs R with $\Sigma_B^{\mathbb{T}}, \Sigma_{Adv}^{\mathbb{T}} \vdash R$, starting from an initial environment, and for all $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ there exists a consistent $\varrho_{B,id} \subseteq \mathcal{T}$ s.t. $R \sim \{\varrho_{B,id}\}_{\mathbb{T}}$ and $Inv(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds.*

Where $\{\varrho_{B,id}\}_{\mathbb{T}} := \{\varrho_{B,id} \mid (id, \mathcal{T}, t_0, spec) \in \mathbb{T}\}$ *and*

$$
\begin{aligned}
&R \sim \{\varrho_{B,id}\}_{\mathbb{T}} :\Leftrightarrow \\
&\forall (sc^x, secret_i(sc^x)) \in contracts_{com}(R, id) : \\
&\exists \varrho_{B,id}, e \in \varrho_{B,id} : sc^x = h(e, id) \\
&\qquad \wedge secret_i(sc^x) = h_{sec}(e, id) \qquad (47) \\
&\wedge \forall \varrho_{B,id}, e \in \varrho_{B,id} : \qquad (48) \\
&\qquad (h(e, id), h_{sec}(e, id)) \in contracts_{com}(R, id)
\end{aligned}
$$

and by recalling from Appendix F we set

$$actions(R) = \{\alpha \mid R = R_1 \xrightarrow{\alpha} R_2\} \quad \textit{(see (44))}$$

$$s\text{-}actions(R) = \{\alpha \mid \exists \vec{\Gamma} : R \xrightarrow{\alpha} \vec{\Gamma}\} \quad \textit{(see (45))}$$

$$com\text{-}actions(R) := actions(R) \cup s\text{-}actions(R)$$

$$
\begin{aligned}
&contracts_{out}(R, id) := \qquad (49) \\
&\qquad \{(sc^x, secret_i(sc^x)) \mid \alpha \in actions(R) \\
&\qquad \wedge B = sender(sc^x) \\
&\qquad \wedge \alpha = claim(c^x, sc^x, secret_i(sc^x))\}
\end{aligned}
$$

$$
\begin{aligned}
&contracts_{in}(R, id) := \qquad (50) \\
&\qquad \{(sc^x, secret_i(sc^x)) \mid \alpha \in com\text{-}actions(R) \\
&\qquad \wedge B = receiver(sc^x) \\
&\qquad \wedge \alpha = claim(c^x, sc^x, secret_i(sc^x))\}
\end{aligned}
$$

$$
\begin{aligned}
&contracts_{com}(R, id) := \\
&\qquad contracts_{out}(R, id) \cup contracts_{in}(R, id)
\end{aligned}
$$

and it holds $Inv(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ *defined as*

$$
\begin{aligned}
&Inv(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) \\
&\quad := \bigwedge_{S \in InvNames} Inv_S(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))
\end{aligned}
$$

consisting of the invariants defined below. We define ten invariants

$$
\begin{aligned}
&InvNames := \\
&\{\textit{in-secrets, secrets, in-schedule, levels, liveness,} \\
&\textit{init-liveness, deposits, setup, tree, auth}\}
\end{aligned}
$$

The first invariant ensures that whenever the secret assigned to an ingoing edge has been revealed, the edge will get executed.

$$
\begin{aligned}
&Inv_{\textit{in-secrets}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) :\Leftrightarrow \\
&\forall Y, e_{in}, \vec{\Gamma} \textit{ with } \vec{\Gamma} = lastEnv(R) : \\
&e_{in} = (Y, B)_{\vec{a}} \in \mathcal{T}, secret(e_{in}, id) \in \vec{\Gamma}.S_{rev} \\
&\Longrightarrow e_{in} \in \varrho_{B,id}
\end{aligned}
$$

It also should be assured that if an edge is executed, all secrets of the edges on the path to the root have been revealed.

$$
\begin{aligned}
&Inv_{secrets}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) :\Leftrightarrow \\
&\forall \vec{\Gamma} \textit{ with } \vec{\Gamma} = lastEnv(R) : \\
&\big(e \in \varrho_{B,id} \\
&\Longrightarrow \forall e' \in onPathToRoot(\mathcal{T}, e) : \\
&\qquad secret(e', id) \in \Gamma_{channel(e)}.S_{rev}\big)
\end{aligned}
$$

Additionally, the secret of an ingoing edge should only be published if the outgoing edge, if existent, had been claimed before.

$$
\begin{aligned}
&Inv_{\textit{in-schedule}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) :\Leftrightarrow \\
&\forall X, Y, e_{in}, \vec{\Gamma} \textit{ with } \vec{\Gamma} = lastEnv(R) : \\
&\big(e_{in} = (Y, B)_{\vec{a}} \in \mathcal{T}, depth(e_{in}) > 1, \\
&secret(e_{in}, id) \in \vec{\Gamma}.S_{rev} \\
&\Longrightarrow \forall e_{out} = (B, X)_{\vec{a}'} \in \mathcal{T} \textit{ with } \vec{a} = [(Y, B)_{\vec{a}}] \cdot \vec{a}' : \\
&\qquad claim(c^x, h(e_{out}, id), h_{sec}(e_{out}, id)) \in actions(R)\big)
\end{aligned}
$$

The levels invariant ensures that once the timeout for a $h(e, id)$ *has run out, there is no channel in which the subcontract is enabled, and its fund is reserved.*

$$
\begin{aligned}
&Inv_{levels}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) :\Leftrightarrow \\
&\forall e \in \mathcal{T} \textit{ with } B = sender(e), \vec{\Gamma} \textit{ with } \vec{\Gamma} = lastEnv(R) : \\
&\Big(\vec{\Gamma}.t > t_0 + depth(e)\Delta \\
&\Longrightarrow \nexists ch, c^x \in \Gamma_{ch}.C_{en} : h(e, id) \in c^x \\
&\qquad \wedge fund(c^x) \in \Gamma_{ch}.F_{res}\Big)
\end{aligned}
$$

For the liveness of the protocol, we need to ensure that if an outgoing edge has been claimed and there is an ingoing one, its timelock has not run out yet, or a representative higher-up has been decided before.

$$
\begin{aligned}
&Inv_{liveness}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) :\Leftrightarrow \\
&\forall Y, e_{out}, \vec{\Gamma} \textit{ with } \vec{\Gamma} = lastEnv(R) : \\
&\Big(e_{out} = (B, X)_{\vec{a}} \in \mathcal{T}, j = depth(e_{out}), \\
&claim(c^x, h(e_{out}, id), h_{sec}(e_{out}, id)) \in actions(R) \\
&\Longrightarrow \forall e_{in} = (Y, B)_{\vec{a}'} \in \mathcal{T} \textit{ with } \vec{a}' = [(Y, B)] \cdot \vec{a} : \\
&\big(\exists j' \leq j + 1, \vec{a}'' : (Y, B)_{\vec{a}''} \in \varrho_{B,id} \\
&\wedge depth((Y, B)_{\vec{a}''}) = j'\big) \\
&\vee \big(\exists \dot{c}^{x'} \in \Gamma_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in \dot{c}^{x'} \\
&\wedge \vec{\Gamma}.t < t_0 + (j + 1)\Delta\big)\Big)
\end{aligned}
$$

Note that $\vec{a}'' = \vec{a}'$ *is possible. Similarly, it should be ensured that if* $e \in \varrho_{B,id}$*, it either has been claimed already or there is still enough time to do so before a respective timeout*

action can happen.

$$
\begin{aligned}
&Inv_{\textit{init-liveness}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) :\Leftrightarrow \\
&\forall e \in \mathcal{T}, \vec{\Gamma} \textit{ with } \vec{\Gamma} = \textit{lastEnv}(R) \,:\, e \in \varrho_{B,id} \\
&\Rightarrow \exists c^x : \textit{claim}(c^x, h(e,id), h_{sec}(e,id)) \in \textit{actions}(R) \\
&\quad \vee \vec{\Gamma}.t < \textit{timelock}(h(e,id))
\end{aligned}
$$

The corresponding funds should be available to execute enabled and claimed contracts. This is ensured in the following invariant.

$$
\begin{aligned}
&Inv_{\textit{deposits}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) :\Leftrightarrow \\
&\forall \vec{\Gamma} \textit{ with } \vec{\Gamma} = \textit{lastEnv}(R), \Gamma_{ch} \in \vec{\Gamma} \,: \\
&\big(\exists c^x \in \Gamma_{ch}.C_{en} \cup \Gamma_{ch}.C_{cla} \Longrightarrow \textit{fund}(c^x) \in \Gamma_{ch}.F_{res}\big)
\end{aligned}
$$

The next invariant says that if a contract for an outgoing edge is enabled or authorized by both the sender and receiver, then the contract for the ingoing edge is enabled, or another representative at the same pr smaller tree level has already been executed. This is because if a contract was enabled for an outgoing edge, we need to show that the ingoing edge is either enabled or has already been executed.

$$
\begin{aligned}
&Inv_{\textit{setup}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) :\Leftrightarrow \\
&\forall X, Y, e_{out} = (B, X)_{\vec{a}} \in \mathcal{T}, \vec{\Gamma} \textit{ with } \vec{\Gamma} = \textit{lastEnv}(R) \,: \\
&\big(h(e_{out}, id) \in c^x \in \Gamma_{channel(e_{out})}.C_{en} \\
&\vee (B, \hat{c}^x) \in \Gamma_{channel(e_{out})}.C_{aut}\,, h(e_{out}, id) \in \hat{c}^x \\
&\Longrightarrow \forall e_{in} = (Y, B)_{\vec{a}'} \in \mathcal{T} \textit{ with } \vec{a}' = [(Y, B)_{\vec{a}}] \cdot \vec{a} \,: \\
&(\exists (c^x)' \in \Gamma_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in (c^x)' \\
&\vee \exists j' \leq \textit{depth}(e_{in}) : (Y, B)_{\vec{a}''} \in \varrho_{B,id} \\
&\quad \wedge \textit{depth}((Y, B)_{\vec{a}''}) = j')\big)
\end{aligned}
$$

The following invariant ensures all subcontracts, including the honest user B, are linked to at least one edge in a tree.

$$
\begin{aligned}
&Inv_{\textit{tree}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) :\Leftrightarrow \\
&\forall \vec{\Gamma} = \textit{lastEnv}(R) \,: \\
&(\,(B, \hat{c}^x) \in \vec{\Gamma}.C_{aut} \\
&\Longrightarrow \forall sc^x \in \hat{c}^x \,\exists e \in \mathcal{T} : sc^x = h(e, id)) \\
&\wedge (c^x \in \vec{\Gamma}.C_{en} \cup \vec{\Gamma}.C_{cla} \wedge B \in \textit{users}(c^x) \\
&\Longrightarrow \forall sc^x \in c^x \,\exists e \in \mathcal{T} : sc^x = h(e, id))
\end{aligned}
$$

Finally, it is ensured that all authorized contracts also get enabled afterwards.

$$
\begin{aligned}
&Inv_{\textit{auth}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec)) :\Leftrightarrow \\
&\forall \Gamma_{ch} \in \vec{\Gamma} = \textit{lastEnv}(R), (B, \hat{c}^x) \in \Gamma_{ch}.C_{aut}, s := |\hat{c}^x| \in \mathbb{N}: \\
&(B = \textit{sender}(\hat{c}^x), \\
&c^x = \{sc^x \mid sc^x \in \hat{c}^x \ \wedge \ \textit{position}(sc^x) = s\} \\
&\Longrightarrow \textit{enableCTLC}\, c^x \in \Sigma_B^{\mathbb{T}}(R)
\end{aligned}
$$

Remark H.15. *Note that for the honest user B, only the claimed outgoing contracts are relevant, not the scheduled ones. Hence, only these are included in $\textit{contracts}_{out}(R, id)$. For the ingoing ones, though, both already claimed and possibly claimed contracts are relevant, which is why they are both included in $\textit{contracts}_{in}(R, id)$.*

Proof: Let $B, \mathbb{T}, \Sigma_B^{\mathbb{T}}, \Sigma_{Adv}^{\mathbb{T}}$ as stated in the Theorem and let R be a run with $\Sigma_B^{\mathbb{T}}, \Sigma_{Adv}^{\mathbb{T}} \vdash R$ and $|R| = n$.

We prove by induction on n:

$$
\begin{aligned}
&\forall (id, \mathcal{T}, t_0, spec) \in \mathbb{T}\, \exists \varrho_{B,id} \subseteq \mathcal{T} \text{ consistent :} \\
&\quad R \sim \{\varrho_{B,id}\}_{\mathbb{T}} \wedge Inv(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))
\end{aligned}
$$

Case $n = 0$:
By definition of R and $|R| = 0$ we have $\textit{actions}(R) = \emptyset$ and that $\Gamma = \textit{lastEnv}(R)$ is an initial environment, see Definition E.1. We show that $a)\, R \sim \{\varrho_{B,id}\}_{\mathbb{T}}$ and $b)\, Inv(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$.

$a)$ Since $\textit{actions}(R) = \emptyset$ by definition also $\textit{contracts}_{in}(R, id) = \emptyset$ and so it is sufficient to show that $\textit{contracts}_{out}(R, id) = \emptyset$ to show the statement (since trivially $R \sim \emptyset$ in this case). By the definition of $\textit{contracts}_{out}(R, id)$ it is left to show

$$
\forall \alpha \in \textit{s-actions}(R) : \alpha \neq \textit{claim}(c^x, sc^x, \textit{secret}_i(sc_\iota^x)).
$$

The set $\textit{s-actions}(R)$ contains all α s.t. $\exists \vec{\Gamma}' : \vec{\Gamma} \xrightarrow{\alpha} \vec{\Gamma}'$. By the definition of the *CTLC* semantics, the action *claim* is impossible in an initial environment since $\vec{\Gamma}.C_{en} = \vec{\Gamma}.C_{cla} = \emptyset$.

$b)$ To show that $Inv(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds, we go through the individual invariants. Let $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ be given then:

- $Inv_{\text{in-secrets}}(R, \emptyset, (id, \mathcal{T}, t_0, spec))$ trivially holds since $\vec{\Gamma}.S_{rev} = \emptyset$.
- $Inv_{\text{secrets}}(R, \emptyset, (id, \mathcal{T}, t_0, spec))$ holds since there is no $e \in \varrho_{B,id} = \emptyset$ and therefore $\textit{onPathToRoot}(T, e)$ is also empty.
- $Inv_{\text{in-schedule}}(R, \emptyset, (id, \mathcal{T}, t_0, spec))$ holds because there are no revealed secrets yet, thus $\vec{\Gamma}.S_{rev} = \emptyset$. Hence an $e_{in} \in \mathcal{T}$ with $\textit{secret}(e_{in}) \in \vec{\Gamma}.S_{rev}$ does not exist.
- $Inv_{\text{levels}}(R, \emptyset, (id, \mathcal{T}, t_0, spec))$ holds because no *CTLC*s have been enabled yet. Thus, $\vec{\Gamma}.C_{en} = \emptyset$ and the conclusion of the invariant trivially holds.
- $Inv_{\text{liveness}}(R, \emptyset, (id, \mathcal{T}, t_0, spec))$ holds since $\textit{actions}(R) = \emptyset$ and therefore no *claim* action has happened yet.
- $Inv_{\text{init-liveness}}(R, \emptyset, (id, \mathcal{T}, t_0, spec))$ has no items to apply to since $\varrho_{B,id} = \emptyset$, thus it holds.
- $Inv_{\text{deposits}}(R, \emptyset, (id, \mathcal{T}, t_0, spec))$ is true because $\vec{\Gamma}.C_{en} = \vec{\Gamma}.C_{cla} = \emptyset$.
- $Inv_{\text{setup}}(R, \emptyset, (id, \mathcal{T}, t_0, spec))$ also holds because
$$\vec{\Gamma}.C_{en} = \vec{\Gamma}.C_{aut} = \emptyset.$$
- $Inv_{\text{tree}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ is true because
$$\vec{\Gamma}.C_{aut} = \vec{\Gamma}.C_{en} = \vec{\Gamma}.C_{cla} = \emptyset.$$
- $Inv_{\text{auth}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds for the same reason.

Case $n > 0$:
With $|R| > 0$ we know that it exists a run R' with $R = R' \xrightarrow{\alpha} \Gamma$. By induction hypothesis (I.H.), we know there is also a $\{\varrho'_{B,id}\}_{\mathbb{T}}$ such that $R' \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$ and $Inv(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds for all $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$. Let $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ be given. We show that there exists $\{\varrho_{B,id}\}_{\mathbb{T}}$ s.t. $R \sim \{\varrho_{B,id}\}_{\mathbb{T}}$ and $Inv(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds. We proceed by case distinction on α.

Case $\alpha =$ *advBatch*:
We show that

a) $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$,
b) $Inv(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ hold.

a) According to the *advBatch* rule only the batch set B changes between $\vec{\Gamma}'$ and $\vec{\Gamma}$, more specifically it changes between all Γ_{ch} as it is a global action. Therefore $contracts_{out}(R, id) = contracts_{out}(R', id)$, $contracts_{in}(R, id) = contracts_{in}(R', id)$ for all id and thus

$$R' \sim \{\varrho'_{B,id}\}_{\mathbb{T}} \Rightarrow R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}.$$

b) All components in the environment remain unchanged except B. Additionally all invariants within $Inv(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ only refer to actions in $actions(R)$ that already existed in $actions(R')$. Thus $Inv(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is implied directly from $Inv(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$, i.e. the I.H..

Case $\alpha = A :$ *commitBatch*:
Similar to the previous case $contracts_{out}(R, id) = contracts_{out}(R', id)$, $contracts_{in}(R, id) = contracts_{in}(R', id)$ remain unchanged for all id and so $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$. Also analogously, *commitBatch* is a global action, and $\vec{\Gamma}$ remains unchanged except $\Gamma.S_{com}$, which is not accessed by the invariants, and they refer only to actions in $actions(R)$ that already existed in $actions(R')$.

Case $\alpha =$ *advCTLC*:
The same reasoning applies here, except that *advCTLC* is a local action that only affects one $\Gamma_{ch} \in \vec{\Gamma}$ where $\Gamma_{ch}.C_{adv}$ has been changed. Since none of the invariants accesses this component, they follow immediately from I.H..

Case $\alpha = A :$ *authCTLC* $\hat{c}^x$:
Similar to the previous cases we argue that $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$ holds. Out of the nine invariants from which $Inv(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is constructed, only $Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ and $Inv_{\text{setup}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ access this component for a $\Gamma_{ch} \in \vec{\Gamma}$ and id with $\hat{c}^x \in \Psi^{id}$.

To show $Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$, we only need to consider the case $A = B$ because the invariant only considers authorizations by the honest user B. In this case, for $(B, \hat{c}^x) \in \Gamma_{ch}.C_{aut} \backslash \Gamma'_{ch}.C_{aut}$ we have to show

$$\forall sc^x \in \hat{c}^x \, \exists e \in \mathcal{T} : sc^x = h(e, id).$$

Since this authorization is given by honest user B, it cannot have been scheduled by the adversary and is hence determined by the honest user strategy (of B). From Equation (37) we see that B will only execute this action if c^x is part of a batch determined by

$$treetoCTLCBadge((id, \mathcal{T}, t_0, spec), \mathcal{S}).$$

Hence, Equation (30) gives us the desired property.

The second affected invariant $Inv_{\text{setup}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ also follows from the honest user strategy, see Equation (35). This is because, again, we only need to consider the case $A = B$ and according to the honest user strategy of B, it only schedules an authorization action for a CTLC representing an outgoing edge if contracts corresponding to all ingoing edges have been enabled (this is specified in Equation (35)). Accordingly, the left disjunct of the invariant's conclusion must hold since $\vec{\Gamma}.C_{en} = \vec{\Gamma}'.C_{en}$.

For $Inv_{\text{auth}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we argue that since the authorization $(B, \hat{c}^x) \in \vec{\Gamma}.C_{aut}$ is given by B, the action to do so is determined by the honest user strategy. Based on Equation (37), we know that the action $B :$ *authCTLC* $\hat{c}^x$ will only be scheduled before t_0. For this $newC(e, R)$ (36) ensures that all conditions for also enabling this contract are met.

Case $\alpha =$ *enableCTLC* c^x:
We first show that $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$. This trivially follows from the inductive hypothesis if we can show that for all id

$$contracts_{out}(R', id) = contracts_{out}(R, id) \text{ and}$$
$$contracts_{in}(R', id) = contracts_{in}(R, id).$$

For id with $c^x \notin \Psi^{id}$, this follows immediately from I.H. so we consider the specific id with $c^x \in \Psi^{id}$. The first claim follows directly from the definition of $contracts_{out}(R, id)$. To show

$$contracts_{in}(R', id) = contracts_{in}(R, id)$$

we assume towards contradiction that there exists

$$(sc^x, secret_i(sc^x)) \in contracts_{in}(R, id) \backslash contracts_{in}(R', id). (*)$$

Then there is $\vec{\Gamma}^*$ such that $R \xrightarrow{\alpha'} \vec{\Gamma}^*$ for

$$\alpha' = claim(c^x, sc^x, secret_i(sc^x_\iota))$$

for some c^x with $sender(sc^x) = B$. If $\alpha' = claim(c^x, sc^x, secret_i(sc^x_\iota))$ then by Equation (23) we know that $sc^x \in c^x \in \Gamma_{ch}.C_{en}$ and there is a i such that $secret_i(sc^x_\iota) \subseteq \Gamma'_{ch}.S_{rev}$. From

$$Inv_{\text{tree}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec)),$$

we know that $\exists e \in \mathcal{T} : sc^x = h(e, id)$. From the way that h is defined (Equation (29)) we can conclude that $sender(e) = B$. And that there is some e' such that $sender(e) = sender(e')$, $receiver(e) = receiver(e')$, $depth(e) = depth(e')$, $h(e', id) = h(e, id)$ and

$$secret_i(sc^x) = h_{sec}(e', id).$$

Consequently, $secret(e', id) \in \Gamma'_{ch}.S_{rev}$. From the inductive

hypothesis, we know that

$$Inv_{\text{in-secrets}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$$

holds and consequently also $e' \in \varrho'_{B,id}$. Since the inductive hypothesis also gives that $R' \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$, we get

$$(sc^x = h(e, id) = h(e', id), \mathit{secret}_i(sc^x)) \in \mathit{contracts}_{in}(R', id),$$

contradicting (*).

