Abstraction for Conflict-Free Replicated Data Types

Hongjin Liang
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, Jiangsu, China
hongjin@nju.edu.cn

Xinyu Feng[*]
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, Jiangsu, China
xyfeng@nju.edu.cn

Abstract

Strong eventual consistency (SEC) has been used as a classic notion of correctness for Conflict-Free Replicated Data Types (CRDTs). However, it does not give proper abstractions of *functionality*, thus is not helpful for modular verification of client programs using CRDTs. We propose a new correctness formulation for CRDTs, called Abstract Converging Consistency (ACC), to specify both data consistency and functional correctness. ACC gives abstract atomic specifications (as an abstraction) to CRDT operations, and establishes consistency between the concrete execution traces and the execution using the abstract atomic operations. The abstraction allows us to verify the CRDT implementation and its client programs separately, resulting in more modular and elegant proofs than monolithic approaches for whole program verification. We give a generic proof method to verify ACC of CRDT implementations, and a rely-guarantee style program logic to verify client programs. Our Abstraction theorem shows that ACC is equivalent to contextual refinement, linking the verification of CRDT implementations and clients together to derive functional correctness of whole programs.

CCS Concepts: • **Theory of computation** → **Program verification**; **Abstraction**; *Distributed algorithms*; • **Software and its engineering** → **Correctness**; **Semantics**.

Keywords: Replicated Data Types, Eventual Consistency, Contextual Refinement, Program Logic, Modular Verification

ACM Reference Format:
Hongjin Liang and Xinyu Feng. 2021. Abstraction for Conflict-Free Replicated Data Types. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '21), June 20–25, 2021, Virtual, Canada.* ACM, New York, NY, USA, 92 pages. https://doi.org/10.1145/3453483.3454067

[*]Corresponding author.

1 Introduction

Replicated data types are distributed implementations of data types that replicate data in different nodes of geographically distributed systems to improve availability and performance. A correct implementation needs to ensure that clients accessing different replicas have a consistent view of the data. Unfortunately, the CAP theorem [7] shows that, in the presence of network partitions, it is impossible to achieve both availability and strong consistency.

Conflict-Free Replicated Data Types (CRDTs) [19] are recently proposed to address the tensions between availability and consistency. On the one hand, CRDTs are designed to have availability. The nodes executing CRDTs can process client requests without synchronization. Later the updates are sent to other nodes, asynchronously and possibly in different orders. On the other hand, since concurrent updates may conflict, CRDTs follow certain carefully-designed strategies to resolve conflicts and provide a weak form of consistency. For instance, the last-writer-wins registers [19] resolve conflicts between concurrent writes by enforcing a global total order among the writes using time-stamps. The main strategy of add-wins sets [19] is to enforce that an add always wins over a concurrent remove of the same element. Benefiting from the conflict resolution strategies, CRDTs guarantee *strong eventual consistency* (SEC) [19], where two nodes are guaranteed to converge (i.e., having identical states) once they have received the same set of updates.

Unfortunately, SEC fails to specify the functional correctness of CRDTs. It is unclear to what extent a CRDT algorithm really implements the desired data type. For instance, can the last-writer-wins registers ensure that every read receives the most recent write, and what is the most recent write? Do the add-wins sets always behave like sequential sets, and what does "behaving like sequential sets" mean exactly? More importantly, without proper abstraction about functionality of CRDTs, it is difficult to verify *client programs* of CRDTs in a modular and layered way.

We use "**let** Π **in** $C_1 \parallel \ldots \parallel C_n$" to represent a program consisting of client programs $C_1, \ldots, C_n$, and the implementation Π of a CRDT. The clients run on distributed nodes and access the CRDT by invoking the operations defined in Π. To reason about the behaviors of the whole program, we need to verify both the correctness of the CRDT implementation Π and the behaviors of the client programs. A proper abstraction Γ for the CRDT would allow us to verify them separately. As shown in Fig. 1, we only need to verify the correctness of the CRDT implementation Π with respect to the abstraction Γ once and for all, no matter in what context (i.e., the collection of clients) it is used. Then we reason about the clients as if they were using the abstract object Γ, without worrying about the implementation details in Π (e.g., time-stamps or various auxiliary data).

However, building a general abstraction mechanism and a framework for verifying functional correctness of CRDTs and their clients turns out to be extremely challenging, mostly because of the diversity of conflict resolution strategies. We observe that the strategies can be divided into two classes. Most CRDTs use uniform conflict resolution strategies (UCR), such as time-stamps, which do not give privilege to particular operations, while add-wins sets and remove-wins sets use operation-dependent conflict resolution strategies "X-wins". The latter case relies on the functionality and the semantic relationship between operations, which makes the reasoning much more difficult than the former case.

Contributions. In this paper, we try to build abstraction and verification frameworks for CRDTs of both classes. The abstraction is in the form of atomic object specifications Γ, which are traditionally used for sequential data types and shared-memory concurrent objects. To facilitate the client reasoning, each Γ is also accompanied with a conflict relation ⋈ which specifies non-commutative abstract operations of the object (see Sec. 4). Our specifications are simple, allowing one to easily tell what abstract data type a CRDT algorithm really implements. They are also abstract enough to hide low-level implementation details such as time-stamps.

For UCR-CRDTs, Fig. 1 gives an overview of our framework. We propose **Abstract Converging Consistency** (ACC), a new formulation of correctness (① in Fig. 1, also in Sec. 5). ACC establishes an abstract view of execution based on the atomic specifications Γ, so reflects the desired functionality. The abstract views of execution sequences may be different on different nodes, but they must be coherent on conflicting abstract operations (related in ⋈) so that SEC is guaranteed.

We prove the **Abstraction Theorem** (see Sec. 6), showing that ACC is equivalent to a contextual refinement between the concrete implementation Π of CRDT operations and the atomic specification Γ, where the specification is executed in *a novel abstract operational semantics*. The Abstraction Theorem allows one to reason about client programs at a high abstraction level, by replacing concrete CRDT implementations with the specifications. It decouples the verification of clients and CRDTs, as shown in Fig. 1. The contextual refinement can be viewed as an alternative and more client-friendly correctness formulation for UCR-CRDTs.

Based on the abstraction, we present a **rely-guarantee-style program logic** for verifying client programs at the high abstraction level (② in Fig. 1, also in Sec. 7). Together with the contextual refinement, our logic offers a way to

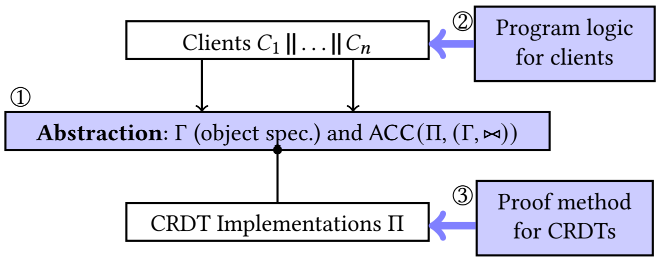

Clients $C_1 \parallel \ldots \parallel C_n$
②
Program logic for clients
①
Abstraction: Γ (object spec.) and ACC(Π, (Γ, ⋈))
③
CRDT Implementations Π
Proof method for CRDTs

Figure 1. Our abstraction and verification framework.

verify the functional correctness of the whole system. We have applied our logic to reason about several interesting client programs (see Appendix F).

We also develop **a proof method** for systematically verifying ACC (③ in Fig. 1, also in Sec. 8). *We have applied it to verify seven major UCR-CRDT algorithms [19]*, including the replicated counter (with both increment and decrement operations), the grow-only set, the last-writer-wins (LWW) register, the LWW-element set, the 2P-set, the continuous sequence, and the replicated growable array (RGA).

To the best of our knowledge, our work gives the first framework for compositional verification of whole programs, including both UCR-CRDT implementations and client code, based on contextual refinement and the abstraction theorem. We actually show that different implementation algorithms for the same data type, such as the continuous sequence and RGA for lists, or the LWW-element set and the 2P-set for sets, can be verified using the *same* abstract specification. Verifying a client program of the data type in our framework guarantees its correctness no matter which specific implementation algorithm it uses.

For X-wins CRDTs, we extend the specification with the explicit operation-dependent conflict resolution strategy, and propose XACC as an extension of ACC for correctness definition. We still establish the Abstraction Theorem, by giving a more relaxed abstract semantics to clients with object specifications. We also verify the functional correctness of the add-wins set and remove-wins set with respect to XACC.

2 Informal Development

Below we discuss the main challenges to formalize the correctness of CRDTs, and give an overview of our approaches.

2.1 The RGA Example

As a motivating example, Fig. 2 shows a simplified version [1] of the RGA algorithm [17] which in practice is the core algorithm for collaboratively edited documents. RGA implements a list object with three operations: `addAfter(a,b)` adds the element b after a in the list, `remove(a)` removes the element a from the list, and `read()` returns the whole list. For simplicity, we assume that the elements are unique, an element is added or removed at most once, and the list always contains a sentinel element ∘.

```
 1 var N := ∅, T := ∅;
 2 var ts := (0, cid);

 3 operation addAfter(a, b){
 4   assume(a = ∘ ∨
 5     a ≠ ∘ ∧ (_,_,a) ∈ N ∧ a ∉ T);
 6   local i := (ts.fst+1, cid);
 7   return;
 8   gen_eff AddAft(a, i, b);
 9 }

10 effector AddAft(a, i, b){
11   N := N ∪ {(a, i, b)};
12   if (ts < i) ts := i;
13 }

14 operation read(){
15   return trav(N,T);
16   gen_eff IdEff;
17 }

18 operation remove(a){
19   assume((_,_,a) ∈ N
20     ∧ a ∉ T ∧ a ≠ ∘);
21   return;
22   gen_eff Rmv(a);
23 }

24 effector Rmv(a){
25   T := T ∪ {a};
26 }
```

Figure 2. The Replicated Growable Array (RGA).

For CRDTs, each operation has *two phases*. In the *first* phase, a client on the node issues the operation. We call the node the *origin* of the operation. The origin node performs some initial local computation and responds to the client's request using the `return` command. It also generates an *effector* (see `gen_eff` in lines 8, 16 and 22), which captures the updates on the shared (replicated) state. The effector is executed immediately at the origin node, and is broadcast to all other nodes. In the *second* phase, each node applies the effector asynchronously over its local replica. Note that *read-only queries* (e.g., the `read()` operation) generate the identity effector `IdEff` (line 16 in Fig. 2). We do not need to broadcast `IdEff` since it does not change the state.

RGA represents the list using a time-stamped tree. Every tree node (a, i, b) consists of a key element b, a time-stamp i associated with b, and the key element a of its *parent node*. It is added by the operation `addAfter(a, b)`. Then a tree is encoded as a set of triples. For instance, the tree above can be represented by the set N:

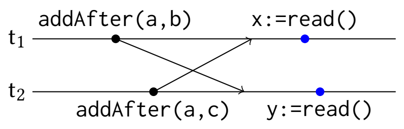
$$\begin{tikzcd}
& \circ & \\
& (\mathsf{ts}_0, \mathsf{a}) \arrow[u] & \\
(\mathsf{ts}_1, \mathsf{e}) \arrow[ur] & (\mathsf{ts}_2, \mathsf{b}) \arrow[u] & (\mathsf{ts}_3, \mathsf{c}) \arrow[ul] \\
& & (\mathsf{ts}_4, \mathsf{d}) \arrow[u]
\end{tikzcd}$$

$$\mathsf{N} = \{(\circ, \mathsf{ts}_0, \mathsf{a}), (\mathsf{a}, \mathsf{ts}_1, \mathsf{e}), (\mathsf{a}, \mathsf{ts}_2, \mathsf{b}), (\mathsf{a}, \mathsf{ts}_3, \mathsf{c}), (\mathsf{c}, \mathsf{ts}_4, \mathsf{d})\}$$

We assume ∘ is the root node of the tree. Besides the tree N, the algorithm also uses T as a *tombstone set* recording all the elements that are removed. Each replica state also contains `ts` to record the newest time-stamp at the replica.

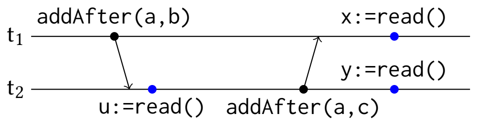
The read-only *query* operation `read()` calls the function `trav`. It first orders the sibling nodes on the tree N in decreasing time-stamp order, and then traverses the tree by depth-first search. From the resulting list, all the elements in the tombstone set T are removed and the list consisting of the remaining elements is returned. For instance, suppose the tombstone set T for the tree N shown above is {e}. The `read()` should return acdb if $\mathsf{ts}_0 < \mathsf{ts}_1 < \mathsf{ts}_2 < \mathsf{ts}_3 < \mathsf{ts}_4$.

```
addAfter(a, b);        ||   addAfter(a, c);
x := read();           ||   y := read();
```



(a)

```
                       ||   u := read();
addAfter(a, b);        ||   if (b ∈ u)
x := read();           ||     addAfter(a, c);
                       ||   y := read();
```



(b)

Figure 3. Clients of RGA and their executions.

The `addAfter(a,b)` operation generates the time-stamp `i` for b. Here time-stamps are implemented using pairs (n, t), where n is a natural number and t is a node ID (we write `cid` for the current node ID). Every two time-stamps are comparable: $(n_1, \mathsf{t}_1) > (n_2, \mathsf{t}_2)$ holds if $(n_1 > n_2)$ or $(n_1 = n_2) \wedge (\mathsf{t}_1 > \mathsf{t}_2)$. The effector of `addAfter(a,b)` simply adds `(a,i,b)` into the tree N and refreshes the time-stamp `ts` at the recipient node. The effector of `remove(a)` adds a into T.

Clients. The top of Fig. 3(a) shows a simple client program of RGA. It consists of two client threads calling the RGA operations. We represent the whole program as **let** Π_{RGA} **in** $C_1 \parallel C_2$, where Π_{RGA} denotes the RGA implementation in Fig. 2.

The bottom of Fig. 3(a) shows an execution of the program, assuming the clients running on two distinct nodes t_1 and t_2. The dots denote the client requests at the origin node (and the blue dots denote read-only queries). An arrow means sending an effector to a certain node.

We model an execution trace as a sequence $\mathcal{E}$ of events recording the execution of all the operations (both originals and effectors), and $\mathcal{E}|_{\mathsf{t}}$ as the subsequence consisting of only events occurring on the node t. So the execution shown in Fig. 3(a) is defined as the following trace $\mathcal{E}$ (assuming $\mathsf{ts}_1 < \mathsf{ts}_2$ and the initial list contains a only). We also record the arguments and return values (if any) of each operation.

$$(\mathsf{t}_1, \mathtt{addAfter}(\mathsf{a}, \mathsf{b}), \mathsf{ts}_1), (\mathsf{t}_2, \mathtt{addAfter}(\mathsf{a}, \mathsf{c}), \mathsf{ts}_2),$$
$$(\mathsf{t}_2, \mathtt{AddAft}(\mathsf{a}, \mathsf{ts}_1, \mathsf{b})), (\mathsf{t}_1, \mathtt{AddAft}(\mathsf{a}, \mathsf{ts}_2, \mathsf{c})),$$
$$(\mathsf{t}_1, \mathtt{read}(), \mathsf{acb}), (\mathsf{t}_2, \mathtt{read}(), \mathsf{acb})$$

The event $(\mathsf{t}_1, \mathtt{addAfter}(\mathsf{a}, \mathsf{b}), \mathsf{ts}_1)$ represents the invocation of an operation on the origin node t_1, where the time-stamp ts_1 is generated for the corresponding effector. The event

$(\mathtt{t}_2, \mathtt{AddAft(a, ts_1, b)})$ represents the execution of an effector on $\mathtt{t}_2$ (sent from other nodes). Then the local traces $\mathcal{E}|_{\mathtt{t}_1}$ and $\mathcal{E}|_{\mathtt{t}_2}$ are the following:

$(\mathtt{t}_1, \mathtt{addAfter(a, b)}, \mathtt{ts}_1), (\mathtt{t}_1, \mathtt{AddAft(a, ts_2, c)}), (\mathtt{t}_1, \mathtt{read()}, \mathtt{acb})$

$(\mathtt{t}_2, \mathtt{addAfter(a, c)}, \mathtt{ts}_2), (\mathtt{t}_2, \mathtt{AddAft(a, ts_1, b)}), (\mathtt{t}_2, \mathtt{read()}, \mathtt{acb})$

Note that each node only sees its own read-only queries.

2.2 Functional Correctness (FC) of CRDTs

Correctness of CRDTs should capture both SEC and functionality of the data types, so that we can reason about the behaviors of clients (e.g., those in Fig. 3) without looking into the code of CRDT implementation (e.g., the RGA algorithm in Fig. 2), assuming the correctness of CRDT. It is easy to see that the RGA algorithm guarantees SEC since all the effectors produced by the algorithm are commutative with each other, but what is the expected functionality? From clients' point of view, the object is shared by all client threads and may be updated concurrently through the provided operations. Ideally we want to allow the client to maintain a simple *atomic* view of each object operation, so that we can interpret the client's behaviors in terms of executions of a sequence of these abstract atomic operations. For instance, the nodes $\mathtt{t}_1$ and $\mathtt{t}_2$ in Fig. 3(a) may both interpret their local execution traces as the following sequential execution of atomic operations:

$\mathtt{addAfter\text{-}atom(a, b)}, \mathtt{addAfter\text{-}atom(a, c)}, (\mathtt{read()}, \mathtt{acb})$

Here `addAfter-atom(x, y)` represents an abstract atomic specification of `addAfter(x, y)`. Its effects are applied atomically to the RGA object. It is abstract and does not generate any effectors or time-stamps. Note that the result `acb` of the final `read` determines the order between `addAfter-atom(a, b)` and `addAfter-atom(a, c)`. Therefore, for the node $\mathtt{t}_2$, the abstract operations have to be executed in a different order from the order of the effectors in its concrete trace $\mathcal{E}|_{\mathtt{t}_2}$.

Unlike SEC, which is about the consistency of data replica on *different* nodes, the functional correctness (FC) is defined from the viewpoint of each *individual* node (or client). It specifies the consistency between the execution trace of concrete operations on a node and the corresponding abstract execution trace.

Defining FC. The above example shows that each node t may interpret an execution $\mathcal{E}$ in terms of a sequential execution of the corresponding atomic operations, which we describe by a total order $ar_{\mathtt{t}}$ over these operations. Our FC requires, for every prefix $\mathcal{E}'$ of $\mathcal{E}$, the sub-trace $\mathcal{E}'|_{\mathtt{t}}$ that t sees locally may correspond to an abstract trace $\mathcal{E}''$ following the total order $ar_{\mathtt{t}}$, such that performing $\mathcal{E}'|_{\mathtt{t}}$ has the same effects as performing $\mathcal{E}''$, that is, *they generate the same state (modulo the state abstraction), and the same return value if* $\mathcal{E}'|_{\mathtt{t}}$ *ends with a query operation.*

In the example both $ar_{\mathtt{t}_1}$ and $ar_{\mathtt{t}_2}$ order `addAfter-atom(a, b)` before `addAfter-atom(a, c)`. For $\mathtt{t}_2$, we consider its local traces of all the prefixes of $\mathcal{E}$:

$\mathcal{E}_1 : (\mathtt{t}_2, \mathtt{addAfter(a, c)}, \mathtt{ts}_2)$
$\mathcal{E}_2 : (\mathtt{t}_2, \mathtt{addAfter(a, c)}, \mathtt{ts}_2), (\mathtt{t}_2, \mathtt{AddAft(a, ts_1, b)})$
$\mathcal{E}_3 : (\mathtt{t}_2, \mathtt{addAfter(a, c)}, \mathtt{ts}_2), (\mathtt{t}_2, \mathtt{AddAft(a, ts_1, b)}),$
$(\mathtt{t}_2, \mathtt{read()}, \mathtt{acb})$

We can check that $\mathcal{E}_1$ generates the same state as the atomic execution of `addAfter-atom(a, c)` (since the trace consists of only one event, it trivially satisfies the total order $ar_{\mathtt{t}_2}$), and $\mathcal{E}_2$ corresponds to

`addAfter-atom(a, b), addAfter-atom(a, c)`

For $\mathcal{E}_3$, we also check the final return value is the same with such a query in the abstract trace.

2.3 Ordering of Operations and ACC

Both SEC and FC above are defined in a declarative manner and are not very informative to the clients of CRDTs. For instance, FC only requires the *existence* of an order $ar_{\mathtt{t}}$ on each node t to order the abstract operations, and says nothing about what the $ar_{\mathtt{t}}$ is like. So the clients still cannot tell the execution orders between CRDT operations.

To help reason about client programs, we want to specify the ordering of operations that CRDTs can enforce. More specifically, for each total order $ar_{\mathtt{t}}$ of abstract operations on each node t, we want to give more constraints to tell how to relate it to the concrete execution order, and how to relate different $ar_{\mathtt{t}}$ on different nodes so that SEC is guaranteed.

For instance, a direct mapping of each concrete step to the corresponding abstract atomic one following the real-time order on a node usually does not work. In the example shown in Fig. 3(a), $ar_{\mathtt{t}_2}$ has to order `addAfter-atom(a, b)` before `addAfter-atom(a, c)`, which is different from the real-time order of concrete operations in $\mathcal{E}|_{\mathtt{t}_2}$. *Then what are the appropriate orders of the abstract operations?*

Preserving the visibility order. Consider the client of RGA in Fig. 3(b). In the execution, the first `read` of $\mathtt{t}_2$ is made after the arrival of the effector of `addAfter(a,b)` from $\mathtt{t}_1$. In this case we say `addAfter(a,b)` is *visible* to `u:=read()`. In general, an operation *a* is visible to an operation *b* at the node t if the effector of *a* has been applied at t before t issues *b*. The visibility order encodes the "happens-before" relations between operations for a certain node.

Naturally we expect `u`, `x` and `y` to read out `ab`, `acb` and `acb` respectively (assuming the initial list contains `a` only). This means, when we map the concrete steps at a thread to a sequence of abstract atomic operations, the abstract executions should follow the visibility order.

Different nodes may observe different orders. In FC we require each node t to maintain an order $ar_{\mathtt{t}}$ of abstract operations. SEC would be obvious if all $ar_{\mathtt{t}}$ are the same. However, as we would see below, this requirement is overly restrictive and cannot be satisfied by some CRDTs.

```
addAfter(a, p);      ||   addAfter(c, e);
addAfter(c, d);      ||   addAfter(a, q);
u := read();         ||   v := read();
```

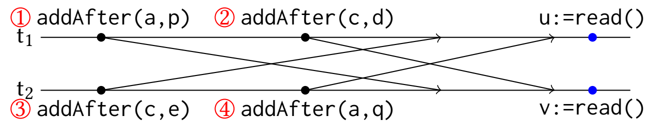


Figure 4. A client of continuous sequence. Assuming the initial sequence is ac, is it possible for u and v to read apqced?

Consider the program in Fig. 4. It is also a client of CRDT sequence, but implemented using the continuous sequence algorithm [19] instead of RGA. The continuous sequence tags each `addAfter` operation with a real number, the value of which reflects the intended position of the newly added element (assuming tags of elements on the sequence are in increasing order). For instance, assuming the initial sequence is ac, operation ① will tag p with a real number between the tags of a and its subsequent element c. The `read` operation then orders the elements by their tags and returns the resulting sequence. Note that the tags are different from the time stamps in RGA, and the happens-before order does *not* imply the order of tags. For instance, we know the tag generated by ② is greater than ①, but the tag of ④ is smaller than ③.

In this example it is possible to read apqced at the end, as long as the tag generated by ① happens to be smaller than that of ④, while the tag of ③ is smaller than that of ②. To interpret the final sequence apqced, node t_1 has to order the abstract operation ④ before ①, and order ② before ③. In addition, it needs to preserve the visibility order, as we explained before. So it needs to order ① before ②. Therefore, the only acceptable order for t_1 is ④①②③. Similarly, the only possible order for t_2 is ②③④①. So ① and ② (also ③ and ④) must be ordered differently by t_1 and t_2.

Therefore we should allow different nodes to have different local views of the abstract executions. In particular, *the visibility orders of operations originated in other nodes may not be respected.* We can also find similar examples in other CRDTs such as the add-wins set.

However, the orders cannot be arbitrarily different because we need to guarantee SEC. They have to be consistent in some way. *What kind of consistency should be enforced then?*

Conflicting operations should follow the same order. CRDTs achieve SEC by turning non-commutative abstract operations into commutative effectors. Arbitrary orderings of commutative operations always lead to the same state.

We say two abstract operations f_1 and f_2 are *conflicting*, represented as $f_1 \bowtie f_2$, if they are not commutative. In Fig. 4, `addAfter(a, p)` and `addAfter(a, q)` are conflicting, but `addAfter(a, p)` and `addAfter(c, d)` are not.

Naturally, to reach the same state, we require the abstract executions on different nodes execute conflicting operations in the same order. In Fig. 4, the abstract executions ④①②③ and ②③④① order ④ and ① (② and ③) the same way.

Abstract Converging Consistency (ACC). We formalize our correctness notion of CRDTs as Abstract Converging Consistency (ACC), which is a relation between the concrete implementation of a CRDT (represented as Π) and its abstract specification (represented as a pair $(\Gamma, \bowtie)$, where Γ is the abstract atomic specification of the operations, and $\bowtie$ is a symmetric binary relation between conflicting operations).

ACC requires FC defined in Sec. 2.2, and the order constraints over abstract executions described in this section. More specifically, ACC$(\Pi, (\Gamma, \bowtie))$ requires that, for any execution trace $\mathcal{E}$, each node t can find a total order ar_t over abstract atomic operations, such that:

- For each prefix $\mathcal{E}'$, there is a corresponding sequence $\mathcal{E}''$ of abstract operations. $\mathcal{E}''$ follows the order ar_t and generates the same effects with $\mathcal{E}'|_t$; and
- ar_t preserves the local visibility order on t; and
- For any two nodes t_1 and t_2, ar_{t_1} and ar_{t_2} can be different, but they must assign the same order for conflicting operations specified in $\bowtie$.

We can prove that ACC defined above guarantees SEC.

Note that the last point only requires the *existence* of a consistent ordering of conflicting operations, with no further constraints. This is not a problem for UCR-CRDTs that use uniform operation-independent conflict resolving strategies. However, for CRDTs like add-wins and remove-wins sets, we may rely on the specific strategy (X-wins) to reason about the behaviors of clients. In this case we need to further refine the above ACC definition.

2.4 Extended ACC for X-Wins CRDTs

We show an execution of add-wins sets in Fig. 5(a). A set provides three operations: `lookup(e)`, `add(e)` and `remove(e)`. The add-wins set algorithm assigns a unique tag to each element when it is added. In Fig. 5 we highlight the tags by labeling the dots effectors rather than originals. We use 0 and 1 to represent the elements in the set, and a and b for the tags. So an element may be added to the set multiple times but each with a different tag. The `remove` operation removes all the occurrences of the element in the local replica. The effector of `remove` carries the set of element-tag pairs removed locally. On receiving the effector, the remote hosts remove only these pairs from their local replicas.

For instance, in Fig. 5(a) when t_2 issues a `remove(1)` request (operation ⑥), it sees only `(1, b)` in the local replica and sends the effector `Rmv((1, b))` to t_1. When it arrives at t_1, the pairs `(1, b)` and `(1, c)` are both in t_1's replica, but only `(1, b)` is removed. Therefore the subsequent `lookup(1)`

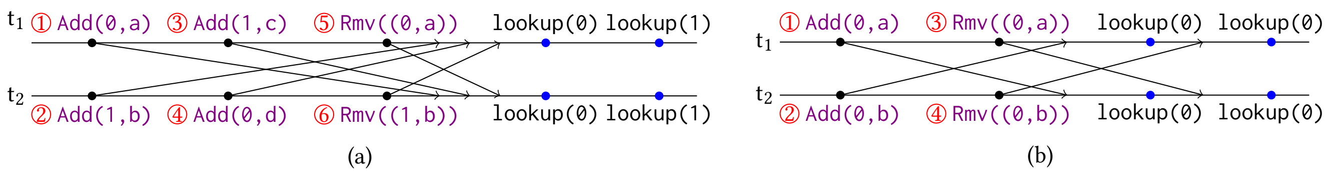


Figure 5. Executions of the add-wins set.

still returns true. This illustrates the *add-wins* conflict resolving strategy: for concurrent add (③) and remove (⑥), the abstract view is to execute add after remove.

It is interesting to see that the *add-wins* conflict resolving strategy is different from the time-stamp-based approaches since it is tied with the functionality of specific operations. As the dual, there is also the remove-wins set algorithm which applies the *remove-wins* strategy. Note that the add-wins set and the remove-wins set assume *causal delivery* between add and remove operations. This is also different from other CRDTs, which do not need to rely on causal delivery.

The add-wins sets and remove-wins sets may have different behaviors, which are observable by clients. If the client relies on the specific strategy and cares about the difference, our above ACC definition would be too abstract to distinguish them. We solve this problem by introducing a *won-by* relation $\blacktriangleleft$ in the abstract specification to describe the conflict resolving strategy. We have remove(e) $\blacktriangleleft$ add(e) for add-wins set, and the reverse for remove-wins set. Since we only need to resolve conflicts for conflicting operations, the $\blacktriangleleft$ relation is a subset of the conflict relation $\bowtie$. Correspondingly, we refine the third point of ACC in 2.3 with an extra requirement that all the ar_t respect the $\blacktriangleleft$ order.

Unfortunately, this simple extension of ACC would not work. Consider the execution shown in Fig. 5(b). For each node, we can see the two lookup operations return true and false respectively. However, we cannot find a total order ar satisfying ACC. For t_1, we have to order ① before ③ (to preserve the visibility order), and ③ before ② (to respect the $\blacktriangleleft$ order). Therefore ④ has to be the last operation, otherwise the abstract execution cannot generate the same return values as the concrete one, failing FC. However, ordering ④ after the concurrent ① would violate the $\blacktriangleleft$ order.

This problem is caused by our over-simplified interpretation of the "add-wins" conflict-resolving strategy, which says we should *always* order remove(e) before add(e) if they are *concurrent*. However, in our example, when ④ arrives at t_1, the effect of ① has already been canceled out by ③. Therefore at this moment whether ① has been executed before or not should make no difference.

To address this problem, we give a more precise description of the strategy, which says concurrent remove(e) should be ordered before add(e) only if the effect of add(e) is still reflected in the state (i.e., its effect has not been canceled out by others). Since the cancellation of effects is functionality dependent, we introduce another *canceled-by* relation $\triangleright$ over abstract operations in the specification. Informally, we let the operation f be canceled by f' ($f \triangleright f'$) if the following two requirements hold:

- f may win others as specified in $\blacktriangleleft$; and
- for any other abstract operations $f_1, \ldots, f_n$ ($n \geq 0$) in between, the abstract operation sequence $f, f_1, \ldots, f_n, f'$ has the same effects as $f_1, \ldots, f_n, f'$.

Therefore, for add-wins sets, we have add(e) $\triangleright$ remove(e) but not the inverse (which violates the first requirement).

We relax the third point of ACC accordingly, and ignore the canceled operations when we check the consistency between the total orders ar_t for different nodes t. This relaxed ACC allows the total orders ar_{t_1} and ar_{t_2} in Fig. 5(b) to be defined as ①③②④ and ②④①③, respectively. When ④ is executed at t_1, we only need to check that ③ and ④ are ordered consistently, and ignore ① and ② because they have been canceled (by ③ and ④ respectively) at this moment. Also because ③ and ④ are not conflicting (they are commutative), it is okay to order them differently in ar_{t_1} and ar_{t_2}.

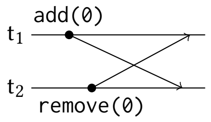
With the more refined specification, we can redefine the correctness as $\mathsf{XACC}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$. It also assumes causal delivery of messages, as required by add-wins and remove-wins sets. Note that UCR-CRDTs satisfying $\mathsf{ACC}(\Pi, (\Gamma, \bowtie))$ in Sec. 2.3 also satisfy $\mathsf{XACC}(\Pi, (\Gamma, \bowtie, \emptyset, \emptyset))$ — Since their conflict resolving policies are not tied with particular operations, we can simply set $\blacktriangleleft$ and $\triangleright$ to be empty.

Compositionality. Like linearizability, our definition of ACC/XACC is compositional. That is, for a set of CRDTs $\Pi_1, \ldots, \Pi_n$, if every Π_i satisfies $\mathsf{XACC}(\Pi_i, (\Gamma_i, \bowtie_i, \blacktriangleleft_i, \triangleright_i))$, then the clients can use them together and view them as a single big object satisfying $\mathsf{XACC}(\overrightarrow{\Pi}, (\overrightarrow{\Gamma}, \overrightarrow{\bowtie}, \overrightarrow{\blacktriangleleft}, \overrightarrow{\triangleright}))$, where $\overrightarrow{\Pi}$ represents the disjoint union of all the operations $\Pi_1 \uplus \ldots \uplus \Pi_n$, and $\overrightarrow{\Gamma}$, $\overrightarrow{\bowtie}$, $\overrightarrow{\blacktriangleleft}$ and $\overrightarrow{\triangleright}$ are defined similarly. Note here we assume the CRDTs do not share data.

2.5 Abstraction and Client Reasoning

It is important to note that the goal of this work is *not* to give axiomatic definitions to tell the validity of a single execution trace, although we use traces above (e.g., those shown in Figs. 3 and 4) to explain the key ideas. Our goal is to support *static program verification*, where we need to consider all the execution traces that can be possibly generated by the program, and the reasoning is based on the program text without actually running it. This is much more challenging than reasoning about a single trace.

For instance, if we look at the execution of a CRDT set on the right, it is easy to tell what the final state is: it must contain 0 for add-wins sets, but mustn't for remove-wins sets. Knowing the concrete implementation mechanism, the result can be easily predicted. The deceiving simplicity may make one doubt the need of abstraction. However, if we consider the simple client program (`add(0); || remove(0);`) which generates the trace, we know it may generate both results (since there are other possible executions where one operation happens before the other), no matter which CRDT set we use[1]. This example shows that we have to consider all possible ordering of operations for program reasoning, which can be very complicated in non-trivial clients. Abstracting away the implementation details and taking an atomic view of operations can greatly simplify the reasoning.



Remark. Picking the appropriate abstraction level for CRDT specifications is one of the key challenges we need to address. On the one hand, the abstractions need to hide as much implementation detail as possible. On the other hand, they need to be useful for client reasoning, i.e., it does not abstract away important functionality properties of the data type.

For X-wins CRDTs, we need to decide whether or not to hide the functionality-dependent "X-wins" strategies. It might be possible to have a weaker ACC definition that unifies UCR and X-Wins CRDTs, but it would not support the reasoning about some special clients whose functionality depends on the differences between add-wins sets, remove-wins sets and UCR sets. Consider the following client:

```
add(0);remove(0);   ||  add(0);remove(0);
x := read();        ||  y := read();
```

At the end the post-condition $0 \in x \Rightarrow 0 \notin y$ holds when the client uses the remove-wins set or UCR sets (e.g., the LWW-element set) but not when it uses the add-wins set. Abstracting away the differences of these sets would prevent the verification of the above program.

3 Basic Technical Settings

Figure 6 shows the syntax of the language. The whole program P consists of n clients C, each running on different nodes. They share the *object* Π, which is replicated on all the nodes. Each client executes sequentially, accessing the local *client* state in the node. It can also access the *object* state through the command $x := f(E)$, which calls the operation f of the object with the argument E.

We model the object Π as a mapping from an operation name f and its argument to the actual operation over the object state. When a client calls an operation, it executes in two steps. First the operation is applied over the object state and generates a return value and an effector δ. The

$$\begin{array}{rrcl}
(\mathit{OpName}) & f & \in & \mathit{String} \\
(\mathit{Effector}) & \delta & \in & \mathit{LocalState} \rightharpoonup \mathit{LocalState} \\
(\mathit{ODecl}) & \Pi & \in & \mathit{OpName}\times \mathit{Val} \rightharpoonup \mathit{LocalState} \rightharpoonup \mathit{Val}\times\mathit{Effector} \\
\\
(\mathit{Expr}) & E & ::= & x \mid n \mid E{+}E \mid \ldots \\
(\mathit{CltStmt}) & C & ::= & x := f(E) \mid \textbf{skip} \mid C;C \mid \textbf{if}\ (E)\ C\ \textbf{else}\ C \mid \ldots \\
(\mathit{Prog}) & P & ::= & \textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n
\end{array}$$

Figure 6. Syntax of the programming language.

effector δ captures the operation's effect over the object state. It is *broadcast* to all nodes, including the one where the client request originates. Then the effector δ is applied on the local replica of the object data on each node. Note that on the origin node of the client request, the generation of the effector and the execution of it over the local replica are done atomically. To simplify the presentation we assume each program uses only one object. As we explained in Sec. 2.4, our correctness definition ACC is compositional and the results still hold when there are more objects.

We assume an effector is delivered to a node at most once, but it may never reach a target node. Also we do *not* assume FIFO message channels. Most of the CRDTs can work under these assumptions. When stronger assumptions are needed (e.g., causal delivery), we can add extra constraints over execution traces.

Events and event traces. The clients C_i in the program **let** Π **in** $C_1 \parallel \ldots \parallel C_n$ are executed following the standard interleaving semantics. The semantics generates events when CRDT operations are executed. An execution trace is the sequence of events generated during the interleaving execution. We define the events e and execution traces $\mathcal{E}$ below:

$$\begin{array}{rrcl}
(\mathit{Event}) & e & ::= & (\mathit{mid}, \mathsf{t}, (f, n, n', \delta)) \mid (\mathit{mid}, \mathsf{t}, (f, n), \delta) \\
(\mathit{ETrace}) & \mathcal{E} & ::= & \epsilon \mid e :: \mathcal{E}
\end{array}$$

Here ϵ represents an empty list. The event $(\mathit{mid}, \mathsf{t}, (f, n, n', \delta))$ is called an *origin event*. It is generated when the object operation f is called on the node t with the argument n, and the return value n' and the effector δ are generated by applying $\Pi(f, n)$ over the local replica. It also contains a unique ID *mid* for the original request of the operation. When the effector δ is delivered to and executed at another node t', the node t' generates the event $(\mathit{mid}, \mathsf{t}', (f, n), \delta)$. It records not only the local node ID t' and the effector, but also the information about the original operation, including the operation name f, the argument n, and the ID *mid*.

We define $\mathcal{T}(P, \mathcal{S})$ as the *prefix closure* of the event traces that can be generated by executing P from the initial state $\mathcal{S}$. We also define $\mathcal{T}(\Pi, \mathcal{S})$ as the prefix closure of the event traces that can be generated by *any* set of clients accessing Π with the initial state $\mathcal{S}$.

4 Specifications for CRDTs

The specification of a CRDT object consists of two parts, the operation specification Γ and the conflict relation $\bowtie$, as shown in Fig. 7. Γ maps operation names and arguments

[1] Note it is indeed possible to construct clients that can distinguish add-wins sets from remove-wins sets, as discussed in the following remarks.

$$\begin{array}{rrl} (\mathit{OSpec}) & \Gamma \in & \mathit{OpName} \times \mathit{Val} \rightharpoonup \mathit{AbsState} \to \mathit{Val} \times \mathit{AbsState} \\ (\mathit{Action}) & \alpha \in & \mathit{AbsState} \to \mathit{AbsState} \\ & \bowtie \in & \mathscr{P}(\mathit{Action} \times \mathit{Action}) \end{array}$$

Figure 7. Object specifications $(\Gamma, \bowtie)$.

to *abstract atomic operations* of the type $\mathit{AbsState} \to \mathit{Val} \times \mathit{AbsState}$. That is, each atomic operation applies over an abstract object state and generates the resulting abstract state and a return value. We assume it is a total function because as a specification we do not want it to get stuck whenever a client applies the operation.

We use `AbsState` to represent the set of object states $\mathcal{S}$ at the abstract level. They may abstract away the implementation dependent information of the concrete states. For instance, the concrete state of RGA consists of a time-stamped tree `N` and a tombstone `T`, as shown in Sec. 2.1, while the abstract state is simply a sequence (e.g., `acdb`).

Since the return value of an operation is meaningful only to the origin node, while the state transformation needs to be performed on all replicas, we use $\mathsf{opr}(\Gamma(f, n))$ to represent the effects of $\Gamma(f, n)$, which does a state transformation. We call the transformation an action (represented as α).

The conflict relation $\bowtie$ needs to be a *symmetric* binary relation over *non-commutative* actions. For sets, `add(x)` and `remove(x)` conflict with each other. For RGA,

$$\begin{array}{l} \mathtt{addAfter}(a, b) \bowtie \mathtt{addAfter}(c, d) \ \text{ iff } \ \{a, b\} \cap \{c, d\} \neq \emptyset, \\ \mathtt{addAfter}(a, b) \bowtie \mathtt{remove}(c) \ \text{ iff } \ c \in \{a, b\}\ . \end{array}$$

Well-defined specifications must satisfy $\mathsf{nonComm}(\Gamma, \bowtie)$, which requires that all the non-commutative actions in Γ should be specified in $\bowtie$.

Definition 1. $\mathsf{nonComm}(\Gamma, \bowtie)$ iff $\forall f_1, n_1, f_2, n_2, \alpha_1, \alpha_2,$

$$\begin{array}{l} \alpha_1 = \mathsf{opr}(\Gamma(f_1, n_1)) \land \alpha_2 = \mathsf{opr}(\Gamma(f_2, n_2)) \land \neg(\alpha_1 \bowtie \alpha_2) \\ \implies \alpha_1 \mathbin{;} \alpha_2 = \alpha_2 \mathbin{;} \alpha_1 \end{array}$$

where $\alpha \mathbin{;} \alpha' \overset{\text{def}}{=} \lambda \mathcal{S}.\, \alpha'(\alpha(\mathcal{S}))$.

As we explained in Sec. 2.5, add-wins and remove-wins sets should be specified with further information about the conflicting resolving strategies, i.e., the won-by ($\blacktriangleleft$) and canceled-by ($\triangleright$) relations over conflicting actions. In the following sections we first present our results for UCR-CRDTs that do not need $\blacktriangleleft$ and $\triangleright$, and show the extension of them to support these X-wins algorithms in Sec. 9.

We assume $\bowtie$ is symmetric and $\mathsf{nonComm}(\Gamma, \bowtie)$ holds throughout the paper. We overload $\bowtie$ over operations, and also over events, written as $(f, n) \bowtie_\Gamma (f', n')$ and $e \bowtie_\Gamma e'$ respectively (the subscript Γ is used to extract actions corresponding to (f, n), (f', n'), e and e').

5 Abstract Converging Consistency

As shown in Def. 2, $\mathsf{ACC}_\varphi(\Pi, (\Gamma, \bowtie))$ is parameterized with an abstraction function φ, which maps concrete object states to abstract ones, i.e., $\varphi \in \mathit{LocalState} \rightharpoonup \mathit{AbsState}$.

$$\begin{array}{l} \mathsf{ExecRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathit{ar})) \ \text{ iff } \ \forall \mathcal{E}' \leqslant \mathcal{E}. \\ \quad \forall (\mathcal{S}_a', n') = \mathsf{aexec}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright \mathit{ar}). \\ \qquad \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathcal{S}_a' \land \\ \qquad\qquad (\forall e = \mathsf{last}(\mathcal{E}'|_{\mathsf{t}}).\, \mathsf{is\_orig}_{\mathsf{t}}(e) \implies \mathsf{rval}(e) = n') \\ \mathsf{Coh}(\mathit{ar}, \mathit{ar}', (\Gamma, \bowtie)) \ \text{ iff } \\ \quad \forall e_1, e_2.\, (e_1 \ \mathit{ar}\ e_2) \land (e_2\ \mathit{ar}'\ e_1) \implies \neg(e_1 \bowtie_\Gamma e_2) \end{array}$$

Figure 8. Auxiliary definitions for ACC.

Definition 2. $\mathsf{ACC}_\varphi(\Pi, (\Gamma, \bowtie))$ iff

$$\forall \mathcal{S}, \mathcal{E}.\ \mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S}) \land \mathcal{S} \in \mathit{dom}(\varphi) \implies \mathsf{ACT}_\varphi(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie))$$

It requires every event trace $\mathcal{E}$ of Π to satisfy ACT shown in Def. 3, which formalizes the idea in Sec. 2.3.

Definition 3. $\mathsf{ACT}_\varphi(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie))$ iff $\exists \mathit{ar}_1, \ldots, \mathit{ar}_n,$

$$\begin{array}{l} \forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(\mathit{ar}_{\mathsf{t}}) \land (\underset{\mathsf{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}} \ \subseteq \mathit{ar}_{\mathsf{t}}) \land \\ \mathsf{ExecRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathit{ar}_{\mathsf{t}})) \land \ \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{Coh}(\mathit{ar}_{\mathsf{t}}, \mathit{ar}_{\mathsf{t}'}, (\Gamma, \bowtie)) \end{array}$$

where we define ExecRelated and Coh in Fig. 8.

Before explaining ACT, we first introduce the notations for visibility of events. In the execution $\mathcal{E}$ an origin event e is *visible to* another event e' originated from the node t (i.e., $e \underset{\mathsf{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}} e'$), if the effector of e has reached t before e' is issued. We also use $\mathsf{visible}(\mathcal{E}, \mathsf{t})$ to represent the set of origin events whose effectors have reached t.

ACT says that each node t may have its own arbitration order ar_{t}, which is a total order over the origin events on $\mathcal{E}$ visible to t. Each ar_{t} must preserve the visibility order on t (i.e., $\underset{\mathsf{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}} \ \subseteq \mathit{ar}_{\mathsf{t}}$).

On functional correctness, ACT requires that the concrete execution on node t should correspond to the execution of the abstract events following the arbitration order ar_{t} (see $\mathsf{ExecRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathit{ar}_{\mathsf{t}}))$). As defined in Fig. 8, ExecRelated says that every state in t's concrete execution can be mapped (via φ) to the state in the abstract execution trace, and that every request issued by t gets the same return value as the abstract one. The definition checks on every prefix $\mathcal{E}'$ of the concrete trace $\mathcal{E}$. We use $\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright \mathit{ar}$ to represent a serialization of the set $\mathsf{visible}(\mathcal{E}', \mathsf{t})$ following the total order ar. Then $\mathsf{aexec}(\Gamma, \mathcal{S}_a, \mathcal{E})$ executes the sequence of abstract operations on $\mathcal{E}$, starting from the initial abstract state $\mathcal{S}_a$. It returns the final state $\mathcal{S}_a'$ and the return value n' of the last operation. Similarly, we use $\mathsf{exec\_st}(\mathcal{S}, \mathcal{E})$ to represent the final state generated by executing the effectors on $\mathcal{E}$ from the initial state $\mathcal{S}$. We omit their definitions here.

The arbitration orders on different nodes can be different, but must be coherent to guarantee SEC. The coherence requires that conflicting actions are given the same arbitration order by all the nodes (see $\mathsf{Coh}(\mathit{ar}_{\mathsf{t}}, \mathit{ar}_{\mathsf{t}'}, (\Gamma, \bowtie))$), as defined in

Fig. 8). Combined with ($\xrightarrow[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \subseteq ar_{\mathsf{t}}$) for every t, Coh actually ensures that ar_{t} must agree with other nodes' visibility orders on *conflicting* operations.

Properties of ACC. Our ACC guarantees SEC. Below we first define the convergence of event traces in Def. 4. It is a property about the concrete level execution only, and it captures the SEC requirement.

Definition 4. $\mathsf{CvT}_{\varphi}(\mathcal{E}, \mathcal{S})$ iff

$$\begin{aligned}&\forall \mathcal{E}', \mathcal{E}'', \mathsf{t}, \mathsf{t}'.\ \mathcal{E}' \leqslant \mathcal{E} \wedge \mathcal{E}'' \leqslant \mathcal{E} \wedge \mathsf{visible}(\mathcal{E}', \mathsf{t}) = \mathsf{visible}(\mathcal{E}'', \mathsf{t}') \\ &\qquad \implies \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}''|_{\mathsf{t}'}))\end{aligned}$$

$\mathsf{CvT}_{\varphi}(\mathcal{E}, \mathcal{S})$ says, whenever the two nodes t and t′ see the same set of operations, executing the corresponding sub-traces on t and t′ results in states corresponding to the same abstract state. Note we allow t and t′ to pick different time points in the execution trace $\mathcal{E}$ (see $\mathcal{E}' \leqslant \mathcal{E}$ and $\mathcal{E}'' \leqslant \mathcal{E}$, which says $\mathcal{E}'$ and $\mathcal{E}''$ can be different prefixes of $\mathcal{E}$), because there is no global time on the nodes. Besides, the two resulting states do not have to be identical. Instead, they only need to be mapped to the same abstract state. This way we allow the implementation-dependent data in the concrete states to be different. The convergence of an object Π, written as $\mathsf{Cv}_{\varphi}(\Pi)$, requires every event trace $\mathcal{E}$ of Π to satisfy CvT.

Lemma 5. If $\mathsf{ACC}_{\varphi}(\Pi, (\Gamma, \bowtie))$, then $\mathsf{Cv}_{\varphi}(\Pi)$.

Another important property of $\mathsf{ACC}_{\varphi}(\Pi, (\Gamma, \bowtie))$ is its *compositionality*, as we explained in Sec. 2.4.

6 Abstraction Theorem

To simplify the reasoning of clients of CRDTs, we give an *abstract operational semantics* of client programs, based on the abstract specification $(\Gamma, \bowtie)$. The abstract version of the client program is defined below:

$$(AProg)\ \mathbb{P} ::= \textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \parallel \ldots \parallel C_n$$

It is safe to reason about clients at the abstract level as long as the CRDT implementation Π contextually refines $(\Gamma, \bowtie)$.

Definition 6. $\Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$ iff, for all clients $C_1, \ldots, C_n$ and state $\mathcal{S} \in dom(\varphi)$, for all $\lfloor\mathcal{E}\rfloor$ and σ_c,

$$\begin{aligned}&(\lfloor\mathcal{E}\rfloor, \sigma_c) \in \mathcal{T}_{\mathsf{s}}(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, \mathcal{S}) \implies \\ &\qquad (\mathsf{obsv}_{\varphi}(\lfloor\mathcal{E}\rfloor), \sigma_c) \in \mathcal{T}_{\mathsf{s}}(\textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \parallel \ldots \parallel C_n, \varphi(\mathcal{S}))\end{aligned}$$

Informally, $\Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$ says, for any clients and initial states, executing the clients with Π does not generate more *observable behaviors* than the execution using $(\Gamma, \bowtie)$ in the abstract operational semantics (presented below). $\mathcal{T}_{\mathsf{s}}(P, \mathcal{S})$ and $\mathcal{T}_{\mathsf{s}}(\mathbb{P}, \mathcal{S})$ are defined similarly as $\mathcal{T}(P, \mathcal{S})$ (Sec. A), but they additionally record the final client state σ_c. Also in the extended trace $\lfloor\mathcal{E}\rfloor$ they record all the intermediate *object* states together with the events. The function $\mathsf{obsv}_{\varphi}(\lfloor\mathcal{E}\rfloor)$ maps the extended trace $\lfloor\mathcal{E}\rfloor$ in the concrete semantics to an abstract trace. Each concrete event is mapped to an abstract one, and every recorded object state is mapped through the state abstraction function φ to an abstract object state.

Theorem 7 (Abstraction Theorem).
$\mathsf{ACC}_{\varphi}(\Pi, (\Gamma, \bowtie)) \iff \Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$.

Abstract operational semantics describes the execution of programs in the form of **with** $(\Gamma, \bowtie)$ **do** $C_1 \parallel \ldots \parallel C_n$. Clients are executed following the interleaving semantics. On each node, we always keep the initial object state $\mathcal{S}_0$. We also maintain a sequence ξ_{t} of the abstract operations that the node t has received. We can view ξ_{t} as a *runtime representation* of the arbitration order ar_{t} used in ACC. Given $\mathcal{S}_0$ and ξ_{t}, we can always generate the current object state on the fly by executing all the operations on ξ_{t} from $\mathcal{S}_0$.

When a node issues an operation, it puts the operation at the very end of its local ξ to get a new sequence ξ'. This reflects the preservation of the visibility order, as required in ACC, because at this moment the node has seen all the operations on ξ and therefore they all need to be ordered before the new operation. We also start from $\mathcal{S}_0$ and execute all the operations on ξ' to get the return value of the last operation. The node then broadcasts the operation itself (instead of effectors) to all the other nodes.

When a node receives an operation sent from others, it can non-deterministically insert the operation into any position of the local sequence ξ, as long as the resulting ξ' is *coherent* with every other ξ_{t} on node t. The coherence requirement is similar to $\mathsf{Coh}(ar_{\mathsf{t}}, ar_{\mathsf{t}'}, (\Gamma, \bowtie))$ defined in Fig. 8. It requires that conflicting operations follow the same order in all sequences (ξ' and all the other ξ_{t}). If we cannot find an insertion position in the local ξ so that the resulting ξ' satisfies the coherence requirement, the execution gets stuck. The semantics of the program can be viewed as the set of the stuck-free executions.