To show the invariants, we first analyze the environment changes induced by the *enableCTLC* action.

From the inference rule of the *enableCTLC* action, we know that there is $\Gamma'_{ch} \in \vec{\Gamma}'$ and $\Gamma_{ch} \in \vec{\Gamma}$ with

$$\Gamma_{ch}.C_{en} \supsetneq \Gamma'_{ch}.C_{en},\ \Gamma_{ch}.F_{av} \subsetneq \Gamma'_{ch}.F_{av},\ \Gamma_{ch}.F_{res} \supsetneq \Gamma'_{ch}.F_{res}.$$

Affected by these changes are the invariants

- $Inv_{\text{levels}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$,
- $Inv_{\text{liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$,
- $Inv_{\text{deposits}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$,
- $Inv_{\text{setup}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$,
- $Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$, and
- $Inv_{\text{auth}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$.

By the semantics of *enableCTLC* there is a new $c^x \in \Gamma_{ch}.C_{en} \backslash \Gamma'_{ch}.C_{en}$ for some channel *ch*.

For $Inv_{\text{levels}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we show towards contradiction that if $B = \mathit{sender}(e)$ and $\mathit{fund}(c^x) \in \Gamma_{ch}.F_{res}$ for this $c^x \in \Gamma_{ch}.C_{en}$ with $h(e, id) \in c^x$ then $\vec{\Gamma}.t < t_0 + \mathit{depth}(e)\Delta$ holds.

Note that from $h(e, id) \in c^x$ and $B = \mathit{sender}(e)$, we also know that $\mathit{sender}(c^x) = B$. The preconditions for executing *enableCTLC* in this case require that for some $\hat{c}^x$ it holds that $(B, \hat{c}^x) \in \Gamma_{ch}.C_{aut}$ and for $s = |\hat{c}^x|$ it holds that $c^x = \{sc_i^x \mid sc_i^x \in \hat{c}^x \wedge i = s\}$. Consequently, we know that $h(e, id) = sc_i^x$ and from $Inv_{\text{auth}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we can conclude that *enableCTLC* $c^x \in \Sigma_B^{\mathbb{T}}(R')$. By definition of Σ_B, this only is the case if $\vec{\Gamma}'.t < t_0$. So, the claim follows since $\vec{\Gamma}'.t = \vec{\Gamma}.t$.

$Inv_{\text{liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ follows from I.H. since $\varrho_{B,id}$ has not been changed between R' and R and $\Gamma_{ch}.C_{en} \supsetneq \Gamma'_{ch}.C_{en}$ holds.

For the $Inv_{\text{deposits}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ invariant there is now exactly one new $c^x \in \Gamma_{ch}.C_{en} \cup \Gamma_{ch}.C_{cla}$ which also includes exactly one sc^x. By the preconditions of the *enableCTLC* inference rule $\mathit{fund}(c^x) \in \Gamma'_{ch}.F_{res} = \Gamma_{ch}.F_{res}$.

For $Inv_{\text{setup}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ there is now exactly one more $c^x \in \vec{\Gamma}.C_{en}$ to consider. From the inference rule for *enableCTLC* (Equation (17)), we can conclude that $(B, \hat{c}^x) \in \Gamma'_{channel(e_{out})}.C_{aut}$ for a $\hat{c}^x \supseteq c^x$. Consequently, if $h(e_{out}, id) \in c^x$ with $e_{out} \widehat{\in} \mathcal{T}$ as the invariant's precondition desires it, then also $h(e_{out}, id) \in \hat{c}^x$. We can hence use $Inv_{\text{setup}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ (given by the inductive hypothesis) to immediately show the invariant's conclusion (using the fact that $\Gamma'_{ch}.C_{en} \subseteq \Gamma'_{ch}.C_{en}$ for all channels *ch*).

For $Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we look at $\vec{\Gamma}' = \mathit{lastEnv}(R')$ and $\vec{\Gamma} = \mathit{lastEnv}(R)$. From the inference rule of *enableCTLC* (17) we know $(B, \hat{c}^x) \in \vec{\Gamma}'.C_{aut}$ in case $\alpha = \mathit{enableCTLC}\, c^x$. Therefore this invariant follows from $Inv_{\text{tree}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$.

For $Inv_{\text{auth}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we look at the case of c^x where

$$\mathit{enableCTLC}\, c^x \in \Sigma_B^{\mathbb{T}}(R') \wedge \mathit{enableCTLC}\, c^x \notin \Sigma_B^{\mathbb{T}}(R).$$

Here the *enableCTLC* rule (17) implies $(B, \hat{c}^x) \notin \vec{\Gamma}.C_{aut}$, so this invariant still holds. For the other cases, we still have *enableCTLC* in the honest user strategy since the set of enabled *CTLC*s has only gotten larger, see $newC(e, R)$ (36).

Case $\alpha = A : \mathit{enableSubC}\ sc^x$:

Here $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$ holds with analogous reasoning as in the previous case. Again, this is because enabling a new (sub)contract representing an ingoing edge e of B does not allow for claiming such a contract immediately. Instead, this option is only available after B revealed the corresponding edge secret $\mathit{secret}(e, id)$, which honest user B will only do after enabling (this connection is made formal by $Inv_{\text{in-secrets}}$).

Also, similar to the previous case, all components of the environment remain untouched except $\Gamma'_{ch}.C'_{en} \neq \Gamma_{ch}.C_{en}$, which implies that only

- $Inv_{\text{levels}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$,
- $Inv_{\text{liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$,
- $Inv_{\text{deposits}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$,
- $Inv_{\text{setup}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$,
- $Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$, and
- $Inv_{\text{auth}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$.

can be affected for the specific id with $sc^x \in \hat{c}^x \in \Psi^{id}$.

For Inv_{levels}, we show towards contradiction that if $B = \mathit{sender}(e)$ and $\mathit{fund}(c^x) \in \Gamma_{ch}.F_{res}$ for this $c^x \in \Gamma_{ch}.C_{en}$ with $h(e, id) \in c^x$ then

$$\vec{\Gamma}.t < t_0 + \mathit{depth}(e)\Delta$$

holds.

For all $h(e, id) \in c^x \in \Gamma'_{ch}.C_{en}$, this immediately follows from the inductive hypothesis. So we only need to consider the case where $h(e, id) = sc^x$. Since the preconditions for executing *enableSubC* in this case require that it holds that $sc^x \in c^x$ and $\mathit{sender}(c^x) = A$ we know that $A = \mathit{sender}(c^x) = \mathit{sender}(e) = B$. Consequently, we know that $B : \mathit{enableSubC}\ sc^x \in \Sigma_B^{\mathbb{T}}(R')$ (since only the honest strategy of B can schedule such actions). By definition of Σ_B, this only is the case if $\vec{\Gamma}'.t < t_0$. So, the claim follows since $\vec{\Gamma}'.t = \vec{\Gamma}.t$.

$Inv_{\text{liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ follows from the fact that there are more subcontracts in $\Gamma_{ch}.C_{en}$ than in $\Gamma'_{ch}.C_{en}$ and that $\varrho'_{B,id}$ remained unchanged.

For $Inv_{\text{deposits}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we look at its preconditions: The *enableSubC* action adds a sub-contract

to an already enabled *CTLC* and thus

$$\exists c^x \in \Gamma'_{ch}.C_{en} \cup \Gamma'_{ch}.C_{cla} : \exists sc^x \in c^x, e \in \mathcal{T} : sc^x = h(e, id)$$
$$\Leftrightarrow$$
$$\exists c^x \in \Gamma_{ch}.C_{en} \cup \Gamma_{ch}.C_{cla} : \exists sc^x \in c^x, e \in \mathcal{T} : sc^x = h(e, id).$$

Additionally, $\Gamma'_{ch}.F_{res} = \Gamma_{ch}.F_{res}$ and so

$$Inv_{\text{deposits}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$$

is implied by I.H..

For $Inv_{\text{setup}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ there is now exactly one more $sc^x \in c^x \in \vec{\Gamma}.C_{en}$ to consider. Consequently, we need to consider that $sc^x = h(e_{out}, id) \in c^x$ with $e_{out} = (B, X)_{\vec{a}} \widehat{\in} \mathcal{T}$ and show that the invariant's conclusion holds for this sc^x. We show that indeed for all $e_{in} = (Y, B)_{\vec{a}'} \in \mathcal{T}$ with $\vec{a}' = [(Y, B)_{\vec{a}'}] \cdot \vec{a}$ there is $(c^x)' \in \Gamma_{channel(e_{in})}.C_{en}$ such that $h(e_{in}, id) \in (c^x)'$ (left disjunct of the conclusion). Since the preconditions for executing *enableSubC* in this case require that it holds that $sc^x \in c^x$ and $sender(c^x) = A$ we know that $A = sender(c^x) = sender(e_{out}) = B$. Consequently, we know that $B : enableSubC\ sc^x \in \Sigma_B^{\mathbb{T}}(R')$ (since only the honest strategy of B can schedule such actions). By definition of Σ_B, this is only the case if $ingoing(e_{out}, R') = 1$. This implies that there is some $(c^x)' \in \Gamma'_{channel(e_{in})}.C_{en}$ such that $h(e_{in}, id) \in (c^x)'$. Since $\Gamma_{ch}.C_{en}$ contains strictly more subcontracts than $\Gamma'_{ch}.C_{en}$ it is also ensured that there is some $(c^x)^* \in \Gamma_{channel(e_{in})}.C_{en}$ such that $h(e_{in}, id) \in (c^x)^*$, concluding the case.

$Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds since if for the newly enabled subcontract we have

$$c^x \in \vec{\Gamma}.C_{en} \wedge B \in users(c^x)$$

the *authCTLC* rule (16) implies $(B, \hat{c}^x) \in \vec{\Gamma}'.C_{aut}$. For this $Inv_{\text{tree}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ applies for all $sc^x \in \hat{c}^x$. By Lemma H.3 we have that $c^x \subseteq \hat{c}^x$ always holds and so $Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is implied.

$Inv_{\text{auth}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds with analogous reasoning as in the previous case $\alpha = enableCTLC\ c^x$.

Case $\alpha = A : revealSecret_{ch}\ s_{\vec{a}}^{id}$:

For the specific id of $s_{\vec{a}}^{id}$ we define

$$\varrho_{B,id} := \varrho'_{B,id} \cup \{e \in \widehat{\varrho}_{B,id} \mid \exists \vec{\Gamma}^* : R \xrightarrow{\alpha'} \vec{\Gamma}^*\} \tag{51}$$

with

$$\alpha' = claim(c^x, h(e, id), h_{sec}(e, id)),$$
$$s_{\vec{a}}^{id} \in h_{sec}(e, id), receiver(e) = B$$

(for the definition of h_{sec} see (28)) and

$$\{\varrho_{B,id}\}_{\mathbb{T}} := (\{\varrho'_{B,id}\}_{\mathbb{T}} \backslash \{\varrho'_{B,id}\}) \cup \{\varrho_{B,id}\}$$

(only replacing this specific $\varrho'_{B,id}$ for the id of $s_{\vec{a}}^{id}$) and show that

a) $R \sim \{\varrho_{B,id}\}_{\mathbb{T}}$,
b) $\varrho_{B,id}$ is consistent for all id and
c) $Inv(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds.

a) To show $R \sim \{\varrho_{B,id}\}_{\mathbb{T}}$ we start by stating that

$$contracts_{in}(R', id) \subseteq contracts_{in}(R, id), \tag{52}$$
$$contracts_{out}(R', id) = contracts_{out}(R, id) \tag{53}$$

holds. Revealing $s_{\vec{a}}^{id}$ can allow for a new α' in $s\text{-}actions(R)$. Having less possible *claim* actions for $\varrho_{B,id}$ is not possible since revealing a secret cannot remove the possibility of claiming a contract. This action moves a secret from a $\Gamma_{ch}.S_{com}$ to $\Gamma_{ch}.S_{rev}$. We now show the two parts (47) and (48) of $R \sim \{\varrho_{B,id}\}_{\mathbb{T}}$ individually. Nothing changes for all id with $\varrho_{B,id} = \varrho'_{B,id}$, so we look at the one specific id with $\varrho_{B,id} \neq \varrho'_{B,id}$.

(47) Let $(sc^x, secret_i(sc^x)) \in contracts_{in}(R, id)$ then by definition of $contracts_{in}(R, id)$ either $(sc^x, secret_i(sc^x)) \in contracts_{in}(R', id)$ or the most recent *revealSecret* action has made such an α' available with $sc^x = h(e, id)$ and $secret_i(sc^x) = h_{sec}(e, id)$. Further, since $receiver(e) = B$ and $sc^x = h(e, id)$, we know that $receiver(sc^x) = B$. Therefore $Inv_{\text{tree}}(R', \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ applies and we get that there is some e such that $h(e, id) = sc^x$. So sc^x with $secret_i(sc^x) = h_{sec}(e, id)$ is the only possible element in

$$contracts_{in}(R, id) \backslash contracts_{in}(R', id).$$

If $(sc^x, secret_i(sc^x)) \in contracts_{in}(R', id)$ then by I.H. also

$$\exists e \in \varrho'_{B,id} : h(e, id) = sc^x \wedge h_{sec}(e, id) = secret_i(sc^x)$$

and thus $e \in \varrho_{B,id}$. If $sc^x \in contracts_{in}(R, id) \backslash contracts_{in}(R', id)$ the above reasoning applies and gives us

$$\exists e \in \varrho_{B,id} : h(e, id) = sc^x \wedge h_{sec}(e, id) = secret_i(sc^x).$$

(48) Let $e \in \varrho_{B,id}$. Then either $e \in \varrho'_{B,id}$ or $e \notin \varrho'_{B,id}$ and

$$e \in \{e \in \widehat{\varrho}_{B,id} \mid \exists \vec{\Gamma}^* : R \xrightarrow{\alpha'} \vec{\Gamma}^*\}. \tag{54}$$

If $e \in \varrho'_{B,id}$ then by I.H. also

$$(h(e, id), h_{sec}(e, id)) \in contracts_{in}(R', id) \cup contracts_{out}(R', id)$$
$$\subseteq contracts_{in}(R, id) \cup contracts_{out}(R, id).$$

In the case of Equation (54) we have

$$(h(e, id), h_{sec}(e, id)) \in contracts_{in}(R, id)$$

by definition of $contracts_{in}(R, id)$ and Equation (51) given that $receiver(e) = B$.

b) Again, nothing changes for all id with $\varrho_{B,id} = \varrho'_{B,id}$, so we look at the one specific id with $\varrho_{B,id} \neq \varrho'_{B,id}$. To show that $\varrho_{B,id}$ is consistent we assume towards contradiction that $\varrho_{B,id}$ is not consistent meaning that there exists some $e \in \varrho_{B,id}$ and a predecessor $e^* \in onPathToRoot(\mathcal{T}, e) \cap \widehat{\varrho}_{B,id}$ such that $e^* \notin \varrho_{B,id}$ or

$$\exists e^{**} \in \varrho_{B,id} : spec(e^{**}) = spec(e).$$

For the first case, we distinguish whether e^* is an ingoing

or an outgoing edge.

Let $e^* = (Y, B)_{\vec{a}}$ be an ingoing edge. By assumption $e^* \notin \varrho_{B,id}$ and by $Inv_{\text{in-secrets}}$ we know that $\mathit{secret}(e^*) \not\sqsubseteq \vec{\Gamma}.S_{rev}$ holds. This contradicts that $e \in \varrho_{B,id} \wedge e \notin \varrho'_{B,id}$ since for that it would need to hold that it exists a $\vec{\Gamma}'$ and $sc^x = h(e, id)$ s.t.

$$\vec{\Gamma} \xrightarrow{\mathit{claim}(c^x, h(e,id), h_{sec}(e,id))} \vec{\Gamma}'$$

is possible, implying $\mathit{secret}(e) \sqsubseteq \Gamma_{channel(e)}.S_{rev}$. However, if e^* is a predecessor of e we have $h_{sec}(e^*, id) \sqsubseteq h_{sec}(e, id)$, see (28).

Let $e^* = (B, X)_{\vec{a}}$ be an outgoing edge. Then there must be a corresponding ingoing edge $e' = (Y, B)_{\vec{a}'}$ with $\vec{a}' = [(Y, B)] \cdot \vec{a}$. Then either $e' \notin \varrho_{B,id}$ and the previous reasoning applies or $e' \in \varrho_{B,id}$. In that case, we know from Inv_{secrets} that $\mathit{secret}(e', id) \in \Gamma_{channel(e')}.S_{rev}$ and so from $Inv_{\text{in-schedule}}$ that $\mathit{claim}(c^x, h(e^*, id), h_{sec}(e^*, id)) \in \mathit{actions}(R)$ for $h(e^*, id) \in c^x$. However, this implies

$$(h(e^*, id), h_{sec}(e^*, id)) \in \mathit{contracts}_{out}(R, id)$$

and because of $R \sim \{\varrho_{B,id}\}_{\mathbb{T}}$ from point a), also $e^* \in \varrho_{B,id}$. This is a contradiction.

For the second case, we first show that for $e \in \varrho_{B,id} \backslash \varrho'_{B,id}$, we have that $s^{id}_{\vec{a}} = \mathit{secret}(e, id)$. If $s^{id}_{\vec{a}} \neq \mathit{secret}(e, id)$ then $\mathit{secret}(e, id) \in \Gamma'_{ch}.S_{rev}$ because for $R \xrightarrow{\alpha'} \vec{\Gamma}^*$, it needs to hold that for all $s \in h_{sec}(e, id)$, $s \in \Gamma_{ch}.S_{rev}$, and so in particular, $\mathit{secret}(e, id) \in \Gamma_{ch}.S_{rev}$ (by Equation (23)). However, since $\Gamma_{ch}.S_{rev} = \Gamma'_{ch}.S_{rev} \cup \{s^{id}_{\vec{a}}\}$ (by Equation (19)), if $s^{id}_{\vec{a}} \neq \mathit{secret}(e, id)$ then it must hold that $\mathit{secret}(e, id) \in \Gamma'_{ch}.S_{rev}$. In this case, however, by $Inv_{\textit{in-secrets}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$, we also know that $e \in \varrho'_{B,id}$. By construction, we know that

$$\mathit{owner}(\mathit{secret}(e, id)) = \mathit{receiver}(e) = B$$

and so by Equation (19), we can conclude that $A = B$. Consequently, we know that $B : \mathit{revealSecret}_{ch}\, s^{id}_{\vec{a}} \in \Sigma^{\mathbb{T}}_B(R')$ (since only the honest strategy of B can schedule such actions). By definition of Σ_B, this only is the case if $no{-}dupl(e, R')$ holds. If now there would be an edge $e^{**} \in \varrho_{B,id} : spec(e^{**}) = spec(e)$ then $\mathit{sender}(e) = \mathit{sender}(e^{**})$ and $\mathit{receiver}(e) = \mathit{receiver}(e^{**})$ which would immediately contradict $no - dupl(e, R)$.

c) To show that $Inv(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds, we go through the invariants individually. In the following $\vec{\Gamma}$ will always notate $\vec{\Gamma} = \mathit{lastEnv}(R)$. Again, nothing changes for all id with $\varrho_{B,id} = \varrho'_{B,id}$, so we look at the one specific id with $\varrho_{B,id} \neq \varrho'_{B,id}$.

$\underline{Inv_{\text{in-secrets}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))}$ Let $e_{in} = (Z, B)_{\vec{a}} \in \mathcal{T}$ for some Z and $\mathit{secret}(e_{in}, id) \in \vec{\Gamma}.S_{rev}$. By the *revealSecret* rule we know that $\Gamma_{ch}.S_{rev} = \Gamma'_{ch}.S_{rev} \cup \{s^{id}_{\vec{a}}\}$. If $ch \neq channel(e_{in})$ this invariant remains unaffected so let $ch = channel(e_{in})$. Either $\mathit{secret}(e_{in}, id) \in \Gamma'_{ch}.S_{rev}$ or $s^{id}_{\vec{a}} = \mathit{secret}(e_{in}, id)$. In the first case, we know by the I.H. that $e_{in} \in \varrho'_{B,id}$ and thus $e_{in} \in \varrho_{B,id}$.

In the second case, we know $A = B$ since $B = \mathit{receiver}(e_{in})$ and Equation (26) from the honest user strategy. Consequentially $\mathit{revealSecret}_{ch}\, s^{id}_{\vec{a}} \in \Sigma^{\mathbb{T}}_B(R')$. By the definition of the honest user strategy, this is only the case if $chContract(e_{in}, R') = 2$, see Equation (41). We show that this implies all the preconditions for executing $R \xrightarrow{\alpha'} \vec{\Gamma}^*$ according to Equation (23). $chContract(e_{in}, R') = 2$ ensures that the subcontract is enabled (there is some $c^x \in \Gamma_{ch}.C_{en}$ such that $h(e, id) \in c^x$) and that the secrets of all edges on the path to the root (but $h(e, id)$) are available in $\Gamma'_{ch}.S_{rev}$, and so all relevant secrets are available $\Gamma_{ch}.S_{rev}$. From this, we can also conclude by Lemma H.3 that a corresponding contract has been advertised, and (using $Inv_{deposits}$) that the corresponding fund is reserved. We are hence left to show that $h(e, id)$ is the top-level contract in c^x. $chContract(e_{in}, R') = 2$ also gives us that there is no $sc^{x\prime} \in c^x$ with a timeout smaller than the one of $h(e, id)$. By construction, this means that $h(e, id)$ is the top-level contract in c^x. Correspondingly $R \xrightarrow{\alpha'} \vec{\Gamma}^*$ holds by Equation (23).

$\underline{Inv_{\text{secrets}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))}$ Let e, e' such that $e \in \varrho_{B,id}$ and $e' \in \mathit{onPathToRoot}(\mathcal{T}, e)$. We need to show that

$$\mathit{secret}(e', id) \in \Gamma_{channel(e)}.S_{rev}.$$

If $e \in \varrho'_{B,id}$ this holds by I.H., because once a secret is in $\Gamma_{channel(e)}.S_{rev}$, it cannot be removed from there.

If $e \in \varrho_{B,id} \backslash \varrho'_{B,id}$ then we know by (51) that $\exists \vec{\Gamma}^* : R \xrightarrow{\alpha'} \vec{\Gamma}^*$ for $\alpha' = \mathit{claim}(c^x, h(e, id), h_{sec}(e, id)), s^{id}_{\vec{a}} \in h_{sec}(e, id)$. Consequently, by the inference rule for claim Equation (23), we know that

$$h_{sec}(e, id) \subseteq \Gamma_{channel(e)}.S_{rev}.$$

By definition of h_{sec}(28) we have

$$\forall e' \in \mathit{onPathToRoot}(\mathcal{T}, e) : \mathit{secret}(e', id) \in h_{sec}(e, id)$$

and so $\mathit{secret}(e', id) \in \Gamma_{channel(e)}.S_{rev}$.

$\underline{Inv_{\text{in-schedule}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))}$ Let Z, e_{in} s.t. $e_{in} = (Z, B)_{\vec{a}} \in \mathcal{T}, \mathit{depth}(e_{in}) > 1, \mathit{secret}(e_{in}) \in \vec{\Gamma}.S_{rev}$ and let e_{out}, X s.t. $e_{out} \in \mathcal{T}, e_{out} = (B, X)_{\vec{a}'}$, $\vec{a} = [e_{in}] \cdot \vec{a}'$. We need to show that

$$\mathit{claim}(c^x, h(e_{out}, id), h_{sec}(e_{out}, id)) \in \mathit{actions}(R). \quad (55)$$

If $\mathit{secret}(e_{in}, id) \in \vec{\Gamma}'.S_{rev}$, then the claim immediately follows from I.H. since $\mathit{actions}(R') \sqsubseteq \mathit{actions}(R)$. If $\mathit{secret}(e_{in}, id) \notin \vec{\Gamma}'.S_{rev}$ then $\mathit{secret}(e_{in}, id) = s^{id}_{\vec{a}}$ with $\mathit{owner}(s^{id}_{\vec{a}}) = B$. Therefore, this action is determined by the honest user strategy and hence Equation (41) and Equation (40) imply Equation (55).

$\underline{Inv_{\text{levels}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))}$
Let $e \in \mathcal{T}, B = \mathit{sender}(e)$, $\vec{\Gamma}.t > t_0 + \mathit{depth}(e)\Delta$. Since $\vec{\Gamma}.t = \vec{\Gamma}'.t$, $\vec{\Gamma}.C_{en} = \vec{\Gamma}'.C_{en}$ and $\vec{\Gamma}.F_{res} = \vec{\Gamma}'.F_{res}$ we get $Inv_{\text{levels}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ directly from

$Inv_{\text{levels}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$.

$\underline{Inv_{\text{liveness}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))}$ Let $X, e_{out} = (B, X)_{\vec{a}} \in \mathcal{T}$ and $j := \mathit{depth}(e_{out})$, $\mathit{claim}(c^x, h(e_{out}, id), h_{sec}(e_{out}, id) \in \mathit{actions}(R)$. We need to show that for every $e_{in}(Z, B)_{\vec{a}'} \in \mathcal{T}$ with $\vec{a}' = [(Y, B)_{\vec{a}}] \cdot \vec{a}$:

$$\exists j' \leq j+1, \vec{a}'' : (Y, B)_{\vec{a}''} \in \varrho_{B,id} \wedge \mathit{depth}((Y, B)_{\vec{a}''}) = j' \tag{56}$$

or

$$\exists c^x \in \Gamma_{channel(e_{in})}.C_{en} : \quad h(e_{in}, id) \in c^x \wedge \vec{\Gamma}.t < t_0 + (j+1)\Delta \tag{57}$$

By I.H. we know that Equation (56) or Equation (57) holds for R' with $\vec{\Gamma}'$ and $\varrho'_{B,id}$. In case Equation (56) holds for R' it also holds for R and $\varrho_{B,id}$ since

$$\mathit{claim}(c^x, h(e_{out}, id), \mathit{secret}_i(sc^x_\iota)) \in \mathit{actions}(R) \Rightarrow \mathit{claim}(c^x, h(e_{out}, id), \mathit{secret}_i(sc^x_\iota)) \in \mathit{actions}(R')$$

and $\varrho'_{B,id} \subseteq \varrho_{B,id}$. In case Equation (57) holds for R' and $\varrho'_{B,id}$ it also holds for R and $\varrho_{B,id}$ because $\vec{\Gamma}.C_{en} = \vec{\Gamma}'.C_{en}$, and $\vec{\Gamma}.t = \vec{\Gamma}'.t$.

$\underline{Inv_{\text{init-liveness}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))}$ Assume $e \in \varrho_{B,id} \backslash \varrho'_{B,id}$, as for all others the invariant is implied directly by I.H.. Since α' from (51) is not yet in $\mathit{actions}(R)$

$$\nexists c^x : \mathit{claim}(c^x, h(e, id), h_{sec}(e, id)) \in \mathit{actions}(R).$$

Since $\mathit{receiver}(e) = B$ we also have $\mathit{owner}(s^{id}_{\vec{a}}) = B$ by the h_{sec} function (28). Therefore $\alpha \in \Sigma^{\mathbb{T}}_B(R')$ which implies

$$\vec{\Gamma}.t = \vec{\Gamma}'.t < \mathit{timeout}(h(e, id)),$$

see (41).

$\underline{Inv_{\text{deposits}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))}$ For all $\Gamma_{ch} \in \vec{\Gamma}$ we have

$$\Gamma_{ch}.C_{en} \cup \Gamma_{ch}.C_{cla} = \Gamma'_{ch}.C_{en} \cup \Gamma'_{ch}.C_{cla}.$$

Furthermore $\mathcal{T}$ stays constant throughout and for all $\Gamma_{ch} \in \vec{\Gamma}$ it holds $\Gamma_{ch}.F_{res} = \Gamma'_{ch}.F_{res}$, thus $Inv_{\text{deposits}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ follows directly from I.H..

$\underline{Inv_{\text{setup}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))}$
Let $X, Y, e_{out} = (B, X)_{\vec{a}} \in \mathcal{T}, c^x, h(e_{out}, id) \in c^x \in \Gamma_{channel(e_{out})}.C_{en}$ or $(B, \hat{c}^x) \in \Gamma_{channel(e_{out})}.C_{aut}, h(e_{out}, id) \in \hat{c}^x$. We need to show that $\forall e_{in} = (Y, B)_{\vec{a}'} \in \mathcal{T}$ with $\vec{a}' = [(Y, B)_{\vec{a}}] \cdot \vec{a}$:

$$\exists (c^x)' \in \Gamma_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in (c^x)' \vee \exists j' \leq \mathit{depth}(e_{in}) : (Y, B)_{\vec{a}''} \in \varrho_{B,id} \wedge \mathit{depth}((Y, B)_{\vec{a}''}) = j'.$$

By I.H., we know this holds for R' with $\vec{\Gamma}'$ and $\varrho'_{B,id}$. Additionally

$$\Gamma_{channel(e_{in})}.C_{en} = \Gamma'_{channel(e_{in})}.C_{en}, \quad \Gamma_{channel(e_{out})}.C_{aut} = \Gamma'_{channel(e_{out})}.C_{aut},$$

and $\varrho'_{B,id} \subseteq \varrho_{B,id}$, thus it also holds for R with $\vec{\Gamma}$ and $\varrho_{B,id}$. $Inv_{\text{tree}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ remains unaffected since

$$\vec{\Gamma}.C_{aut} = \vec{\Gamma}'.C_{aut}, \ \vec{\Gamma}.C_{en} = \vec{\Gamma}'.C_{en}, \quad \vec{\Gamma}.C_{cla} = \vec{\Gamma}'.C_{cla}.$$

$\underline{Inv_{\text{auth}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))}$ remains unaffected since the honest user strategy does not rely on revealed secrets for its decision to enable a contract, see $newC(e, R)$ (36).

Case $\alpha = A : \boldsymbol{\mathit{shareSecret}^{ch'}_{ch}}\, s^{id}_{\vec{a}}$**:**

According to the inference rule of $\mathit{shareSecret}^{ch'}_{ch}$ only Γ'_{ch} is effected by this action, all other elements of $\vec{\Gamma}'$ remain untouched, especially $\Gamma'_{ch'}$ with $ch' \neq ch$. Furthermore, $\Gamma'_{ch}.S_{rev} \subseteq \Gamma_{ch}.S_{rev}$. For showing $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$ we first notice that

$$\mathit{contracts}_{out}(R', id) = \mathit{contracts}_{out}(R, id), \quad \mathit{contracts}_{in}(R', id) \subseteq \mathit{contracts}_{in}(R, id)$$

for the id of $s^{id}_{\vec{a}}$. In the following, we look at this specific id only, since nothing changes for the others. Assume towards contradiction that there is a

$$(sc^x, \mathit{secret}_i(sc^x)) \in \mathit{contracts}_{in}(R, id) \backslash \mathit{contracts}_{in}(R', id).$$

Then, there needs to be a

$$\alpha' = \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{s\text{-}actions}(R) \backslash \mathit{s\text{-}actions}(R'),$$

so $\exists \vec{\Gamma}^* : R \xrightarrow{\alpha'} \vec{\Gamma}^*$. By the inference rule of *claim* (Equation (23)), this implies that $sc^x \in c^x \in \Gamma_{ch'}.S_{en}$ and $\mathit{secret}_i(sc^x) \subseteq \Gamma_{ch'}.S_{rev}$. Since $B = \mathit{receiver}(sc^x)$ the invariant $Inv_{\text{tree}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ applies and hence there is a $e \in \mathcal{T}$ with $sc^x = h(e, id)$ and hence by construction of h there is an edge $e' \in \mathcal{T}$ such that $sc^x = h(e', id)$ and $h_{sec}(e', id) = \mathit{secret}_i(sc^x)$ and $B = \mathit{receiver}(e')$. Consequently, $\mathit{secret}(e', id) \in \Gamma_{ch'}.S_{rev}$. But then also $\mathit{secret}(e', id) \in \Gamma'_{ch^*}.S_{rev}$ for some ch^* because either $\mathit{secret}(e', id) = s^{id}_{\vec{a}}$ and then by Equation (20) $\mathit{secret}(e', id) \in \Gamma'_{ch'}.S_{rev}$ or $\mathit{secret}(e', id) \neq s^{id}_{\vec{a}}$ and then $\mathit{secret}(e', id) \in \Gamma'_{ch'}.S_{rev}$ (since by Equation (20) $\Gamma_{ch'}.S_{rev} = \Gamma'_{ch'}.S_{rev} \cup \{s^{id}_{\vec{a}}\}$). But then $Inv_{\text{in-secrets}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ implies $e' \in \varrho'_{B,id}$ and hence by $R' \sim \varrho'_{B,id}$ we know that $(sc^x, \mathit{secret}_i(sc^x)) \in \mathit{contracts}_{in}(R', id)$ (contradicting our initial assumption).

We now go through the invariants individually. Since $\vec{\Gamma}.S_{rev} = \vec{\Gamma}'.S_{rev}$ (so the set of revealed secrets over all channels stays constant) the invariant $Inv_{\text{in-secrets}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is not affected. $Inv_{\text{secrets}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is also not affected since $\mathit{shareSecret}^{ch'}_{ch}$ does not remove secrets from channels. For $Inv_{\text{in-schedule}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ the same argument applies as for $Inv_{\text{in-secrets}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$. The remaining invariants do not depend on any $\Gamma_{ch}.S_{rev}$ and so are implied directly by I.H..

$Inv_{\text{auth}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ remains unaffected since the honest user strategy does not rely on revealed

secrets for its decision to enable a contract, see $newC(e, R)$ (36).

Case $\alpha = \textit{timeout}(\dot{c}^x, \dot{sc}^x)$:

We show that $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$. Since no *claim* or *withdraw* action is executed it holds immediately that

$$\textit{contracts}_{out}(R, id) = \textit{contracts}_{out}(R', id)$$

for all id. We additionally show for all id that

$$\textit{contracts}_{in}(R, id) = \textit{contracts}_{in}(R', id).$$

For all id with $\widehat{\dot{c}^x} \notin \Psi^{id}$ it holds trivially, for the one id with $\widehat{\dot{c}^x} \in \Psi^{id}$ we assume towards contradiction that

$$\textit{contracts}_{in}(R, id) \neq \textit{contracts}_{in}(R', id)$$

and consider the cases

$$(sc^x, \textit{secret}_i(sc^x)) \in \textit{contracts}_{in}(R, id) \backslash \textit{contracts}_{in}(R', id)$$

and

$$(sc^x, \textit{secret}_i(sc^x)) \in \textit{contracts}_{in}(R', id) \backslash \textit{contracts}_{in}(R, id).$$

If $(sc^x, \textit{secret}_i(sc^x)) \in \textit{contracts}_{in}(R, id) \backslash \textit{contracts}_{in}(R', id)$ then (since no new *claim* or *withdraw* action is executed) this means that $R \xrightarrow{\alpha'} \vec{\Gamma}^*$ for some $\vec{\Gamma}^*$ and $\alpha' = \textit{claim}(c^x, sc^x, \textit{secret}_i(sc^x))$ and $\textit{receiver}(sc^x) = B$. Since by the inference rule for *claim* (Equation (23)) we know that $sc^x \in c^x \in \Gamma_{ch}.C_{en}$ for some channel *ch* we can conclude using Inv_{tree} that there is some $e \in \mathcal{T}$ such that $sc^x = h(e, id)$ and hence by construction also that there is some $e' \in \mathcal{T}$ with $h(e', id) = sc^x$ and $h_{sec}(e', id) = \textit{secret}_i(sc^x)$ and $\textit{receiver}(sc^x) = \textit{receiver}(e) = \textit{receiver}(e') = B$. If $R \xrightarrow{\alpha'} \vec{\Gamma}^*$ then by the inference rule for *claim* (Equation (23)), we know that $\textit{secret}_i(sc^x) \subseteq \vec{\Gamma}.S_{rev} = \vec{\Gamma}'.S_{rev}$ and so also $\textit{secret}(e', id) \in \vec{\Gamma}.S_{rev}$ since by construction $\textit{secret}(e', id) \in h_{sec}(e', id) = \textit{secret}_i(sc^x)$. This allows us to conclude using $Inv_{\textit{in-secrets}}$ (for R' and $\varrho'_{B,id}$) that $e' \in \varrho'_{B,id}$ and so by $R' \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$ that

$$(h(e', id), h_{sec}(e', id)) \in \textit{contracts}_{in}(R', id),$$

contradicting the original assumption.

If $(sc^x, \textit{secret}_i(sc^x)) \in \textit{contracts}_{in}(R', id) \backslash \textit{contracts}_{in}(R, id)$ then this means that $R' \xrightarrow{\alpha'} \vec{\Gamma}^*$ for some $\vec{\Gamma}^*$ and

$$\alpha' = \textit{claim}(c^x, sc^x, \textit{secret}_i(sc^x))$$

and $\textit{receiver}(sc^x) = B$. Since by the inference rule for *claim* (Equation (23)) we know that $sc^x \in c^x \in \Gamma'_{ch}.C_{en}$ for some channel *ch* we can conclude using Inv_{tree} that there is some $e \in \mathcal{T}$ such that $sc^x = h(e, id)$ and hence by construction also that there is some $e' \in \mathcal{T}$ with $h(e', id) = sc^x$ and $h_{sec}(e', id) = \textit{secret}_i(sc^x)$ and $\textit{receiver}(sc^x) = \textit{receiver}(e) = \textit{receiver}(e') = B$. From $R' \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$ we hence also know that $e' \in \varrho'_{B,id}$ and consequently using $Inv_{\text{init-liveness}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ that either

$$\textit{claim}(c^x, h(e', id), h_{sec}(e', id)) \in \textit{actions}(R')$$

or $\vec{\Gamma}'.t < t_0 + \textit{depth}(e')\Delta$. The first case gives us that

$$(h(e', id), h_{sec}(e', id)) \in \textit{contracts}_{in}(R, id),$$

contradicting the original assumption. The second case immediately contradicts the precondition of Equation (21) since

$$\textit{timelock}(h(e, id)) = t_0 + \textit{depth}(e')\Delta.$$

To show $Inv(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$, we look at the 4 invariants that are potentially affected, which are Inv_{levels}, Inv_{liveness}, Inv_{deposits} and Inv_{setup} and Inv_{setup}. The changes from $\alpha = \textit{timeout}(\dot{c}^x, \dot{sc}^x)$ do not influence the other invariants.

We first show Inv_{levels}: Assume that $e \in \mathcal{T}$ with $B = \textit{sender}(e)$ and $\vec{\Gamma}.t > t_0 + \textit{depth}(e)\Delta$. Since $\vec{\Gamma}.t = \vec{\Gamma}'.t$ we get from the inductive hypothesis that

$$(*)\ \nexists ch, c^x \in \Gamma'_{ch}.C_{en} : h(e, id) \in c^x \wedge \textit{fund}(c^x) \in \Gamma'_{ch}.F_{res}.$$

Since the inference rule for *timeout* (Equation (21)) only removes $\dot{sc}^x$ from $\dot{c}^x \in \vec{\Gamma}'.C_{en}$, if there would be a $c^x \in \Gamma_{ch}.C_{en}$ with $h(e, id) \in c^x$ and $\textit{fund}(c^x) \in \Gamma_{ch}.F_{res}$ then there would also be some $c^x* \in \Gamma'_{ch}.C_{en}$ with $h(e, id) \in c^x*$ and also $\textit{fund}(c^x) \in \Gamma'_{ch}.F_{res}$ (since $\Gamma_{ch}.F_{res} = \Gamma'_{ch}.F_{res}$ for all channels *ch*). This immediately contradicts $(*)$.

To show Inv_{liveness}, let $e_{out} = (B, X)_{\vec{a}} \in \mathcal{T}, j = \textit{depth}(e_{out})$ and $e_{in} = (Y, B)_{\vec{a}'} \in \mathcal{T}$ with $\vec{a}' = [(Y, B)_{\vec{a}}] \cdot \vec{a}$ be given as stated in $Inv_{\text{liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ which are the same as in $Inv_{\text{liveness}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$. By I.H. we either have

$$\exists j' \leq j + 1 : (Y, B)_{\vec{a}''} \in \varrho'_{B,id} \wedge \textit{depth}((Y, B)_{\vec{a}''}) = j' \textbf{ or} \tag{58}$$

$$\exists c^x \in \vec{\Gamma}'.C_{en} : h(e_{in}, id) \in c^x \wedge \vec{\Gamma}'.t < t_0 + (j + 1)\Delta. \tag{59}$$

In the case of Equation (58), the conclusion trivially holds. In the case of Equation (59), we make a case distinction on $\dot{sc}^x = h(e_{in}, id)$. If $\dot{sc}^x = h(e_{in}, id)$ we immediately arrive at a contradiction because the precondition of Equation (21) requires that $\textit{timelock}(\dot{sc}^x) \leq \vec{\Gamma}'.t$ and by construction $\textit{timelock}(\dot{sc}^x) = \textit{timelock}(h(e_{in}, id)) = t_0 + \textit{depth}(e_{in})\Delta$ and $\textit{depth}(e_{in}) = j + 1$. If $\dot{sc}^x \neq h(e_{in}, id)$ then we know from $c^x \in \vec{\Gamma}'.C_{en} : h(e_{in}, id) \in c^x$ that also there is some $c^x* \in \vec{\Gamma}.C_{en} : h(e_{in}, id) \in c^x*$ since $\vec{\Gamma}.C_{en}$ coincides with $\vec{\Gamma}'.C_{en}$ with the only exception of $\dot{sc}^x$ being removed from one channel *ch*. With $\vec{\Gamma}.t = \vec{\Gamma}'.t$ this shows the conclusion.

For showing $Inv_{\text{deposits}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we notice that $\vec{\Gamma}.F_{res} = \vec{\Gamma}'.F_{res}$ and that by the inference rule of *timeout* (Equation (21)) it holds that if $c^x \in \Gamma'_{ch}.C_{en}$ then also for some $c^x* \in \Gamma'_{ch}.C_{en}$ with $\textit{fund}(c^x*) = \textit{fund}(c^x)$ (because the rule removes at most one subcontract from c^x, which leaves the contract's funds unchanged). Consequently,

the invariant follows directly from the inductive hypothesis.

To show $Inv_{\textit{setup}}$, assume $e_{\textit{out}} = (B, X)_{\vec{a}} \in \mathcal{T}$. We distinguish the cases $h(e_{\textit{out}}, id) \in c^x \in \Gamma_{channel(e_{\textit{out}})}.C_{\textit{en}}$ and $(B, \hat{c}^x) \in \Gamma_{channel(e_{\textit{out}})}.C_{\textit{aut}}$ such that $h(e_{\textit{out}}, id) \in \hat{c}^x$.