Since the operation lists ξ on all nodes must be coherent during the execution, we can prove that the abstract semantics inherently guarantees the convergence of the abstract object states. Then, the contextual refinement $\Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$ can ensure $\mathsf{Cv}_{\varphi}(\Pi)$, the convergence of the concrete object. With the Abstraction Theorem (Thm 7), we can derive Lem. 5 again: $\mathsf{ACC}_{\varphi}(\Pi, (\Gamma, \bowtie))$ can ensure $\mathsf{Cv}_{\varphi}(\Pi)$ too.

7 Program Logic for Client Verification

To reason about clients using a CRDT object Π, we apply the Abstraction Theorem, and verify the clients using the more abstract object specifications $(\Gamma, \bowtie)$ instead.

We design a Hoare-style program logic to verify *functional correctness* of client programs, specified in the form of pre- and post-conditions. The top level judgment is in the form of $\vdash \{\mathcal{P}\}\textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \parallel \ldots \parallel C_n\{\mathcal{Q}\}$, where $\mathcal{P}$ and $\mathcal{Q}$ are traditional Hoare-logic state assertions over both client and object states. To enable thread-local reasoning, we borrow ideas from shared-memory concurrency verification and base our logic on rely-guarantee reasoning [11]. Each C_{t} is verified in the form of $R, G; \Gamma, \bowtie \vdash_{\mathsf{t}} \{p\}C_{\mathsf{t}}\{q\}$, where R and G

```
                   {s = a}
                 || u := read();         || v := read();
addAfter(a, b);  || if (b ∈ u)           || if (c ∈ v)
                 ||   addAfter(a, c);    ||   addAfter(c, d);
                 || x := read();         || y := read();
          {d ∈ x ⇒ (s = x = acdb) ∧ (y = x ∨ y = acd)}
```

Figure 9. Correctness of a client program of RGA.

$$
\begin{array}{lrcl}
(\mathit{StateAssn}) & \mathcal{P}, \mathcal{Q} & ::= & B \mid \neg\mathcal{P} \mid \mathcal{P} \wedge \mathcal{Q} \mid \mathcal{P} \vee \mathcal{Q} \mid \ldots \\
(\mathit{Assn}) & p, q & ::= & \mathcal{P} \mid \mathsf{emp} \mid [\alpha]^i_{\mathsf{t}} \mid \boxed{\alpha}^i_{\mathsf{t}} \mid p \sqcup q \mid p \ltimes [\alpha]^i_{\mathsf{t}} \\
 & & \mid & p \ltimes \boxed{\alpha}^i_{\mathsf{t}} \mid (p, \bowtie) \ltimes [\alpha]^i_{\mathsf{t}} \mid (p, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}} \\
 & & \mid & p \Rrightarrow q \mid \neg p \mid p \wedge q \mid p \vee q \mid p \Rightarrow q \mid \ldots \\
(\mathit{RGAssn}) & R, G & ::= & \mathsf{Emp} \mid p \rightsquigarrow [\alpha]^i_{\mathsf{t}} \mid R \vee R \mid R \Rightarrow R \mid \ldots
\end{array}
$$

Figure 10. Syntax of the assertion language.

are rely and guarantee assertions, specifying the interactions between the current thread t and its environment threads.

The key challenge for the logic is to deal with the *weak* behaviors produced by the abstract semantics in Sec. 6, where client threads can reorder actions, which is reminiscent of weak memory models of languages like C11.

A motivating example. Figure 9 shows a client program of RGA and its specification. The precondition says the initial list s is a. The postcondition shows that x and y must be equal, if all the operations have been applied before the reads. It also tells which values x and y may read. Since we do *not* assume causal delivery, when the thread t_3 receives `addAfter(a,c)` from the thread t_2, it may *not* have received `addAfter(a,b)` from the thread t_1, though `addAfter(a,c)` is issued only after t_2 receives `addAfter(a,b)`. As a result, it is possible that y reads acd. But, when t_3 finally receives `addAfter(a,b)`, it must insert `addAfter(a,b)` *before* `addAfter(a,c)` (in the abstract semantics) to restore the causality (required by the coherence check). It is impossible for y to read abcd.

Assertions. It seems difficult to use traditional state assertions to express the insertion of an action into the past execution. Our idea is to introduce action assertions. We extend the syntax of Hoare logic assertions, p, with several new assertion forms, to specify the set of actions (originate from either the current thread t_c or its environment) and their orders of which t_c has knowledge at each program point. Figure 10 gives the syntax of our assertion language.

The assertions $[\alpha]^i_{\mathsf{t}}$ and $\boxed{\alpha}^i_{\mathsf{t}}$ describe singleton action sets containing only the action α. The former says the action α (with ID i) has been issued from its origin t, but we do not care whether it's on the way or it has arrived at the current node, while the latter says the current node has received α. We may omit the superscript action ID in an assertion when it is clear from the context what the action denotes. For the motivating example of Fig. 9, after t_3 succeeds in the check c ∈ v, its assertion must contain $\boxed{\texttt{addAfter(a,c)}}_{\mathsf{t}_2}$, but only $[\texttt{addAfter(a,b)}]_{\mathsf{t}_1}$.

We write emp for an empty action set. The assertion $p \sqcup q$ allows us to merge two action sets without enforcing new ordering. It can be used to describe non-conflicting actions. For instance, $[\texttt{addAfter(a,b)}]_{\mathsf{t}_1} \sqcup \boxed{\texttt{remove(e)}}_{\mathsf{t}_2}$ says `addAfter(a,b)` and `remove(e)` can be ordered either way. It can also describe a set of conflicting but concurrently issued actions, so that we do not need to enumerate all the possible execution traces. For instance, when the program (`addAfter(a, b); || addAfter(a, c)`) terminates, we have $\boxed{\texttt{addAfter(a,b)}}_{\mathsf{t}_1} \sqcup \boxed{\texttt{addAfter(a,c)}}_{\mathsf{t}_2}$.

We use $p \ltimes [\alpha]^i_{\mathsf{t}}$, $p \ltimes \boxed{\alpha}^i_{\mathsf{t}}$, $(p, \bowtie) \ltimes [\alpha]^i_{\mathsf{t}}$ and $(p, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}}$ to add a new action α and some new orders about α. The assertion $p \ltimes [\alpha]^i_{\mathsf{t}}$ requires α to be ordered after all the actions in p, while $(p, \bowtie) \ltimes [\alpha]^i_{\mathsf{t}}$ enforces the ordering between α and only the actions which have arrived (e.g., boxed actions) in the current view of p and conflict ($\bowtie$) with α. The assertions $p \ltimes \boxed{\alpha}^i_{\mathsf{t}}$ and $(p, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}}$ have similar meanings, but they also say that α has arrived at the current node. For the thread t_3 of Fig. 9, if the test of c ∈ v is true, it knows the following p_c: $[\texttt{addAfter(a,b)}]_{\mathsf{t}_1} \ltimes \boxed{\texttt{addAfter(a,c)}}_{\mathsf{t}_2}$. It says, t_3 can infer that `addAfter(a,b)` must be inserted before `addAfter(a,c)` even though `addAfter(a,b)` may not have arrived at t_3. After t_3 calls `addAfter(c,d)`, the assertion becomes $(p_c, \bowtie) \ltimes \boxed{\texttt{addAfter(c,d)}}_{\mathsf{t}_3}$. Here t_3 adds only the ordering between the conflicting `addAfter(a,c)` and `addAfter(c,d)`.

It is always safe to discard some ordering information. That is, $(p \ltimes [\alpha]^i_{\mathsf{t}}) \Rightarrow (p \sqcup [\alpha]^i_{\mathsf{t}})$ holds. It is also safe to branch on the ordering of actions:

$$([\alpha]^i_{\mathsf{t}} \sqcup [\alpha']^j_{\mathsf{t}'}) \Rightarrow [\alpha]^i_{\mathsf{t}} \ltimes [\alpha']^j_{\mathsf{t}'} \vee [\alpha']^j_{\mathsf{t}'} \ltimes [\alpha]^i_{\mathsf{t}}$$

Standard state assertions, $\mathcal{P}$, can be lifted to action assertions. A set of partially ordered actions satisfies $\mathcal{P}$ if all the final states resulting from executing these actions satisfy $\mathcal{P}$ (as a state assertion). For instance, the following holds:

$$(\mathsf{s} = \mathsf{a} \wedge \mathsf{emp}) \sqcup (\boxed{\texttt{addAfter(a,b)}}_{\mathsf{t}_1} \ltimes \boxed{\texttt{addAfter(a,c)}}_{\mathsf{t}_2}) \Rightarrow \mathsf{s} = \mathsf{acb}$$

When executing the actions, we only execute the actions that have arrived in the current view. As a result,

$$(\mathsf{s} = \mathsf{a} \wedge \mathsf{emp}) \sqcup ([\texttt{addAfter(a,b)}]_{\mathsf{t}_1} \ltimes \boxed{\texttt{addAfter(a,c)}}_{\mathsf{t}_2}) \Rightarrow \mathsf{s} = \mathsf{ac} \vee \mathsf{s} = \mathsf{acb}$$

The assertion $p \Rrightarrow q$ specifies that the states satisfying q result from receiving and applying all the actions on the way in p. It is used when the whole client program terminates (see the PAR rule in Fig. 11, where in Q_{t} all the actions must have arrived at node t). For instance, the following holds:

$$(\mathsf{s} = \mathsf{a} \wedge \mathsf{emp}) \sqcup ([\texttt{addAfter(a,b)}]_{\mathsf{t}_1} \ltimes \boxed{\texttt{addAfter(a,c)}}_{\mathsf{t}_2}) \Rrightarrow \mathsf{s} = \mathsf{acb}$$

Rely/guarantee assertions. The assertions R and G (see Fig. 10) specify the interface between a thread and its environment. The guarantee G specifies the invocations of object actions made by the thread itself. The rely R specifies the thread's expectations of the object actions that originate from its environment.

The assertion Emp says there is no action issued. The assertion $p \rightsquigarrow [\alpha]^i_{\mathsf{t}}$ says that t invokes the action α when p holds, i.e., p is the prerequisite for t to issue the request α.

Threads can cooperate if the rely condition of a thread t is implied by the guarantee of the other t′. We stabilize the assertion p at each program point of t under its rely R, so that it is resistant to interference from the environment. To stabilize an assertion p with respect to $R = (p' \rightsquigarrow [\alpha]^i_{\mathsf{t}'})$, we do the following steps:

(1) Check that the prerequisite p' for the invocation of α is met at p. This requires p to contain the knowledge of all the received actions $\boxed{\alpha'}^{j}_{\mathsf{t}''}$ in p', though it is possible that some of these actions have not arrived at the current node yet (i.e. they are in brackets in p).
(2) If the check in (1) is passed, we add $[\alpha]^i_{\mathsf{t}'}$ to the action set of the current node. We do not need to know whether or not α has arrived at the current node.
(3) The knowledge of the action ordering at the current node should also be expanded. For those α' in p' that are prerequisite of α and are also in conflict ($\bowtie$) with α, α' should be ordered before α on all the nodes, since we require all the nodes to observe the same ordering of conflicting actions.

For instance, $p \overset{\text{def}}{=} [\texttt{addAfter(a,b)}]_{\mathsf{t}_1}$ is stabilized to the following p_1 under R_1, for the RGA object:

$$\begin{aligned} R_1 &\overset{\text{def}}{=} \boxed{\texttt{addAfter(a,b)}}_{\mathsf{t}_1} \rightsquigarrow [\texttt{addAfter(a,c)}]_{\mathsf{t}_2} \\ p_1 &\overset{\text{def}}{=} p \vee ([\texttt{addAfter(a,b)}]_{\mathsf{t}_1} \ltimes [\texttt{addAfter(a,c)}]_{\mathsf{t}_2}) \end{aligned} \tag{7.1}$$

In the inference rules (see the CALL-R and LOCAL rules in Fig. 11), we use the stability check $\mathsf{Sta}(p, R, \bowtie)$. It is passed by stabilized assertions only. For (7.1), $\mathsf{Sta}(p_1, R_1, \bowtie)$ holds.

Inference rules. Figure 11 presents the key inference rules. The PAR rule is almost the standard parallel composition rule in rely-guarantee reasoning. We let each thread start its execution from an empty action set (see $\mathcal{P} \wedge \mathsf{emp}$). At the end, we derive the state assertion $\mathcal{Q}_{\mathsf{t}}$ by receiving all the actions in q_{t} (see $q_{\mathsf{t}} \Rrightarrow \mathcal{Q}_{\mathsf{t}}$). In the state assertions, we merge the client state and the object state into one, assuming their variables are from different name spaces. We also assume that the rely/guarantee conditions specify object states only.

In the CALL rule, we first compute the return value n' of the call, using $p \overset{\mu}{\twoheadrightarrow} n'$, where $\mu \in \mathit{AbsState} \rightarrow \mathit{Val}$ is the return value generator of $\Gamma(f, n)$. $p \overset{\mu}{\twoheadrightarrow} n'$ says, applying μ over any final state of executing the actions following the specified order in p returns n'. We then assign n' to x. The assertion q holds after the assignment, following the forward assignment rule in Hoare logic. Finally we add the newly generated action α to the action set in q, and use the resulting assertion $(q, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}}$ as the postcondition. The invocation of α following q (i.e., $q \rightsquigarrow [\alpha]^i_{\mathsf{t}}$) needs to satisfy G. The superscript i needs to be the same as specified in G.

$$\frac{\forall \mathsf{t} \in [1..n]: \quad R_{\mathsf{t}}, G_{\mathsf{t}}; \Gamma, \bowtie \vdash_{\mathsf{t}} \{\mathcal{P} \wedge \mathsf{emp}\} C_{\mathsf{t}} \{q_{\mathsf{t}}\} \qquad (\bigvee_{\mathsf{t}' \neq \mathsf{t}} G_{\mathsf{t}'}) \Rightarrow R_{\mathsf{t}} \qquad q_{\mathsf{t}} \Rrightarrow \mathcal{Q}_{\mathsf{t}}}{\vdash \{\mathcal{P}\} \textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \parallel \ldots \parallel C_n \{\bigwedge_{\mathsf{t}} \mathcal{Q}_{\mathsf{t}}\}} \ (\textsc{par})$$

$$\frac{p \Rightarrow E = n \qquad \mathsf{split}(\Gamma(f, n)) = (\mu, \alpha) \qquad p \overset{\mu}{\twoheadrightarrow} n' \qquad x = n' \wedge \exists v.\, p[v/x] \Rightarrow q \qquad q \rightsquigarrow [\alpha]^i_{\mathsf{t}} \Rightarrow G}{\mathsf{Emp}, G; \Gamma, \bowtie \vdash_{\mathsf{t}} \{p\} x := f(E) \{(q, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}}\}} \ (\textsc{call})$$

$$\frac{\mathsf{Emp}, G; \Gamma, \bowtie \vdash_{\mathsf{t}} \{p\} x := f(E) \{q\} \qquad \mathsf{Sta}(\{p, q\}, R, \bowtie) \qquad \mathsf{cmt\text{-}closed}(\{p, q\})}{R, G; \Gamma, \bowtie \vdash_{\mathsf{t}} \{p\} x := f(E) \{q\}} \ (\textsc{call-r})$$

$$\frac{R', G'; \Gamma, \bowtie \vdash_{\mathsf{t}} \{p'\} C \{q'\} \qquad p \Rightarrow p' \qquad R \Rightarrow R' \qquad q' \Rightarrow q \qquad G' \Rightarrow G}{R, G; \Gamma, \bowtie \vdash_{\mathsf{t}} \{p\} C \{q\}} \ (\textsc{csq})$$

$$\frac{\mathsf{Sta}(p, R, \bowtie) \qquad \mathsf{cmt\text{-}closed}(p)}{R, G; \Gamma, \bowtie \vdash_{\mathsf{t}} \{p\} x := E \{\exists v.\, x = E[v/x] \wedge p[v/x]\}} \ (\textsc{local})$$

Figure 11. Selected inference rules.

One may wonder that it is too restrictive for the CALL rule to require the argument n and return value n' to be constant values. When the precondition p cannot determine a unique argument or return value (i.e., $(p \Rightarrow E = n)$ or $(p \overset{\mu}{\twoheadrightarrow} n')$ does not hold), we can first apply a standard disjunction rule to branch on p, and apply the CALL rule on each branch.

Note that in this step we only reason about the behavior of the function call without considering the environment. Therefore we use an empty rely condition Emp here. To allow a weaker R, we can apply the CSQ rule to stabilize the postcondition by weakening $(q, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}}$. Then we apply CALL-R rule, which requires the pre- and post-conditions be stable with respect to R and satisfy cmt-closed. Here $\mathsf{cmt\text{-}closed}(p)$ iff p is preserved after receiving one or more actions that are already issued in p.

The LOCAL rule allows us to reason about local computation of a thread. The pre- and post-conditions are the same as those in the forward assignment rule in Hoare logic.

Verification of the motivating example. In Fig. 12 we sketch the proof of t_3 in the motivating example of Fig. 9. More examples are in Appendix F.

We first define the rely/guarantee conditions of each thread. G_{t_1} says that the thread t_1 guarantees the invocation of α_b unconditionally. G_{t_2} says that t_2 calls α_c after it receives α_b. Similarly, G_{t_3} says that t_3 calls α_d after it receives α_c. Here we write $\diamond\boxed{\alpha}^i_{\mathsf{t}}$ for $\boxed{\alpha}^i_{\mathsf{t}} \sqcup \mathsf{true}$.

By the PAR rule, we only need to verify each thread independently. For thread t_3, we first stabilize p_a under R_{t_3}, resulting in the assertion (1) in Fig. 12. After finding $\mathsf{c} \in \mathsf{v}$, we

$$
\begin{array}{llll}
p_a \overset{\text{def}}{=} (\mathsf{s} = \mathsf{a}) \wedge \mathsf{emp} & & \alpha_b \overset{\text{def}}{=} \mathtt{addAfter(a,b)} \\
\alpha_c \overset{\text{def}}{=} \mathtt{addAfter(a,c)} & & \alpha_d \overset{\text{def}}{=} \mathtt{addAfter(c,d)} \\
G_{\mathsf{t}_1} \overset{\text{def}}{=} \mathsf{true} \rightsquigarrow [\alpha_b]_{\mathsf{t}_1} & & R_{\mathsf{t}_1} \overset{\text{def}}{=} G_{\mathsf{t}_2} \vee G_{\mathsf{t}_3} \\
G_{\mathsf{t}_2} \overset{\text{def}}{=} (\Diamond \boxed{\alpha_b}_{\mathsf{t}_1}) \rightsquigarrow [\alpha_c]_{\mathsf{t}_2} & & R_{\mathsf{t}_2} \overset{\text{def}}{=} G_{\mathsf{t}_1} \vee G_{\mathsf{t}_3} \\
G_{\mathsf{t}_3} \overset{\text{def}}{=} (\Diamond \boxed{\alpha_c}_{\mathsf{t}_2}) \rightsquigarrow [\alpha_d]_{\mathsf{t}_3} & & R_{\mathsf{t}_3} \overset{\text{def}}{=} G_{\mathsf{t}_1} \vee G_{\mathsf{t}_2}
\end{array}
$$

$\{p_a \vee p_a \sqcup [\alpha_b]_{\mathsf{t}_1} \vee p_a \sqcup ([\alpha_b]_{\mathsf{t}_1} \ltimes [\alpha_c]_{\mathsf{t}_2})\}$ (1)

```
v := read();
if (c ∈ v)
```

  $\{p_a \sqcup ([\alpha_b]_{\mathsf{t}_1} \ltimes \boxed{\alpha_c}_{\mathsf{t}_2})\}$ (2)

```
  addAfter(c, d);
```

  $\{p_a \sqcup ([\alpha_b]_{\mathsf{t}_1} \ltimes \boxed{\alpha_c}_{\mathsf{t}_2} \ltimes \boxed{\alpha_d}_{\mathsf{t}_3})\}$ (3)

```
y := read();
```

$\{\mathsf{s} = \mathsf{acdb} \Rightarrow \mathsf{y} = \mathsf{s} \vee \mathsf{y} = \mathsf{acd}\}$ (4)

Figure 12. Verification of the client with RGA.

can discard the branches where α_c is not arrived. So we get the assertion (2). Then, t_3 calls `addAfter(c,d)`. The immediate post-condition $(p \sqcup ([\alpha_b]_{\mathsf{t}_1} \ltimes \boxed{\alpha_c}_{\mathsf{t}_2}), \bowtie) \ltimes \boxed{\alpha_d}_{\mathsf{t}_3}$ can be derived from the CALL rule. Using the CSQ rule, we weaken it to the assertion (3), which is stable and cmt-closed. Finally we get the assertion (4). It has the branch $\mathsf{y} = \mathsf{acd}$ because it is possible that t_3 has not yet received α_b by the read.

Logic soundness: If $\vdash \{\mathcal{P}\}\mathbb{P}\{Q\}$, then $\models \{\mathcal{P}\}\mathbb{P}\{Q\}$. The Hoare triple $\models \{\mathcal{P}\}\mathbb{P}\{Q\}$ is defined using the abstract semantics in Sec. 6. The formal model and the soundness proofs are in Appendix E.

Invariant-based reasoning. Our logic can be easily extended to verify object invariants. We can add an extra invariant assertion I in the judgment, which will be in the form of $I, R, G; \Gamma, \bowtie \vdash_{\mathsf{t}} \{p\}C\{q\}$. Then in the CALL-R rule in Fig. 11 we add the extra requirements $p \Rightarrow I$ and $q \Rightarrow I$.

8 Verifying CRDT Implementations

Our proof method for ACC asks users to first provide specifications $\rightarrowtail$ and $\mathcal{V}$ about implementations:

$$
\begin{array}{lll}
\rightarrowtail \; \in \; \mathscr{P}(\mathit{Effector} \times \mathit{Effector}) & & \text{(the time-stamp order)} \\
\mathcal{V} \; \in \; \mathit{LocalState} \to \mathscr{P}(\mathit{Effector}) & & \text{(the view function)}
\end{array}
$$

The time-stamp order $\rightarrowtail$ is a *partial order between effectors*. It describes the algorithm's conflict-resolution strategy, e.g., the write with a larger time-stamp wins. For the RGA algorithm, we instantiate $\rightarrowtail$ as follows:

$$
\begin{array}{rl}
\delta \rightarrowtail \delta' \text{ iff } & \exists \mathsf{a}, \mathsf{i}, \mathsf{b}, \mathsf{a}', \mathsf{i}', \mathsf{b}'. \; \delta = \mathtt{AddAft(a,i,b)} \\
& \wedge \; (\delta' = \mathtt{AddAft(a',i',b')} \wedge \mathtt{i} < \mathtt{i'} \\
& \quad \vee \; \delta' = \mathtt{Rmv(a)} \; \vee \; \delta' = \mathtt{Rmv(b)})
\end{array}
$$

Here $\rightarrowtail$ orders the `AddAft` effectors by comparing their time-stamps. It also orders an `AddAft` before the conflicting `Rmv` effectors (which is not time-stamped). Note that $\rightarrowtail$ is specified at the implementation level. One should not confuse it with the won-by order $\blacktriangleleft$ over abstract operations, which we introduce in Sec. 2.4 and Sec. 9.

The view function $\mathcal{V}$ maps each local state $\mathcal{S}$ to a set of effectors that must have been applied before reaching $\mathcal{S}$. With it, our proof method can be local, in that the reasoning of each execution step relies on the current local state on the node only, without referring to the execution traces. For the RGA algorithm, $\mathcal{V}$ is instantiated as follows:

$$
\begin{array}{rl}
\mathcal{V}(\mathcal{S}) \overset{\text{def}}{=} \{\, \delta \mid & \exists \mathsf{a}, \mathsf{i}, \mathsf{b}. \; (\mathsf{a}, \mathsf{i}, \mathsf{b}) \in \mathcal{S}(\mathsf{N}) \;\wedge\; \delta = \mathtt{AddAft(a,i,b)} \\
& \vee \; \exists \mathsf{a}. \; \mathsf{a} \in \mathcal{S}(\mathsf{T}) \;\wedge\; \delta = \mathtt{Rmv(a)} \,\}
\end{array}
$$

Our proof method, $\text{CRDT-TS}_{\varphi}(\Pi, (\Gamma, \bowtie), \rightarrowtail, \mathcal{V})$, is a conjunction of the following proof obligations:

- Commutative effectors: the effectors generated by Π are all commutative.
- Same return value: the corresponding operations in Π and Γ have the same return value if executed at φ-related states.
- State correspondence: starting from φ-related states $\mathcal{S}$ and $\mathcal{S}_a$, executing a *valid* effector δ (generated from Π) and the corresponding abstract operation should lead to φ-related states. δ is valid if $\rightarrowtail$ does not order it before any δ' visible from $\mathcal{S}$, i.e. $\delta' \in \mathcal{V}(\mathcal{S})$.
- Some simple well-formedness checks for $\rightarrowtail$ and $\mathcal{V}$ to ensure the user-specified $\rightarrowtail$ and $\mathcal{V}$ make sense.

Theorem 8.
$\text{CRDT-TS}_{\varphi}(\Pi, (\Gamma, \bowtie), \rightarrowtail, \mathcal{V}) \implies \text{ACC}_{\varphi}(\Pi, (\Gamma, \bowtie))$.

Examples. Using Theorem 8, we have verified seven CRDT algorithms [19], including the replicated counter (with both increment and decrement operations), the grow-only set, the last-writer-wins (LWW) register, the LWW-element set, the 2P-set, the continuous sequence, and the replicated growable array (RGA). To verify algorithms whose $\bowtie$ is empty (such as the counter), we let $\rightarrowtail$ be $\emptyset$ and $\mathcal{V}$ be $\lambda \mathcal{S}. \, \emptyset$. Proofs of the examples are in Appendix H.

Using the verification framework. Our verification framework consists of the program logic (in Sec. 7) and the proof method (in Sec. 8). As Fig. 1 shows, one needs to do the following to verify a whole program **let** Π **in** $C_1 \parallel \ldots \parallel C_n$:

- Provide the specifications for CRDTs. The operation specification Γ is the same as the one for sequential data types. It is also easy to come up with the conflict relation $\bowtie$, which is between all the non-commutative abstract operations in Γ.
- Apply the program logic for client reasoning. Similar to standard rely-guarantee reasoning, the user needs to provide the rely/guarantee conditions, intermediate assertions, and do the proofs following the logic rules.
- Apply the proof method for CRDT implementations. All one needs to do is to provide $\rightarrowtail$ and $\mathcal{V}$, and prove the set of proof obligations. The proof obligations are all first-order formulae. They do not universally quantify over execution traces, but only over states and

effectors. Thus they can be discharged without induction, and can potentially be discharged by SMT solvers.