Assume that $h(e_{\textit{out}}, id) \in c^x \in \Gamma_{channel(e_{\textit{out}})}.C_{\textit{en}}$ (*). Then we know that there is also some $c^x* \in \Gamma'_{channel(e_{\textit{out}})}.C_{\textit{en}}$ such that $h(e_{\textit{out}}, id) \in c^x*$ (since the *timeout* rule atmost removes $\dot{sc}^x$ from $\dot{c}^x$). Consequently, from $Inv_{\text{setup}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we get that

$$\exists (c^x)' \in \Gamma'_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in (c^x)' \tag{60}$$

$$\vee \exists j' \leq \textit{depth}(e_{in}) : (Y, B)_{\vec{a}''} \in \varrho'_{B,id} \tag{61}$$
$$\wedge \textit{depth}((Y, B)_{\vec{a}''}) = j'$$

In the case of Equation (60) we could have $h(e_{in}, id) = \dot{sc}^x$ and thus $\nexists (c^x)' \in \Gamma_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in (c^x)'$. If $\dot{sc}^x = h(e_{in}, id)$ by the precondition of *timeout* (21) and the definition of h we have

$$\begin{aligned}\vec{\Gamma}.t &\geq \textit{timelock}(h(e_{in}, id)) \\ &> \textit{timelock}(h(e_{out}, id)) = t_0 + \textit{depth}(e_{out})\Delta\end{aligned}$$

So we get from $Inv_{\textit{levels}}$ that

$$\nexists c^x \in \Gamma_{channel(e_{out})}.C_{en} : \; h(e_{out}, id) \in c^x$$

which contradicts our assumption (*). In the case of Equation (61) the conclusion trivially holds.

Next, assume that $(B, \hat{c}^x) \in \Gamma_{channel(e_{out})}.C_{\textit{aut}}$ such that $h(e_{out}, id) \in \hat{c}^x$. In this case we also know that $(B, \hat{c}^x) \in \Gamma'_{channel(e_{out})}.C_{\textit{aut}}$ (since the *timeout* rule does not change authorizations).

Consequently, from $Inv_{\text{setup}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we again get that

$$\exists (c^x)' \in \Gamma'_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in (c^x)' \textbf{ or} \tag{62}$$

$$\exists j' \leq \textit{depth}(e_{in}) : (Y, B)_{\vec{a}''} \in \varrho'_{B,id} \wedge \textit{depth}((Y, B)_{\vec{a}''}) = j' \tag{63}$$

Again, the claim immediately follows for Equation (63). For the case of Equation (62) we only need to consider the case $h(e_{in}, id) = \dot{sc}^x$ where have by the precondition of *timeout* (21) and the definition of h:

$$\begin{aligned}\vec{\Gamma}.t &\geq \textit{timelock}(h(e_{in}, id)) > \textit{timelock}(h(e_{out}, id)) \\ &= t_0 + \textit{depth}(e_{out})\Delta > t_0\end{aligned}$$

Therefore, $Inv_{\text{auth}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ contradicts the assumption which is proven in the next paragraph independently. This is because

$$\vec{\Gamma}.t \geq t_0 \Rightarrow \nexists \hat{c}^x \in \vec{\Gamma}.C_{adv} : (B, \hat{c}^x) \in \vec{\Gamma}.C_{\textit{aut}} \wedge B = \textit{sender}(\hat{c}^x)$$

is a direct implication of this invariant. In the invariant we implied $\vec{\Gamma}.t < t_0$, which is the negation of $\vec{\Gamma}.t \geq t_0$, the precondition here. Therefore, implying that such an $(B, \hat{c}^x) \in \vec{\Gamma}.C_{\textit{aut}}$ does not exist, which is the negation of the invariants precondition, follows directly.

$Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ only applies to less subcontracts as $Inv_{\text{tree}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ and so it is implied directly by I.H.. For $Inv_{\text{auth}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we look at $\vec{\Gamma}.t = \vec{\Gamma}'.t$. For

$$(B, \widehat{\dot{c}^x}) \in \vec{\Gamma}.C_{\textit{aut}} = \vec{\Gamma}'.C_{\textit{aut}}$$

a substrategy (37) of the honest user strategy implies that B schedules

$$\textit{enableCTLC}\, \dot{c}^x \tag{64}$$

right after B : $\textit{authCTLC}\, \widehat{\dot{c}^x}$. The only action removing authorizations is *enableCTLC* (17) itself. Based on the same substrategy we know that $B : \textit{authCTLC}\, \widehat{\dot{c}^x}$ only gets scheduled if $\vec{\Gamma}.t < t_0$. By Definition F.2, we know that no time elapses as long as B does not agree, and so $\vec{\Gamma}.t < t_0$ still holds. By Equation (29) we have $\textit{timelock}(\dot{sc}^x) > t_0$ and so $\vec{\Gamma}.t > t_0$, which is a contradiction to the given action α.

Case $\alpha = \textit{refund}\, \dot{c}^x$:

To show that $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$ the same reasoning can be applied as in the *timeout* case.

From the *refund* rule, we also know

$$\begin{aligned}\vec{\Gamma}.C_{\textit{en}} \subseteq \vec{\Gamma}'.C_{\textit{en}} \wedge \vec{\Gamma}.C_{\textit{adv}} \subseteq \vec{\Gamma}'.C_{\textit{adv}} \\ \wedge \vec{\Gamma}.F_{\textit{res}} \subseteq \vec{\Gamma}'.F_{\textit{res}} \wedge \vec{\Gamma}.F_{\textit{av}} \supseteq \vec{\Gamma}'.F_{\textit{av}}.\end{aligned} \tag{65}$$

Similar to the previous case $\alpha = \textit{timeout}(\dot{c}^x, \dot{sc}^x)$ out of $Inv(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ only 5 invariants are affected for the id with $\widehat{\dot{c}^x} \in \Psi^{id}$:

$$Inv_{\text{levels}}, Inv_{\text{liveness}}, Inv_{\text{deposits}}, Inv_{\text{setup}}, Inv_{\text{tree}}, \text{ and } Inv_{\text{auth}}$$

For proving $Inv_{\text{levels}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we note that no time has passed, $\vec{\Gamma}.t = \vec{\Gamma}'.t$, and the set of enabled contracts as well as reserved funds have only gotten smaller, see (65). Therefore, it follows immediately from I.H..

The proof for $Inv_{\text{liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ works analogously to its proof in the $\alpha = \textit{timeout}(\dot{c}^x, \dot{sc}^x)$ case.

For showing $Inv_{\text{deposits}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we look at the *refund* rule (22) and notice that exactly $\textit{fund}(c^x)$ gets removed from $\Gamma'_{ch}.F_{\textit{res}}$ alongside c^x from $\Gamma'_{ch}.C_{\textit{en}}$. Thus $Inv_{\text{deposits}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ applies to one less contract than $Inv_{\text{deposits}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$. The fund for no other contract can be missing in $\Gamma_{ch}.F_{\textit{res}}$ since every contract has a unique fund according to Definition E.1.

Again, the proof for $Inv_{\text{setup}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ works analogously to its proof in the $\alpha = \textit{timeout}(\dot{c}^x, \dot{sc}^x)$ case considering that $\dot{c}^x = \{\dot{sc}^x\}$ (which is given since the preconditions of the *refund* rule require $\dot{c}^x$ to be a singleton set).

$Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is implied by I.H. as $\Gamma_{ch}.C_{en}$ only got smaller or stayed equal compared to $\Gamma'_{ch}.C_{en}$. Their implication in this invariant is unaffected.

$Inv_{\text{auth}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is implied by analogous reasoning as in the previous case $\alpha = \textit{timeout}(\dot{c}^x, \dot{sc}^x)$.

Case $\alpha = \textit{claim}(\dot{c}^x, \dot{sc}^x, \textit{secret}_i(\dot{sc}^x))$:

For the specific id with $\widehat{\dot{c}^x} \in \Psi^{id}$ we define

$$\begin{aligned}\varrho_{B,id} := \varrho'_{B,id} \cup \{e \in \mathcal{T} \mid e = (B, X)_{\vec{a}} \wedge \dot{sc}^x = h(e, id) \\ \wedge \textit{secret}_i(\dot{sc}^x) = h_{sec}(e, id)\}\end{aligned}$$

and

$$\{\varrho_{B,id}\}_{\mathbb{T}} := (\{\varrho'_{B,id}\}_{\mathbb{T}} \backslash \{\varrho'_{B,id}\}) \cup \{\varrho_{B,id}\}$$

(only replacing this specific $\varrho'_{B,id}$ for the id of $\widehat{\dot{c}^x} \in \Psi^{id}$) and show that

a) $R \sim \{\varrho_{B,id}\}_{\mathbb{T}}$,
b) $\varrho_{B,id}$ is consistent and
c) $Inv(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds.

a) For showing that $R \sim \{\varrho_{B,id}\}_{\mathbb{T}}$, we first show that

$$\mathit{contracts}_{in}(R, id) = \mathit{contracts}_{in}(R', id).$$

Assume towards contradiction that (for the id of Ψ^{id} specifically, for the others it holds by definition)

$$\mathit{contracts}_{in}(R, id) \neq \mathit{contracts}_{in}(R', id).$$

We consider the cases

$$(sc^x, \mathit{secret}_i(\dot{sc}^x)) \in \mathit{contracts}_{in}(R, id) \backslash \mathit{contracts}_{in}(R', id)$$

and

$$(sc^x, \mathit{secret}_i(\dot{sc}^x)) \in \mathit{contracts}_{in}(R', id) \backslash \mathit{contracts}_{in}(R, id)$$

individually. Assume that there is some

$$(sc^x, \mathit{secret}_i(\dot{sc}^x)) \in \mathit{contracts}_{in}(R, id) \backslash \mathit{contracts}_{in}(R', id).$$

This means that $R \xrightarrow{\alpha'} \vec{\Gamma}^*$ for some $\vec{\Gamma}^*$ and

$$\alpha' = \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x))$$

and $\mathit{receiver}(sc^x) = B$. Since by the inference rule for *claim* (Equation (23)) we know that $sc^x \in c^x \in \Gamma_{ch}.C_{en}$ for some channel *ch* we can conclude using Inv_{tree} that there is some $e \in \mathcal{T}$ such that $sc^x = h(e, id)$ and hence by construction also that there is some $e' \in \mathcal{T}$ with $h(e', id) = sc^x$ and $h_{sec}(e', id) = \mathit{secret}_i(sc^x)$ and $\mathit{receiver}(sc^x) = \mathit{receiver}(e) = \mathit{receiver}(e') = B$. If $R \xrightarrow{\alpha'} \vec{\Gamma}^*$ then by the inference rule for *claim* (Equation (23)), we know that $\mathit{secret}_i(sc^x) \subseteq \vec{\Gamma}.S_{rev} = \vec{\Gamma}'.S_{rev}$ and so also $\mathit{secret}(e', id) \in \vec{\Gamma}.S_{rev}$ since by construction $\mathit{secret}(e', id) \in h_{sec}(e', id) = \mathit{secret}_i(sc^x)$. This allows us to conclude using $Inv_{in\text{-}secrets}$ (for R' and $\varrho'_{B,id}$) that $e' \in \varrho'_{B,id}$ and so by $R' \sim \varrho'_{B,id}$ that $sc^x = h(e', id) \in \mathit{contracts}_{in}(R', id)$, contradicting the original assumption.

If $(sc^x, \mathit{secret}_i(\dot{sc}^x)) \in \mathit{contracts}_{in}(R', id) \backslash \mathit{contracts}_{in}(R, id)$ then this means that $R' \xrightarrow{\alpha'} \vec{\Gamma}^*$ for some $\vec{\Gamma}^*$ and

$$\alpha' = \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x))$$

and $\mathit{receiver}(sc^x) = B$. We further know that there is no $\vec{\Gamma}^\dagger$ such that $R' \xrightarrow{\alpha'} \vec{\Gamma}^\dagger$ and also $sc^x \neq \dot{sc}^x$ (since otherwise by definition $sc^x \in \mathit{contracts}_{in}(R, id)$). By the inference rule of *claim* (Equation (23)), we know that all environment components influencing the rules precondition but $\Gamma_{ch}.C_{en}$ and $\Gamma.C_{adv}$ stay unaffected. Further, for $sc^x \notin \dot{c}^x$ we know that $R' \xrightarrow{\alpha'}$ since if $c^x \neq \dot{c}^x$ then $c^x \in \Gamma'_{ch}.C_{en}$ implies $c^x \in \Gamma_{ch}.C_{en}$ and $c^x \in \Gamma'_{ch}.C_{adv}$ implies $c^x \in \Gamma_{ch}.C_{adv}$ for some channel *ch*. We, hence, are left to consider the case that $sc^x \in \dot{c}^x$. By the inference rule for *claim* (Equation (23)) we know that $sc^x \in c^x \in \Gamma'_{ch}.C_{en}$, so we can conclude using Inv_{tree} that there is some $e \in \mathcal{T}$ such that $sc^x = h(e, id)$ and $\mathit{receiver}(sc^x) = \mathit{receiver}(e)$ and $\mathit{sender}(sc^x) = \mathit{sender}(e)$. Similarly, we can conclude that there is some $e' \in \mathcal{T}$ such that $\dot{sc}^x = h(e', id)$ and $\mathit{receiver}(\dot{sc}^x) = \mathit{receiver}(e')$ and $\mathit{sender}(\dot{sc}^x) = \mathit{sender}(e')$. Since all subcontracts in the same CTLC have the same sender and receiver, we can also conclude that $\mathit{receiver}(e) = \mathit{receiver}(e')$, $\mathit{sender}(e) = \mathit{sender}(e')$. And consequently from Definition F.4 that $spec(e) = spec(e')$. Since by definition, $\dot{sc}^x \in \mathit{contracts}_{in}(R', id)$ and by assumption $sc^x \in \mathit{contracts}_{in}(R', id)$, from $R' \sim \varrho'_{B,id}$, we know that $e, e' \in \varrho'_{B,id}$. However, since $\varrho'_{B,id}$ is consistent (by I.H.) this is contradicting given that $spec(e) = spec(e')$.

For $\mathit{contracts}_{out}(R, id)$ though we have

$$\mathit{contracts}_{out}(R, id) = \mathit{contracts}_{out}(R', id) \cup \{(\dot{sc}^x, \mathit{secret}_i(\dot{sc}^x))\}$$

if $B = \mathit{sender}(\dot{sc}^x)$, otherwise

$$\mathit{contracts}_{out}(R, id) = \mathit{contracts}_{out}(R', id).$$

So if $sc^x \in \mathit{contracts}_{out}(R, id)$ then either $sc^x \in \mathit{contracts}_{out}(R', id)$ and hence (by $R' \sim \varrho'_{B,id}$) also $e \in \varrho'_{B,id} \subseteq \varrho_{B,id}$ for some $e \in \mathcal{T}$ with $sc^x = h(e, id)$. Or $sc^x = \dot{sc}^x$ with $B = \mathit{sender}(sc^x)$.

Since by the inference rule of the *claim* action (Equation (23)) we know that $\dot{sc}^x \in \dot{c}^x \in \Gamma'_{ch}.C_{en}$ for some channel *ch*, $Inv_{\text{tree}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ implies

$$\exists e \in \mathcal{T} : e = (B, X)_{\vec{a}} \wedge \dot{sc}^x = h(e, id).$$

By construction of h there then is also some $e' \in \mathcal{T}$ with $h(e', id) = \dot{sc}^x$ and $h_{sec}(e', id) = \mathit{secret}_i(\dot{sc}^x)$ and $\mathit{sender}(sc^x) = \mathit{sender}(e) = \mathit{sender}(e') = B$. So by definition of $\varrho_{B,id}$ also $e' \in \varrho_{B,id}$.

Correspondingly, if $e \in \varrho_{B,id}$ then either $e \in \varrho'_{B,id}$ and (by $R' \sim \varrho'_{B,id}$) $h(e, id) \in \mathit{contracts}_{out}(R', id) \subseteq \mathit{contracts}_{out}(R, id)$. Or $h(e, id) = \dot{sc}^x$ and hence $h(e, id) \in \mathit{contracts}_{out}(R, id)$ by construction.

b) To show that $\varrho_{B,id}$ is consistent we assume towards contradiction that $\varrho_{B,id}$ is not consistent meaning that there exists a predecessor $e^* \in \mathit{onPathToRoot}(\mathcal{T}, e) \cap \widehat{\varrho}_{B,id}$ of the one $e \in \varrho_{B,id} \backslash \varrho'_{B,id}$ such that (1) $e^* \notin \varrho_{B,id}$ or (2) $\exists e^{**} \in \varrho'_{B,id} : spec(e^{**}) = spec(e)$. The situation for (1) is similar to the one we have covered for

$$\alpha = A : \mathit{revealSecret}_{ch}\, s^{id}_{\vec{a}}.$$

Therefore, the argumentation is analogous.

For (2), assume towards contradiction that there is $e^{**} \in \varrho'_{B,id}$ such that $spec(e^{**}) = spec(e)$. Since $spec(e^{**}) = spec(e)$ we know from Definition F.4 also that $\mathit{sender}(e) = \mathit{sender}(e^{**})$ and $\mathit{receiver}(e) = \mathit{receiver}(e^{**})$. Consequently, by construction, e^{**} and e are part of the exact same *CTLC*, see Equa-

tion (30). This c^x has a unique identifier, and according to Lemma H.10, cannot be enabled again after it has been claimed. Therefore,

$$\alpha = \textit{claim}(\dot{c}^x, h(e, id), h_{sec}(e, id))$$

is not possible, which is a contradiction.

c) Firstly, we note that for all $\Gamma_{ch} \in \vec{\Gamma}$ all sets are the same as they are in $\Gamma'_{ch} \in \vec{\Gamma}'$ except for one Γ_{ch}, where we have

$$\Gamma_{ch}.C_{adv} \subsetneq \Gamma'_{ch}.C_{adv} \ \wedge \ \Gamma_{ch}.C_{en} \subsetneq \Gamma'_{ch}.C_{en} \\ \wedge \ \Gamma_{ch}.C_{cla} \supsetneq \Gamma'_{ch}.C_{cla}.$$

$Inv(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is shown by going through the invariants individually.

$\overline{Inv_{\text{in-secrets}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))}$ immediately follows from $Inv_{\text{in-secrets}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ since $\varrho'_{B,id} \subseteq \varrho_{B,id}$ and $\Gamma_{ch}.S_{rev} = \Gamma'_{ch}.S_{rev}$.

$\overline{Inv_{\text{secrets}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))}$ is implied for all $e \in \varrho'_{B,id}$ by I.H. and for $e \in \varrho_{B,id} \backslash \varrho'_{B,id}$ we have $h_{sec}(e, id) \subseteq \Gamma_{channel(e)}.S_{rev}$ by definition of $\varrho_{B,id}$ for this case. The invariant is then implied by the definition of h_{sec} in Equation (28).

$\overline{Inv_{\text{in-schedule}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))}$ is implied directly from I.H. as $\vec{\Gamma}.S_{rev} = \vec{\Gamma}'.S_{rev}$ and $\textit{actions}(R') \subsetneq \textit{actions}(R)$.

$\overline{Inv_{\text{levels}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))}$ directly follows from the inductive hypothesis, because the *claim* rule (Equation (23)) only removes contracts from $\vec{\Gamma}.C_{en}$.

$\overline{Inv_{\text{liveness}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))}$ Is proven by first setting $e_{out} = (B, X)_{\vec{a}} \in \mathcal{T}$ and $j = \textit{depth}(e_{out})$ and $e_{in} = (Y, B)_{\vec{a}'}$ with $\vec{a}' = [(e_{in})] \cdot \vec{a}$. We distinguish the cases $h(e_{out}, id) \neq \dot{sc}^x$ and $h(e_{out}, id) = \dot{sc}^x$.

First consider $h(e_{out}, id) \neq \dot{sc}^x$. We know from

$$Inv_{\text{liveness}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$$

that either

$$\exists j' \leq j + 1, \vec{a}'' : (Y, B)_{\vec{a}''} \in \varrho'_{B,id} \wedge \textit{depth}((Y, B)_{\vec{a}''}) = j'$$

or

$$\exists c^x \in \Gamma'_{channel(e_{in})}.C_{en} : \\ h(e_{in}, id) \in c^x \wedge \vec{\Gamma}'.t < t_0 + (j + 1)\Delta.$$

In the first case, the claim immediately follows since $\varrho'_{B,id} \subseteq \varrho_{B,id}$ and $\textit{depth}(e_{in}) = 1 + \textit{depth}(e_{out})$. In the second case, we distinguish whether $h(e_{in}, id) \in \dot{c}^x$. If $h(e_{in}, id) \notin \dot{c}^x$ then we know that $\dot{c}^x \neq (c^x)'$ and hence $h(e_{in}, id) \in (c^x)'$ and $\vec{\Gamma}.t = \vec{\Gamma}'.t < t_0 + (j + 1)\Delta$. If $h(e_{in}, id) \in \dot{c}^x$ then we know that $\textit{sender}(\dot{sc}^x) = \textit{sender}(h(e_{in}, id)) = \textit{sender}(e_{in}) = Y$ and

$$\textit{receiver}(\dot{sc}^x) = \textit{receiver}(h(e_{in}, id)) = \textit{receiver}(e_{in}) = Y$$

(since they are subcontracts of the same CTLC). Further, by $Inv_{tree}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we know that there is some $e \in \mathcal{T}$ such that $h(e, id) = \dot{sc}^x$ (since by Equation (23) we know that $\dot{sc}^x \in \dot{c}^x \in \Gamma'_{channel(e_{in})}.C_{en}$). Consequently, $e = (Y, B)_{\vec{a}''}$ for some $\vec{a}''$ (since $\textit{sender}(e) = \textit{sender}(\dot{sc}^x) = Y$ and $\textit{receiver}(e) = \textit{receiver}(\dot{sc}^x) = B$). By definition of $\textit{contracts}_{in}(\cdot, id)$, $(\dot{sc}^x, h_{sec}(e, id)) \in \textit{contracts}_{in}(R', id)$ and so by $R' \sim \varrho'_{B,id}$, also $e \in \varrho'_{B,id} \subseteq \varrho_{B,id}$. So, we are left to show that $\textit{depth}(e) \leq j + 1$. Assume towards contradiction that $\textit{depth}(e) > j + 1$. By construction, we know that $\textit{timelock}(\dot{sc}^x) = t_0 + \textit{depth}(e)\Delta$, so in this case $\textit{timelock}(\dot{sc}^x) > t_0 + (j + 1)\Delta$. However, $\textit{timelock}(h(e_{in}, id)) = t_0 + \textit{depth}(e_{in})\Delta = t_0 + j\Delta$, so $\textit{timelock}(\dot{sc}^x) > \textit{timelock}(h(e_{in}, id))$. But then, by construction $\textit{position}(\dot{sc}^x) > \textit{position}(h(e_{in}, id))$. Since $h(e_{in}, id), \dot{sc}^x \in \dot{c}^x \in \Gamma'_{channel(e_{in})}.C_{en}$ we know by Lemma H.3 that $h(e_{in}, id), \dot{sc}^x \in \hat{c}^x \in \Gamma'_{channel(e_{in})}.C_{adv}$. By the inference rule of *claim* we know that if $\textit{position}(\dot{sc}^x) > 1$ then there is no $(sc^x)^* \in \hat{c}^x$ with $\textit{position}((sc^x)^*) < \textit{position}(\dot{sc}^x)$. This is contradicted by $h(e_{in}, id) \in \hat{c}^x$.