9 X-Wins CRDTs

Algorithms like add-wins sets and remove-wins sets resolve conflicts following a specific X-wins strategy, while the operation X wins only when its effect is not canceled. We generalize ACC to support these algorithms, by enforcing the X-wins strategy specified using the won-by ($\blacktriangleleft$) and canceled-by ($\rhd$) relations. Like $\bowtie$ (see Fig. 7), they are also binary relation over actions. The full specification is now a quadruple $(\Gamma, \bowtie, \blacktriangleleft, \rhd)$.

For add-wins sets, `add(x)` wins over concurrent `remove(x)` (`remove(x)` $\blacktriangleleft$ `add(x)`), but it can also be canceled by subsequent `remove(x)` (`add(x)` $\rhd$ `remove(x)`); while for remove-wins sets, we have the inverse.

$\blacktriangleleft$ and $\rhd$ can only relate conflicting operations, that is, $\blacktriangleleft \subseteq \bowtie$ and $\rhd \subseteq \bowtie$. Also $\rhd$ should be valid in that α' indeed nullifies the effects of α if $\alpha \rhd \alpha'$. Like $\bowtie$, we also overload $\blacktriangleleft$ and $\rhd$ over operations and events.

We generalize ACC with the extended specification, and define $\mathsf{XACC}_\varphi(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$. It requires every trace $\mathcal{E}$ of Π to satisfy XACT if $\mathsf{causalDelivery}(\mathcal{E})$. Here we assume causal delivery of messages, which is required by both add-wins and remove-wins sets. It says, if an origin event e_1 happens before another origin event e_2, then for any node t the effector of e_1 reaches t earlier than that of e_2.

Definition 9. $\mathsf{XACT}_\varphi(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$ iff $\exists ar_1, \ldots, ar_n$,
$\forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(ar_\mathsf{t}) \wedge (\xrightarrow[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \subseteq ar_\mathsf{t})$
$\wedge\ \mathsf{PresvCancel}(ar_\mathsf{t}, \mathsf{t}, \mathcal{E}, (\Gamma, \rhd)) \wedge \mathsf{ExecRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_\mathsf{t}))$
$\wedge\ \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((ar_\mathsf{t}, ar_{\mathsf{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$
where we define RCoh in Fig. 13.

XACT (see Def. 9) is similar to ACT, but it enforces the more relaxed coherence relation RCoh between the arbitration orders on different nodes. As defined in Fig. 13, RCoh requires that the arbitration orders ar_t and $ar_{\mathsf{t}'}$ of the nodes t and t′ enforce the same ordering for conflicting events e_0 and e_1, if neither e_0 or e_1 are canceled (i.e., $\{e_0, e_1\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}', \mathsf{t}, (\Gamma, \rhd)) \cap \mathsf{nc\text{-}vis}(\mathcal{E}'', \mathsf{t}', (\Gamma, \rhd))$). Moreover, the ordering must follow the won-by order $\blacktriangleleft$ if these two events are concurrent (i.e., neither one happens before the other). It is more relaxed than Coh in that, if either e_0 or e_1 is canceled by others, they can be ordered differently in ar_t and $ar_{\mathsf{t}'}$.

XACT also requires $\mathsf{PresvCancel}(ar_\mathsf{t}, \mathsf{t}, \mathcal{E}, (\Gamma, \rhd))$. It says, if e_1 is canceled by e_2 and is also visible to e_2 on certain node, the arbitration order ar_t must order e_1 before e_2.

Similar to ACC, XACC also ensures SEC, and is compositional. We prove that both the add-wins and remove-wins sets satisfy XACC.

The Abstraction Theorem. We also revise the abstract operational semantics in Sec. 6, to give clients an abstract view of the X-wins strategy. The contextual refinement $\Pi \sqsubseteq_\varphi$

$$\begin{aligned}
&\mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((ar_\mathsf{t}, ar_{\mathsf{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd)) \text{ iff } \forall \mathcal{E}', \mathcal{E}'', e_0, e_1.\\
&\quad \mathcal{E}' \leqslant \mathcal{E} \wedge \mathcal{E}'' \leqslant \mathcal{E} \ \wedge\ e_0 \bowtie_\Gamma e_1 \wedge\\
&\qquad \{e_0, e_1\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}', \mathsf{t}, (\Gamma, \rhd)) \cap \mathsf{nc\text{-}vis}(\mathcal{E}'', \mathsf{t}', (\Gamma, \rhd))\\
&\quad \Longrightarrow ((e_0, e_1) \in ar_\mathsf{t} \cap ar_{\mathsf{t}'} \vee (e_1, e_0) \in ar_\mathsf{t} \cap ar_{\mathsf{t}'}) \wedge\\
&\qquad (\mathsf{Concurrent}_{\mathcal{E}}(e_0, e_1) \wedge (e_0 \blacktriangleleft_\Gamma e_1) \Longrightarrow (e_0, e_1) \in ar_\mathsf{t})\\
&\mathsf{nc\text{-}vis}(\mathcal{E}, \mathsf{t}, (\Gamma, \rhd)) \overset{\text{def}}{=} \{e \mid e \in \mathsf{visible}(\mathcal{E}, \mathsf{t}) \wedge\\
&\qquad \neg(\exists e'.\ e' \in \mathsf{visible}(\mathcal{E}, \mathsf{t}) \wedge (e \rhd_\Gamma e') \wedge (e \xrightarrow{\mathsf{vis}}_{\mathcal{E}} e'))\}
\end{aligned}$$

Figure 13. Auxiliary definitions for XACC.

$(\Gamma, \bowtie, \blacktriangleleft, \rhd)$ is defined similarly as $\Pi \sqsubseteq_\varphi (\Gamma, \bowtie)$ (Def. 6), but uses the new abstract semantics and assumes causal delivery on concrete executions. It is equivalent to XACC.

10 Related Work

Attiya et al. [1] propose a functional correctness criterion specifically for the RGA algorithm. They do not use an operational atomic specification as we do, but instead characterize the lists' functionality axiomatically (e.g., by requiring an element be in the list if it has been inserted but not deleted). Both our ACC and their work require different nodes to take the same arbitration order between `addAfter` events. Our ACC is more general and can apply to other data types too.

Jagadeesan and Riely [10] propose a correctness criterion encoding both SEC and functional correctness for CRDTs. Their "sequential specification" is a set of legal sequential traces. It is accompanied with a dependency relation between abstract operations, which plays a similar role as our $\bowtie$ relation. Their correctness definition computes the *dependent cuts* of an execution of CRDT, which is similar to our $\mathsf{visible}(\mathcal{E}, \mathsf{t})$ projected to conflicting actions. They require all nodes to have the same arbitration orders (i.e., linearizations), but over dependent actions only. This is in spirit similar to our approach, which requires the arbitration orders of different nodes to be coherent on conflicting actions. To support add-wins sets, their linearizations view different calls to the same operation as interchangeable (a.k.a. puns). By contrast, our XACC encodes the X-wins strategy of these algorithms directly, taking the effects of cancellation into account. Similar ideas of cancellation can be found in the earlier work on checking serializability [4]. Note that Jagadeesan and Riely [10] do *not* give a proof method for *client* reasoning. Also, they verify the CRDT algorithms case by case without giving a generic proof method.

Wang et al. [21] propose RA-linearizability for CRDTs. Their specifications are *non-atomic*, and often have to expose some low-level implementation details. For instance, their specification for RGA needs the tombstone set of removed elements, and their specification for add-wins sets splits a `remove` into two abstract operations. By contrast, we use atomic and implementation-independent specifications. They also give proof methods for RA-linearizability, which contain some trace-based proof obligations such as commutativity, while our proof obligations for ACC are state-based.

Besides, they do *not* provide formal solutions for program logic for client verification.

Gotsman et al. [9] verify data integrity invariants for clients of replicated data types. They do not prove pre- and post-conditions as we do, which can be used specify more interesting functional properties. They introduce a token system with a conflict relation $\bowtie$ to relate operations that need to be causally dependent. We use the same symbol $\bowtie$ to relate non-commmutative abstract operations.

Lewchenko et al. [14] propose conflict-aware replicated data types (CARD), and design a refinement type system that enables verification of pre- and post-conditions for clients of CARD. There is also much work about general verification approaches for distributed systems and their clients (e.g., [18, 23]). Our program logic is customized for clients of CRDTs only. We can utilize certain properties (e.g., SEC) of CRDTs in the verification of clients.

Several papers (e.g., [3, 5, 6, 20, 24]) use concurrent specifications for replicated data types. On the one hand, concurrent specifications are more general than sequential specifications, so they can in principle support any replicated data types. On the other hand, it is unclear how to utilize the concurrent specifications in client reasoning.

Gomes et al. [8] verify SEC of CRDTs in Isabelle/HOL. Their method is based on global execution traces. Our proof method is local and state-based, and verifies functional correctness as well as SEC. Nagar and Jagannathan [15] verify SEC of CRDTs automatically. Their verification is parameterized with consistency policies offered by the underlying network (e.g., whether message delivery is causal). Kaki et al. [12] verify invariants for clients of replicated data types. Their approach is based on symbolic execution with a bound on concurrent operations. They also repair the invariant violations of clients by strengthening the network's consistency policies. It would be interesting to also study our ACC and our client logic with various network consistency policies.

There is also work that verifies eventual consistency and/or causal consistency by model-checking (e.g., [2, 3]), or for some particular data types such as key-value stores [13].

11 Conclusion and Future Work

We develop a theory of data abstraction for CRDTs, with independent proof methods to verify CRDT implementations and client programs respectively. Our Abstraction Theorem, as one of the key results in the theory, decomposes the verification of the two sides so that they can be done independently and modularly. It forms a semantic basis for understanding CRDTs, based on which we believe more proof techniques and tools can be developed in the future.

Limitations. This paper mostly focuses on UCR-CRDTs. For X-wins CRDTs, we formulate XACC and prove both the add-wins set and remove-wins set satisfy XACC. It might be possible to develop a general proof method for verifying XACC, similar to CRDT-TS for verifying ACC (Sec. 8), but one needs to be careful to avoid overfitting, since we do not have many interesting X-wins CRDTs to test the generality. Also, we leave the program logic for clients using X-wins CRDTs as future work. To reason about their clients, one needs to take into account the X-wins strategy specified using the won-by ($\blacktriangleleft$) and canceled-by ($\triangleright$) relations, and ensure soundness of the logic w.r.t. the new abstract operational semantics discussed in Sec. 9.

Our verification of CRDTs is done at the algorithm level. To bridge the real code with the operations defined in Π (see Fig. 6), one only needs refinement proofs for sequential programs since the real implementation code runs sequentially on individual hosts.

This paper considers only *operation-based* CRDTs. Our results may be adapted to support *state-based* CRDTs when assuming causal delivery, but it seems nontrivial to build abstractions that on the one hand reflect the algorithms' resistance to unreliable networks, and on the other hand are still useful for client reasoning. Nair et al. [16] recently propose a proof method for verifying invariant preservation of state-based replicated objects. It would be interesting to incorporate their ideas into our work.

We would also like to further test the applicability of our results by considering new operation-based CRDT algorithms (e.g., those constructed by semidirect products [22]). It is also interesting to mechanize our results in proof assistants and explore the possibility of building tools to automate the verification process.

Acknowledgments

We thank our shepherd Hongseok Yang and anonymous referees for their suggestions and comments on earlier versions of this paper. This work is supported in part by grants from National Natural Science Foundation of China (NSFC) under Grant Nos. 61922039 and 61632005.

References

[1] Hagit Attiya, Sebastian Burckhardt, Alexey Gotsman, Adam Morrison, Hongseok Yang, and Marek Zawirski. 2016. Specification and Complexity of Collaborative Text Editing. In *PODC 2016*. 259–268.

[2] Ahmed Bouajjani, Constantin Enea, Rachid Guerraoui, and Jad Hamza. 2017. On Verifying Causal Consistency. In *POPL 2017*. 626–638.

[3] Ahmed Bouajjani, Constantin Enea, and Jad Hamza. 2014. Verifying Eventual Consistency of Optimistic Replication Systems. In *POPL 2014*. 285–296.

[4] Lucas Brutschy, Dimitar Dimitrov, Peter Müller, and Martin Vechev. 2017. Serializability for Eventual Consistency: Criterion, Analysis, and Applications. In *POPL 2017*. 458–472.

[5] Sebastian Burckhardt. 2014. Principles of Eventual Consistency. *Found. Trends Program. Lang.* 1, 1-2 (Oct. 2014), 1–150.

[6] Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. 2014. Replicated Data Types: Specification, Verification, Optimality. In *POPL 2014*. 271–284.

[7] Seth Gilbert and Nancy Lynch. 2002. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services.

SIGACT News 33, 2 (June 2002), 51–59.

[8] Victor B. F. Gomes, Martin Kleppmann, Dominic P. Mulligan, and Alastair R. Beresford. 2017. Verifying strong eventual consistency in distributed systems. *PACMPL* 1, OOPSLA (2017), 109:1–109:28.

[9] Alexey Gotsman, Hongseok Yang, Carla Ferreira, Mahsa Najafzadeh, and Marc Shapiro. 2016. 'Cause I'm Strong Enough: Reasoning About Consistency Choices in Distributed Systems. In *POPL 2016*. 371–384.

[10] Radha Jagadeesan and James Riely. 2018. Eventual Consistency for CRDTs. In *ESOP 2018*. 968–995.

[11] Cliff B. Jones. 1983. Tentative Steps Toward a Development Method for Interfering Programs. *ACM Trans. Program. Lang. Syst.* 5, 4 (1983), 596–619.

[12] Gowtham Kaki, Kapil Earanky, KC Sivaramakrishnan, and Suresh Jagannathan. 2018. Safe Replication through Bounded Concurrency Verification. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 164 (2018).

[13] Mohsen Lesani, Christian J. Bell, and Adam Chlipala. 2016. Chapar: Certified Causally Consistent Distributed Key-value Stores. In *POPL 2016*. 357–370.

[14] Nicholas V. Lewchenko, Arjun Radhakrishna, Akash Gaonkar, and Pavol Černý. 2019. Sequential Programming for Replicated Data Stores. *Proc. ACM Program. Lang.* 3, ICFP, Article 106 (2019).

[15] Kartik Nagar and Suresh Jagannathan. 2019. Automated Parameterized Verification of CRDTs. In *CAV 2019*. 459–477.

[16] Sreeja S. Nair, Gustavo Petri, and Marc Shapiro. 2020. Proving the Safety of Highly-Available Distributed Objects. In *ESOP 2020*. 544–571.

[17] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, and Joonwon Lee. 2011. Replicated abstract data types: Building blocks for collaborative applications. *J. Parallel and Distrib. Comput.* 71, 3 (2011), 354 – 368.

[18] Ilya Sergey, James R. Wilcox, and Zachary Tatlock. 2017. Programming and Proving with Distributed Protocols. *Proc. ACM Program. Lang.* 2, POPL, Article 28 (2017).

[19] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. A comprehensive study of Convergent and Commutative Replicated Data Types. Research Report RR-7506, INRIA.

[20] Paolo Viotti and Marko Vukolić. 2016. Consistency in Non-Transactional Distributed Storage Systems. *ACM Comput. Surv.* 49, 1 (June 2016), 19:1–19:34.

[21] Chao Wang, Constantin Enea, Suha Orhun Mutluergil, and Gustavo Petri. 2019. Replication-aware Linearizability. In *PLDI 2019*. 980–993.

[22] Matthew Weidner, Heather Miller, and Christopher Meiklejohn. 2020. Composing and Decomposing Op-Based CRDTs with Semidirect Products. *Proc. ACM Program. Lang.* 4, ICFP, Article 94 (Aug. 2020).

[23] James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas Anderson. 2015. Verdi: A Framework for Implementing and Formally Verifying Distributed Systems. In *PLDI 2015*. 357–368.

[24] Peter Zeller, Annette Bieniusa, and Arnd Poetzsch-Heffter. 2014. Formal Specification and Verification of CRDTs. In *FORTE 2014*. 33–48.

$$
\begin{array}{rrcl}
(\mathit{NodeID}) & \mathsf{t} & \in & \mathit{Nat} \qquad (\mathit{MsgID})\ \mathit{mid} \in \mathit{Nat}\\
(\mathit{MsgSoup}) & M & \in & \mathit{MsgID} \rightharpoonup (\mathit{OpName} \times \mathit{Val}) \times \mathit{Effector}\\
(\mathit{LocalState}) & \mathcal{S} & \in & \mathit{PVar} \rightharpoonup \mathit{Val}\\
(\mathit{RtNode}) & \varsigma & ::= & (C, \mathcal{S}) \mid (\Pi, \mathcal{S}, M)\\
(\mathit{State}) & \sigma & ::= & \{\mathsf{t}_1 \rightsquigarrow \varsigma_1, \ldots, \mathsf{t}_n \rightsquigarrow \varsigma_n\}\\
(\mathit{World}) & W & ::= & (\sigma_c, \sigma_o, M)\\
(\mathit{Event}) & e & ::= & (\mathit{mid}, \mathsf{t}, (f, n, n', \delta)) \mid (\mathit{mid}, \mathsf{t}, (f, n), \delta)\\
(\mathit{Label}) & \iota & ::= & e \mid \tau \qquad (\mathit{ETrace})\ \mathcal{E} ::= \epsilon \mid e :: \mathcal{E}
\end{array}
$$

Figure 14. States and events.

$$
\frac{\forall \mathsf{t} \in [1..n].\ \sigma_c(\mathsf{t}) = (C_\mathsf{t}, \emptyset) \qquad \forall \mathsf{t} \in [1..n].\ \sigma_o(\mathsf{t}) = (\Pi, \mathcal{S}_0 \uplus \{\mathtt{cid} \rightsquigarrow \mathsf{t}\}, \emptyset)}{(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, \mathcal{S}_0) \overset{\text{load}}{\longmapsto} (\sigma_c, \sigma_o, \emptyset)}
$$

$$
\frac{\sigma_c(\mathsf{t}) = \varsigma_c \quad \sigma_o(\mathsf{t}) = \varsigma_o \quad (\varsigma_c, \varsigma_o, M) \xrightarrow{\iota}_{\mathsf{t}} (\varsigma_c', \varsigma_o', M')}{(\sigma_c, \sigma_o, M) \overset{\iota}{\longmapsto} (\sigma_c\{\mathsf{t} \rightsquigarrow \varsigma_c'\}, \sigma_o\{\mathsf{t} \rightsquigarrow \varsigma_o'\}, M')}
$$

$$
\frac{\forall \mathsf{t}.\ \sigma_c(\mathsf{t}) = (\mathbf{skip}, \mathcal{S}_c^{\mathsf{t}}) \quad \forall \mathsf{t}.\ \sigma_o(\mathsf{t}) = (\Pi, \mathcal{S}_o^{\mathsf{t}}, M)}{(\sigma_c, \sigma_o, M) \longmapsto (\mathbf{end}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathcal{S}_o^1, \ldots, \mathcal{S}_o^n))}
$$

(a) world transitions

$$
\frac{\begin{array}{lll}\llbracket E \rrbracket_{\mathcal{S}_c} = n & \Pi(f, n)(\mathcal{S}_o) = (n', \delta) & \delta(\mathcal{S}_o) = \mathcal{S}_o' \\ \mathit{mid} \notin \mathit{dom}(M_s) & M_s' = M_s \uplus \{\mathit{mid} \rightsquigarrow ((f, n), \delta)\} & \\ & M_d' = M_d \uplus \{\mathit{mid} \rightsquigarrow ((f, n), \delta)\} & \end{array}}{\begin{array}{l}((x := f(E), \mathcal{S}_c), (\Pi, \mathcal{S}_o, M_d), M_s) \xrightarrow{(\mathit{mid}, \mathsf{t}, (f, n, n', \delta))}_{\mathsf{t}} \\ \qquad ((\mathbf{skip}, \mathcal{S}_c\{x \rightsquigarrow n'\}), (\Pi, \mathcal{S}_o', M_d'), M_s')\end{array}}
$$

$$
\frac{\begin{array}{ll}M_s(\mathit{mid}) = ((f, n), \delta) & \delta(\mathcal{S}_o) = \mathcal{S}_o' \\ \mathit{mid} \notin \mathit{dom}(M_d) & M_d' = M_d \uplus \{\mathit{mid} \rightsquigarrow ((f, n), \delta)\}\end{array}}{(\varsigma_c, (\Pi, \mathcal{S}_o, M_d), M_s) \xrightarrow{(\mathit{mid}, \mathsf{t}, (f, n), \delta)}_{\mathsf{t}} (\varsigma_c, (\Pi, \mathcal{S}_o', M_d'), M_s)}
$$

(b) local transitions

Figure 15. Selected operational semantics rules.

A The Basic Technical Settings

Figure 6 shows the syntax of the language. The program P consists of n clients C running on different nodes. They share the *object* Π, which is replicated on all the nodes. Each client executes sequentially, accessing the local *client* state in the node. It can also access the *object* state through the command $x := f(E)$, which calls the operation f of the object with the argument E.

We model the object Π as a mapping from an operation name f and its argument to the actual operation over the object state. When a client calls an operation, it executes in two steps. First the operation is applied over the object state and generates a return value and an effector δ. The effector δ captures the operation's effect over the object state. It is broadcast to all nodes, including the one where the client request originates. Then the effector δ is applied on the local replica of the object data on each node. Note that on the origin node of the client request, the generation of the effector and the execution of it over the local replica are done atomically.

We give the state model in Fig. 14. The whole program configuration (a world W) consists of the client configuration σ_c, the object configuration σ_o, and a *global* message pool M consisting of all the broadcast messages, containing both delivered and undelivered. The client configuration σ_c maps a node ID t to the pair ς consisting of the client code C and its local state $\mathcal{S}$. A local state $\mathcal{S}$ is a mapping from program variables to their values. The object configuration σ_o maps a node ID t to the triple $(\Pi, \mathcal{S}, M)$, an alternative form of ς. Here Π is the set of object operations, $\mathcal{S}$ is the local replica of the object, and M is a *local* message pool containing all the incoming messages. Note that the client state and the object state on each node are disjoint. The only way that a client can access the object state is to call the operations in Π.

$$
\begin{array}{lll}
e \xrightarrow{\mathsf{t}}_{\mathcal{E}} e' & \text{iff} & \exists \mathit{mid}, \mathsf{t}_0, f, n, n', \delta. \\
 & & \quad e = (\mathit{mid}, \mathsf{t}_0, (f, n, n', \delta)) \wedge e \in \mathcal{E} \wedge \\
 & & \quad e' = (\mathit{mid}, \mathsf{t}, (f, n), \delta) \wedge e' \in \mathcal{E} \\
e \overset{\mathsf{t}}{\underset{\mathcal{E}}{\Rightarrow}} e' & \text{iff} & e \xrightarrow{\mathsf{t}}_{\mathcal{E}} e' \vee e = e' \wedge \mathsf{is\_orig}_{\mathsf{t}}(e) \wedge e \in \mathcal{E} \\
e_1 \prec^{\mathsf{t}}_{\mathcal{E}} e_2 & \text{iff} & \exists e_1', e_2'. (e_1 \overset{\mathsf{t}}{\underset{\mathcal{E}}{\Rightarrow}} e_1') \wedge (e_2 \overset{\mathsf{t}}{\underset{\mathcal{E}}{\Rightarrow}} e_2') \wedge (e_1' <_{\mathcal{E}} e_2') \\
e \overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_{\mathcal{E}} e' & \text{iff} & \left(e \prec^{\mathsf{t}}_{\mathcal{E}} e'\right) \wedge \mathsf{is\_orig}_{\mathsf{t}}(e') \\
e \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e' & \text{iff} & \exists \mathsf{t}.\, e \overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_{\mathcal{E}} e' \\
\overset{\mathsf{hb}}{\longmapsto}_{\mathcal{E}} & \overset{\text{def}}{=} & (\overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}})^{+} \\
\mathsf{visible}(\mathcal{E}, \mathsf{t}) & \overset{\text{def}}{=} & \{e \mid \exists e'.\, e \overset{\mathsf{t}}{\underset{\mathcal{E}}{\Rightarrow}} e'\} \\
es\lfloor ar & \overset{\text{def}}{=} & \begin{cases} \epsilon & \text{if } es = \emptyset \\ e :: (es' \lfloor ar) & \text{if } es = es' \uplus \{e\} \wedge \\ & \qquad \forall e' \in es'.\, (e', e) \notin ar \end{cases} \\
\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}) & \overset{\text{def}}{=} & \begin{cases} \mathcal{S} & \text{if } \mathcal{E} = \epsilon \\ \mathsf{exec\_st}(\mathcal{S}', \mathcal{E}') & \text{if } \mathcal{E} = e :: \mathcal{E}' \wedge \\ & \qquad \mathsf{eff}(e)(\mathcal{S}) = \mathcal{S}' \end{cases}
\end{array}
$$

Figure 16. Events and event traces.

We present some key operational semantics rules in Fig. 15. The first rule in Fig. 15(a) creates the initial program configuration by replicating the initial object state $\mathcal{S}_0$ on all the nodes. The third rule says the whole program terminates if each client terminates, and each node has received all the messages (therefore each local message buffer is the same with the global one).

The second rule says whenever a node takes one step, the whole program steps accordingly. The local stepping relation on each node is in the form of $(\varsigma_c, \varsigma_o, M) \xrightarrow{\iota}_{\mathsf{t}} (\varsigma_c', \varsigma_o', M')$, which is defined in Fig. 15(b). Here the label ι indicates the type of the transition (see Fig. 14).

When a client makes an object operation call $x := f(E)$, as shown in the first rule of Fig. 15(b), the operation defined in Π is applied over the object replica $\mathcal{S}_o$ and generates the return value n' and the effector δ. Then the message $((f, n), \delta)$ is put into the global message pool, associated with a fresh message ID. Here n is the value of the argument E. This message is put into the local message pool immediately and the effector δ is applied over the object replica to generate a new object state $\mathcal{S}_o'$. We use the event $(\mathit{mid}, \mathsf{t}, (f, n, n', \delta))$ as the label of the transition to record the call of the operation (f, n) from the node t. We call the event the *origin* of the operation (f, n).

The next rule says when a node receives a broadcast message $((f, n), \delta)$, i.e., the message is in the global message pool but not in the local one yet, it is put into the local message pool and the effector δ is applied to the object replica to update the state. This step generates the event $(\mathit{mid}, \mathsf{t}, (f, n), \delta)$ as the label to show the processing on the replica t of a state update request δ originated from a different node. Other client steps affect the client state $\mathcal{S}_c$ only. Such steps are called silent steps. We omit the rules here.

Our semantics does not guarantee delivery of messages. A message can stay in the global message pool forever but not being put into the local message pool of certain node. It does not guarantee any order of delivery either, since messages in the global pool can be processed in any order.

Notations on events and event traces. Events e are defined in Fig. 14. We use $\mathsf{msgid}(e)$, $\mathsf{tid}(e)$, $\mathsf{op}(e)$, and $\mathsf{eff}(e)$ to get the mid, t, (f, n) and δ components in e respectively. $\mathsf{rval}(e)$ gets the return value n' if e is in the form of $(\mathit{mid}, \mathsf{t}, (f, n, n', \delta))$, undefined otherwise.

The event trace $\mathcal{E}$ is a sequence of events. We use $W \xrightarrow{\mathcal{E}}{}^{*} W'$ to represent that the zero-step or multiple-step transition from W to W' generates the event trace $\mathcal{E}$. Then we define $\mathcal{T}(P, \mathcal{S})$ as the prefix closure of the event traces that can be generated by the execution of P starting from $\mathcal{S}$, as shown below. We also define $\mathcal{T}(\Pi, \mathcal{S})$ as the prefix closure of the event traces that can be generated by *any* set of clients accessing Π with the initial state $\mathcal{S}$.

$$
\begin{array}{lll}
\mathcal{T}(P, \mathcal{S}) & \overset{\text{def}}{=} & \{\mathcal{E} \mid \exists W, W'.\, ((P, \mathcal{S}) \overset{\mathsf{load}}{\longmapsto} W) \wedge (W \overset{\mathcal{E}}{\longmapsto}{}^{*} W')\} \\
\mathcal{T}(\Pi, \mathcal{S}) & \overset{\text{def}}{=} & \{\mathcal{E} \mid \exists C_1, \ldots, C_n.\, \mathcal{E} \in \mathcal{T}(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, \mathcal{S})\}
\end{array}
$$

We use $e \in \mathcal{E}$ to represent that e is on the trace $\mathcal{E}$, $e <_{\mathcal{E}} e'$ to denote e occurs before e' on $\mathcal{E}$, and $e \leq_{\mathcal{E}} e'$ to mean either $e <_{\mathcal{E}} e'$, or e and e' are the same event. $\mathcal{E}' \leqslant \mathcal{E}$ means $\mathcal{E}'$ is a prefix of $\mathcal{E}$. $\mathsf{is\_orig}_{\mathsf{t}}(e)$ says e is an origin event on the node t. Then $\mathsf{orig}(\mathcal{E})$ represents the set of all the origin events on $\mathcal{E}$. Fig. 16 defines various relations between events on $\mathcal{E}$. $e \xrightarrow{\mathsf{t}}_{\mathcal{E}} e'$ says that the node t receives the e', which is an effector of the origin e, and $e \overset{\mathsf{t}}{\underset{\mathcal{E}}{\Longrightarrow}} e'$ requires either $e \xrightarrow{\mathsf{t}}_{\mathcal{E}} e'$, or e' and e are the same event originated at t. In the framed box in Fig. 16 we show an example execution, where we use black dots to represent the origins and white ones their effectors. So we know $e_1 \xrightarrow{\mathsf{t}_2}_{\mathcal{E}} e_1'$ and $e_1 \overset{\mathsf{t}_1}{\underset{\mathcal{E}}{\Longrightarrow}} e_1$. The order $e_1 \prec^{\mathsf{t}}_{\mathcal{E}} e_2$ says the node t receives the effector of e_1 earlier than that of e_2. In the example execution, we know $e_1 \prec^{\mathsf{t}_1}_{\mathcal{E}} e_2$, $e_2 \prec^{\mathsf{t}_2}_{\mathcal{E}} e_1$, $e_2 \prec^{\mathsf{t}_1}_{\mathcal{E}} e_3$, and $e_2 \prec^{\mathsf{t}_2}_{\mathcal{E}} e_3$. The visibility order $e \overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_{\mathcal{E}} e'$ says, when the origin event e' is issued on t, t has already received the effector of e. In the example execution, we have $e_1 \overset{\mathsf{vis}}{\underset{\mathsf{t}_2}{\longmapsto}}_{\mathcal{E}} e_3$ and $e_2 \overset{\mathsf{vis}}{\underset{\mathsf{t}_2}{\longmapsto}}_{\mathcal{E}} e_3$. The relation $e \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e'$ hides the node where e' occurs. The happens-before order $\overset{\mathsf{hb}}{\longmapsto}_{\mathcal{E}}$ over origin events on $\mathcal{E}$ is the transitive closure of $\overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}}$. The set $\mathsf{visible}(\mathcal{E}, \mathsf{t})$ of origin events visible on the node t are the events whose effectors have reached t.

$$
\begin{array}{rrcl}
(\mathit{AProg}) & \mathbb{P} & ::= & \textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \parallel \ldots \parallel C_n \\
(\mathit{AOpEvent}) & \mathbbm{e} & ::= & (\mathit{mid}, (f, n)) \quad (\mathit{AOpHist})\ \xi\ ::=\ \epsilon \mid \mathbbm{e} :: \xi \\
(\mathit{ARtNode}) & \mathbb{R} & ::= & (\Gamma, \mathcal{S}, \xi) \quad (\mathit{ANdSet})\ \Sigma ::= \{\mathsf{t}_1 \rightsquigarrow \mathbb{R}_1, \ldots, \mathsf{t}_n \rightsquigarrow \mathbb{R}_n\} \\
(\mathit{AMsgSoup}) & \mathbb{M} & ::= & \{\mathit{mid}_1 \rightsquigarrow (f_1, n_1), \ldots, \mathit{mid}_k \rightsquigarrow (f_k, n_k)\} \\
(\mathit{AWorld}) & \mathbb{W} & ::= & (\sigma_c, \Sigma, \mathbb{M}, \bowtie) \\
(\mathit{ObsvEvent}) & \mathbbm{o} & ::= & (\mathit{mid}, \mathsf{t}, (f, n, n')) \mid (\mathit{mid}, \mathsf{t}, (f, n)) \\
(\mathit{ALabel}) & \mathbbm{l} & ::= & \mathbbm{o} \mid \tau \qquad (\mathit{ObsvTrace})\ \mathbb{O}\ ::=\ \epsilon \mid \mathbbm{o} :: \mathbb{O}
\end{array}
$$

(a) world and event trace

$$
\frac{\begin{array}{c}\text{for all } \mathsf{t} \in [1..n]: \quad \sigma_c(\mathsf{t}) = (\textbf{skip}, \mathcal{S}_c^{\mathsf{t}}) \quad \Sigma(\mathsf{t}) = (\Gamma, \mathcal{S}_0, \xi_{\mathsf{t}}) \\ \mathit{dom}(\mathbb{M}) = \mathit{dom}(\xi_{\mathsf{t}}) \quad \mathcal{S}_o^{\mathsf{t}} = \mathsf{aexecST}(\Gamma, \mathcal{S}_0, \xi_{\mathsf{t}})\end{array}}{(\sigma_c, \Sigma, \mathbb{M}, \bowtie) \Longmapsto (\textbf{end}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathcal{S}_o^1, \ldots, \mathcal{S}_o^n))}
$$

$$
\frac{\begin{array}{c}(\sigma_c(\mathsf{t}), \Sigma(\mathsf{t}), \mathbb{M}) \circ\!\!\xrightarrow{\mathbbm{l}}_{\mathsf{t}} (\varsigma', \mathbb{R}', \mathbb{M}') \quad \mathbb{R}' = (\Gamma, \mathcal{S}_0, \xi) \\ \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{AbsCoh}(\xi, \Sigma(\mathsf{t}').\xi, (\Gamma, \bowtie))\end{array}}{(\sigma_c, \Sigma, \mathbb{M}, \bowtie) \overset{\mathbbm{l}}{\Longmapsto} (\sigma_c\{\mathsf{t} \rightsquigarrow \varsigma'\}, \Sigma\{\mathsf{t} \rightsquigarrow \mathbb{R}'\}, \mathbb{M}', \bowtie)}
$$

(b) world transitions

$$
\frac{\begin{array}{c}\varsigma_c = (x := f(E), \mathcal{S}_c) \quad [\![E]\!]_{\mathcal{S}_c} = n \quad \mathit{mid} \notin \mathit{dom}(\mathbb{M}) \\ \mathbb{M}' = \mathbb{M} \uplus \{\mathit{mid} \rightsquigarrow (f, n)\} \quad \xi' = \xi +\!\!+ [(\mathit{mid}, (f, n))] \\ \mathsf{aexecRV}(\Gamma, \mathcal{S}, \xi') = n' \quad \varsigma_c' = (\textbf{skip}, \mathcal{S}_c\{x \rightsquigarrow n'\})\end{array}}{(\varsigma_c, (\Gamma, \mathcal{S}, \xi), \mathbb{M}) \circ\!\!\xrightarrow{(\mathit{mid}, \mathsf{t}, (f, n, n'))}_{\mathsf{t}} (\varsigma_c', (\Gamma, \mathcal{S}, \xi'), \mathbb{M}')}
$$

$$
\frac{\begin{array}{c}\mathbb{M}(\mathit{mid}) = (f, n) \qquad \mathit{mid} \notin \mathit{dom}(\xi) \\ \xi = \xi_0 +\!\!+ \xi_1 \qquad \xi' = \xi_0 +\!\!+ [(\mathit{mid}, (f, n))] +\!\!+ \xi_1\end{array}}{(\varsigma_c, (\Gamma, \mathcal{S}, \xi), \mathbb{M}) \circ\!\!\xrightarrow{(\mathit{mid}, \mathsf{t}, (f, n))}_{\mathsf{t}} (\varsigma_c, (\Gamma, \mathcal{S}, \xi'), \mathbb{M})}
$$

(c) local transitions

Figure 17. Abstract operational semantics for CRDTs.

B Proofs of the Abstraction Theorems

B.1 For ACC

B.1.1 The Abstract Operational Semantics Fig. 17 defines the high-level small-step operational semantics for the abstract program $\mathbb{P}$, where clients interact with the specification Γ. It also relies on the conflict relation $\bowtie$ to order the incoming operations. At the abstract level, each replica $\mathbb{R}$ consists of the code Γ, the *initial object state* $\mathcal{S}$, and the list ξ of operations that have been done on this replica. Here ξ is similar to the local message pool M in the concrete object replica, but it also maintains the *abstract execution order*. Note that in $\mathbb{R}$ we do not record the current object state, which can always be generated from the initial state $\mathcal{S}$ and the list ξ of operations.

The rules for world transitions in Fig. 17(b) are similar to those in Fig. 15(a). We omit the load rule which creates the initial world from $\mathbb{P}$. The first rule in Fig. 17(b) says, when the whole program terminates, we replay the operations on the list ξ to generate the final object state on each node. The next rule requires that the local operation sequences ξ generated in each local step $(\varsigma, \mathbb{R}, \mathbb{M}) \circ\!\!\xrightarrow{\mathbbm{l}}_{\mathsf{t}} (\varsigma', \mathbb{R}', \mathbb{M}')$ (the transition rules are shown in Fig. 17(c)) must be coherent with the operation sequences on all other nodes. Here the coherence is the abstract counterpart of Coh in Fig. 8:

$$
\begin{array}{l}
\mathsf{AbsCoh}(\xi, \xi', (\Gamma, \bowtie)) \ \text{ iff} \\
\quad \forall \mathbbm{e}_1, \mathbbm{e}_2.\ \mathbbm{e}_1 <_{\xi} \mathbbm{e}_2 \wedge \mathbbm{e}_2 <_{\xi'} \mathbbm{e}_1 \implies \neg(\mathbbm{e}_1 \bowtie_{\Gamma} \mathbbm{e}_2) \\
\quad \text{where } \mathbbm{e} <_{\xi} \mathbbm{e}' \ \text{ iff } \ \exists i, j.\ \xi(i) = \mathbbm{e} \wedge \xi(j) = \mathbbm{e}' \wedge i < j
\end{array}
$$

It says that conflicting operations must be ordered the same way in ξ of all nodes. We can see ξ here can be viewed as the arbitration order *ar* in our ACC definition in Sec. 5.

Figure 17(c) shows the local transition rules. Each node maintains the initial object state $\mathcal{S}$. When a client issues a request, as shown in the first rule, the message is appended at the end of ξ. Then all the operations on the resulting ξ' are executed on the fly from the initial state $\mathcal{S}$ to calculate the return value n'. Here the definition of $\mathsf{aexecRV}(\Gamma, \mathcal{S}, \xi)$ is similar to $\mathsf{aexecRV}(\Gamma, \mathcal{S}, \mathcal{E})$ in Sec. 5.

When a node receives an operation request sent from others, the message is non-deterministically inserted into ξ, as shown in the second rule. In this case we do not execute the operation, since the node is not the origin of the request and does not need the return value. Note that, although the insertion is non-deterministic, the resulting ξ needs to be coherent with the lists on other nodes, as the last world transition rule in Fig. 17(b) requires.

Since the local operation lists ξ on all nodes must be coherent during the execution, we can prove that the abstract semantics inherently guarantees the convergence of the abstract object states. Then, the contextual refinement $\Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$ can ensure $\mathsf{Cv}_{\varphi}(\Pi)$, the convergence of the concrete object. With the Abstraction Theorem, we can derive Lem. 5 again: $\mathsf{ACC}_{\varphi}(\Pi, (\Gamma, \bowtie))$ can ensure $\mathsf{Cv}_{\varphi}(\Pi)$ too.

B.1.2 Proofs for the Abstraction Theorem (Theorem 7) The contextual refinement $\Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$ says, for any object states and event traces generated by an execution at the concrete level, the corresponding abstract states and traces can be generated at the abstract level (using the *abstract semantics* presented below). $\mathcal{T}_{\mathsf{s}}(P, \mathcal{S})$ and $\mathcal{T}_{\mathsf{s}}(\mathbb{P}, \mathcal{S})$ are defined similarly as $\mathcal{T}(P, \mathcal{S})$ (Sec. A), but they additionally record the final states and all the intermediate object states ($\mathcal{S}_c^i$ for the final client-local state and $\mathbb{S}_o^i$ for the trace of object states) of every node i of an execution. The function $\mathsf{obsv}(\mathcal{E})$ maps the event trace $\mathcal{E}$ at the concrete level to the one at the abstract level. We also lift φ to state traces. $\varphi(\mathbb{S}_o)$ maps every object state in the sequence $\mathbb{S}_o$ to an abstract state.

Definition 10. $\Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$ iff, for all clients $C_1, \ldots, C_n$ and state $\mathcal{S} \in dom(\varphi)$, for all $\mathcal{E}, \mathcal{S}_c^1, \ldots, \mathcal{S}_c^n, \mathbb{S}_o^1, \ldots, \mathbb{S}_o^n$,

$$\begin{array}{l}(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n)) \in \mathcal{T}_{\mathsf{s}}(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n, \mathcal{S}) \\ \implies (\mathsf{obsv}(\mathcal{E}), (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\varphi(\mathbb{S}_o^1), \ldots, \varphi(\mathbb{S}_o^n))) \\ \qquad\qquad\qquad \in \mathcal{T}_{\mathsf{s}}(\textbf{with}\ (\Gamma, \bowtie)\ \textbf{do}\ C_1 \parallel \ldots \parallel C_n, \varphi(\mathcal{S}))\end{array}$$

Figure 18 shows the formal definitions of $\mathcal{T}_{\mathsf{s}}(P, \mathcal{S})$ and $\mathcal{T}_{\mathsf{s}}(\mathbb{P}, \mathcal{S}_a)$ used in contextual refinement. $\mathcal{T}_{\mathsf{s}}(P, \mathcal{S})$ and $\mathcal{T}_{\mathsf{s}}(\mathbb{P}, \mathcal{S})$ are defined similarly as $\mathcal{T}(P, \mathcal{S})$ (Sec. A), but they additionally record the final states and all the intermediate object states ($\mathcal{S}_c^i$ for the final client-local state and $\mathbb{S}_o^i$ for the trace of object states) of every node i of an execution. The function $\mathsf{obsv}(\mathcal{E})$ maps the event trace $\mathcal{E}$ at the concrete level to the one at the abstract level. We also lift φ to state traces. $\varphi(\mathbb{S}_o)$ maps every object state in the sequence $\mathbb{S}_o$ to an abstract state.

Below we prove separately the two directions of the equivalence: $\mathsf{ACC}_{\varphi}(\Pi, (\Gamma, \bowtie)) \iff \Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$.

Proof of Theorem 7 ($\Longrightarrow$). For any $C_1, \ldots, C_n, \mathcal{S}, \mathcal{S}_a, \mathcal{S}_c^1, \ldots, \mathcal{S}_c^n, \mathbb{S}_o^1, \ldots, \mathbb{S}_o^n$ and $\mathcal{E}$, suppose $(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n)) \in \mathcal{T}_{\mathsf{s}}(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n, \mathcal{S})$ and $\varphi(\mathcal{S}) = \mathcal{S}_a$. Let $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$. We want to prove that

$$(\mathbb{O}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\varphi(\mathbb{S}_o^1), \ldots, \varphi(\mathbb{S}_o^n))) \in \mathcal{T}_{\mathsf{s}}(\textbf{with}\ (\Gamma, \bowtie)\ \textbf{do}\ C_1 \parallel \ldots \parallel C_n, \varphi(\mathcal{S})).$$

From $\mathsf{ACC}_{\varphi}(\Pi, (\Gamma, \bowtie))$, we know

$$\mathsf{ACT}_{\varphi}(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie)) .$$

Thus we know

$$\begin{array}{l}\exists ar_1, \ldots, ar_n. \\ \quad \forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(ar_{\mathsf{t}}) \wedge (\overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_{\mathcal{E}} \subseteq ar_{\mathsf{t}}) \wedge \mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathsf{t}})) \\ \qquad \wedge\ \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{Coh}(ar_{\mathsf{t}}, ar_{\mathsf{t}'}, (\Gamma, \bowtie))\end{array}$$

Since $(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n)) \in \mathcal{T}_{\mathsf{s}}(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n, \mathcal{S})$, we know there exist m, W and W' such that

$$\begin{array}{c}((\textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n, \mathcal{S}) \xrightarrow{\mathsf{load}} W) \wedge (W \xRightarrow{(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n))}{}^{m}\ W'), \\ W = (\sigma_c, \sigma_o, \emptyset),\ W' = (\sigma_c', \sigma_o', M_s'), \\ \forall \mathsf{t} \in [1..n].\ \sigma_c'(\mathsf{t}) = (\_, \mathcal{S}_c^{\mathsf{t}}), \\ \forall \mathsf{t} \in [1..n].\ \sigma_o'(\mathsf{t}) = (\Pi, \mathcal{S}_o^{\mathsf{t}}, M_{\mathsf{t}}').\end{array}$$

Let

$$\begin{array}{l}\qquad\qquad \mathbb{W} = (\sigma_c, \Sigma, \emptyset, \bowtie) \text{ and } \mathbb{W}' = (\sigma_c', \Sigma', \mathbb{M}_s', \bowtie), \text{ where} \\ \forall \mathsf{t} \in [1..n].\ \Sigma(\mathsf{t}) = (\Gamma, \mathcal{S}_a, \epsilon) \\ \forall \mathsf{t} \in [1..n].\ \Sigma'(\mathsf{t}) = (\Gamma, \mathcal{S}_a, \xi_{\mathsf{t}}') \\ \mathbb{M}_s' = \mathsf{abs}(M_s'), \quad \text{where } \mathsf{abs}(M_s') \overset{\text{def}}{=} \{(mid, (f, n)) \mid \exists \delta.\ M_s'(mid) = ((f, n), \delta)\} \\ \forall \mathsf{t} \in [1..n].\ \xi_{\mathsf{t}}' = \mathsf{abs}(M_{\mathsf{t}}' \lfloor ar_{\mathsf{t}})\end{array}$$

$$(\mathit{StSeq})\ \mathbb{S} ::= \epsilon \mid \mathcal{S} :: \mathbb{S}$$

$$\frac{W = (\sigma_c, \sigma_o, \_) \qquad \mathit{dom}(\sigma_c) = \mathit{dom}(\sigma_o) = [1..n] \qquad \forall i.\ \sigma_c(i) = (\_, \mathcal{S}_c^i) \qquad \forall i.\ \sigma_o(i) = (\_, \mathcal{S}_o^i, \_)}{W \xRightarrow{(\epsilon, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), ([\mathcal{S}_o^1], \ldots, [\mathcal{S}_o^n]))}{}^{0}\, W}$$

$$\frac{\begin{array}{c} W \overset{e}{\longmapsto} W'' \qquad W'' \xRightarrow{(\mathcal{E}', (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_1'', \ldots, \mathbb{S}_n''))}{}^{k}\, W' \\ W = (\sigma_c, \sigma_o, \_) \qquad \forall i.\ \sigma_o(i) = (\_, \mathcal{S}_o^i, \_) \qquad \mathcal{E} = e :: \mathcal{E}' \qquad \forall i.\ \mathbb{S}_o^i = \mathcal{S}_o^i :: \mathbb{S}_i'' \end{array}}{W \xRightarrow{(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n))}{}^{k+1}\, W'}$$

$$\frac{W \overset{\tau}{\longmapsto} W'' \qquad W'' \xRightarrow{(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n))}{}^{k}\, W'}{W \xRightarrow{(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n))}{}^{k+1}\, W'}$$

$$\mathcal{T}_{\mathsf{s}}(P, \mathcal{S}) \overset{\text{def}}{=} \{(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n)) \mid \exists W, W'.\ ((P, \mathcal{S}) \overset{\text{load}}{\longmapsto} W) \wedge (W \xRightarrow{(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n))}{}^{*}\, W')\}$$

(a) at the concrete level

$$\frac{\mathbb{W} = (\sigma_c, \Sigma, \_, \_) \qquad \mathit{dom}(\sigma_c) = \mathit{dom}(\Sigma) = [1..n] \qquad \forall i.\ \sigma_c(i) = (\_, \mathcal{S}_c^i) \qquad \forall i.\ \mathsf{aexecST}(\Sigma(i)) = \mathcal{S}_o^i}{\mathbb{W} \xRightarrow{(\epsilon, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), ([\mathcal{S}_o^1], \ldots, [\mathcal{S}_o^n]))}{}^{0}\, \mathbb{W}}$$

$$\frac{\begin{array}{c} \mathbb{W} \overset{\mathbb{o}}{\longmapsto} \mathbb{W}'' \qquad \mathbb{W}'' \xRightarrow{(\mathbb{O}', (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_1'', \ldots, \mathbb{S}_n''))}{}^{k}\, \mathbb{W}' \\ \mathbb{W} = (\sigma_c, \Sigma, \_, \_) \qquad \forall i.\ \mathsf{aexecST}(\Sigma(i)) = \mathcal{S}_o^i \qquad \mathbb{O} = \mathbb{o} :: \mathbb{O}' \qquad \forall i.\ \mathbb{S}_o^i = \mathcal{S}_o^i :: \mathbb{S}_i'' \end{array}}{\mathbb{W} \xRightarrow{(\mathbb{O}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n))}{}^{k+1}\, \mathbb{W}'}$$

$$\frac{\mathbb{W} \overset{\tau}{\longmapsto} \mathbb{W}'' \qquad \mathbb{W}'' \xRightarrow{(\mathbb{O}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n))}{}^{k}\, \mathbb{W}'}{\mathbb{W} \xRightarrow{(\mathbb{O}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n))}{}^{k+1}\, \mathbb{W}'}$$

$$\mathcal{T}_{\mathsf{s}}(\mathbb{P}, \mathcal{S}) \overset{\text{def}}{=} \{(\mathbb{O}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n)) \mid \exists \mathbb{W}, \mathbb{W}'.\ ((\mathbb{P}, \mathcal{S}) \overset{\text{load}}{\longmapsto} \mathbb{W}) \wedge (\mathbb{W} \xRightarrow{(\mathbb{O}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n))}{}^{*}\, \mathbb{W}')\}$$

(b) at the abstract level

$$\mathsf{obsv}(\mathcal{E}) \overset{\text{def}}{=} \begin{cases} (\mathit{mid}, \mathsf{t}, (f, n, n')) :: \mathsf{obsv}(\mathcal{E}') & \text{if } \mathcal{E} = (\mathit{mid}, \mathsf{t}, (f, n, n', \delta)) :: \mathcal{E}' \\ (\mathit{mid}, \mathsf{t}, (f, n)) :: \mathsf{obsv}(\mathcal{E}') & \text{if } \mathcal{E} = (\mathit{mid}, \mathsf{t}, (f, n), \delta) :: \mathcal{E}' \\ \epsilon & \text{if } \mathcal{E} = \epsilon \end{cases}$$

$$\varphi(\mathbb{S}) \overset{\text{def}}{=} \begin{cases} \epsilon & \text{if } \mathbb{S} = \epsilon \\ \varphi(\mathcal{S}) :: \varphi(\mathbb{S}') & \text{if } \mathbb{S} = \mathcal{S} :: \mathbb{S}' \end{cases}$$

(b) the functions $\mathsf{obsv}(\mathcal{E})$ and $\varphi(\mathbb{S})$

Figure 18. The trace sets used in contextual refinements.

Since $(\textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \parallel \ldots \parallel C_n, \mathcal{S}_a) \xrightarrow{\text{load}} \mathbb{W}$, we only need to prove

$$\mathbb{W} \xRightarrow{(\mathbb{O},(\mathcal{S}_c^1,\ldots,\mathcal{S}_c^n),(\varphi(\mathbb{S}_o^1),\ldots,\varphi(\mathbb{S}_o^n)))}{}^m\, \mathbb{W}'.$$

The proof is by induction over m.