Next, we consider the case $h(e_{out}, id) = \dot{sc}^x$. Then we know from $Inv_{\text{setup}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ that either

$$\exists (c^x)' \in \Gamma'_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in (c^x)' \textbf{ or} \\ \exists j' \leq \textit{depth}(e_{in}) : (Y, B)_{\vec{a}''} \in \varrho'_{B,id} \wedge \textit{depth}((Y, B)_{\vec{a}''}) = j'.$$

In the second case, the claim immediately follows since $\varrho'_{B,id} \subseteq \varrho_{B,id}$ and $\textit{depth}(e_{in}) = 1 + \textit{depth}(e_{out})$. In the first case, we need to first show that $\exists (c^x)' \in \Gamma_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in (c^x)'$. This trivially holds since $\Gamma_{channel(e_{in})}.C_{en}$ is the same as $\Gamma'_{channel(e_{in})}.C_{en}$ with the only exception that $\dot{c}^x$ got removed. However, $\dot{c}^x$ could not have been $(c^x)'$ (the contract containing $h(e_{in}, id)$) since

$$\textit{sender}(h(e_{in}, id)) = \textit{sender}(e_{in}) \\ \neq \textit{receiver}(e_{in}) = B = \textit{sender}(\dot{c}^x)$$

but if it would hold that $h(e_{in}, id) \in \dot{c}^x$ then we would need to have that $\textit{sender}(h(e_{in}, id)) = \textit{sender}(\dot{c}^x)$. We are hence, left to show that $\vec{\Gamma}.t < t_0 + (j + 1)\Delta$. Assume towards contradiction that $\vec{\Gamma}.t \geq t_0 + (j + 1)\Delta$ then also $\vec{\Gamma}'.t = \vec{\Gamma}.t > t_0 + j\Delta$ and because $j = \textit{depth}(e_{out})$ consequently by $Inv_{\textit{levels}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we get that

$$\nexists ch, c^x \in \Gamma'_{ch}.C_{en} : h(e_{out}, id) \in c^x \wedge \textit{fund}(c^x) \in \Gamma'_{ch}.F_{res}$$

However, since $\dot{sc}^x = h(e_{out}, id)$ by inference rule of *claim* (Equation (23)) we know that $h(e_{out}, id) \in \dot{c}^x \in \Gamma'_{ch}.C_{en}$ for some channel *ch*. By $Inv_{\text{deposits}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we also have $\textit{fund}(\dot{c}^x) \in \Gamma'_{ch}.F_{res}$ which leads to a contradiction.

$\overline{Inv_{\text{deposits}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))}$ follows directly from I.H. as $\Gamma_{ch}.C_{en} \cup \Gamma_{ch}.C_{cla}$ only shrinks while $\Gamma_{ch}.F_{res}$ remains unchanged.

$\overline{Inv_{\text{setup}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))}$ is shown by setting $e_{out} = (B, X)_{\vec{a}} \in \mathcal{T}$ and $j = \textit{depth}(e_{out})$ and $e_{in} = (Y, B)_{\vec{a}'}$ with $\vec{a}' = [(e_{in})] \cdot \vec{a}$. Further, let either be

$$h(e_{out}, id) \in c^x \in \Gamma_{channel(e_{out})}.C_{en} \tag{66}$$

or

$$(B, \hat{c}^x) \in \Gamma_{channel(e_{out})}.C_{aut}, h(e_{out}, id) \in \hat{c}^x \quad (67)$$

If Equation (66) holds then also

$$h(e_{out}, id) \in c^x \in \Gamma'_{channel(e_{out})}.C_{en}$$

(since *claim* only removes CTLCs from $\Gamma'_{channel(e_{out})}.C_{en}$) and similarly if Equation (67) holds also

$$(B, \hat{c}^x) \in \Gamma'_{channel(e_{out})}.C_{aut}, h(e_{out}, id) \in \hat{c}^x.$$

Hence, in both cases we know by $Inv_{\text{setup}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ that either

$$\exists (c^x)' \in \Gamma'_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in (c^x)' \quad (68)$$

or

$$\exists j' \leq \mathit{depth}(e_{in}) : (Y, B)_{\vec{a}''} \in \varrho'_{B,id} \wedge \mathit{depth}((Y, B)_{\vec{a}''}) = j'. \quad (69)$$

If Equation (69) holds then the claim follows immediately since $\varrho'_{B,id} \subseteq \varrho_{B,id}$. If Equation (68) holds we distinguish whether $h(e_{in}, id) \in \dot{c}^x$. If $h(e_{in}, id) \notin \dot{c}^x$ then we know that $\dot{c}^x \neq (c^x)'$ and hence $h(e_{in}, id) \in (c^x)' \in \Gamma_{channel(e_{in})}.C_{en}$. If $h(e_{in}, id) \in \dot{c}^x$ then we know that $\mathit{sender}(\dot{sc}^x) = \mathit{sender}(h(e_{in}, id)) = \mathit{sender}(e_{in}) = Y$ and $\mathit{receiver}(\dot{sc}^x) = \mathit{receiver}(h(e_{in}, id)) = \mathit{receiver}(e_{in}) = Y$ (since they are subcontracts of the same CTLC). Further, by $Inv_{tree}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we know that there is some $e \in \mathcal{T}$ such that $h(e, id) = \dot{sc}^x$ and $\mathit{secret}_i(\dot{sc}^x) = h_{sec}(e, id)$ (since by Equation (23) we know that $\dot{sc}^x \in \dot{c}^x \in \Gamma'_{channel(e_{in})}.C_{en}$). Consequently, $e = (Y, B)_{\vec{a}''}$ for some $\vec{a}''$ (since $\mathit{sender}(e) = \mathit{sender}(\dot{sc}^x) = Y$ and $\mathit{receiver}(e) = \mathit{receiver}(\dot{sc}^x) = B$). By definition of $\mathit{contracts}_{in}(\cdot, id)$, $(\dot{sc}^x, h_{sec}(e, id)) \in \mathit{contracts}_{in}(R', id)$ and so by $R' \sim \varrho'_{B,id}$, also $e \in \varrho'_{B,id} \subseteq \varrho_{B,id}$. So, we are left to show that $\mathit{depth}(e) \leq j + 1$. Assume towards contradiction that $\mathit{depth}(e) > j + 1$. By construction, we know that $\mathit{timelock}(\dot{sc}^x) = t_0 + \mathit{depth}(e)\Delta$, so in this case $\mathit{timelock}(\dot{sc}^x) > t_0 + (j + 1)\Delta$. However, $\mathit{timelock}(h(e_{in}, id)) = t_0 + \mathit{depth}(e_{in})\Delta = t_0 + j\Delta$, so $\mathit{timelock}(\dot{sc}^x) > \mathit{timelock}(h(e_{in}, id))$. But then, by construction $\mathit{position}(\dot{sc}^x) > \mathit{position}(h(e_{in}, id))$. Since $h(e_{in}, id), \dot{sc}^x \in \dot{c}^x \in \Gamma'_{channel(e_{in})}.C_{en}$ we know by Lemma H.3 that $h(e_{in}, id), \dot{sc}^x \in \hat{c}^x \in \Gamma'_{channel(e_{in})}.C_{adv}$. By the inference rule of *claim* we know that if $\mathit{position}(\dot{sc}^x) > 1$ then there is no $(sc^x)^* \in \hat{c}^x$ with $\mathit{position}((sc^x)^*) < \mathit{position}(\dot{sc}^x)$. This is contradicted by $h(e_{in}, id) \in \hat{c}^x$.

$Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is not affected as the rule only removes elements from $\vec{\Gamma}'.C_{en}$ and only adds an element to set $\vec{\Gamma}.C_{cla}$, which previously was in $\vec{\Gamma}'.C_{en}$ and hence is guaranteed to satisfy the condition.

$Inv_{\text{init-liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds since we only need to consider the case of the newly added $e \in \varrho_{B,id} \setminus \varrho'_{B,id}$. In this case by construction of $\varrho_{B,id}$, we know that

$$\mathit{claim}(\dot{c}^x, h(e, id), h_{sec}(e, id)) \in \mathit{actions}(R).$$

$Inv_{\text{auth}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is implied by $Inv_{\text{init-liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ combined with the fact that

$$\exists j' \leq \mathit{depth}(e_{in}) : (Y, B)_{\vec{a}''} \in \varrho_{B,id} \wedge \mathit{depth}((Y, B)_{\vec{a}''}) = j')$$

from this invariant cannot happen since this implies that B revealed $\mathit{secret}((Y, B)_{\vec{a}''})$, see Inv_{tree}. This only happens if $t_0 < \vec{\Gamma}.t$ according to Equation (41) which would be a contradiction based on the reasoning we applied in Equation (64).

Case $\alpha = \mathit{withdraw}(c^x, sc^x)$:
We show that

a) $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$ and
b) $Inv(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ hold.

As $\{\varrho'_{B,id}\}_{\mathbb{T}}$ stays the same for R' and R we know by I.H. that all $\varrho'_{B,id}$ are consistent.
a) The relation $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$ also follows from I.H. given that we can show for all id

$$\mathit{contracts}_{out}(R, id) = \mathit{contracts}_{out}(R', id) \text{ and}$$
$$\mathit{contracts}_{in}(R, id) = \mathit{contracts}_{in}(R', id).$$

We look at the id with $\hat{c}^x \in \Psi^{id}$ specifically since this holds for all others by I.H.. By definition we have $\mathit{contracts}_{out}(R', id) \subseteq \mathit{contracts}_{out}(R, id)$. We show that also $\mathit{contracts}_{out}(R, id) \subseteq \mathit{contracts}_{out}(R', id)$ holds.

For $\vec{\Gamma}' = \mathit{lastEnv}(R')$ all Γ'_{ch} remain untouched except for one specific *ch*. Let this *ch* be fixed then the following statements are true:

$$\Gamma_{ch}.S_{rev} = \Gamma'_{ch}.S_{rev},\ \Gamma_{ch}.C_{en} = \Gamma'_{ch}.C_{en},\ \Gamma_{ch}.C_{adv} = \Gamma'_{ch}.C_{adv},$$
$$\Gamma_{ch}.C_{cla} \subsetneq \Gamma'_{ch}.C_{cla},\ \Gamma_{ch}.F_{res} \subsetneq \Gamma'_{ch}.F_{res},\ \Gamma_{ch}.t = \Gamma'_{ch}.t$$

Since

$$\mathit{contracts}_{out}(R, id)$$
$$= \mathit{contracts}_{out}(R', id) \cup \{(sc^x, \mathit{secret}_i(sc^x))\}$$

we only need to show that $(sc^x, \mathit{secret}_i(sc^x)) \in \mathit{contracts}_{out}(R', id)$. From the *withdraw* rule (24) combined with Lemma H.2 we know that

$$c^x = \{sc^x\} \in \vec{\Gamma}'.C_{cla}$$

and thus, $\mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R')$ is implied. Consequently, $sc^x \in \mathit{contracts}_{out}(R', id)$.

We next show that $\mathit{contracts}_{in}(R, id) = \mathit{contracts}_{in}(R', id)$. To this end, we first show that $\mathit{contracts}_{in}(R', id) \subseteq \mathit{contracts}_{in}(R, id)$. Assume towards contradiction that there is some

$$(\dot{sc}^x, \mathit{secret}_i(\dot{sc}^x)) \in \mathit{contracts}_{in}(R', id) \setminus \mathit{contracts}_{in}(R, id).$$

Then this can only be the case if $R' \xrightarrow{\alpha'} \vec{\Gamma}^*$ for some $\vec{\Gamma}^*$ and $\alpha' = \mathit{claim}(\dot{c}^x, \dot{sc}^x, \mathit{secret}_i(\dot{sc}^x))$ but there is no $\vec{\Gamma}^*$ such

that $R \xrightarrow{\alpha'} \vec{\Gamma}^*$. By the definition of the *withdraw* rule (Equation (24)) this could only be the case if $\mathit{fund}(\dot{c}^x) = \mathit{fund}(c^x)$ (so the action is not possible anymore because the required funds got removed from $\vec{\Gamma}'.F_{av}$). This is ruled out by the uniqueness of funds from Definition E.1.

We next show that $\mathit{contracts}_{in}(R, id) \subseteq \mathit{contracts}_{in}(R', id)$. Assume towards contradiction that there is some

$$(\dot{sc}^x, \mathit{secret}_i(\dot{sc}^x)) \in \mathit{contracts}_{in}(R, id) \backslash \mathit{contracts}_{in}(R', id).$$

This could only be the case if $R \xrightarrow{\alpha'} \vec{\Gamma}^*$ for some $\vec{\Gamma}^*$ and

$$\alpha' = \mathit{claim}(\dot{c}^x, \dot{sc}^x, \mathit{secret}_i(\dot{sc}^x))$$

but there is no $\vec{\Gamma}^*$ such that $R' \xrightarrow{\alpha'} \vec{\Gamma}^*$. However, this cannot be the case since the *withdraw* rule only removes elements from $\vec{\Gamma}'.C_{cla}$ and $\vec{\Gamma}'.F_{av}$. Note in particular

$$(sc^x, \mathit{secret}_i(sc^x)) \in \mathit{contracts}_{in}(R', id)$$

by Lemma H.5.

b) From the aforementioned changes in $\vec{\Gamma}$ compared to $\vec{\Gamma}'$ only Inv_{levels}, $Inv_{deposits}$ and Inv_{tree} are affected.

- $Inv_{\text{levels}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ holds since the preconditions have not changed and $\Gamma_{ch}.F_{res} \subseteq \Gamma'_{ch}.F_{res}$.
- $Inv_{\text{deposits}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is implied by the *withdraw* rule (24). Let $c^x \in \Gamma'_{ch}.C_{cla} \backslash \Gamma_{ch}.C_{cla}$ then

  $$\Gamma'_{ch}.F_{res} \backslash \Gamma_{ch}.F_{res} = \{\mathit{fund}(c^x)\}.$$

  Therefore, it still holds for all other contracts based on $Inv_{\text{deposits}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ and the uniqueness of funds (Definition E.1).
- $Inv_{\text{tree}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is also implied by I.H. because only $\vec{\Gamma}.C_{cla}$ got smaller. With $\vec{\Gamma}.C_{cla} \subseteq \vec{\Gamma}'.C_{cla}$ this invariant applies to fewer contracts now and is unchanged for the remaining. Therefore, it still holds.

Case $\alpha = \mathit{elapse}\,\delta$:

We show that

a) $R \sim \{\varrho'_{B,id}\}_{\mathbb{T}}$,

b) $Inv(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ hold.

As $\{\varrho'_{B,id}\}_{\mathbb{T}}$ stays the same for R' and R we know by I.H. that all $\varrho'_{B,id}$ are consistent.

a) For all id the equality

$$\mathit{contracts}_{out}(R, id) = \mathit{contracts}_{out}(R', id).$$

directly follows from its definition since α is not a *claim* action. To show

$$\mathit{contracts}_{in}(R, id) = \mathit{contracts}_{in}(R', id).$$

for all id we look at the rule for *claim* (23). Here, we see that it does not depend on $\vec{\Gamma}.t$. Hence, since none of the actions relevant for $\mathit{contracts}_{in}(R, id)$ are time-dependent, this set is not influenced by $\alpha = \mathit{elapse}\,\delta$.

b) The only invariants affected by $\vec{\Gamma}.t > \vec{\Gamma}'.t$ are Inv_{levels}, Inv_{liveness}, $Inv_{\text{init-liveness}}$, and Inv_{auth}. The following arguments apply to all $\varrho'_{B,id} \in \{\varrho'_{B,id}\}_{\mathbb{T}}$.

$Inv_{\text{levels}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$: By Definition F.2 of adversarial strategies, we know that an *elapse* δ action can only be executed if all honest users proposed an action *elapse* δ_i such that $\delta_i \geq \delta$. So, in particular, we know that $\Sigma_B^{\mathbb{T}}(R') = \mathit{elapse}\,\delta_B$ with $\delta_B \geq \delta$ from the honest user strategy (46). Now let $e \in \mathcal{T}$ with $B = \mathit{sender}(e)$ and

$$\vec{\Gamma}.t > t_0 + \mathit{depth}(e)\Delta \tag{70}$$

We show that $\nexists ch, c^x \in \Gamma_{ch}.C_{en} : h(e, id) \in c^x$ with $\mathit{fund}(c^x) \in \Gamma_{ch}.F_{res}$. We proceed by case distinction on $\vec{\Gamma}'.t > t_0 + \mathit{depth}(e)\Delta$.

$\vec{\Gamma}'.t > t_0 + \mathit{depth}(e)\Delta$: In this case, the claim immediately follows from $Inv_{\text{levels}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$, nothing has changed.

$\vec{\Gamma}'.t < t_0 + \mathit{depth}(e)\Delta$: In this case, by definition of $\Sigma_B^{\mathbb{T}}$, we know that $\delta_B = t_0 + j\Delta - \vec{\Gamma}'.t$ for the minimal j such that $t_0 + j\Delta - \vec{\Gamma}'.t > 0$ (since the honest user strategy makes time progress only to the next time step). If $j \leq \mathit{depth}(e)$, then $\vec{\Gamma}.t \leq t_0 + \mathit{depth}(e)\Delta$ since $\vec{\Gamma}.t = \vec{\Gamma}'.t + \delta \leq \vec{\Gamma}'.t + \delta_B = \vec{\Gamma}'.t + t_0 + j\Delta - \vec{\Gamma}'.t \leq t_0 + \mathit{depth}(e)\Delta$. With this, we have a contradiction to (70). If $j > \mathit{depth}(e)$ then $t_0 + \mathit{depth}(e)\Delta - \vec{\Gamma}'.t \leq 0$ and so $\vec{\Gamma}'.t \geq t_0 + \mathit{depth}(e)\Delta$ immediately contradicting the assumption of the case.

$\vec{\Gamma}'.t = t_0 + \mathit{depth}(e)\Delta$: In this case, we show that $\Sigma_B^{\mathbb{T}}$ would schedule an action different from *elapse* δ if

$$\exists ch, c^x \in \Gamma_{ch}.C_{en} : h(e, id) \in c^x$$

with $\mathit{fund}(c^x) \in \Gamma_{ch}.F_{res}$ holds. Assume towards contradiction that $\exists ch, c^x \in \Gamma_{ch}.C_{en} : h(e, id) \in c^x$ with $\mathit{fund}(c^x) \in \Gamma_{ch}.F_{res}$. By definition of $h(e, id)$ (29) we have

$$\mathit{timelock}(h(e, id)) = t_0 + \mathit{depth}(e)\Delta.$$

Consequently it holds that $\mathit{timelock}(h(e, id)) \leq \vec{\Gamma}'.t$. Based on Lemma H.1 we distinguish the cases whether $|c^x| = 1$ or $|c^x| > 1$.

If $|c^x| = 1$ by definition of $\widetilde{\Sigma}_B^e$ (Equation (38)), we would have that $\widetilde{\Sigma}_B^e(R') \ni \mathit{refund}\, c^x$ and so $\Sigma_B^{\mathbb{T}}(R') \neq \mathit{elapse}(\delta_B)$.

If $|c^x| > 1$, we show that also there is no $\dot{sc}^x \in c^x$ with

$$\mathit{position}(\dot{sc}^x) < \mathit{position}(h(e, id)).$$

If there would be such a contract, we would know by Inv_{tree} that there is some $e' \in \mathcal{T}$ such that $\dot{sc}^x = h(e', id)$. Further, we would know that

$$\mathit{sender}(e') = \mathit{sender}(\dot{sc}^x) = \mathit{sender}(e) = B.$$

By well-formedness of CTLCs (Equation (12)) we would further know that also $\mathit{timelock}(\dot{sc}^x) < \mathit{timelock}(h(e, id))$ and so consequently that $\mathit{depth}(e') < \mathit{depth}(e)$. So also $\vec{\Gamma}'.t > t_0 + \mathit{depth}(e')\Delta$ Applying $Inv_{\text{levels}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ would immediately contradict that $h(e', id) = \dot{sc}^x \in c^x \in \Gamma'_{ch}.C_{en} = \Gamma_{ch}.C_{en}$. Consequently, by the definition of $\widetilde{\Sigma}_B^{e'}$ (Equation (38)), we would have that $\widetilde{\Sigma}_B^{e'}(R') \ni \mathit{timeout}(c^x, h(e', id))$ and so $\Sigma_B^{\mathbb{T}}(R') \neq \mathit{elapse}(\delta_B)$.

$Inv_{\text{liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$:

Let

$e_{out} = (B, X)_{\vec{a}} \in \mathcal{T}, j = \mathit{depth}(e_{out})$,
$\mathit{claim}(c^x, h(e_{out}, id), h_{sec}(e_{out}, id)) \in \mathit{actions}(R)$ and
$e_{in} = (Y, B)_{\vec{a}'} \in \mathcal{T}$ with $\vec{a}' = [(Y, B)_{\vec{a}}] \cdot \vec{a}$.

According to the preconditions of $Inv_{\text{liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$. We show that either

$$(i)\ \exists j' \leq j+1, \vec{a}'' : (Y, B)_{\vec{a}''} \in \varrho'_{B,id} \wedge \mathit{depth}((Y, B)_{\vec{a}''}) = j' \textbf{ or}$$
$$(ii)\ \exists c^x \in \Gamma_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in c^x \wedge \vec{\Gamma}.t < t_0 + (j+1)\Delta.$$

From $Inv_{\text{liveness}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we know that either

$$\text{(a)}\ \exists j' \leq j+1, \vec{a}'' : (Y, B)_{\vec{a}''} \in \varrho'_{B,id} \wedge \mathit{depth}((Y, B)_{\vec{a}''}) = j' \text{ or}$$
$$\text{(b)}\ \exists c^x \in \Gamma'_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in c^x \wedge \vec{\Gamma}'.t < t_0 + (j+1)\Delta.$$

In the case of (a), (i) is implied immediately as they are the same. In the case of (b), we also have

$$\forall \Gamma_{ch} \in \vec{\Gamma} : \Gamma_{ch}.C_{en} = \Gamma'_{ch}.C_{en},$$

from the *elapse* δ rule (25). Thus (ii) holds whenever

$$\vec{\Gamma}.t < t_0 + (j+1)\Delta.$$

We now show that whenever $\vec{\Gamma}.t \geq t_0 + (j+1)\Delta$ statement (i) is implied, which concludes the proof. For this, we argue that $e_{in} \in \varrho'_{B,id}$ which fulfills (i) with $j' = j+1$ and $\vec{a}' = \vec{a}''$ except if $no - dupl(e_{in}, R)$ is not true. The case in which $no - dupl(e_{in}, R)$ is not true will be dealt with individually at the end of the proof.

To do so, due to $Inv_{\text{in-secrets}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$, it is sufficient to show that $\mathit{secret}(e_{in}) \in \vec{\Gamma}.S_{rev}$.

Assume towards contradiction that $\mathit{secret}(e_{in}) \notin \vec{\Gamma}.S_{rev}$. We will show that in this case $\Sigma^{\mathbb{T}}_B(R') \not\ni \mathit{elapse}(\delta_B)$, which would contradict that $\alpha = \mathit{elapse}(\delta)$. From (b) we know that

$$\exists c^x \in \Gamma'_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in c^x.$$

For this c^x we continue by case distinction based on the existence of another sub-contract with lower timeout, in other words we distinguish the cases in which $h(e_{in}, id)$ is the current top-contract in c^x and in which it is not.

Assume that there exists a subcontract $sc^x \in c^x$ with

$$\mathit{timelock}(sc^x) < \mathit{timelock}(h(e_{in}, id)). \tag{71}$$

From $Inv_{\text{tree}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we know $\exists e \in \mathcal{T}$ with $sc^x = h(e, id)$ and consequently $\mathit{depth}(e) =: j_e < j+1$ since we know by construction of h (29) that $\mathit{timelock}(sc^x) = t_0 + j_e\Delta$. Further, we have that $\vec{\Gamma}.t = \vec{\Gamma}'.t + \delta$ and $\delta \leq \Delta$ and $\vec{\Gamma}.t \geq t_0 + (j+1)\Delta$. So it holds that $\Gamma'.t = \Gamma.t - \delta \geq \Gamma.t - \Delta \geq t_0 + \Delta j \geq t_0 + j_e\Delta$. Consequently, we would have that $\widetilde{\Sigma}^{e_{out}}_B(R') \ni \mathit{timeout}(c^x, h(e_{out}, id))$ and so $\Sigma^{\mathbb{T}}_B(R') \not\ni \mathit{elapse}(\delta_B)$, see (38).

Assume that there exists no subcontract $sc^x \in c^x$ fulfilling Equation (71). Since $\mathit{claim}(h(e_{out}, id)) \in \mathit{actions}(R)$, we know that $\mathit{claim}(h(e_{out}, id)) \in \mathit{actions}(R')$ and hence also (since $R' \sim \varrho'_{B,id}$) that $e_{out} \in \varrho'_{B,id}$. From $Inv_{\text{secrets}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$, we know that for all $e' \in \mathit{onPathToRoot}(\mathcal{T}, e_{out})$ it holds

$$\mathit{secret}(e', id) \in \Gamma_{channel(e_{out})}.S_{rev}.$$

Therefore $chContract(e_{in}, R) = 2$ and if $no - dupl(e_{in}, R)$ is true all preconditions for $\widetilde{\Sigma}^{e_{in}}_B(R') = \mathit{revealSecret}_{ch}(\mathit{secret}(e_{in}, id))$ with $ch = channel(e_{in})$, see (41), are fulfilled. Hence $\Sigma^{\mathbb{T}}_B(R')$ outputs this action instead of $\mathit{elapse}(\delta_B)$.

If $no - dupl(e_{in}, R)$ is not true we have

$$\exists e'' \widehat{\in} \mathcal{T} : e'' \neq e_{in} \wedge \mathit{sender}(e'') = \mathit{sender}(e_{in}) \wedge \mathit{receiver}(e'') = \mathit{receiver}(e_{in}) \wedge \mathit{secret}(e'', id) \in \vec{\Gamma}.S_{rev}.$$

From $Inv_{\text{in-schedule}}(R', \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ we know then $e'' \in \varrho'_{B,id}$. Since $R' \sim \varrho'_{B,id}$, we know that either $\alpha' \in R'$ or $R' \xrightarrow{\alpha'} \vec{\Gamma}^*$ for some $\vec{\Gamma}^*$ for $\alpha' = \mathit{claim}(\dot{c}^x, h(e'', id), h_{sec}(e'', id))$. If $\alpha' \in R'$ then we know that there was some prefix R^* of R' such that $R^* \xrightarrow{\alpha'} \vec{\Gamma}^\dagger$ for some $\vec{\Gamma}^\dagger$ and hence by the inference rule of *claim* (Equation (23)), also $h(e'', id) \in \dot{c}^x$. Since c^x has the same identifier as $\dot{c}^x$, it cannot be enabled after α' due to Lemma H.10. So we are left to consider the case that $R' \xrightarrow{\alpha'} \vec{\Gamma}^*$. In this case by the inference rule of *claim* (Equation (23)), we know that all preconditions for the execution of $h(e'', id)$ are satisfied. In addition, using $Inv_{in-schedule}$ for $\mathit{secret}(e'', id) \in \vec{\Gamma}.S_{rev} = \vec{\Gamma}'.S_{rev}$, we can conclude that the corresponding outgoing edges of e'' have been claimed in R'. We further can show that $\vec{\Gamma}'.t < \mathit{timelock}(h(e'', id))$ because by $Inv_{\textit{init-liveness}}$ we know that either $\alpha' \in R'$ (leading to a contradiction as shown above) or $\vec{\Gamma}'.t < \mathit{timelock}(h(e'', id))$. Therefore, $chContract(e'', R) = 2$ and all preconditions for $\widetilde{\Sigma}^{e''}_B(R') = \alpha'$, see (41), are fulfilled. Hence, $\Sigma^{\mathbb{T}}_B(R')$ outputs this α' instead of $\mathit{elapse}(\delta_B)$.

$Inv_{\text{init-liveness}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$:

Let $e \in \varrho'_{B,id}$. From the I.H. we know that either

$$\exists c^x : \mathit{claim}(c^x, h(e, id), h_{sec}(e, id)) \in \mathit{actions}(R') \text{ or} \tag{72}$$
$$\vec{\Gamma}'.t < \mathit{timeout}(h(e, id)). \tag{73}$$

If Equation (72) holds, the claim immediately follows because if $\mathit{claim}(c^x, h(e, id), h_{sec}(e, id)) \in \mathit{actions}(R')$ also

$$\mathit{claim}(c^x, h(e, id), h_{sec}(e, id)) \in \mathit{actions}(R)$$

(since $\mathit{actions}(R') \subseteq \mathit{actions}(R)$). Assume that Equation (73) holds. By I.H. we also know that $R' \sim \varrho'_{B,id}$. Consequently,

$$(h(e, id), h_{sec}(e, id)) \in \mathit{contracts}_{out}(R', id) \cup \mathit{contracts}_{in}(R', id).$$

We distinguish the cases

$$(h(e,id), h_{sec}(e,id)) \in \textit{contracts}_{out}(R', id) \text{ and}$$
$$(h(e,id), h_{sec}(e,id)) \in \textit{contracts}_{in}(R', id).$$

If $(h(e,id), h_{sec}(e,id)) \in \textit{contracts}_{out}(R', id)$ then by Definition (49)

$$\textit{claim}(c^x, h(e,id), h_{sec}(e,id)) \in \textit{actions}(R').$$

Next, we consider the case that

$$(h(e,id), h_{sec}(e,id)) \in \textit{contracts}_{in}(R', id).$$

By the definition of $\textit{contracts}_{in}(R', id)$ we hence know that either $\textit{claim}(c^x, h(e,id), h_{sec}(e,id)) \in \textit{actions}(R')$ or $R' \xrightarrow{\alpha'} \vec{\Gamma}^*$ for some $\vec{\Gamma}^*$, $\alpha' = \textit{claim}(c^x, h(e,id), h_{sec}(e,id))$ and some c^x. For the first case, it again follows from $\textit{actions}(R') \subseteq \textit{actions}(R)$.

In the second case, we know that all preconditions for executing α' are met. We show that in this case, the honest user strategy would schedule α' instead of *elapse* δ. Using $Inv_{in-schedule}$ for $\textit{secret}(e,id) \in h_{sec}(e,id) \subseteq \vec{\Gamma}'.S_{rev}$, we can conclude that the corresponding outgoing edges of e have been claimed in R'. Therefore, $chContract(e'', R) = 2$ and all preconditions for $\widetilde{\Sigma}^e_B(R') = \alpha'$, see (41), are fulfilled. Hence, $\Sigma^{\mathbb{T}}_B(R')$ outputs this α' instead of $\textit{elapse}(\delta_B)$.

$Inv_{\text{auth}}(R, \varrho'_{B,id}, (id, \mathcal{T}, t_0, spec))$ is implied by I.H. combined with the reasoning we applied in Equation (64).

Protocol Security.

Theorem H.16 (Protocol Security)**.** *Let B be an honest user, $\mathbb{T}$ be a set of tuples of the form $(id, \mathcal{T}, t_0, spec)$, which is well-formed, and $\Sigma^{\mathbb{T}}_B$ the honest user strategy for B executing $\mathbb{T}$. Let $\Sigma^{\mathbb{T}}_{Adv}$ be an arbitrary adversarial strategy. Then for all final runs R with $\Sigma^{\mathbb{T}}_B, \Sigma^{\mathbb{T}}_{Adv} \vdash R$, starting from an initial environment, with $\vec{\Gamma} := \textit{lastEnv}(R)$, and for all $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ there exists $\widetilde{\omega}_{id} \in \mathcal{O}^{\mathcal{T}}_B \cup \{\emptyset\}$ s.t. for their family*

$$\{\widetilde{\omega}_{id}\}_{\mathbb{T}} := \{\widetilde{\omega}_{id} \mid (id, \mathcal{T}, t_0, spec) \in \mathbb{T}\}$$

it holds

$$\begin{aligned}&\forall id, e \in \widetilde{\omega}_{id} : e \in \widehat{\varrho}_{B,id}\\ &\Rightarrow \exists c^x : \textit{claim}(c^x, h(e,id), h_{sec}(e,id)) \in \textit{actions}(R)\end{aligned}$$

and

$$\begin{aligned}&\forall \textit{claim}(c^x, sc^x, \textit{secret}_i(sc^x)) \in \textit{actions}(R) :\\ &B \in \textit{users}(c^x) \Rightarrow \exists id, e \in \widetilde{\omega}_{id} : sc^x = h(e,id)\\ &\qquad\qquad \wedge \textit{secret}_i(sc^x) = h_{sec}(e,id).\end{aligned}$$

Proof: By Theorem H.14, for every $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ it exists a consistent $\varrho_{B,id} \subseteq \mathcal{T}$ s.t. for their family $\{\varrho_{B,id}\}_{\mathbb{T}}$ we have $R \sim \{\varrho_{B,id}\}_{\mathbb{T}}$. By Equation (47) we then have

$$\begin{aligned}&\forall \textit{claim}(c^x, sc^x, \textit{secret}_i(sc^x)) \in \textit{actions}(R) : \\ &\exists \varrho_{B,id}, e \in \varrho_{B,id} :\\ &sc^x = h(e,id) \wedge \textit{secret}_i(sc^x) = h_{sec}(e,id).\end{aligned} \tag{74}$$

Since R is final, the $\varrho_{B,id}$ stay constant for all extensions of R. Let $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ be given. For this, we now show

$$\exists \widetilde{\omega}_{id} \in \mathcal{O}^{\mathcal{T}}_B \cup \{\emptyset\} : \widehat{\varrho}_{B,id} \cap \widetilde{\omega}_{id} = \varrho_{B,id}.$$

For this, we construct

$$\widetilde{\omega}_{id} := \{e \in \mathcal{T} \mid \exists e' \in \varrho_{B,id} : e \in \textit{onPathToRoot}(\mathcal{T}, e')\},$$

which, since $\varrho_{B,id}$ is consistent, fulfills $\widehat{\varrho}_{B,id} \cap \widetilde{\omega}_{id} = \varrho_{B,id}$. Therefore, it is left to show that $\widetilde{\omega}_{id} \in \mathcal{O}^{\mathcal{T}}_B \cup \{\emptyset\}$ holds. In case $\varrho_{B,id} = \emptyset$ we also get $\widetilde{\omega}_{id} = \emptyset$ by the above definition.

In case $\varrho_{B,id} \neq \emptyset$ it follows immediately $\widetilde{\omega}_{id} \neq \emptyset$. Hence, it is left to show that if $\widetilde{\omega}_{id} \neq \emptyset$ we have $\widetilde{\omega}_{id} \in \mathcal{O}^{\mathcal{T}}_B$. We recall from Definition D.3:

$$\begin{aligned}\mathcal{O}^{\mathcal{T}}_B := \{\omega \in \mathcal{O}_{\textit{full}}(\mathcal{T}) \mid\ & \textit{NoDup}(\mathcal{T}, B, \omega)\\ \wedge\ & \textit{HonestRoot}(\mathcal{T}, B, \omega)\\ \wedge\ & \textit{EagerPull}(\mathcal{T}, B, \omega)\}\end{aligned}$$

$\textit{NoDup}(\mathcal{T}, B, \widetilde{\omega}_{id})$ follows from $\widehat{\varrho}_{B,id} \cap \widetilde{\omega}_{id} = \varrho_{B,id}$ and the fact that $\varrho_{B,id}$ is consistent and hence does not contain any duplicate edges (so since all edges involving B in $\widetilde{\omega}_{id}$ are also contained in $\varrho_{B,id}$ also $\widetilde{\omega}_{id}$ cannot contain any duplicate edges involving B).

For $\textit{HonestRoot}(\mathcal{T}, B, \widetilde{\omega}_{id})$, we look at the situation where B sits in the root of $\mathcal{T}$. Assume that there is an edge $e = (X, B)_{[X,B]} \widehat{\in} \mathcal{T}$. We show that then also $e \in \widetilde{\omega}_{id}$. Since $\varrho_{B,id}$ is consistent and not empty there also needs to be an $e^* := (Y, B)_{[(Y,B)]} \in \mathcal{T} \cap \varrho_{B,id}$ for some user Y. The edges e and e^* could coincide but don't need to. Assume towards contradiction that $e \notin \widetilde{\omega}_{id}$.

Since $e^* \in \varrho_{B,id}$ by

$$Inv_{\text{secrets}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$$

we know that

$$s = \textit{secret}(e^*, id) \in \Gamma_{channel(e^*)}.S_{rev}.$$

Then Lemma H.9 implies

$$(B : \textit{revealSecret}_{ch}\, s) \in \textit{actions}(R)$$

with $ch = channel(e)$. Therefore it exist $R^\dagger$ and R^* with

$$R = R^\dagger \xrightarrow{B:\textit{revealSecret}_{ch}\, s} R^*$$

and so $(B : \textit{revealSecret}_{ch}\, s) \in \Sigma^{\mathbb{T}}_B(R^\dagger)$. By definition of $\Sigma^{\mathbb{T}}_B$, see Equation (41), we know that $chContract(e^*, R^\dagger) = 2$ holds in this case. We show that $chContract(e, R^\dagger) = 2$ also holds. The condition $secretsAv(e)$ follows immediately from the fact that $h_{sec}(e,id)$ has only one secret owned by B. The condition $isIngoing(e, R^\dagger)$ follows immediately from

$isIngoing(e^*, R^\dagger)$ and since

$$\textit{timelock}(h(e,id)) = \textit{timelock}(h(e^*,id))$$

the *timelock* condition is also implied. Thus it is left to show that $enabled(e)$ holds. Because

$$\textit{depth}(e) = \textit{depth}(e^*) = 1$$

and it is dictated by the definition of h (29) that no smaller *timelock* is possible, we can imply that there can be no subcontract with a smaller timelock than the one of $h(e,id)$ in the same c^x.

Further, from $isIngoing(e^*, R^\dagger)$ we know that either $h(e,id)$ must be enabled (in $h(e,id) \in c^x \in \textit{lastEnv}(R^\dagger).C_{en}$) or $\textit{claim}(c^x, h(e,id), h_{sec}(e,id)) \in \textit{actions}(R^\dagger)$. In the second case, we immediately know from $R \sim \varrho_{B,id}$ that $e \in \varrho_{B,id}$ and hence $e \in \widetilde{\omega}_{id}$, which would contradict the assumption. If $h(e,id) \in c^x \in \textit{lastEnv}(R^\dagger).C_{en}$ is enabled then $chContract(e, R^\dagger) = 2$ holds.

We next show that $no{-}dupl(e, R^\dagger)$. For this, we assume towards contradiction that $no - dupl(e, R^\dagger)$ does not hold. This means

$$\begin{aligned}\exists e' \,\widehat{\in}\, \mathcal{T} : e' \neq e \wedge \textit{sender}(e') = \textit{sender}(e)\\ \wedge \textit{receiver}(e') = \textit{receiver}(e)\\ \wedge s' := \textit{secret}(e', id) \in \vec{\Gamma}^\dagger.S_{rev}\end{aligned}$$

where $\vec{\Gamma}^\dagger = \textit{lastEnv}(R^\dagger)$.

By Lemma H.9 we then have

$$R^\dagger = \dot{R} \overset{B:\textit{revealSecret}_{ch}\, s'}{\longrightarrow} \widetilde{R} \tag{75}$$

for some $\dot{R}$ and $\widetilde{R}$. From Equation (39) of the honest user strategy we know $\exists c^x : h(e,id) \in c^x \in \vec{\Gamma}^\dagger.C_{en}$. Hence, if there exists some $\dot{c}^x$ (so a contract with the same id as c^x) with $h(e',id) \in \dot{c}^x \in \textit{lastEnv}(\dot{R}).C_{en}$, which needs to be the case for the action in (75), we also have

$$h(e,id) \in \dot{c}^x \in \textit{lastEnv}(\dot{R}).C_{en},$$

by Lemma H.11 because

$$\textit{position}(h(e,id)) > \textit{position}(h(e',id))$$

by construction of h. This contradicts

$$(B : \textit{revealSecret}_{ch}\, s') \in \Sigma_B^{\mathbb{T}}(\dot{R})$$

because $\Sigma_B^{\mathbb{T}}$ would not schedule $B : \textit{revealSecret}_{ch}\, s'$ if $h(e',id)$ is not the top-level contract in $\dot{c}^x$ and hence cannot be executed.

Consequently, $no - dupl(e, R^\dagger)$ holds. And so that $\Sigma_B^{\mathbb{T}}$ would schedule $B : \textit{revealSecret}_{ch}\, s^*$ on $R^\dagger$ given that s^* has not yet been revealed.

More formally, for $ch = channel(e)$ we have that either

$$\begin{aligned}&(a)\ s^* = \textit{secret}(e,id) \in \Gamma^\dagger_{ch}.S_{rev} \textbf{ or}\\ &(b)\ (B : \textit{revealSecret}_{ch}\, s^*) \in \Sigma_B^{\mathbb{T}}(R^\dagger).\end{aligned}$$

In case (a) from $Inv_{\text{in-schedule}}$ we get immediately that $e \in \varrho_{B,id}$ (since also $s^* \in \Gamma_{ch}.S_{rev}$ because the secrets sets monotonically increase, see Lemma H.8), which would contradict our assumption $e \notin \widetilde{\omega}_{id}$.

In case (b) we argue why during the run R^* an environment must be reached where $\textit{revealSecret}_{ch}\, s^*$ is finally executed and hence we can use $Inv_{\text{in-schedule}}$ again to show that $e \in \varrho_{B,id}$ (and thus arrive at a contradiction). To this end, we show that there must be R' and R'' with

$$R^* = R' \overset{\textit{elapse}\,\delta}{\longrightarrow} R''$$

and $s^* \in \Gamma'_{ch}.S_{rev}$. Consequently

$$\textit{secret}(e,id) \in \Gamma''_{ch}.S_{rev} = \Gamma_{ch}.S_{rev}.$$

Thus $e \in \varrho_{B,id}$. Clearly, such R' and R'' must exist since $\vec{\Gamma}''.t > \vec{\Gamma}^\dagger.t$ (because $\vec{\Gamma}^\dagger.t \leq \textit{timelock}(h(e,id))$) and elapse is the only rule to increase time. Assume towards contradiction that $s^* \notin \Gamma'_{ch}.S_{rev}$. Then

$$(B : \textit{revealSecret}_{ch}\, s^*) \in \Sigma_B^{\mathbb{T}}(R')$$

by the persistence of the honest user strategy (46) and the fact that for any $\widetilde{\vec{\Gamma}}$ and any channel *ch* the set $\widetilde{\Gamma}_{ch}.S_{rev}$ can only increase in size (if there would have been a point in R' where $B : \textit{revealSecret}_{ch}\, s^*$ was not possible anymore this would imply starting from this point, s^* would reside in the S_{rev} set, contradicting $s^* \notin \Gamma'_{ch}.S_{rev}$).