- $m = 0$. Trivial.
- $m = k + 1$.
  Since $W \xRightarrow{(\mathcal{E},(\mathcal{S}_c^1,\ldots,\mathcal{S}_c^n),(\mathbb{S}_o^1,\ldots,\mathbb{S}_o^n))}{}^m\, W'$, we know there exist $\mathcal{E}'$, ι and W'' such that
  $$W \xRightarrow{(\mathcal{E}',(\mathcal{S}_1',\ldots,\mathcal{S}_n'),(\mathbb{S}_1'',\ldots,\mathbb{S}_n''))}{}^k\, W'' \text{ and } W'' \xrightarrow{\iota} W',$$
  and, if $\iota = \tau$, then $\mathcal{E} = \mathcal{E}'$ and $\forall i.\ \mathbb{S}_o^i = \mathbb{S}_i''$; otherwise, $\mathcal{E} = \mathcal{E}'{+}{+}[\iota]$ and $\forall i.\ \mathbb{S}_o^i = \mathbb{S}_i''{+}{+}[\mathcal{S}_o^i]$.
  Suppose $W'' = (\sigma_c'', \sigma_o'', M_s'')$, where $\forall \mathrm{t} \in [1..n].\ \sigma_o''(\mathrm{t}) = (\Pi, \mathcal{S}_\mathrm{t}'', M_\mathrm{t}'')$. Let
  $$\begin{aligned}\mathbb{O}' = \mathsf{obsv}(\mathcal{E}'),\ \mathbb{I} = \mathsf{obsv}(\iota) \text{ and } \mathbb{W}'' = (\sigma_c'', \Sigma'', \mathbb{M}_s'', \bowtie), \text{ where}\\ \forall \mathrm{t} \in [1..n].\ \Sigma''(\mathrm{t}) = (\Gamma, \mathcal{S}_a, \xi_\mathrm{t}'')\\ \mathbb{M}_s'' = \mathsf{abs}(M_s'')\\ \forall \mathrm{t} \in [1..n].\ \xi_\mathrm{t}'' = \mathsf{abs}(M_\mathrm{t}'' \downharpoonleft ar_\mathrm{t}')\\ \forall \mathrm{t} \in [1..n].\ ar_\mathrm{t}' = ar_\mathrm{t}|_{\mathsf{visible}(\mathcal{E}',\mathrm{t})}\end{aligned}$$
  To apply the induction hypothesis, we prove:
  $$\begin{aligned}&\forall \mathrm{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}',\mathrm{t})}(ar_\mathrm{t}') \wedge (\overset{\mathsf{vis}}{\underset{\mathrm{t}}{\longmapsto}}_{\mathcal{E}'} \subseteq ar_\mathrm{t}') \wedge \mathsf{ExecRelated}_\varphi(\mathrm{t}, (\mathcal{E}', \mathcal{S}), (\Gamma, ar_\mathrm{t}'))\\ &\wedge \forall \mathrm{t}' \neq \mathrm{t}.\ \mathsf{Coh}(ar_\mathrm{t}', ar_{\mathrm{t}'}', (\Gamma, \bowtie))\end{aligned}$$
  By the induction hypothesis, we know
  $$\mathbb{W} \xRightarrow{(\mathcal{E}',(\mathcal{S}_1',\ldots,\mathcal{S}_n'),(\varphi(\mathbb{S}_1''),\ldots,\varphi(\mathbb{S}_n'')))}{}^k\, \mathbb{W}''.$$
  Since $\forall \mathrm{t}.\ \mathsf{ExecRelated}_\varphi(\mathrm{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_\mathrm{t}))$, we know
  $$\forall \mathrm{t}.\ \varphi(\mathcal{S}_o^\mathrm{t}) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}|_\mathrm{t})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}, \mathrm{t}) \downharpoonleft ar_\mathrm{t}) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \xi_\mathrm{t}').$$
  Thus we only need to prove $\mathbb{W}'' \xrightarrow{\mathbb{I}} \mathbb{W}'$.
  - Suppose $\iota = \tau$ and $\mathcal{E} = \mathcal{E}'$. Then $W'' \longrightarrow W'$ is a client step. Thus $\mathbb{W}'' \longrightarrow \mathbb{W}'$.
  - Suppose $\iota = e$ and $\mathcal{E} = \mathcal{E}'{+}{+}[e]$.
    Suppose $\mathsf{tid}(e) = \mathrm{t}$. We first prove $(\sigma_c''(\mathrm{t}), \Sigma''(\mathrm{t}), \mathbb{M}_s'') \xrightarrow{\mathbb{I}}_\mathrm{t} (\sigma_c'(\mathrm{t}), \Sigma'(\mathrm{t}), \mathbb{M}_s')$.
    The proof is by case analysis over the event e.
    - $e = (mid, \mathrm{t}, (f, n, n', \delta))$.
      From the operational semantics, we know there exists x, E and C_t' such that
      $$\begin{gathered}\sigma_c''(\mathrm{t}) = ((x := f(E); C_\mathrm{t}'), \mathcal{S}_c'') \qquad \mathcal{S}_c' = \mathcal{S}_c''\{x \rightsquigarrow n'\} \qquad \sigma_c' = \sigma_c''\{\mathrm{t} \rightsquigarrow (C_\mathrm{t}', \mathcal{S}_c')\}\\ [\![E]\!]_{\mathcal{S}_c''} = n \qquad \Pi(f, n)(\mathcal{S}'') = (n', \delta) \qquad mid \notin dom(M_s'')\\ M_s' = M_s'' \uplus \{mid \rightsquigarrow ((f, n), \delta)\} \qquad \delta(\mathcal{S}'') = \mathcal{S}' \qquad M_\mathrm{t}' = M_\mathrm{t}'' \uplus \{mid \rightsquigarrow ((f, n), \delta)\}\end{gathered}$$
      Since $mid \notin dom(M_s'')$, we know
      $$mid \notin dom(\mathbb{M}_s'')$$
      Also, from the concrete operational semantics, we can prove:
      $$\lfloor (\mathcal{E}'|_\mathrm{t}) \rfloor = M_\mathrm{t}''$$
      Thus we know
      $$\forall e' \in M_\mathrm{t}''.\ e' \overset{\mathsf{vis}}{\underset{\mathrm{t}}{\longmapsto}}_{\mathcal{E}}\ e$$
      Then, since $\overset{\mathsf{vis}}{\underset{\mathrm{t}}{\longmapsto}}_{\mathcal{E}} \subseteq ar_\mathrm{t}$, we know
      $$\forall e' \in M_\mathrm{t}''.\ (e', e) \in ar_\mathrm{t}$$
      Let
      $$\mathbb{M}_s' = \mathbb{M}_s'' \uplus \{mid \rightsquigarrow (f, n)\} \text{ and } \xi_\mathrm{t}' = \xi_\mathrm{t}''{+}{+}[(mid, (f, n))]$$
      Thus
      $$\mathbb{M}_s' = \mathsf{abs}(M_s')$$
      Since $\xi_\mathrm{t}'' = \mathsf{abs}(M_\mathrm{t}'' \downharpoonleft ar_\mathrm{t})$ and $M_\mathrm{t}' = M_\mathrm{t}'' \uplus \{mid \rightsquigarrow ((f, n), \delta)\}$, we know
      $$\xi_\mathrm{t}' = \mathsf{abs}(M_\mathrm{t}' \downharpoonleft ar_\mathrm{t})$$

Also, since $\mathsf{ExecRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_\mathsf{t}))$, we know
$$n' = \mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma, \mathcal{S}_a, M'_\mathsf{t} \downharpoonleft ar_\mathsf{t}).$$
Thus we know
$$\mathsf{aexecRV}(\Gamma, \mathcal{S}_a, \xi'_\mathsf{t}) = n'$$
Thus by the abstract operational semantics, we know
$$(\sigma''_c(\mathsf{t}), \Sigma''(\mathsf{t}), \mathbb{M}''_s) \overset{\mathbb{0}}{\circ\!\!\longrightarrow}_\mathsf{t} (\sigma'_c(\mathsf{t}), \Sigma'(\mathsf{t}), \mathbb{M}'_s).$$

- $e = (mid, \mathsf{t}, (f, n), \delta)$.
  From the operational semantics, we know
  $$M''_s(mid) = ((f,n), \delta) \qquad mid \notin dom(M''_\mathsf{t}) \qquad \delta(\mathcal{S}'') = \mathcal{S}'$$
  $$M'_\mathsf{t} = M''_\mathsf{t} \uplus \{mid \rightsquigarrow ((f,n),\delta)\} \qquad M'_s = M''_s$$
  Thus we know
  $$\mathbb{M}''_s(mid) = (f,n) \qquad mid \notin dom(\xi''_\mathsf{t})$$
  Let
  $$\mathbb{M}'_s = \mathbb{M}''_s$$
  Thus
  $$\mathbb{M}'_s = \mathsf{abs}(M'_s)$$
  Let
  $$\xi'_\mathsf{t} = \mathsf{abs}(M'_\mathsf{t} \downharpoonleft ar_\mathsf{t})$$
  Then, since $\xi''_\mathsf{t} = \mathsf{abs}(M''_\mathsf{t} \downharpoonleft ar_\mathsf{t})$, we know there exist ξ_1 and ξ_2 such that
  $$\xi''_\mathsf{t} = \xi_1{+\!+}\xi_2 \text{ and } \xi'_\mathsf{t} = \xi_1{+\!+}[(mid, (f,n))]{+\!+}\xi_2$$
  Thus by the abstract operational semantics, we know
  $$(\sigma''_c(\mathsf{t}), \Sigma''(\mathsf{t}), \mathbb{M}''_s) \overset{\mathbb{0}}{\circ\!\!\longrightarrow}_\mathsf{t} (\sigma'_c(\mathsf{t}), \Sigma'(\mathsf{t}), \mathbb{M}'_s).$$

Next we prove $\forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{AbsCoh}(\xi'_\mathsf{t}, \xi'_{\mathsf{t}'}, (\Gamma, \bowtie))$.
For any $\mathbb{e}_1$ and $\mathbb{e}_2$, suppose $\mathbb{e}_1 <_{\xi'_\mathsf{t}} \mathbb{e}_2$ and $\mathbb{e}_2 <_{\xi'_{\mathsf{t}'}} \mathbb{e}_1$. Since $\xi'_\mathsf{t} = \mathsf{abs}(M'_\mathsf{t} \downharpoonleft ar_\mathsf{t})$ and $\xi'_{\mathsf{t}'} = \mathsf{abs}(M'_{\mathsf{t}'} \downharpoonleft ar_{\mathsf{t}'})$, we know there exist e_1 and e_2 such that
$$\mathsf{abs}(e_1) = \mathbb{e}_1, \mathsf{abs}(e_2) = \mathbb{e}_2, e_1 \in M'_\mathsf{t} \cap M'_{\mathsf{t}'}, e_2 \in M'_\mathsf{t} \cap M'_{\mathsf{t}'}, e_1\ ar_\mathsf{t}\ e_2, e_2\ ar_{\mathsf{t}'}\ e_1.$$
Since $\mathsf{Coh}(ar_\mathsf{t}, ar_{\mathsf{t}'}, (\Gamma, \bowtie))$, we know
$$\neg(e_1 \bowtie_\Gamma e_2)$$
Thus we know $\neg(\mathbb{e}_1 \bowtie_\Gamma \mathbb{e}_2)$. As a result, we know
$$\mathsf{AbsCoh}(\xi'_\mathsf{t}, \xi'_{\mathsf{t}'}, (\Gamma, \bowtie)).$$
Thus we know
$$\mathbb{W}'' \overset{\mathbb{0}}{\circ\!\!\!\longrightarrow} \mathbb{W}'.$$

Thus we are done. □

Proof of Theorem 7 ($\Longleftarrow$). For any $\mathcal{S}$, $\mathcal{S}_a$ and $\mathcal{E}$, suppose $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$ and $\varphi(\mathcal{S}) = \mathcal{S}_a$. We want to prove $\mathsf{ACT}_\varphi(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie))$. That is, we want to prove:
$$\begin{aligned}&\exists ar_1, \ldots, ar_n.\\ &\quad \forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(ar_\mathsf{t}) \wedge (\underset{\mathsf{t}}{\overset{\mathsf{vis}}{\longmapsto}}_\mathcal{E} \subseteq ar_\mathsf{t}) \wedge \mathsf{ExecRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_\mathsf{t}))\\ &\qquad \wedge \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{Coh}(ar_\mathsf{t}, ar_{\mathsf{t}'}, (\Gamma, \bowtie))\end{aligned}$$

From $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, we know there exist $\mathcal{S}^1_c, \ldots, \mathcal{S}^n_c$ and $\mathbb{S}^1_o, \ldots, \mathbb{S}^n_o$ such that
$$(\mathcal{E}, (\mathcal{S}^1_c, \ldots, \mathcal{S}^n_c), (\mathbb{S}^1_o, \ldots, \mathbb{S}^n_o)) \in \mathcal{T}_\mathbf{s}(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, \mathcal{S}).$$

Let $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$. From $\Pi \sqsubseteq_\varphi (\Gamma, \bowtie)$, we know
$$(\mathbb{O}, (\mathcal{S}^1_c, \ldots, \mathcal{S}^n_c), (\varphi(\mathbb{S}^1_o), \ldots, \varphi(\mathbb{S}^n_o))) \in \mathcal{T}_\mathbf{s}(\mathbf{with}\ (\Gamma, \bowtie)\ \mathbf{do}\ C_1 \parallel \ldots \parallel C_n, \varphi(\mathcal{S})).$$

Thus we know there exist $\mathbb{W}_0$ and $\mathbb{W}$ such that
$$(\mathbf{with}\ (\Gamma, \bowtie)\ \mathbf{do}\ C_1 \parallel \ldots \parallel C_n, \mathcal{S}_a) \overset{\mathsf{load}}{\circ\!\!\longrightarrow} \mathbb{W}_0,\ \ \mathbb{W}_0 \xRightarrow{(\mathbb{O}, (\mathcal{S}^1_c, \ldots, \mathcal{S}^n_c), (\varphi(\mathbb{S}^1_o), \ldots, \varphi(\mathbb{S}^n_o)))}{}^* \mathbb{W}$$
$$\mathbb{W}_0 = (\sigma_0, \Sigma_0, \emptyset, \bowtie), \mathbb{W} = (\sigma, \Sigma, \mathbb{M}_s, \bowtie),$$
$$\forall \mathsf{t}.\ \sigma_0(\mathsf{t}) = (C_\mathsf{t}, \emptyset), \forall \mathsf{t}.\ \Sigma_0(\mathsf{t}) = (\Gamma, \varphi(\mathcal{S}), \epsilon), \forall \mathsf{t}.\ \Sigma(\mathsf{t}) = (\Gamma, \varphi(\mathcal{S}), \xi_\mathsf{t}).$$

For any t, let

$$ar_{\mathrm{t}} = \{(e_1, e_2) \mid \{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}, \mathrm{t}) \wedge \mathsf{abs}(e_1) <_{\xi_{\mathrm{t}}} \mathsf{abs}(e_2)\}$$

where $\mathsf{abs}(e) \overset{\mathrm{def}}{=} (mid, (f, n))$ if $e = (mid, \mathrm{t}, (f, n, n', \delta))$.

- By the abstract operational semantics, we know $dom(\mathsf{visible}(\mathbb{O}, \mathrm{t})) = dom(\xi_{\mathrm{t}})$. Then, since $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$, we know $dom(\mathsf{visible}(\mathcal{E}, \mathrm{t})) = dom(\xi_{\mathrm{t}})$. Thus $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathrm{t})}(ar_{\mathrm{t}})$ holds.
- For any e_1 and e_2, if $e_1 \underset{\mathrm{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}} e_2$, since $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$, we know $\mathsf{obsv}(e_1) \underset{\mathrm{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathbb{O}} \mathsf{obsv}(e_2)$. Then, from the abstract operational semantics, we know $\mathsf{abs}(e_1) <_{\xi_{\mathrm{t}}} \mathsf{abs}(e_2)$. Thus $(e_1, e_2) \in ar_{\mathrm{t}}$. So, $\underset{\mathrm{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}} \subseteq ar_{\mathrm{t}}$.
- Below we prove $\mathsf{ExecRelated}_{\varphi}(\mathrm{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathrm{t}}))$.
  - For any $\mathcal{E}' \leqslant \mathcal{E}$, we prove $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathrm{t}})) = \mathsf{aexecST}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathrm{t}) \downharpoonright ar_{\mathrm{t}})$.
    Suppose the length of $\mathcal{E}'$ is k, and the $(k+1)$-th state in the sequence $\mathbb{S}_o^{\mathrm{t}}$ is $\mathcal{S}_o^{\mathrm{t}}$. Since $(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n)) \in \mathcal{T}_{\mathrm{s}}(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, \mathcal{S})$, we know
    $$\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathrm{t}}) = \mathcal{S}_o^{\mathrm{t}}.$$
    Let $\mathbb{O}' = \mathsf{obsv}(\mathcal{E}')$. Since $\mathcal{E}' \leqslant \mathcal{E}$ and $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$, we know $\mathbb{O}' \leqslant \mathbb{O}$. Thus
    $$(\mathsf{visible}(\mathbb{O}', \mathrm{t}) \downharpoonright ar_{\mathrm{t}}) = (\xi_{\mathrm{t}}|_{\mathsf{visible}(\mathbb{O}',\mathrm{t})})$$
    Since $\mathbb{W}_0 \xRightarrow{(\mathbb{O},(\mathcal{S}_c^1,\ldots,\mathcal{S}_c^n),(\varphi(\mathbb{S}_o^1),\ldots,\varphi(\mathbb{S}_o^n)))}{}^{*}\ \mathbb{W}$, we know
    $$\mathsf{aexec\_st}(\Gamma, \varphi(\mathcal{S}), (\xi_{\mathrm{t}}|_{\mathsf{visible}(\mathbb{O}',\mathrm{t})})) = \varphi(\mathcal{S}_o^{\mathrm{t}})$$
    Thus $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathrm{t}})) = \mathsf{aexecST}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathrm{t}) \downharpoonright ar_{\mathrm{t}})$.
  - For any $\mathcal{E}' \leqslant \mathcal{E}$, for any e such that $\mathsf{last}(\mathcal{E}') = e$ and $\mathsf{is\_orig}_{\mathrm{t}}(e)$, we prove $\mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathrm{t}) \downharpoonright ar_{\mathrm{t}})$.
    Let $\mathbb{O}' = \mathsf{obsv}(\mathcal{E}')$. Since $\mathbb{W}_0 \xRightarrow{(\mathbb{O},(\mathcal{S}_c^1,\ldots,\mathcal{S}_c^n),(\varphi(\mathbb{S}_o^1),\ldots,\varphi(\mathbb{S}_o^n)))}{}^{*}\ \mathbb{W}$, we know there exist $\mathbb{W}_1$, $\mathbb{W}_2$, $\mathbb{O}_1$, $\mathbb{l}$, $\mathbb{O}_2$ such that
    $$\mathbb{W}_0 \overset{\mathbb{O}_1}{\multimapdotbothA\!\!\longrightarrow}{}^{*} \mathbb{W}_1,\ \mathbb{W}_1 \overset{\mathbb{l}}{\multimapdotbothA\!\!\longrightarrow} \mathbb{W}_2,\ \mathbb{W}_2 \overset{\mathbb{O}_2}{\multimapdotbothA\!\!\longrightarrow}{}^{*} \mathbb{W},$$
    $$\mathbb{O} = \mathbb{O}_1{+}{+}[\mathbb{l}]{+}{+}\mathbb{O}_2,\ \mathbb{O}_1{+}{+}[\mathbb{l}] = \mathbb{O}',\ \mathbb{l} = \mathsf{obsv}(e).$$
    Suppose $\mathbb{W}_2 = (\sigma_2, \Sigma_2, \mathbb{M}_s'', \bowtie)$, and $\forall \mathrm{t}.\ \Sigma_2(\mathrm{t}) = (\Gamma, \varphi(\mathcal{S}), \xi_{\mathrm{t}}'')$. Thus
    $$\begin{aligned}&\mathsf{rval}(e) = \mathsf{rval}(\mathbb{l})\\ &= \mathsf{aexec\_rv}(\Gamma, \varphi(\mathcal{S}), \xi_{\mathrm{t}}'')\\ &= \mathsf{aexec\_rv}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathbb{O}_1{+}{+}[\mathbb{l}], \mathrm{t}) \downharpoonright ar_{\mathrm{t}})\\ &= \mathsf{aexec\_rv}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathrm{t}) \downharpoonright ar_{\mathrm{t}})\end{aligned}$$
- Below we prove $\forall \mathrm{t}' \neq \mathrm{t}.\ \mathsf{Coh}(ar_{\mathrm{t}}, ar_{\mathrm{t}'}, (\Gamma, \bowtie))$. That is, for any e_1 and e_2, if $(e_1, e_2) \in ar_{\mathrm{t}}$ and $(e_2, e_1) \in ar_{\mathrm{t}'}$, we want to prove $\neg(e_1 \bowtie_{\Gamma} e_2)$.
  Since $(e_1, e_2) \in ar_{\mathrm{t}}$ and $(e_2, e_1) \in ar_{\mathrm{t}'}$, we know
  $$\mathsf{abs}(e_1) <_{\xi_{\mathrm{t}}} \mathsf{abs}(e_2) \text{ and } \mathsf{abs}(e_2) <_{\xi_{\mathrm{t}'}} \mathsf{abs}(e_1).$$
  By the abstract operational semantics, we know there exist $\mathrm{t}_0 \in \{\mathrm{t}, \mathrm{t}'\}$ and $\mathbb{W}_1$, $\mathbb{W}_2$, $\mathbb{O}_1$, $\mathbb{l}$, $\mathbb{O}_2$ such that
  $$\mathbb{W}_0 \overset{\mathbb{O}_1}{\multimapdotbothA\!\!\longrightarrow}{}^{*} \mathbb{W}_1,\ \mathbb{W}_1 \overset{\mathbb{l}}{\multimapdotbothA\!\!\longrightarrow} \mathbb{W}_2,\ \mathbb{W}_2 \overset{\mathbb{O}_2}{\multimapdotbothA\!\!\longrightarrow}{}^{*} \mathbb{W},$$
  $$\mathbb{O} = \mathbb{O}_1{+}{+}[\mathbb{l}]{+}{+}\mathbb{O}_2,\ \mathsf{tid}(\mathbb{l}) = \mathrm{t}_0,$$
  $$\mathbb{W}_2 = (\sigma_2, \Sigma_2, \mathbb{M}_s'', \bowtie),\ \forall \mathrm{t}.\ \Sigma_2(\mathrm{t}) = (\Gamma, \varphi(\mathcal{S}), \xi_{\mathrm{t}}''),$$
  $$\{\mathsf{abs}(e_1), \mathsf{abs}(e_2)\} \subseteq \lfloor \xi_{\mathrm{t}}'' \rfloor,\ \{\mathsf{abs}(e_1), \mathsf{abs}(e_2)\} \subseteq \lfloor \xi_{\mathrm{t}'}'' \rfloor.$$
  So we know
  $$\mathsf{AbsCoh}(\xi_{\mathrm{t}}'', \xi_{\mathrm{t}'}'', (\Gamma, \bowtie)),\ \mathsf{abs}(e_1) <_{\xi_{\mathrm{t}}''} \mathsf{abs}(e_2),\ \mathsf{abs}(e_2) <_{\xi_{\mathrm{t}'}''} \mathsf{abs}(e_1).$$
  Thus $\neg(\mathsf{abs}(e_1) \bowtie_{\Gamma} \mathsf{abs}(e_2))$. So we know $\neg(e_1 \bowtie_{\Gamma} e_2)$.

Thus we are done. □

B.2 For XACC

B.2.1 Full Definition of XACC Algorithms like add-wins sets and remove-wins sets resolve conflicts following a specific "X-wins" strategy, while the operation X wins only when its effect is not canceled. We generalize ACC to support these algorithms, by enforcing the "X-wins" strategy specified using the won-by ($\blacktriangleleft$) and canceled-by ($\triangleright$) relations. Like $\bowtie$ (see Fig. 7), they are also binary relation over actions. The full specification is now a quadruple $(\Gamma, \bowtie, \blacktriangleleft, \triangleright)$.

For add-wins sets, `add(x)` wins over concurrent `remove(x)` (`remove(x)` $\blacktriangleleft$ `add(x)`), but it can also be cancelled by subsequent `remove(x)` (`add(x)` $\triangleright$ `remove(x)`); while for remove-wins sets, we have the inverse.

$$\begin{array}{l}
\mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((\mathit{ar}_{\mathsf{t}}, \mathit{ar}_{\mathsf{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd)) \text{ iff } \forall \mathcal{E}', \mathcal{E}'', e_0, e_1. \\
\quad \mathcal{E}' \leqslant \mathcal{E} \wedge \mathcal{E}'' \leqslant \mathcal{E} \ \wedge \ e_0 \bowtie_\Gamma e_1 \ \wedge \\
\qquad \{e_0, e_1\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}', \mathsf{t}, (\Gamma, \rhd)) \cap \mathsf{nc\text{-}vis}(\mathcal{E}'', \mathsf{t}', (\Gamma, \rhd)) \\
\quad \implies ((e_0, e_1) \in \mathit{ar}_{\mathsf{t}} \cap \mathit{ar}_{\mathsf{t}'} \vee (e_1, e_0) \in \mathit{ar}_{\mathsf{t}} \cap \mathit{ar}_{\mathsf{t}'}) \ \wedge \\
\qquad\quad (\mathsf{Concurrent}_{\mathcal{E}}(e_0, e_1) \wedge (e_0 \blacktriangleleft_\Gamma e_1) \implies (e_0, e_1) \in \mathit{ar}_{\mathsf{t}})
\end{array}$$

$$\begin{array}{l}
\mathsf{nc\text{-}vis}(\mathcal{E}, \mathsf{t}, (\Gamma, \rhd)) \overset{\text{def}}{=} \{e \mid e \in \mathsf{visible}(\mathcal{E}, \mathsf{t}) \ \wedge \\
\qquad\qquad \neg(\exists e'. \ e' \in \mathsf{visible}(\mathcal{E}, \mathsf{t}) \wedge (e \rhd_\Gamma e') \wedge (e \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e'))\}
\end{array}$$

$$\mathsf{Concurrent}_{\mathcal{E}}(e_0, e_1) \text{ iff } \neg(e_0 \overset{\mathsf{hb}}{\longmapsto}_{\mathcal{E}} e_1) \wedge \neg(e_1 \overset{\mathsf{hb}}{\longmapsto}_{\mathcal{E}} e_0)$$

$$\mathsf{PresvCancel}(\mathit{ar}_{\mathsf{t}}, \mathsf{t}, \mathcal{E}, (\Gamma, \rhd)) \text{ iff } \left(\overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} \cap \rhd_\Gamma\right)|_{\mathsf{visible}(\mathcal{E},\mathsf{t})} \subseteq \mathit{ar}_{\mathsf{t}}$$

Figure 19. Auxiliary Definitions for XACC

$\blacktriangleleft$ and $\rhd$ can only relate conflicting operations, that is, $\blacktriangleleft \subseteq \bowtie$ and $\rhd \subseteq \bowtie$. Also $\rhd$ should indeed capture the cancellation of effects, as defined in Def. 11. Like $\bowtie$, we also overload $\blacktriangleleft$ and $\rhd$ over operations and events.