However, in order to execute the $\textit{elapse}\,\delta$ action, we would need to have

$$\{\textit{elapse}\,\delta'\} = \Sigma_B^{\mathbb{T}}(R')$$

for some δ' (since $\Sigma_B^{\mathbb{T}}$ only schedules an *elapse* action if no other action is scheduled). This gives us the final contradiction.

For $\textit{EagerPull}(\mathcal{T}, B, \widetilde{\omega}_{id})$ let

$$(X,B)_{\vec{a}_1} \,\widehat{\in}\, \mathcal{T} \ \wedge\ (B,Y)_{\vec{a}_2} \in \widetilde{\omega}_{id} \ \wedge\ \vec{a}_1 = [(X,B)] \cdot \vec{a}_2$$

be given. We set $j := \textit{depth}((B,Y)_{\vec{a}_2})$. From $(B,Y)_{\vec{a}_2} \in \widetilde{\omega}_{id}$, we know that also $(B,Y)_{\vec{a}_2} \in \varrho_{B,id}$ and hence invariant $Inv_{\text{liveness}}(R, \varrho_{B,id}, (id, \mathcal{T}, t_0, spec))$ applies. Thus either

$$\begin{aligned}&1)\ \exists j' \leq |\vec{a}_1|, \vec{a}_3 : (X,B)_{\vec{a}_3} \in \varrho_{B,id} \wedge |\vec{a}_3| = j' \textbf{ or}\\ &2)\ \exists c^x \in \Gamma_{channel((X,B)_{\vec{a}_1})}.C_{en} : h((X,B)_{\vec{a}_1}, id) \in c^x\\ &\quad \wedge \vec{\Gamma}.t < t_0 + (j+1)\Delta\end{aligned}$$

In the first case, the claim holds immediately, since by construction $(X,B)_{\vec{a}_3} \in \varrho_{B,id}$ implies $(X,B)_{\vec{a}_3} \in \widetilde{\omega}_{id}$. In the second case, we have a contradiction since we know that R is final and so

$$\vec{\Gamma}.t > t_0 + (j+1)\Delta = \textit{timelock}((X,B)_{\vec{a}_1}).$$

With that, we showed that $\widetilde{\omega}_{id} \in \mathcal{O}_B^{\mathcal{T}}$ for $\widetilde{\omega}_{id} \neq \emptyset$. To show the first implication of the final statement, we still need to show that if $e \in \widetilde{\omega}_{id} \cap \widehat{\varrho}_{B,id}$ then also $\textit{claim}(c^x, h(e,id), h_{sec}(e,id)) \in \textit{actions}(R)$. If $e \in \widetilde{\omega}_{id} \cap \widehat{\varrho}_{B,id}$ then $e \in \varrho_{B,id}$ (by construction) and hence from $\textit{Inv}_{\textit{init-liveness}}$, we know that either $\textit{claim}(c^x, h(e,id), h_{sec}(e,id)) \in \textit{actions}(R)$ or $\vec{\Gamma}.t <$

timelock$(h(e,id))$ In the first case, the claim follows immediately. In the second case, we immediately arrive at a contradiction to R being final.

The second implication of the final statement immediately follows from (74) and $\varrho_{B,id} \subseteq \widetilde{\omega}_{id}$. This concludes the proof.

End-to-end Security. When $\mathcal{T} = \mathit{unfold}(\mathcal{D}, A)$ is given by the unfolding process of a digraph $\mathcal{D}$, which is in-semiconnected w.r.t. $A \in \mathcal{N}$, the execution of the protocol never leaves an honest party *underwater* in $\mathcal{D}$. We use the same notion of *underwater* as in Theorem D.4. To formally state our end-to-end security statement, we assume to be provided a set $\mathbb{D}$ of graph specifications of the form $(id, A, \mathcal{D}, t_0, \mathit{gspec}_{\mathcal{D}})$ where id denotes a unique identifier, $\mathcal{D}$ the graph to be executed, t_0 the execution starting time and $\mathit{gspec}_{\mathcal{D}}$ is a graph specification that maps arcs to a pair $(f_X^{\zeta}, \mathit{ch})$ of funds and channels.

We will denote with $\mathit{spec}_{\mathcal{D}}$ the tree specification induced by the graph specification $\mathit{gspec}_{\mathcal{D}}$, formally defined as

$$\mathit{spec}_{\mathcal{D}}(e) := \mathit{gspec}_{\mathcal{D}}(\mathit{sender}(e), \mathit{receiver}(e))$$

Note that $\mathit{spec}_{\mathcal{D}}$ is by construction a valid tree specification.

We now formally state our end-to-end security statement:

Theorem H.17 (End to End Security)**.** *Let B be an honest user, $\mathbb{D}$ be a set of tuples of the form $(id, A, \mathcal{D}, t_0, \mathit{gspec}_{\mathcal{D}})$ with digraph $\mathcal{D}$ in-semiconnected w.r.t. $A \in \mathcal{N}$. Be $\mathbb{T}$ well-formed, see Definition F.6, and given as*

$$\mathbb{T} := \{(id, \mathcal{T}, t_0, \mathit{spec}) \mid (id, A, \mathcal{D}, t_0, \mathit{gspec}_{\mathcal{D}}) \in \mathbb{D} \ \wedge \ \mathit{spec} = \mathit{spec}_{\mathcal{D}} \ \wedge \ \mathcal{T} = \mathit{unfold}(\mathcal{D}, A)\}$$

Let $\Sigma_B^{\mathbb{T}}$ be the honest user strategy for B executing $\mathbb{T}$. Let $\Sigma_{Adv}^{\mathbb{T}}$ be an arbitrary adversarial strategy. Then, for all final runs R with $\Sigma_B^{\mathbb{T}}, \Sigma_{Adv}^{\mathbb{T}} \vdash R$, starting from an initial environment it holds:

$$\begin{aligned}
&\forall \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R), X \in \mathcal{N} : \\
&\quad \mathit{sender}(c^x) = B \ \wedge \ \mathit{receiver}(c^x) = X \\
&\quad \Rightarrow \exists (id, A, \mathcal{D}, t_0, \mathit{gspec}_{\mathcal{D}}) \in \mathbb{D} : \ (B, X) \in \mathcal{D} \\
&\qquad \wedge\, x = (id, B, X) \\
&\qquad \wedge\, \forall (Y, B) \in \mathcal{D} \, \exists \dot{c}^{x'}, \dot{sc}^{x'}, j : x' = (id, Y, B) \\
&\qquad \wedge\, \mathit{claim}(\dot{c}^{x'}, \dot{sc}^{x'}, \mathit{secret}_j(\dot{sc}^{x'})) \in \mathit{actions}(R)
\end{aligned}$$

Intuitively, every claim action taking money away from B corresponds to an outgoing arc in $\mathcal{D}$ for which all ingoing arcs were claimed.

Proof: Let $\mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R)$ with $\mathit{sender}(c^x) = B$ and $\mathit{receiver}(c^x) = X$, for some user X. From Theorem H.16 we have that for every $(id, \mathcal{T}, t_0, \mathit{spec}) \in \mathbb{T}$ there is some $\widetilde{\omega}_{id} \in \mathcal{O}_B^{\mathcal{T}} \cup \{\emptyset\}$ such that

$$\begin{aligned}
&\forall e \in \widetilde{\omega}_{id} : e \in \widehat{\varrho}_{B,id} \qquad (76)\\
&\qquad \Rightarrow \exists c^x : \mathit{claim}(c^x, h(e, id), h_{sec}(e, id)) \in \mathit{actions}(R)
\end{aligned}$$

and

$$\begin{aligned}
&\forall \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R) : B \in \mathit{users}(c^x) \qquad (77)\\
&\Rightarrow \exists id, e \in \widetilde{\omega}_{id} : sc^x = h(e, id) \wedge \mathit{secret}_i(sc^x) = h_{sec}(e, id).
\end{aligned}$$

So consequently, (77) immediately gives us for $(id, \mathcal{T}, t_0, \mathit{spec}) \in \mathbb{T}$ that there is some $e \in \widetilde{\omega}_{id}$ which fulfills

$$sc^x = h(e, id) \ \wedge \ \mathit{secret}_i(sc^x) = h_{sec}(e, id) \qquad (78)$$

By definition of h, see Equation (29), $e = (B, X)_{\vec{a}}$ for some walk $\vec{a}$ and $x = (id, B, X)$. Since $e \,\widehat{\in}\, \mathcal{T}$ we have $(B, X) \in \mathcal{D}$ for

$$(id, A, \mathcal{D}, t_0, \mathit{gspec}_{\mathcal{D}}) \in \mathbb{D}.$$

Let now $(Y, B) \in \mathcal{D}$. Then Theorem D.4 immediately gives us that there is also some $\vec{a}'$ such that $e' = (Y, B)_{\vec{a}'} \in \widetilde{\omega}_{id}$. From (76) we then also have that $\mathit{claim}(\dot{c}^{x'}, h(e', id), h_{sec}(e', id)) \in \mathit{actions}(R)$ for some $\dot{c}^{x'}$ and so by definition of h, we know that $x' = (id, Y, B)$ and $h_{sec}(e', id) = \mathit{secret}_{\iota}(h(e', id))$ for some ι. This concludes the proof.

Remark H.18. *If we assume the preconditions from Theorem H.17 and*

$$\alpha = \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R)$$

with $\mathit{sender}(c^x) = B$ and $\mathit{receiver}(c^x) = X$ then there exist R', R'' s.t.

$$R = R' \xrightarrow{\alpha} R''.$$

By applying Lemma H.10 we get that there are no $\dot{c}^x$, $\dot{sc}^x$, ι for which

$$\alpha' \in \mathit{actions}(R') \vee \alpha' \in \mathit{actions}(R'')$$

with $\alpha' = \mathit{claim}(\dot{c}^x, \dot{sc}^x, \mathit{secret}_{\iota}(\dot{sc}^x))$ holds. This means that a CTLC (with sender B) with the same identifier can never be claimed twice during a run. By construction of h, contracts with different sender or receiver get assigned different identifiers. Therefore, in Theorem H.17 two different claim actions in $\mathit{actions}(R)$ can never be linked to the same arc in $\mathcal{D}$. This rules out that there could be several claim actions that are mapped to the same arc (B, X) in $\mathcal{D}$.

Protocol Correctness. Next, we will prove the correctness of the tree protocol. Intuitively, the protocol for the tree $\mathcal{T}$ is correct if, given that all users of the tree $\mathcal{T}$ are honest, the protocol computes edges corresponding to the 'optimal' tree that lies in the intersection of the outcome sets of all users.

To ensure protocol correctness, some additional prerequisites need to be satisfied (beyond the users being honest)

- The protocol setup phase needs to be started in time. The time t_0 is part of the protocol specification but denotes the time starting from when the execution of the tree can start. However, to ensure that all tree edges are set up and enabled by t_0, the setup process needs

to start before $\Delta * \mathit{depth}(\mathcal{T})$ since setting up each level of the tree may take up to time Δ.
- The funds of users (according to *spec*) needs to be available in the initial environment.

We further define the user set of a tree set as follows:

$$\mathit{users}(\mathbb{T}) := \bigcup_{e \in \{e \mid \exists (id,\mathcal{T},t_0,spec) \in \mathbb{T} : e \mathrel{\widehat{\in}} \mathcal{T}\}} \mathit{users}(e)$$

Intuitively, liquidity states that for all edges (without duplicates) of all trees (identified by $(\mathit{sender}(e), \mathit{receiver}(e), id)$) there is a unique fund as specified in the spec *spec*.

To prove protocol correctness, we first show that the setup process is correct, meaning that if the protocol is started on time, all tree edges will be enabled by the time the protocol's starting time t_0 is reached.

Theorem H.19 (Setup Correctness)**.** *Let Hon* $= \{B_1, \ldots, B_k\}$ *be a set of honest users. Let* $\mathbb{T}$ *be well-formed, see Definition F.6, and users*$(\mathbb{T}) \subseteq$ *Hon. Let* $\Sigma^{\mathbb{T}}_{Hon} = \{\Sigma^{\mathbb{T}}_{B_1}, \ldots, \Sigma^{\mathbb{T}}_{B_k}\}$ *a set of honest user strategies executing* $\mathbb{T}$*. Let* $\Sigma^{\mathbb{T}}_{Adv}$ *be an arbitrary adversarial strategy (for Hon). Let* R *with* $(\Sigma^{\mathbb{T}}_{Hon}, \Sigma^{\mathbb{T}}_{Adv}) \vdash R$ *be a final run starting from a liquid initial environment* $\vec{\Gamma}_0$ *for* $\mathbb{T}$*, see Definition F.5. Further, let* $\vec{\Gamma} = \mathit{lastEnv}(R)$*. We recall from Equation* (33)

$$\mathit{depth}(\mathcal{T}) = max\{\mathit{depth}(e) \mid e \mathrel{\widehat{\in}} \mathcal{T}\}.$$

Then if the execution started on time, meaning

$$t_0 - \mathit{depth}(\mathcal{T})\Delta > \vec{\Gamma}_0.t$$

we have for all $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ *and* $m \in \mathbb{N}$ *with*

$$\mathit{depth}(\mathcal{T}) \geq m \wedge t_0 > \vec{\Gamma}.t \geq t_0 - (\mathit{depth}(\mathcal{T}) - m)\Delta$$

that the following implication holds for all $e \mathrel{\widehat{\in}} \mathcal{T}$*:*

$$\begin{aligned} &\mathit{depth}(e) + m \geq \mathit{depth}(\mathcal{T}) \\ &\Rightarrow \exists c^x \in \Gamma_{channel(e)}.C_{en} : h(e, id) \in c^x \end{aligned}$$

Proof: Let *Hon*, $\mathbb{T}$, Σ_{Hon} and R as stated in the Theorem with $|R| = n$. The Theorem is proven by induction on n.

Case $n = 0$**:**

Let $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ be given. By definition of R and $|R| = 0$ we have $\vec{\Gamma} = \mathit{lastEnv}(R) = \vec{\Gamma}_0$. Let

$$t_0 > \vec{\Gamma}.t = \vec{\Gamma}_0.t \geq t_0 - (\mathit{depth}(\mathcal{T}) - m)\Delta \tag{79}$$

$$\wedge\, t_0 - \mathit{depth}(\mathcal{T})\Delta > \vec{\Gamma}_0.t \tag{80}$$

according to the preconditions. Combining these inequalities yields:

$$t_0 - (\mathit{depth}(\mathcal{T}) - m)\Delta \overset{(79)}{\leq} \vec{\Gamma}_0.t = \vec{\Gamma}.t \overset{(80)}{<} t_0 - \mathit{depth}(\mathcal{T})\Delta$$

Thus, $t_0 - (\mathit{depth}(\mathcal{T}) - m)\Delta = t_0 - \mathit{depth}(\mathcal{T})\Delta + m\Delta < t_0 - \mathit{depth}(\mathcal{T})\Delta$ has to hold for a potential $m \in \mathbb{N}$, which therefore cannot exist. Hence, the claim trivially holds.

Case $n > 0$**:**

With $|R| > 0$ we know that it exists a run R' with

$$R = R' \xrightarrow{\alpha} \Gamma.$$

By induction hypothesis (I.H.), the Theorem holds for R' and $\vec{\Gamma}' = \mathit{lastEnv}(R')$.

We proceed by case distinction on α. Out of the components of $\vec{\Gamma}$, only $\vec{\Gamma}.t$ and $\vec{\Gamma}.C_{en}$ are relevant for the Theorem. According to the *CTLC* Semantics, see Appendix E, the only actions influencing these components are

enableCTLC, *enableSubC*, *timeout*, *refund*, *claim*, *elapse* δ.

Since all users are honest, all actions are determined by the honest user strategy. By *enableCTLC* and *enableSubC* only additional contracts or subcontracts get added to $\Gamma_{ch}.C_{en}$. Thus, the claim still follows from I.H.. According to the honest user strategy, see Equation (38), *timeout*, and *refund* are only executed if there is an $e \mathrel{\widehat{\in}} \mathcal{T}$ with

$$\mathit{timelock}(h(e, id)) \leq \vec{\Gamma}.t.$$

By definition of h, see Equation (29), we have

$$\mathit{timelock}(h(e, id)) = t_0 + \mathit{depth}(e)\Delta.$$

As it is a precondition for this Theorem that $\vec{\Gamma}.t < t_0$, the actions *timeout*, and *refund* are not performed by honest users yet. For *claim* it is also a precondition that $t_0 \leq \vec{\Gamma}.t$ holds. Thus, honest users do not schedule this action yet. The only relevant action remaining is $\alpha = \mathit{elapse}\,\delta$. As $\alpha = \mathit{elapse}\,\delta$ does only get executed if all parties, in this case, all parties from *Hon*, agree. Hence for all $B \in$ *Hon* we have $\Sigma^{\mathbb{T}}_B(R') = \{\mathit{elapse}\,\delta\}$. According to Equation (43), this is only the case if no other action is possible based on the rules of the honest user strategy. In particular, this includes $newC(e, R) = 0$ for all $e \mathrel{\widehat{\in}} \mathcal{T}$ and hence $ingoing(e, R) = 0$, see Equation (35).

So let $e \mathrel{\widehat{\in}} \mathcal{T}$ be given. Then there exist users $B, X \in$ *Hon* such that $e = (B, X)_{\vec{a}}$, which is an outgoing edge for the honest user B. Hence for $ingoing(e, R) = 0$ inside of the honest user strategy of B we need to rule out that $ingoing(e, R) = 1$, see Equation (35). Since $e \neq (Y, B)_{\vec{a}}$, i.e. it is not an ingoing edge for B, there are 3 remaining conditions which are all concatenated with a logical 'and' condition. For the whole statement to be false, and thus implying $ingoing(e, R) = 0$, at least one of them needs to

be false. These conditions are:

$(a)\, \forall e' \mathrel{\widehat{\in}} \mathcal{T}$ with $\mathit{onPath}(e, e'), \mathit{depth}(e') = \mathit{depth}(e) + 1$
$\exists \widetilde{c^x} \in \Gamma_{channel(e')}.C_{en} : h(e', id) \in \widetilde{c^x}$

$(b)\, \vec{\Gamma}.t < t_0,$
$\exists \Psi^{id} := \mathit{treetoCTLCBadge}((id, \mathcal{T}, t_0, spec), \mathcal{S}) \in \vec{\Gamma}.B :$
$h(e, id) \in c^x \in \Psi^{id} \wedge h(e', id) \in \widetilde{c^x} \in \Psi^{id}$

$(c)\, \nexists ch : h(e, id) \in c^x \in \Gamma_{ch}.C_{en}$

We now argue that (a) and (b) cannot be false, and thus (c) is. The negation of (c) then gives us the desired property for the statement to hold.

For (a) we look at the case where $m \in \mathbb{N}$ is such that

$$\vec{\Gamma}'.t < t_0 - (\mathit{depth}(\mathcal{T}) - m)\Delta \text{ but} \tag{81}$$
$$\vec{\Gamma}.t = \vec{\Gamma}'.t + \delta \geq t_0 - (\mathit{depth}(\mathcal{T}) - m)\Delta \tag{82}$$

since for other $m' \in \mathbb{N}$ the claim follows immediately from I.H.. For these m', the preconditions have not changed. Since $\delta \leq \Delta$, the difference between $\vec{\Gamma}'.t + \delta$ and $\vec{\Gamma}'.t$ is smaller than 1Δ. Thus, there can only be one $m \in \mathbb{N}$ fulfilling both (81) and (82) at the same time. So, let $m \in \mathbb{N}$ be fixed in this way.

Additionally, we only need to consider edges e with

$$\mathit{depth}(e) + m = \mathit{depth}(\mathcal{T})$$

since for an edge e' with

$$\mathit{depth}(e') + m > \mathit{depth}(\mathcal{T})$$

we know that

$$\mathit{depth}(e') + m' \geq \mathit{depth}(\mathcal{T})$$

holds for some $m' < m$ and thus the I.H. applies. Therefore, only if the given $e \mathrel{\widehat{\in}} \mathcal{T}$ and $m \in \mathbb{N}$ fulfill these properties, this case is not implied directly by I.H.

So let e and m be this way. For (a) to be false, there would need to be an e' with $\mathit{depth}(e') = \mathit{depth}(e) + 1$ for which

$$\nexists \widetilde{c^x} \in \Gamma_{channel(e')}.C_{en} : h(e', id) \in \widetilde{c^x}.$$

This is a contradiction to the I.H. since

$$\mathit{depth}(e) + m = \mathit{depth}(\mathcal{T}) \Rightarrow \mathit{depth}(e') + (m-1) \geq \mathit{depth}(\mathcal{T})$$

and so the I.H. applies for $m-1$ and hence, $h(e', id)$ would be enabled. Therefore, this cannot be the case.

For (b) to be false, we either have $\vec{\Gamma}.t \geq t_0$, here the assumption would be contradicted, or

$$\nexists \Psi^{id} := \mathit{treetoCTLCBadge}((id, \mathcal{T}, t_0, spec), \mathcal{S}) \in \vec{\Gamma}.B :$$
$$h(e, id) \in c^x \in \Psi^{id} \wedge h(e', id) \in \widetilde{c^x} \in \Psi^{id}.$$

This means that there is no batch representing all edges in $\mathcal{T}$. In this case, we have $newBatch(\mathcal{T}, R') = 2$, since $\vec{\Gamma}.B = \vec{\Gamma}'.B$ and Lemma H.12 ensures that all necessary funds are still available. Therefore we have

$$\widehat{\Sigma}_B^{\mathcal{T}}(R') = \left\{ \mathit{advBatch}\, \Psi^{id} \right\},$$

see Equation (34). By Equation (43), this implies that the honest user strategy outputs this action instead of *elapse* δ:

$$\mathit{advBatch}\, \Psi^{id} \in \Sigma_B^{\mathbb{T}}(R')$$

And since all users need to agree in order to elapse time, we reach a contradiction.

Thus, the only possibility for $ingoing(e, R) = 0$ to hold is that (c) is false. This means

$$\exists ch : h(e, id) \in c^x \in \Gamma_{ch}.C_{en}.$$

Since all users are honest, Equation (37) dictates $ch = channel(e)$, which concludes the proof.

Using the correctness of the setup, we can prove protocol correctness. Intuitively, protocol correctness states that if all users are honest, and the protocol specification is consistent with the blockchain state, and execution is started in time, then the final run will reflect an execution corresponding to the ideal outcome ω_{id}^* that lies in the intersection of the outcome sets of all honest users.

Theorem H.20 (Protocol Correctness)**.** *Let $\mathit{Hon} = \{B_1, \ldots, B_k\}$ be a set of honest users. Let $\mathbb{T}$ be a well-formed set of tuples of the form $(id, \mathcal{T}, t_0, spec)$ and $\mathit{users}(\mathbb{T}) \subseteq \mathit{Hon}$. Let $\Sigma_{\mathit{Hon}}^{\mathbb{T}} = \{\Sigma_{B_1}^{\mathbb{T}}, \ldots, \Sigma_{B_k}^{\mathbb{T}}\}$ a set of honest user strategies executing $\mathbb{T}$. Let $\Sigma_{Adv}^{\mathbb{T}}$ be an arbitrary adversarial strategy (for Hon). Let R with $(\Sigma_{\mathit{Hon}}^{\mathbb{T}}, \Sigma_{Adv}^{\mathbb{T}}) \vdash R$ be a final run starting from an initial configuration $\vec{\Gamma}_0$ that is liquid w.r.t. $\mathbb{T}$. Further, let $\vec{\Gamma}_0.t < t_0 - \mathit{depth}(\mathcal{T})\Delta$, then for all $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ it exists $\omega_{id}^* \in \bigcap_{B \in \mathit{Hon}} \mathcal{O}_B^{\mathcal{T}}$ s.t. for their family $\{\omega_{id}^*\}_{\mathbb{T}} := \{\omega_{id}^* \mid (id, \mathcal{T}, t_0, spec) \in \mathbb{T}\}$ it holds*

$$\begin{aligned}
&\big(\forall \mathit{id}, e \in \omega_{id}^* \\
&\quad \exists c^x : \mathit{claim}(c^x, h(e, id), h_{sec}(e, id)) \in \mathit{actions}(R)\big) \\
\wedge\ &\big(\forall \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R) : \\
&\quad \mathit{users}(c^x) \cap \mathit{Hon} \neq \emptyset \\
&\qquad \Rightarrow \exists \mathit{id}, e \in \omega_{id}^* : sc^x = h(e, id) \\
&\qquad\quad \wedge\ \mathit{secret}_i(sc^x) = h_{sec}(e, id)\big).
\end{aligned}$$

Proof: Let $(id, \mathcal{T}, t_0, spec) \in \mathbb{T}$ be given. We define

$$\omega_{id}^* := \{e \mathrel{\widehat{\in}} \mathcal{T} \mid \exists c^x : \mathit{claim}(c^x, h(e, id), h_{sec}(e, id)) \in \mathit{actions}(R)\}.$$

For the statement, it is sufficient to show that

$$\omega_{id}^* \in \bigcap_{B \in \mathit{Hon}} \mathcal{O}_B^{\mathcal{T}}.$$

To this end, we need to show the following properties:

a) $\forall e \in \omega_{id}^* \forall e' \in \mathit{onPathToRoot}(\mathcal{T}, e) : e' \in \omega_{id}^*$
b) $\forall B \in \mathit{Hon} : \mathit{NoDup}(\mathcal{T}, B, \omega_{id}^*)$

c) $\forall B \in \mathit{Hon} : \mathit{HonestRoot}(\mathcal{T}, B, \omega^*_{id})$
d) $\forall B \in \mathit{Hon} : \mathit{EagerPull}(\mathcal{T}, B, \omega^*_{id})$

a) Assume towards contradiction

$$\exists e \in \omega^*_{id} \exists e' \in \mathit{onPathToRoot}(\mathcal{T}, e) : e' \notin \omega^*_{id}.$$

Then there needs to be an $e^* = (X, Y)_{\vec{a}^*} \in \omega^*_{id}$ and $e^\dagger = (Y, Z)_{\vec{a}^\dagger} \in \mathit{onPathToRoot}(\mathcal{T}, e)$ with $e^\dagger \notin \omega^*_{id}$ and $\vec{a}^* = [(X, Y)] \cdot \vec{a}^\dagger$. Since $e^* \in \omega^*_{id}$ implies

$$\mathit{claim}(c^x, h(e^*, id), h_{sec}(e^*, id)) \in \mathit{actions}(R).$$

we can use Theorem H.16 with $e^* \in \widehat{\varrho}_{Y,id}$ (and hence $Y \in \mathit{users}(h(e^*, id)) = \mathit{users}(c^x)$) to conclude that there is some $\widetilde{\omega}_Y \in \mathcal{O}^{\mathcal{T}}_Y$ such that $e^* \in \widetilde{\omega}_Y$.

By Definition D.3 of outcome sets, we then have $e^\dagger \in \widetilde{\omega}_Y$ (since outcome sets are constructed from partial trees) which implies (by Theorem H.16)

$$\mathit{claim}(c^x, h(e^\dagger, id), h_{sec}(e^\dagger, id)) \in \mathit{actions}(R).$$

Therefore $e^\dagger \in \omega^*_{id}$ by construction of ω^*_{id}.

b) Let $B \in \mathit{Hon}$. Assume towards contradiction that $e = (X, Y)_{\vec{a}} \in \omega^*_{id}$ and $e' = (X, Y)_{\vec{a}'} \in \omega^*_{id}$ and $B \in \{X, Y\}$ and $\vec{a} \neq \vec{a}'$. By the definition of ω^*_{id} we hence know that $\mathit{claim}(c^x, h(e, id), h_{sec}(e, id)) \in \mathit{actions}(R)$ and $\mathit{claim}(c^x, h(e', id), h_{sec}(e', id)) \in \mathit{actions}(R)$. From Theorem H.16 we know that there is some $\widetilde{\omega}_B \in \mathcal{O}^{\mathcal{T}}_B$ such that

$$\begin{aligned} &\forall e \in \widehat{\varrho}_{B,id} : e \in \widetilde{\omega}_B \\ &\quad \Rightarrow \exists c^x : \mathit{claim}(c^x, h(e, id), h_{sec}(e, id)) \in \mathit{actions}(R) \end{aligned} \tag{83}$$

and

$$\begin{aligned} &\forall \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R) : B \in \mathit{users}(c^x) \\ &\Rightarrow \exists e \in \widetilde{\omega}_B : sc^x = h(e, id) \wedge \mathit{secret}_i(sc^x) = h_{sec}(e, id) \end{aligned} \tag{84}$$

Consequently, we can conclude with Equation (83) that also $e \in \widetilde{\omega}_B$ and $e' \in \widetilde{\omega}_B$ (by definition of h we know that $\mathit{users}(e) = \mathit{users}(h(e, id))$). However, from $\widetilde{\omega}_B \in \mathcal{O}^{\mathcal{T}}_B$ we know that also $\mathit{NoDup}(\mathcal{T}, B, \widetilde{\omega}_B)$ and so $\vec{a} \neq \vec{a}'$, giving a contradiction.

c) Assume towards contradiction

$$\exists e = (X, B)_{[(X,B)]} \in \mathcal{T} \wedge e \notin \omega^*_{id}.$$

Hence $\mathit{claim}(c^x, h(e, id), h_{sec}(e, id)) \notin \mathit{actions}(R)$.

Let $\mathit{lastEnv}(R) =: \vec{\Gamma}$. Since R is final, see Definition G.4, we have $t_0 + \mathit{depth}(\mathcal{T})\Delta < \vec{\Gamma}.t$. By the precondition, we also have

$$\vec{\Gamma}_0.t \leq t_0 - \mathit{depth}(\mathcal{T})\Delta$$

for the initial environment $\vec{\Gamma}_0$.

Then we know that there must have be runs R^1 and R^2 such that $R = R^1 \xrightarrow{\mathit{elapse}\,\delta} R^2$ for some δ and for $R^{1a} := R^1 \xrightarrow{\mathit{elapse}\,\delta} \vec{\Gamma}^2$ and $\vec{\Gamma}^1 = \mathit{lastEnv}(R^1)$ it holds that $\vec{\Gamma}^1.t < t_0$ and $\vec{\Gamma}^2.t \geq t_0$. Since we know from the definition of the honest user strategy that $\delta \leq \Delta$, we also have that $\vec{\Gamma}^1.t = \vec{\Gamma}^2.t - \delta \geq t_0 - \Delta$.

Consequently, we can apply Theorem H.19 using $\mathit{depth}(e) = 1$ and $m := (\mathit{depth}(\mathcal{T}) - 1)$ (then $\mathit{depth}(\mathcal{T}) - m = 1$ and $\mathit{depth}(e) + m = \mathit{depth}(\mathcal{T})$) which results in

$$\exists c^x \in \Gamma^1_{channel(e)}.C_{en} : h(e, id) \in c^x. \tag{85}$$

Then by definition of the *elapse* rule, we also know that

$$c^x \in \Gamma^2_{channel(e)}.C_{en} : h(e, id) \in c^x. \tag{86}$$

We now show that then also $\alpha := X : \mathit{revealSecret}_{ch}\, s^{id}_{[(X,B)]} \in \Sigma^{\mathbb{T}}_X(R^{1a})$. By the definition of $\Sigma^{\mathbb{T}}_X$, for this we need to show that

1) $s^{id}_{[(X,B)]} \notin \Gamma^1_{channel(e)}.S_{rev}$
2) $no - dupl(e, R^{1a})$
3) $chContract(e, R^{1a}) = 2$.

To show the first two conditions, it is sufficient to show that the secret of no edge e' with $\mathit{sender}(e') = X$ is revealed yet. Assume towards contradiction that $\mathit{secret}(e', id) \in \Gamma^1_{channel(e)}.S_{rev}$. Then using Lemma H.9 it follows that $X : \mathit{revealSecret}_{ch}\, \mathit{secret}(e', id) \in R^1$ and so there must be a prefix R^{1b} of R^1 such that

$$R^{1b} \xrightarrow{X:\mathit{revealSecret}_{ch}\,\mathit{secret}(e',id)}$$

is a prefix of R^1. Since the action is restricted by X, we know that then also $X : \mathit{revealSecret}_{ch}\, \mathit{secret}(e', id) \in \Sigma^{\mathbb{T}}_X(R^{1b})$. However, this would imply that $\mathit{lastEnv}(R^{1b}).t \geq t_0$ (according to the definition of $\Sigma^{\mathbb{T}}_X$ secrets are only revealed after t_0). This immediately gives a contradiction, because from $\vec{\Gamma}^1.t < t_0$ we have that also $\mathit{lastEnv}(R^{1b}).t \geq t_0$ since time only increases over a run.

To show the last condition, we are left to show that

- $enabled(e)$
- $secretsAv(e)$
- $isIngoing(e, R^{1b})$
- $t_0 \leq \vec{\Gamma}^2.t < \mathit{timelock}(h(e, id))$

$enabled(e)$ follows from Equation (86) and construction of h since $\mathit{depth}(e) = 1$ and hence $h(e, id)$ is the top-level contract in c^x. We get $secretsAv(e)$ since by construction $h_{sec}(e, id)$ contains only the single secret $\mathit{secret}(e', id)$. Further, $isIngoing(e, R^{1b})$ holds since $\mathit{depth}(e) = 1$ and we can use Theorem H.19 with $m := (\mathit{depth}(\mathcal{T}) - 1)$ to show for all e_{in} with $\mathit{depth}(e_{in}) = 1$ that

$$\exists c^x \in \Gamma^1_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in c^x. \tag{87}$$

and so also

$$\exists c^x \in \Gamma^2_{channel(e_{in})}.C_{en} : h(e_{in}, id) \in c^x. \tag{88}$$

Finally, we already know that $t_0 \leq \vec{\Gamma}^2.t$. This leaves us with showing that $\vec{\Gamma}^2.t < \mathit{timelock}(h(e, id))$. Since by definition of h, we have that $\mathit{timelock}(h(e, id)) = t_0 + \Delta$ this follows immediately from $\vec{\Gamma}^2.t = \vec{\Gamma}^1.t + \delta < t_0 + \delta < t_0 + \Delta$ (since $\delta < \Delta$ and $\vec{\Gamma}^1.t < t_0$).

With this, we know that $X : \mathit{revealSecret}_{ch}\, s^{id}_{[(X,B)]} \in \Sigma^{\mathbb{T}}_X(R^{1b})$

Since we know that $t_0 + \mathit{depth}(\mathcal{T})\Delta < \vec{\Gamma}.t$ and $\vec{\Gamma}^2.t < t_0 + \Delta$ and $\mathit{depth}(\mathcal{T}) \geq 1$, we know that there must be R^3 and R^4 such that $R^2 = R^3 \xrightarrow{\mathit{elapse}\delta'} R^4$.

From the persistence of $\Sigma^{\mathbb{T}}_X$ and the monotonicity (Lemma H.8) of the revealed secrets, we can conclude that either

$$X : \mathit{revealSecret}_{ch}\, s^{id}_{[(X,B)]} \in \Sigma^{\mathbb{T}}_X(R \xrightarrow{\mathit{elapse}\,\delta} R^3)$$

or $s^{id}_{[(X,B)]} \in \mathit{lastEnv}(R^3).S_{rev}$

The first case would immediately lead to a contradiction since then it could not be (by definition of $\Sigma^{\mathbb{T}}_X$) that $\Sigma^{\mathbb{T}}_X$ schedules an *elapse* action. In the second case, we know (by Lemma H.8) that also $s^{id}_{[(X,B)]} \in \vec{\Gamma}.S_{rev}$. Using Theorem H.14, we know that there exists some $\varrho_{B,id}$ such that $R \sim \varrho_{B,id}$ and by $Inv_{insecrets}$ that $ein\varrho_{B,id}$. Further, $Inv_{init\text{-}liveness}$ gives us with $t_0 + \mathit{timelock}(h(e,id) \leq t_0 + \mathit{depth}(\mathcal{T})\Delta < \vec{\Gamma}.t$ that $\mathit{claim}(c^x, h(e,id), h_{sec}(e,id)) \notin \mathit{actions}(R)$, contradicting the original assumption.

d) Let $e = (X,B)_{\vec{a}_1} \widehat{\in} \mathcal{T}$ and $e' = (B,Y)_{\vec{a}_2} \in \omega^*_{id}$ with $\vec{a}_1 = [(X,B)] \cdot \vec{a}_2$.

By the definition of ω^*_{id} we hence know that

$$\mathit{claim}(c^x, h(e',id), h_{sec}(e',id)) \in \mathit{actions}(R).$$

From Theorem H.16 we know that there is some $\widetilde{\omega}_B \in \mathcal{O}^{\mathcal{T}}_B$ such that

$$\forall e \in \widehat{\varrho}_{B,id} : e \in \widetilde{\omega}_B \tag{89}$$
$$\Rightarrow \exists c^x : \mathit{claim}(c^x, h(e,id), h_{sec}(e,id)) \in \mathit{actions}(R)$$

and

$$\forall \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R) : B \in \mathit{users}(c^x) \tag{90}$$
$$\Rightarrow \exists e \in \widetilde{\omega}_B : sc^x = h(e,id) \wedge \mathit{secret}_i(sc^x) = h_{sec}(e,id)$$

Consequently, we can conclude with Equation (90) that also $e' \in \widetilde{\omega}_B$. Since $\widetilde{\omega}_B \in \mathcal{O}^{\mathcal{T}}_B$ we know in particular that $\mathit{EagerPull}(\mathcal{T}, B, \widetilde{\omega}_B)$ holds and consequently also that there exists some $\vec{a}_3$ such that $e'' = (X,B)_{\vec{a}_3} \in \widetilde{\omega}_B$ and $\mathit{depth}(e'') \leq \mathit{depth}(e)$. With this, we can use Equation (89) to obtain that

$$\mathit{claim}(\dot{c}^x, h(e'',id), h_{sec}(e'',id)) \in \mathit{actions}(R)$$

for some $\dot{c}^x$ and so by construction of ω^*_{id} also $e'' \in \omega^*_{id}$ what concludes the case.

End-to-end Correctness. Similar to our end-to-end security statement, we formulate an end-to-end correctness statement that ensures that if a tree protocol resulting from a graph is executed by honest users then the final executions of CTLC contracts involving honest users exactly correspond to the arcs in the graph.

Theorem H.21 (End-to-end Correctness)**.**
Let $\mathit{Hon} = \{B_1, \ldots, B_k\}$ be a set of honest users. Let $\mathbb{D}$ be a set of tuples of the form $(id, A, \mathcal{D}, t_0, \mathit{gspec}_{\mathcal{D}})$ with digraph $\mathcal{D}$ in-semiconnected w.r.t. $A \in \mathcal{N}$ and $\mathcal{N} \subseteq \mathit{Hon}$. Be $\mathbb{T}$ well-formed and given as

$$\mathbb{T} := \{(id, \mathcal{T}, t_0, \mathit{spec}) \mid (id, A, \mathcal{D}, t_0, \mathit{gspec}_{\mathcal{D}}) \in \mathbb{D} \wedge \mathit{spec} = \mathit{spec}_{\mathcal{D}} \wedge \mathcal{T} = \mathit{unfold}(\mathcal{D}, A)\}.$$

Let $\Sigma^{\mathbb{T}}_{\mathit{Hon}} = \{\Sigma^{\mathbb{T}}_{B_1}, \ldots, \Sigma^{\mathbb{T}}_{B_k}\}$ a set of honest user strategies executing $\mathbb{T}$. Let $\Sigma^{\mathbb{T}}_{Adv}$ be an arbitrary adversarial strategy (for Hon). Let R with $(\Sigma^{\mathbb{T}}_{\mathit{Hon}}, \Sigma^{\mathbb{T}}_{Adv}) \vdash R$ be a final run starting from an initial configuration $\vec{\Gamma}_0$ that is liquid w.r.t. $\mathbb{T}$. Further, let $\vec{\Gamma}_0.t < t_0 - \mathit{depth}(\mathcal{T})\Delta$ for all $(id, \mathcal{T}, t_0, \mathit{spec}) \in \mathbb{T}$. Then it holds

$$\begin{aligned}
&\big(\forall \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R), X, Y \in \mathcal{N} : \\
&\mathit{sender}(c^x) = X \wedge \mathit{receiver}(c^x) = Y \wedge \{X,Y\} \cap \mathit{Hon} \neq \emptyset \\
&\Rightarrow \exists (id, A, \mathcal{D}, t_0, \mathit{gspec}_{\mathcal{D}}) \in \mathbb{D} : (X,Y) \in \mathcal{D} \\
&\quad \wedge x = (id, X, Y)\big) \\
&\wedge \big(\forall (id, A, \mathcal{D}, t_0, \mathit{gspec}_{\mathcal{D}}) \in \mathbb{D}\ \forall (X,Y) \in \mathcal{D}\ \exists \dot{c}^{x'}, \dot{sc}^{x'}, j : \\
&x' = (id, X, Y) \wedge \mathit{claim}(\dot{c}^{x'}, \dot{sc}^{x'}, \mathit{secret}_j(\dot{sc}^{x'})) \in \mathit{actions}(R)\big)
\end{aligned}$$

Proof: From Theorem H.20, we know that for all $(id, \mathcal{T}, t_0, \mathit{spec}) \in \mathbb{T}$ there exists some $\omega^*_{id} \in \bigcap_{B \in \mathit{Hon}} \mathcal{O}^{\mathcal{T}}_B$ such that

$$\forall e \in \omega^*_{id} : \exists c^x : \mathit{claim}(c^x, h(e,id), h_{sec}(e,id)) \in \mathit{actions}(R) \tag{91}$$

and

$$\begin{aligned}
&\forall \mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R) : \\
&\mathit{users}(c^x) \cap \mathit{Hon} \neq \emptyset \\
&\Rightarrow \exists e \in \omega^*_{id} : sc^x = h(e,id) \wedge \mathit{secret}_i(sc^x) = h_{sec}(e,id)
\end{aligned} \tag{92}$$

To show the first conjunction of the statement assume that $\mathit{claim}(c^x, sc^x, \mathit{secret}_i(sc^x)) \in \mathit{actions}(R)$, $X, Y \in \mathcal{N}$, $\mathit{sender}(c^x) = X$, $\mathit{receiver}(c^x) = Y$, and $\{X,Y\} \cap \mathit{Hon} \neq \emptyset$. Using Equation (92), we can conclude that there exists some $e \in \omega^*_{id}$ such that $sc^x = h(e,id)$ and $\mathit{secret}_i(sc^x) = h_{sec}(e,id)$. From Theorem D.5, we immediately get that $(X,Y) \in \mathcal{D}$ and by definition of h that $x' = (id, X, Y)$.

To show the second conjunction, assume that $(X,Y) \in \mathcal{D}$ for some $(id, A, \mathcal{D}, t_0, \mathit{gspec}_{\mathcal{D}}) \in \mathbb{D}$. Then from Theorem D.5, we have that there must be some $\vec{a}$ such that $e = (X,Y)_{\vec{a}} \in \omega^*_{id}$. Using Equation (91), we can conclude that there exists some c^x with $\mathit{claim}(c^x, h(e,id), h_{sec}(e,id)) \in \mathit{actions}(R)$ This closes the case because we know that by construction $h(e,id) = sc^x$, $h_{sec}(e,id) = \mathit{secret}_j(sc^x)$ and $x = (id, X, Y)$ for some sc^x and $j \in \mathbb{N}$.

Remark H.22. *The proposed protocol can easily be extended to allow for multiple arcs in the same direction between two users. This means we would have graphs like:*

In the unfolding process, this implies multiple, identical edges on the same level. The only differentiating factor is then the underlying fund. Since their fund is different, they belong to two different CTLCs. In the presented con-

structions and proofs, nothing would change systematically besides adding another index to differentiate the objects.