Definition 11. $\mathsf{cancel}(\rhd)$ iff $\forall \alpha, \alpha'. \ \alpha \rhd \alpha' \implies$

$$\forall \alpha_1, \ldots, \alpha_n. \ \ \alpha \mathbin{⨾} \alpha_1 \mathbin{⨾} \ldots \mathbin{⨾} \alpha_n \mathbin{⨾} \alpha' = \alpha_1 \mathbin{⨾} \ldots \mathbin{⨾} \alpha_n \mathbin{⨾} \alpha'$$

Definition 12 (Causal Delivery)**.** $\mathsf{causalDelivery}(\mathcal{E})$ iff

$$\forall e_1, e_2. \ (e_1 \overset{\mathsf{hb}}{\longmapsto}_{\mathcal{E}} e_2) \implies \forall \mathsf{t}. \ e_2 \in \mathsf{visible}(\mathcal{E}, \mathsf{t}) \implies e_1 \prec^{\mathsf{t}}_{\mathcal{E}} e_2$$

Definition 13. $\mathsf{XACC}_\varphi(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$ iff

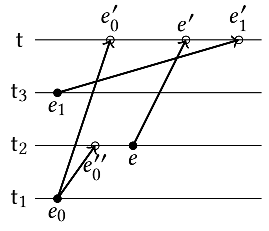
$$\begin{array}{l}
\forall \mathcal{S}, \mathcal{E}. \ \mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S}) \wedge \mathcal{S} \in \mathit{dom}(\varphi) \wedge \mathsf{causalDelivery}(\mathcal{E}) \\
\qquad \implies \mathsf{XACT}_\varphi(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))
\end{array}$$

Definition 14. $\mathsf{XACT}_\varphi(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$ iff $\exists \mathit{ar}_1, \ldots, \mathit{ar}_n$,

$$\begin{array}{l}
\forall \mathsf{t}. \ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(\mathit{ar}_{\mathsf{t}}) \wedge (\underset{\mathsf{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}} \subseteq \mathit{ar}_{\mathsf{t}}) \\
\ \wedge \ \mathsf{PresvCancel}(\mathit{ar}_{\mathsf{t}}, \mathsf{t}, \mathcal{E}, (\Gamma, \rhd)) \wedge \mathsf{ExecRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathit{ar}_{\mathsf{t}})) \\
\ \wedge \ \forall \mathsf{t}' \neq \mathsf{t}. \ \mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((\mathit{ar}_{\mathsf{t}}, \mathit{ar}_{\mathsf{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))
\end{array}$$

where we define PresvCancel and RCoh in Fig. 19.

XACT (see Def. 14) is similar to ACT, but it enforces the more relaxed coherence relation RCoh between the arbitration orders on different nodes. As defined in Fig. 19, RCoh requires that the arbitration orders ar_{t} and $\mathit{ar}_{\mathsf{t}'}$ of the nodes t and t′ enforce the same ordering for conflicting events e_0 and e_1, if neither e_0 or e_1 are canceled (i.e., $\{e_0, e_1\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}', \mathsf{t}, (\Gamma, \rhd)) \cap \mathsf{nc\text{-}vis}(\mathcal{E}'', \mathsf{t}', (\Gamma, \rhd))$). Moreover, the ordering must follow the won-by order $\blacktriangleleft$ if these two events are concurrent (i.e., neither one happens before the other). It is more relaxed in that, if either e_0 or e_1 is cancelled by others, they can be ordered differently in ar_{t} and $\mathit{ar}_{\mathsf{t}'}$. We illustrate one such scenario below.



In the right figure, suppose $e_0 \rhd e$ and $e_1 \blacktriangleleft e_0$ (e.g., e_0, e and e_1 are `add`, `remove` and `remove` operations of an add-wins set). From the figure we see $e_0 \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e$, Therefore $e_0 \notin \mathsf{nc\text{-}vis}(\mathcal{E}, \mathsf{t}, (\Gamma, \rhd))$. Therefore we do not need to care about the ordering between the conflicting e_0 and e_1 in ar_{t}. This is reasonable because, by the assumption of causal delivery we know e'_0 arrives at t earlier than e'. Therefore, e'_0 on t is canceled by e' and its effect is invisible to e'_1, so the order between e_0 and e_1 does not matter from the node t's point of view.

XACT also requires $\mathsf{PresvCancel}(\mathit{ar}_{\mathsf{t}}, \mathsf{t}, \mathcal{E}, (\Gamma, \rhd))$. It says, if e_1 is canceled by e_2 and e_1 is visible to e_2 on certain node (i.e., $e_1 \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e_2$), the arbitration order ar_{t} must order e_1 before e_2. For this reason the node t in the above figure must order e_0 before e.

B.2.2 The New Abstract Operational Semantics Figure 20 shows the new abstract operational semantics rules. For the replica $\mathbb{R}$ on each node t, we add a set of message IDs, ms, to keep track of the actions that t receives and is aware of their cancellation, i.e., t also has received the actions that cancel those in ms. We also add a relation between actions, $\mathbb{cr}$, which removes the ordering on canceled actions in ξ.

In the world $\mathbb{W}$, each message in the global message pool $\mathbb{M}$ now contains not only an action (f, n), but also a set of message IDs, which are the set of actions canceled by (f, n) on its origin node. We also keep track of the visibility relation of actions, $\mathbb{V}$, which is a mapping from a message ID mid to the set of message IDs which are visible to mid. We use $\mathbb{V}$ to enforce causal delivery in the abstract semantics.

The semantics rules are similar to those in Fig. 17, The main changes are made over the third world transition rule in Fig. 17(b) and over the first two local transition rules in Fig. 17(c).

For the new local transition rules in Fig. 20(c), when the client issues an operation request, as shown in the first rule, we calculate the set of operations ms_1 canceled by this operation and record them in ms (i.e., $ms' = ms \cup ms_1$). In addition, we pack ms together with the operation (f, n) into the message and put the message into the global and the local message pools.

The second rule says, if a node receives an operation request, it must have received all those operations that happen before this incoming operation (i.e., $\mathbb{V}(mid) \subseteq dom(\xi)$), due to causal delivery. We also record the set of canceled operations ms_1 in the local ms, and non-deterministically insert the incoming operation into the resulting ξ where we require the canceled operations ms_1 are all ordered before the incoming operation (i.e., $ms_1 \subseteq dom(\xi_0)$).

For the main global transition rule, the third rule in Fig. 20(b), we checks the coherence using AbsCoh-W, which follows RCoh in the definition of XACC.

B.2.3 Proofs for the Abstraction Theorem Below we prove separately the two directions of the equivalence:

Theorem 15. $\mathsf{XACC}_{\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \rhd)) \iff \Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie, \blacktriangleleft, \rhd)$.

Figure 21 gives auxiliary definitions used in the proofs.

Proof of Theorem 15 ($\Longrightarrow$). For any $C_1, \ldots, C_n, \mathcal{S}, \mathcal{S}_a, \mathcal{S}_c^1, \ldots, \mathcal{S}_c^n, \mathbb{S}_o^1, \ldots, \mathbb{S}_o^n$ and $\mathcal{E}$, suppose $(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n)) \in \mathcal{T}_{\mathsf{s}}(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, \mathcal{S})$, $\mathsf{causalDelivery}(\mathcal{E})$ and $\varphi(\mathcal{S}) = \mathcal{S}_a$. Let $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$. We want to prove that

$$(\mathbb{O}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\varphi(\mathbb{S}_o^1), \ldots, \varphi(\mathbb{S}_o^n))) \in \mathcal{T}_{\mathsf{s}}(\textbf{with } (\Gamma, \bowtie, \blacktriangleleft, \rhd) \textbf{ do } C_1 \parallel \ldots \parallel C_n, \varphi(\mathcal{S})).$$

From $\mathsf{XACC}_{\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$, we know there exists φ such that

$$\mathsf{XACT}_{\varphi}(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie, \blacktriangleleft, \rhd)) .$$

Thus we know

$$\begin{array}{l} \exists ar_1, \ldots, ar_n. \\ \quad \forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(ar_{\mathsf{t}}) \wedge (\overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_{\mathcal{E}} \subseteq ar_{\mathsf{t}}) \\ \qquad \wedge\ \mathsf{PresvCancel}(ar_{\mathsf{t}}, \mathsf{t}, \mathcal{E}, (\Gamma, \rhd)) \wedge \mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathsf{t}})) \\ \qquad \wedge\ \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((ar_{\mathsf{t}}, ar_{\mathsf{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd)) \end{array}$$

Since $(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n)) \in \mathcal{T}_{\mathsf{s}}(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, \mathcal{S})$, we know there exist m, W and W' such that

$$\begin{array}{c} ((\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, \mathcal{S}) \overset{\mathsf{load}}{\longmapsto} W) \wedge (W \xRightarrow{(\mathcal{E},(\mathcal{S}_c^1,\ldots,\mathcal{S}_c^n),(\mathbb{S}_o^1,\ldots,\mathbb{S}_o^n))}{}^{m}\, W'), \\ W = (\sigma_c, \sigma_o, \emptyset),\ W' = (\sigma_c', \sigma_o', M_s'), \\ \forall \mathsf{t} \in [1..n].\ \sigma_c'(\mathsf{t}) = (\_, \mathcal{S}_c^{\mathsf{t}}), \\ \forall \mathsf{t} \in [1..n].\ \sigma_o'(\mathsf{t}) = (\Pi, \mathcal{S}_o^{\mathsf{t}}, M_{\mathsf{t}}'). \end{array}$$

Let

$$\begin{array}{l} \mathbb{W} = (\sigma_c, \Sigma, \emptyset, \emptyset, \bowtie, \blacktriangleleft) \text{ and } \mathbb{W}' = (\sigma_c', \Sigma', \mathbb{M}_s', \mathbb{V}', \bowtie, \blacktriangleleft), \text{ where} \\ \forall \mathsf{t} \in [1..n].\ \Sigma(\mathsf{t}) = ((\Gamma, \rhd), \mathcal{S}_a, \epsilon, \emptyset, \emptyset) \\ \forall \mathsf{t} \in [1..n].\ \Sigma'(\mathsf{t}) = ((\Gamma, \rhd), \mathcal{S}_a, \xi_{\mathsf{t}}', ms_{\mathsf{t}}', \mathbb{cr}_{\mathsf{t}}') \\ \mathbb{M}_s' = \mathsf{abs\text{-}ms}(M_s', \mathcal{E}, (\Gamma, \rhd)) \\ \mathbb{V}' = \{(\mathsf{msgid}(e), \{\mathsf{msgid}(e') \mid e' \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e\}) \mid e \in \mathsf{orig}(\mathcal{E})\} \\ \forall \mathsf{t} \in [1..n].\ ms_{\mathsf{t}}' = \mathsf{get\text{-}all\text{-}ms}_{(\Gamma,\rhd)}(\mathcal{E}, \mathsf{t}) \\ \forall \mathsf{t} \in [1..n].\ \xi_{\mathsf{t}}' = \mathsf{abs}(M_{\mathsf{t}}' \downharpoonleft ar_{\mathsf{t}}) \\ \forall \mathsf{t} \in [1..n].\ \mathbb{cr}_{\mathsf{t}}' = \bigcup_{\mathcal{E}' \leqslant \mathcal{E}} (ar_{\mathsf{t}}|_{\mathsf{nc\text{-}vis}(\mathcal{E}',\mathsf{t},(\Gamma,\rhd))}) \end{array}$$

$$
\begin{array}{rrcl}
(\mathit{AbsProg}) & \mathbb{P} & ::= & \textbf{with } (\Gamma, \bowtie, \blacktriangleleft, \triangleright) \textbf{ do } C_1 \,\|\, \ldots \,\|\, C_n \\
(\mathit{AbsOpEvent}) & \mathbb{e} & ::= & (\mathit{mid}, (f, n)) \\
(\mathit{AbsOpHist}) & \xi & ::= & \epsilon \mid \mathbb{e} :: \xi \\
(\mathit{MsgIDSet}) & \mathit{ms} & \in & \mathscr{P}(\mathit{MsgID}) \\
(\mathit{AbsEvtRel}) & \mathbb{er} & \in & \mathscr{P}(\mathit{AbsOpEvent} \times \mathit{AbsOpEvent}) \\
(\mathit{AbsReplica}) & \mathbb{R} & ::= & ((\Gamma, \triangleright), \mathcal{S}, \xi, \mathit{ms}, \mathbb{er}) \\
(\mathit{AbsState}) & \Sigma & ::= & \{\mathrm{t}_1 \rightsquigarrow \mathbb{R}_1, \ldots, \mathrm{t}_n \rightsquigarrow \mathbb{R}_n\} \\
(\mathit{AbsMsgSoup}) & \mathbb{M} & \in & \mathit{MsgID} \rightharpoonup (\mathit{OpName} \times \mathit{Val}) \times \mathit{MsgIDSet} \\
(\mathit{VisMap}) & \mathbb{V} & \in & \mathit{MsgID} \rightharpoonup \mathit{MsgIDSet} \\
(\mathit{AbsWorld}) & \mathbb{W} & ::= & (\sigma_c, \Sigma, \mathbb{M}, \mathbb{V}, \bowtie, \blacktriangleleft) \\
(\mathit{ObsvEvent}) & \mathbb{o} & ::= & (\mathit{mid}, \mathrm{t}, (f, n, n')) \mid (\mathit{mid}, \mathrm{t}, (f, n)) \qquad\qquad (\mathit{AbsLabel})\ \mathbb{l} \ ::= \ \mathbb{o} \mid \tau \\
(\mathit{ObsvTrace}) & \mathbb{O} & ::= & \epsilon \mid \mathbb{o} :: \mathbb{O}
\end{array}
$$

(a) world and event trace

$$
\frac{\forall \mathrm{t} \in [1..n].\ \sigma_c(\mathrm{t}) = (C_{\mathrm{t}}, \emptyset) \qquad \forall \mathrm{t} \in [1..n].\ \Sigma(\mathrm{t}) = ((\Gamma, \triangleright), \mathcal{S}_0, \epsilon, \emptyset, \emptyset)}{(\textbf{with } (\Gamma, \bowtie, \blacktriangleleft, \triangleright) \textbf{ do } C_1 \,\|\, \ldots \,\|\, C_n, \mathcal{S}_0) \mathrel{\diamond\!\!\xrightarrow{\text{load}}} (\sigma_c, \Sigma, \emptyset, \emptyset, \bowtie, \blacktriangleleft)}
$$

$$
\frac{\begin{array}{c} \mathit{dom}(\sigma_c) = [1..n] \qquad \text{for all } \mathrm{t} \in \mathit{dom}(\sigma_c): \qquad \sigma_c(\mathrm{t}) = (\textbf{skip}, \mathcal{S}_{\mathrm{t}}) \qquad \Sigma(\mathrm{t}) = ((\Gamma, \triangleright), \mathcal{S}_0, \xi_{\mathrm{t}}, \mathit{ms}_{\mathrm{t}}, \mathbb{er}_{\mathrm{t}}) \\ \mathit{dom}(\xi_{\mathrm{t}}) = \mathit{dom}(\mathbb{M}) \qquad \mathcal{S}'_{\mathrm{t}} = \mathsf{aexecST}(\Gamma, \mathcal{S}_0, \xi_{\mathrm{t}}) \end{array}}{(\sigma_c, \Sigma, \mathbb{M}, \mathbb{V}, \bowtie, \blacktriangleleft) \mathrel{\diamond\!\!\longrightarrow} (\textbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}'_1, \ldots, \mathcal{S}'_n))}
$$

$$
\frac{\begin{array}{c} \sigma_c(\mathrm{t}) = \varsigma \qquad \Sigma(\mathrm{t}) = \mathbb{R} \qquad (\varsigma, \mathbb{R}, \mathbb{M}, \mathbb{V}) \mathrel{\diamond\!\!\xrightarrow{\mathbb{l}}}_{\mathrm{t}} (\varsigma', \mathbb{R}', \mathbb{M}', \mathbb{V}') \qquad \mathbb{R}' = ((\Gamma, \triangleright), \mathcal{S}_0, \xi, \mathit{ms}, \mathbb{er}) \\ \Sigma(\mathrm{t}') = ((\Gamma, \triangleright), \mathcal{S}_0, \xi', \mathit{ms}', \mathbb{er}') \qquad \forall \mathrm{t}' \neq \mathrm{t}.\ \mathsf{AbsCoh\text{-}W}(\mathbb{er}, \mathbb{er}', \mathbb{V}, (\Gamma, \bowtie, \blacktriangleleft)) \end{array}}{(\sigma_c, \Sigma, \mathbb{M}, \mathbb{V}, \bowtie, \blacktriangleleft) \mathrel{\diamond\!\!\xrightarrow{\mathbb{l}}} (\sigma_c\{\mathrm{t} \rightsquigarrow \varsigma'\}, \Sigma\{\mathrm{t} \rightsquigarrow \mathbb{R}'\}, \mathbb{M}', \mathbb{V}', \bowtie, \blacktriangleleft)}
$$

where $\mathsf{AbsCoh\text{-}W}(\mathbb{er}, \mathbb{er}', \mathbb{V}, (\Gamma, \bowtie, \blacktriangleleft))$ iff

$$
\begin{array}{l}
\forall \mathbb{e}_1, \mathbb{e}_2.\ (\{(\mathbb{e}_1, \mathbb{e}_2), (\mathbb{e}_2, \mathbb{e}_1)\} \cap \mathbb{er} \neq \emptyset) \wedge (\{(\mathbb{e}_1, \mathbb{e}_2), (\mathbb{e}_2, \mathbb{e}_1)\} \cap \mathbb{er}' \neq \emptyset) \wedge (\mathbb{e}_1 \bowtie_\Gamma \mathbb{e}_2) \\
\implies ((\mathbb{e}_1, \mathbb{e}_2) \in \mathbb{er} \cap \mathbb{er}' \vee (\mathbb{e}_2, \mathbb{e}_1) \in \mathbb{er} \cap \mathbb{er}') \\
\quad \wedge (\mathbb{e}_1.\mathit{mid} \notin \mathbb{V}(\mathbb{e}_2.\mathit{mid}) \wedge \mathbb{e}_2.\mathit{mid} \notin \mathbb{V}(\mathbb{e}_1.\mathit{mid}) \wedge (\mathbb{e}_1 \blacktriangleleft_\Gamma \mathbb{e}_2) \implies \mathbb{e}_1\ \mathbb{er}\ \mathbb{e}_2)
\end{array}
$$

(b) world transitions

$$
\frac{\begin{array}{c} [\![E]\!]_{\mathcal{S}_c} = n \qquad \mathit{mid} \notin \mathit{dom}(\mathbb{M}_s) \qquad \mathit{ms}_1 = \mathsf{cancelled}_{(\Gamma, \triangleright)}(\xi, (f, n)) \\ \mathbb{M}'_s = \mathbb{M}_s \uplus \{\mathit{mid} \rightsquigarrow ((f, n), \mathit{ms}_1)\} \qquad \mathit{ms}' = \mathit{ms} \cup \mathit{ms}_1 \qquad \mathbb{V}' = \mathbb{V} \uplus \{\mathit{mid} \rightsquigarrow \mathit{dom}(\xi)\} \\ \mathbb{e} = (\mathit{mid}, (f, n)) \qquad \xi' = \xi{+}{+}[\mathbb{e}] \qquad \mathsf{aexecRV}(\Gamma, \mathcal{S}, \xi') = n' \qquad \mathbb{er}' = \mathbb{er} \cup (\lfloor \xi \backslash \mathit{ms}' \rfloor \times \{\mathbb{e}\}) \end{array}}{((x := f(E), \mathcal{S}_c), ((\Gamma, \triangleright), \mathcal{S}, \xi, \mathit{ms}, \mathbb{er}), \mathbb{M}_s, \mathbb{V}) \mathrel{\diamond\!\!\xrightarrow{(\mathit{mid}, \mathrm{t}, (f, n, n'))}}_{\mathrm{t}} ((\textbf{skip}, \mathcal{S}_c\{x \rightsquigarrow n'\}), ((\Gamma, \triangleright), \mathcal{S}, \xi', \mathit{ms}', \mathbb{er}'), \mathbb{M}'_s, \mathbb{V}')}
$$

where $\mathsf{cancelled}_{(\Gamma, \triangleright)}(\xi, (f, n)) \overset{\text{def}}{=}$
$\{\mathit{mid} \mid \exists f', n'.\ ((\mathit{mid}, (f', n')) \in \xi) \wedge ((f', n') \triangleright_\Gamma (f, n))\}$,
and $(\xi \backslash \mathit{ms})$ removes from ξ those events in ms, and $\lfloor \xi \rfloor$ turns the sequence to a set.

$$
\frac{\begin{array}{c} \mathbb{M}_s(\mathit{mid}) = ((f, n), \mathit{ms}_1) \qquad \mathit{mid} \notin \mathit{dom}(\xi) \qquad \mathbb{V}(\mathit{mid}) \subseteq \mathit{dom}(\xi) \qquad \xi = \xi_0{+}{+}\xi_1 \qquad \mathit{ms}_1 \subseteq \mathit{dom}(\xi_0) \\ \mathbb{e} = (\mathit{mid}, (f, n)) \qquad \xi' = \xi_0{+}{+}[\mathbb{e}]{+}{+}\xi_1 \qquad \mathbb{er}' = \mathbb{er} \cup (\lfloor \xi_0 \backslash \mathit{ms}' \rfloor \times \{\mathbb{e}\}) \cup (\{\mathbb{e}\} \times \lfloor \xi_1 \backslash \mathit{ms}' \rfloor) \qquad \mathit{ms}' = \mathit{ms} \cup \mathit{ms}_1 \end{array}}{(\varsigma_c, ((\Gamma, \triangleright), \mathcal{S}, \xi, \mathit{ms}, \mathbb{er}), \mathbb{M}_s, \mathbb{V}) \mathrel{\diamond\!\!\xrightarrow{(\mathit{mid}, \mathrm{t}, (f, n))}}_{\mathrm{t}} (\varsigma_c, ((\Gamma, \triangleright), \mathcal{S}, \xi', \mathit{ms}', \mathbb{er}'), \mathbb{M}_s, \mathbb{V})}
$$

(c) local transitions

Figure 20. Abstract operational semantics for XACC objects.

$$\begin{array}{l}
\mathsf{abs}(mid, \mathsf{t}, (f, n, n', \delta)) \overset{\text{def}}{=} (mid, (f, n)) \\
\mathsf{get\text{-}ms}_{(\Gamma,\triangleright)}(\mathcal{E}, e) \overset{\text{def}}{=} \{e' \mid (e' \xrightarrow{\mathsf{vis}}_{\mathcal{E}} e) \wedge (e' \triangleright_{\Gamma} e)\} \\
\mathsf{abs\text{-}ms}(M, \mathcal{E}, (\Gamma, \triangleright)) \overset{\text{def}}{=} \\
\quad \{(mid, ((f, n), ms)) \mid \exists \delta.\ M(mid) = ((f, n), \delta) \wedge \exists e.\ e \in \mathsf{orig}(\mathcal{E}) \wedge \mathsf{msgid}(e) = mid \wedge ms = \mathsf{get\text{-}ms}_{(\Gamma,\triangleright)}(\mathcal{E}, e)\} \\
\mathsf{get\text{-}all\text{-}ms}_{(\Gamma,\triangleright)}(\mathcal{E}, \mathsf{t}) \overset{\text{def}}{=} \bigcup\{\mathsf{get\text{-}ms}_{(\Gamma,\triangleright)}(\mathcal{E}, e) \mid e \in \mathsf{visible}(\mathcal{E}, \mathsf{t})\}
\end{array}$$

Figure 21. Auxiliary Definitions for the Proof of Theorem 15.

where we give the definitions of abs, abs-ms and get-all-ms in Fig. 21.

Since $(\textbf{with } (\Gamma, \bowtie, \blacktriangleleft, \triangleright) \textbf{ do } C_1 \parallel \ldots \parallel C_n, \mathcal{S}_a) \xrightarrow{\mathsf{load}} \mathbb{W}$, we only need to prove

$$\mathbb{W} \xRightarrow{(\mathbb{O}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\varphi(\mathbb{S}_o^1), \ldots, \varphi(\mathbb{S}_o^n)))}{}^{m}\ \mathbb{W}'.$$

The proof is by induction over m.

- $m = 0$. Trivial.
- $m = k + 1$.

Since $W \xRightarrow{(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n))}{}^{m} W'$, we know there exist $\mathcal{E}'$, ι and W'' such that

$$W \xRightarrow{(\mathcal{E}', (\mathcal{S}_1', \ldots, \mathcal{S}_n'), (\mathbb{S}_1'', \ldots, \mathbb{S}_n''))}{}^{k} W'' \text{ and } W'' \xrightarrow{\iota} W',$$

and, if $\iota = \tau$, then $\mathcal{E} = \mathcal{E}'$ and $\forall i.\ \mathbb{S}_o^i = \mathbb{S}_i''$; otherwise, $\mathcal{E} = \mathcal{E}'{+\!+}[\iota]$ and $\forall i.\ \mathbb{S}_o^i = \mathbb{S}_i''{+\!+}[\mathcal{S}_o^i]$.

Suppose $W'' = (\sigma_c'', \sigma_o'', M_s'')$, where $\forall \mathsf{t} \in [1..n].\ \sigma_o''(\mathsf{t}) = (\Pi, \mathcal{S}'', M_{\mathsf{t}}'')$. Let

$$\begin{array}{l}
\mathbb{O}' = \mathsf{obsv}(\mathcal{E}'),\ \mathbb{I} = \mathsf{obsv}(\iota) \text{ and } \mathbb{W}'' = (\sigma_c'', \Sigma'', \mathbb{M}_s'', \mathbb{V}'', \bowtie, \blacktriangleleft), \text{ where} \\
\quad \forall \mathsf{t} \in [1..n].\ \Sigma''(\mathsf{t}) = ((\Gamma, \triangleright), \mathcal{S}_a, \xi_{\mathsf{t}}'', ms_{\mathsf{t}}'', \mathbb{er}_{\mathsf{t}}'') \\
\quad \mathbb{M}_s'' = \mathsf{abs\text{-}ms}(M_s'', \mathcal{E}', (\Gamma, \triangleright)) \\
\quad \mathbb{V}'' = \{(\mathsf{msgid}(e), \{\mathsf{msgid}(e') \mid e' \xrightarrow{\mathsf{vis}}_{\mathcal{E}'} e\}) \mid e \in \mathsf{orig}(\mathcal{E}')\} \\
\quad \forall \mathsf{t} \in [1..n].\ ms_{\mathsf{t}}'' = \mathsf{get\text{-}all\text{-}ms}_{(\Gamma,\triangleright)}(\mathcal{E}', \mathsf{t}) \\
\quad \forall \mathsf{t} \in [1..n].\ \xi_{\mathsf{t}}'' = \mathsf{abs}(M_{\mathsf{t}}'' \downharpoonright ar_{\mathsf{t}}') \\
\quad \forall \mathsf{t} \in [1..n].\ \mathbb{er}_{\mathsf{t}}'' = \bigcup_{\mathcal{E}'' \leqslant \mathcal{E}'}(ar_{\mathsf{t}}'|_{\mathsf{nc\text{-}vis}(\mathcal{E}'', \mathsf{t}, (\Gamma, \triangleright))}) \\
\quad \forall \mathsf{t} \in [1..n].\ ar_{\mathsf{t}}' = ar_{\mathsf{t}}|_{\mathsf{visible}(\mathcal{E}', \mathsf{t})}
\end{array}$$

To apply the induction hypothesis, we prove:

$$\begin{array}{l}
\forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}', \mathsf{t})}(ar_{\mathsf{t}}') \wedge (\underset{\mathsf{t}}{\xrightarrow{\mathsf{vis}}}_{\mathcal{E}'} \subseteq ar_{\mathsf{t}}') \\
\wedge\ \mathsf{PresvCancel}(ar_{\mathsf{t}}', \mathsf{t}, \mathcal{E}', (\Gamma, \triangleright)) \\
\wedge\ \mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}', \mathcal{S}), (\Gamma, ar_{\mathsf{t}}')) \\
\wedge\ \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{RCoh}_{(\mathsf{t}, \mathsf{t}')}((ar_{\mathsf{t}}', ar_{\mathsf{t}'}'), \mathcal{E}', (\Gamma, \bowtie, \blacktriangleleft, \triangleright))
\end{array}$$

By the induction hypothesis, we know

$$\mathbb{W} \xRightarrow{(\mathcal{E}', (\mathcal{S}_1', \ldots, \mathcal{S}_n'), (\varphi(\mathbb{S}_1''), \ldots, \varphi(\mathbb{S}_n'')))}{}^{k}\ \mathbb{W}''.$$

Since $\forall \mathsf{t}.\ \mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathsf{t}}))$, we know

$$\forall \mathsf{t}.\ \varphi(\mathcal{S}_o^{\mathsf{t}}) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}, \mathsf{t}) \downharpoonright ar_{\mathsf{t}}).$$

From the concrete operational semantics, we can prove:

$$\mathsf{visible}(\mathcal{E}, \mathsf{t}) = M_{\mathsf{t}}'$$

Thus

$$\forall \mathsf{t}.\ \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}, \mathsf{t}) \downharpoonright ar_{\mathsf{t}}) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \xi_{\mathsf{t}}').$$

Thus

$$\forall \mathsf{t}.\ \varphi(\mathcal{S}_o^{\mathsf{t}}) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \xi_{\mathsf{t}}').$$

So we only need to prove $\mathbb{W}'' \xrightarrow{\mathbb{I}} \mathbb{W}'$.

- Suppose $\iota = \tau$ and $\mathcal{E} = \mathcal{E}'$. Then $W'' \longrightarrow W'$ is a client step. Thus $\mathbb{W}'' \overset{\diamond}{\longmapsto} \mathbb{W}'$.
- Suppose $\iota = e$ and $\mathcal{E} = \mathcal{E}'{+}{+}[e]$.

  Suppose $\mathsf{tid}(e) = \mathrm{t}$. We first prove $(\sigma_c''(\mathrm{t}), \Sigma''(\mathrm{t}), \mathbb{M}_s'', \mathbb{V}'') \overset{\mathbb{I}}{\diamond\!\!\longrightarrow}_{\mathrm{t}} (\sigma_c'(\mathrm{t}), \Sigma'(\mathrm{t}), \mathbb{M}_s', \mathbb{V}')$. The proof is by case analysis over the event e.
  - $e = (\mathit{mid}, \mathrm{t}, (f, n, n', \delta))$.
    From the operational semantics, we know there exists x, E and C_{t}' such that
    $$\sigma_c''(\mathrm{t}) = ((x := f(E); C_{\mathrm{t}}'), \mathcal{S}_c'') \qquad \mathcal{S}_c' = \mathcal{S}_c''\{x \rightsquigarrow n'\} \qquad \sigma_c' = \sigma_c''\{\mathrm{t} \rightsquigarrow (C_{\mathrm{t}}', \mathcal{S}_c')\}$$
    $$[\![E]\!]_{\mathcal{S}_c''} = n \qquad \Pi(f,n)(\mathcal{S}'') = (n', \delta) \qquad \mathit{mid} \notin \mathit{dom}(M_s'')$$
    $$M_s' = M_s'' \uplus \{\mathit{mid} \rightsquigarrow ((f,n),\delta)\} \qquad \delta(\mathcal{S}'') = \mathcal{S}' \qquad M_{\mathrm{t}}' = M_{\mathrm{t}}'' \uplus \{\mathit{mid} \rightsquigarrow ((f,n),\delta)\}$$
    Since $\mathit{mid} \notin \mathit{dom}(M_s'')$, we know
    $$\mathit{mid} \notin \mathit{dom}(\mathbb{M}_s'')$$
    Also, from the concrete operational semantics, we can prove:
    $$\mathsf{visible}(\mathcal{E}', \mathrm{t}) = M_{\mathrm{t}}''$$
    Thus we know
    $$\forall e' \in M_{\mathrm{t}}''.\ e' \underset{\mathrm{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}}\ e$$
    Then, since $\underset{\mathrm{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}} \subseteq \mathit{ar}_{\mathrm{t}}$, we know
    $$\forall e' \in M_{\mathrm{t}}''.\ (e', e) \in \mathit{ar}_{\mathrm{t}}$$
    Let
    $$\mathit{ms} = \mathsf{cancelled}_{(\Gamma,\rhd)}(\xi_{\mathrm{t}}'', (f,n)).$$
    Then we know
    $$\mathit{ms} = \mathsf{get\text{-}ms}_{(\Gamma,\rhd)}(\mathcal{E}, e).$$
    Let
    $$\mathbb{e} = (\mathit{mid}, (f,n)),\ \mathbb{M}_s' = \mathbb{M}_s'' \uplus \{\mathit{mid} \rightsquigarrow ((f,n), \mathit{ms})\},\ \mathit{ms}_{\mathrm{t}}' = \mathit{ms}_{\mathrm{t}}'' \cup \mathit{ms},\ \xi_{\mathrm{t}}' = \xi_{\mathrm{t}}''{+}{+}[\mathbb{e}].$$
    Since $\mathbb{M}_s'' = \mathsf{abs\text{-}ms}(M_s'', \mathcal{E}', (\Gamma, \rhd))$, we know
    $$\mathbb{M}_s' = \mathsf{abs\text{-}ms}(M_s', \mathcal{E}, (\Gamma, \rhd))$$
    Since $\mathit{ms}_{\mathrm{t}}'' = \mathsf{get\text{-}all\text{-}ms}_{(\Gamma,\rhd)}(\mathcal{E}', \mathrm{t})$, we know
    $$\mathit{ms}_{\mathrm{t}}' = \mathsf{get\text{-}all\text{-}ms}_{(\Gamma,\rhd)}(\mathcal{E}, \mathrm{t})$$
    Since $\xi_{\mathrm{t}}'' = \mathsf{abs}(M_{\mathrm{t}}'' \downharpoonleft \mathit{ar}_{\mathrm{t}}')$, we know
    $$\xi_{\mathrm{t}}' = \mathsf{abs}(M_{\mathrm{t}}' \downharpoonleft \mathit{ar}_{\mathrm{t}})$$
    Also, since $\mathsf{ExecRelated}_{\varphi}(\mathrm{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathit{ar}_{\mathrm{t}}))$, we know
    $$n' = \mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma, \mathcal{S}_a, M_{\mathrm{t}}' \downharpoonleft \mathit{ar}_{\mathrm{t}})$$
    Thus we know
    $$\mathsf{aexecRV}(\Gamma, \mathcal{S}_a, \xi_{\mathrm{t}}') = n'$$
    Let
    $$\mathbb{V}' = \mathbb{V}'' \uplus \{\mathit{mid} \rightsquigarrow \mathit{dom}(\xi_{\mathrm{t}}'')\},\ \mathbb{er}' = \mathbb{er}'' \cup (\lfloor \xi_{\mathrm{t}}'' \backslash \mathit{ms}_{\mathrm{t}}' \rfloor \times \{\mathbb{e}\}).$$
    Thus we know
    $$\mathbb{V}' = \{(\mathsf{msgid}(e), \{\mathsf{msgid}(e') \mid e' \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e\}) \mid e \in \mathsf{orig}(\mathcal{E})\},$$
    $$\forall \mathrm{t} \in [1..n].\ \mathbb{er}_{\mathrm{t}}' = \textstyle\bigcup_{\mathcal{E}'' \leqslant \mathcal{E}} (\mathit{ar}_{\mathrm{t}}|_{\mathsf{nc\text{-}vis}(\mathcal{E}'', \mathrm{t}, (\Gamma,\rhd))})$$
    And, by the abstract operational semantics, we know
    $$(\sigma_c''(\mathrm{t}), \Sigma''(\mathrm{t}), \mathbb{M}_s'', \mathbb{V}'') \overset{\mathbb{I}}{\diamond\!\!\longrightarrow}_{\mathrm{t}} (\sigma_c'(\mathrm{t}), \Sigma'(\mathrm{t}), \mathbb{M}_s', \mathbb{V}').$$
  - $e = (\mathit{mid}, \mathrm{t}, (f,n), \delta)$.
    From the operational semantics, we know
    $$M_s''(\mathit{mid}) = ((f,n),\delta) \qquad \mathit{mid} \notin \mathit{dom}(M_{\mathrm{t}}'') \qquad \delta(\mathcal{S}'') = \mathcal{S}'$$
    $$M_{\mathrm{t}}' = M_{\mathrm{t}}'' \uplus \{\mathit{mid} \rightsquigarrow ((f,n),\delta)\} \qquad M_s' = M_s''$$
    Since $\mathbb{M}_s'' = \mathsf{abs\text{-}ms}(M_s'', \mathcal{E}', (\Gamma, \rhd))$, we know there exists ms such that
    $$\mathbb{M}_s''(\mathit{mid}) = ((f,n), \mathit{ms}),\ \exists e.\ e \in \mathsf{orig}(\mathcal{E}') \wedge \mathsf{msgid}(e) = \mathit{mid} \wedge \mathit{ms} = \mathsf{get\text{-}ms}_{(\Gamma,\rhd)}(\mathcal{E}', e)$$
    Since $\mathsf{causalDelivery}(\mathcal{E})$, we know
    $$\mathit{ms} \subseteq \mathit{dom}(\xi_{\mathrm{t}}'') \text{ and } \mathbb{V}''(\mathit{mid}) \subseteq \mathit{dom}(\xi_{\mathrm{t}}'').$$
    Since $\xi_{\mathrm{t}}'' = \mathsf{abs}(M_{\mathrm{t}}'' \downharpoonleft \mathit{ar}_{\mathrm{t}}')$, we know
    $$\mathit{mid} \notin \mathit{dom}(\xi_{\mathrm{t}}'')$$
    Let

$$\mathbb{M}'_s = \mathbb{M}''_s \text{ and } ms'_{\mathrm{t}} = ms''_{\mathrm{t}} \cup ms.$$

Thus

$$\mathbb{M}'_s = \mathsf{abs\text{-}ms}(M'_s, \mathcal{E}, (\Gamma, \rhd)),\ ms'_{\mathrm{t}} = \mathsf{get\text{-}all\text{-}ms}_{(\Gamma,\rhd)}(\mathcal{E}, \mathrm{t}).$$

Let

$$\xi'_{\mathrm{t}} = \mathsf{abs}(M'_{\mathrm{t}} \downharpoonright ar_{\mathrm{t}})$$

Then, since $\xi''_{\mathrm{t}} = \mathsf{abs}(M''_{\mathrm{t}} \downharpoonright ar'_{\mathrm{t}})$, we know there exist ξ_1 and ξ_2 such that

$$\xi''_{\mathrm{t}} = \xi_1 {+\!+} \xi_2 \text{ and } \xi'_{\mathrm{t}} = \xi_1 {+\!+} [(mid, (f, n))] {+\!+} \xi_2$$

From $\mathsf{PresvCancel}(ar_{\mathrm{t}}, \mathrm{t}, \mathcal{E}, (\Gamma, \rhd))$, we know $\left(\xrightarrow{\mathsf{vis}}_{\mathcal{E}} \cap \rhd_\Gamma\right)|_{\mathsf{visible}(\mathcal{E},\mathrm{t})} \subseteq ar_{\mathrm{t}}$. Thus

$$\forall e'.\ e' \in ms \implies (e', e) \in ar_{\mathrm{t}}.$$

Thus

$$ms \subseteq dom(\xi_1).$$

Let

$$\mathbb{er}' = \mathbb{er}'' \cup (\lfloor \xi_1 \backslash ms'_{\mathrm{t}} \rfloor \times \{(mid, (f, n))\}) \cup (\{(mid, (f, n))\} \times \lfloor \xi_2 \backslash ms'_{\mathrm{t}} \rfloor).$$

Thus we know

$$\forall \mathrm{t} \in [1..n].\ \mathbb{er}'_{\mathrm{t}} = \textstyle\bigcup_{\mathcal{E}'' \leqslant \mathcal{E}} (ar_{\mathrm{t}}|_{\mathsf{nc\text{-}vis}(\mathcal{E}'',\mathrm{t},(\Gamma,\rhd))})$$

And, by the abstract operational semantics, we know

$$(\sigma''_c(\mathrm{t}), \Sigma''(\mathrm{t}), \mathbb{M}''_s, \mathbb{V}'') \overset{\mathbb{I}}{\diamond\!\!\longrightarrow}_{\mathrm{t}} (\sigma'_c(\mathrm{t}), \Sigma'(\mathrm{t}), \mathbb{M}'_s, \mathbb{V}').$$

Next we prove $\forall \mathrm{t}' \neq \mathrm{t}.\ \mathsf{AbsCoh\text{-}W}(\mathbb{er}'_{\mathrm{t}}, \mathbb{er}'_{\mathrm{t}'}, \mathbb{V}', (\Gamma, \bowtie, \blacktriangleleft))$.

For any $\mathbb{e}_1$ and $\mathbb{e}_2$, suppose $\{(\mathbb{e}_1, \mathbb{e}_2), (\mathbb{e}_2, \mathbb{e}_1)\} \cap \mathbb{er}'_{\mathrm{t}} \neq \emptyset$, $\{(\mathbb{e}_1, \mathbb{e}_2), (\mathbb{e}_2, \mathbb{e}_1)\} \cap \mathbb{er}'_{\mathrm{t}'} \neq \emptyset$ and $\mathbb{e}_1 \bowtie_\Gamma \mathbb{e}_2$, we want to prove $((\mathbb{e}_1, \mathbb{e}_2) \in \mathbb{er}'_{\mathrm{t}} \cap \mathbb{er}'_{\mathrm{t}'} \vee (\mathbb{e}_2, \mathbb{e}_1) \in \mathbb{er}'_{\mathrm{t}} \cap \mathbb{er}'_{\mathrm{t}'})$ and $(\mathbb{e}_1.mid \notin \mathbb{V}'(\mathbb{e}_2.mid) \wedge \mathbb{e}_2.mid \notin \mathbb{V}'(\mathbb{e}_1.mid) \wedge (\mathbb{e}_1 \blacktriangleleft_\Gamma \mathbb{e}_2) \implies \mathbb{e}_1\ \mathbb{er}'_{\mathrm{t}}\ \mathbb{e}_2)$.

Since $\{(\mathbb{e}_1, \mathbb{e}_2), (\mathbb{e}_2, \mathbb{e}_1)\} \cap \mathbb{er}'_{\mathrm{t}} \neq \emptyset$, from $\mathbb{er}'_{\mathrm{t}} = \bigcup_{\mathcal{E}' \leqslant \mathcal{E}} (ar_{\mathrm{t}}|_{\mathsf{nc\text{-}vis}(\mathcal{E}',\mathrm{t},(\Gamma,\rhd))})$, we know there exist e_1, e_2 and $\mathcal{E}' \leqslant \mathcal{E}$ such that

$$\mathsf{abs}(e_1) = \mathbb{e}_1,\ \mathsf{abs}(e_2) = \mathbb{e}_2,\ \{e_1, e_2\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}', \mathrm{t}, (\Gamma, \rhd))$$

Similarly, we know there exists $\mathcal{E}'' \leqslant \mathcal{E}$ such that

$$\{e_1, e_2\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}'', \mathrm{t}', (\Gamma, \rhd))$$

From $\mathsf{RCoh}_{(\mathrm{t},\mathrm{t}')}((ar_{\mathrm{t}}, ar_{\mathrm{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$, we know

$$((e_1, e_2) \in ar_{\mathrm{t}} \cap ar_{\mathrm{t}'} \vee (e_2, e_1) \in ar_{\mathrm{t}} \cap ar_{\mathrm{t}'}) \text{ and } (\mathsf{Concurrent}_{\mathcal{E}}(e_1, e_2) \wedge (e_1 \blacktriangleleft_\Gamma e_2) \implies (e_1, e_2) \in ar_{\mathrm{t}})$$

Thus we know

$$((\mathbb{e}_1, \mathbb{e}_2) \in \mathbb{er}'_{\mathrm{t}} \cap \mathbb{er}'_{\mathrm{t}'} \vee (\mathbb{e}_2, \mathbb{e}_1) \in \mathbb{er}'_{\mathrm{t}} \cap \mathbb{er}'_{\mathrm{t}'})$$

Besides, if $\mathbb{e}_1.mid \notin \mathbb{V}'(\mathbb{e}_2.mid) \wedge \mathbb{e}_2.mid \notin \mathbb{V}'(\mathbb{e}_1.mid)$, we know

$$\neg(e_1 \xrightarrow{\mathsf{vis}}_{\mathcal{E}} e_2) \wedge \neg(e_2 \xrightarrow{\mathsf{vis}}_{\mathcal{E}} e_1)$$

From $\mathsf{causalDelivery}(\mathcal{E})$, we know

$$\mathsf{Concurrent}_{\mathcal{E}}(e_1, e_2).$$

So $(e_1 \blacktriangleleft_\Gamma e_2) \implies (e_1, e_2) \in ar_{\mathrm{t}}$. Thus

$$(\mathbb{e}_1 \blacktriangleleft_\Gamma \mathbb{e}_2) \implies \mathbb{e}_1\ \mathbb{er}'_{\mathrm{t}}\ \mathbb{e}_2$$

As a result, we know

$$\mathsf{AbsCoh\text{-}W}(\mathbb{er}'_{\mathrm{t}}, \mathbb{er}'_{\mathrm{t}'}, \mathbb{V}', (\Gamma, \bowtie, \blacktriangleleft)).$$

Thus we know

$$\mathbb{W}'' \overset{\mathbb{I}}{\blacklozenge\!\!\longrightarrow} \mathbb{W}'.$$

Thus we are done. □

Proof of Theorem 15 ($\Longleftarrow$). For any $\mathcal{S}$, $\mathcal{S}_a$ and $\mathcal{E}$, suppose $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$ and $\varphi(\mathcal{S}) = \mathcal{S}_a$ and $\mathsf{causalDelivery}(\mathcal{E})$. We want to prove $\mathsf{XACT}_\varphi(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$. That is, we want to prove:

$$\begin{array}{l}
\exists ar_1, \dots, ar_n. \\
\quad \forall \mathrm{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathrm{t})}(ar_{\mathrm{t}}) \wedge (\xrightarrow[\mathrm{t}]{\mathsf{vis}}_{\mathcal{E}} \subseteq ar_{\mathrm{t}}) \\
\qquad \wedge\ \mathsf{PresvCancel}(ar_{\mathrm{t}}, \mathrm{t}, \mathcal{E}, (\Gamma, \rhd)) \wedge \mathsf{ExecRelated}_\varphi(\mathrm{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathrm{t}})) \\
\qquad \wedge\ \forall \mathrm{t}' \neq \mathrm{t}.\ \mathsf{RCoh}_{(\mathrm{t},\mathrm{t}')}((ar_{\mathrm{t}}, ar_{\mathrm{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))
\end{array}$$

From $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, we know there exist $\mathcal{S}^1_c, \dots, \mathcal{S}^n_c$ and $\mathbb{S}^1_o, \dots, \mathbb{S}^n_o$ such that

$$(\mathcal{E}, (\mathcal{S}^1_c, \dots, \mathcal{S}^n_c), (\mathbb{S}^1_o, \dots, \mathbb{S}^n_o)) \in \mathcal{T}_{\mathbf{s}}(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \dots \parallel C_n, \mathcal{S}).$$

Let $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$. From $\Pi \sqsubseteq_\varphi (\Gamma, \bowtie, \blacktriangleleft, \rhd)$, we know

$$(\mathbb{O}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\varphi(\mathbb{S}_o^1), \ldots, \varphi(\mathbb{S}_o^n))) \in \mathcal{T}_s(\textbf{with } (\Gamma, \bowtie, \blacktriangleleft, \rhd) \textbf{ do } C_1 \,\|\, \ldots \,\|\, C_n, \varphi(\mathcal{S})).$$

Thus we know there exist $\mathbb{W}_0$ and $\mathbb{W}$ such that

$$(\textbf{with } (\Gamma, \bowtie, \blacktriangleleft, \rhd) \textbf{ do } C_1 \,\|\, \ldots \,\|\, C_n, \mathcal{S}_a) \overset{\mathsf{load}}{\Longleftrightarrow} \mathbb{W}_0,\ \mathbb{W}_0 \xRightarrow{(\mathbb{O},(\mathcal{S}_c^1,\ldots,\mathcal{S}_c^n),(\varphi(\mathbb{S}_o^1),\ldots,\varphi(\mathbb{S}_o^n)))}{}^* \mathbb{W}$$
$$\mathbb{W}_0 = (\sigma_0, \Sigma_0, \emptyset, \emptyset, \bowtie, \blacktriangleleft),\ \mathbb{W} = (\sigma, \Sigma, \mathbb{M}_s, \mathbb{V}, \bowtie, \blacktriangleleft),$$
$$\forall \mathsf{t}.\ \sigma_0(\mathsf{t}) = (C_\mathsf{t}, \emptyset), \forall \mathsf{t} \in [1..n].\ \Sigma_0(\mathsf{t}) = ((\Gamma, \rhd), \mathcal{S}_a, \epsilon, \emptyset, \emptyset), \forall \mathsf{t} \in [1..n].\ \Sigma(\mathsf{t}) = ((\Gamma, \rhd), \mathcal{S}_a, \xi_\mathsf{t}, ms_\mathsf{t}, \mathbb{er}_\mathsf{t})$$

For any t, let

$$ar_\mathsf{t} = \{(e_1, e_2) \mid \{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}, \mathsf{t}) \wedge \mathsf{abs}(e_1) <_{\xi_\mathsf{t}} \mathsf{abs}(e_2)\}$$

where $\mathsf{abs}(e) \overset{\text{def}}{=} (mid, (f, n))$ if $e = (mid, \mathsf{t}, (f, n, n', \delta))$.

- By the abstract operational semantics, we know $dom(\mathsf{visible}(\mathbb{O}, \mathsf{t})) = dom(\xi_\mathsf{t})$. Then, since $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$, we know $dom(\mathsf{visible}(\mathcal{E}, \mathsf{t})) = dom(\xi_\mathsf{t})$. Thus $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(ar_\mathsf{t})$ holds.
- For any e_1 and e_2, if $e_1 \overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_\mathcal{E} e_2$, since $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$, we know $\mathsf{obsv}(e_1) \overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_\mathbb{O} \mathsf{obsv}(e_2)$. Then, from the abstract operational semantics, we know $\mathsf{abs}(e_1) <_{\xi_\mathsf{t}} \mathsf{abs}(e_2)$. Thus $(e_1, e_2) \in ar_\mathsf{t}$. So, $\overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_\mathcal{E} \subseteq ar_\mathsf{t}$.
- Below we prove $\mathsf{PresvCancel}(ar_\mathsf{t}, \mathsf{t}, \mathcal{E}, (\Gamma, \rhd))$.
  For any e_1 and e_2, if $e_1 \overset{\mathsf{vis}}{\longmapsto}_\mathcal{E} e_2$, $e_1 \rhd_\Gamma e_2$ and $\{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}, \mathsf{t})$, since $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$, we know $\mathsf{obsv}(e_1) \overset{\mathsf{vis}}{\longmapsto}_\mathbb{O} \mathsf{obsv}(e_2)$. From $\mathsf{causalDelivery}(\mathcal{E})$, we know
  $$\mathsf{obsv}(e_1) <^\mathsf{t}_\mathbb{O} \mathsf{obsv}(e_2).$$
  Since $e_1 \rhd_\Gamma e_2$ and $\mathsf{obsv}(e_1) \overset{\mathsf{vis}}{\longmapsto}_\mathbb{O} \mathsf{obsv}(e_2)$, we know $\mathsf{msgid}(e_1) \in \mathbb{M}_s(\mathsf{msgid}(e_2)).ms$. By the abstract operational semantics we know
  $$\mathsf{abs}(e_1) <_{\xi_\mathsf{t}} \mathsf{abs}(e_2).$$
  Thus $(e_1, e_2) \in ar_\mathsf{t}$. So, $\mathsf{PresvCancel}(ar_\mathsf{t}, \mathsf{t}, \mathcal{E}, (\Gamma, \rhd))$.
- Below we prove $\mathsf{ExecRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_\mathsf{t}))$.
  - For any $\mathcal{E}' \leqslant \mathcal{E}$, we prove $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_\mathsf{t})) = \mathsf{aexecST}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar_\mathsf{t})$.
    Suppose the length of $\mathcal{E}'$ is k, and the $k+1$-th state in the sequence $\mathbb{S}_o^\mathsf{t}$ is $\mathcal{S}_o^\mathsf{t}$. Since $(\mathcal{E}, (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n)) \in \mathcal{T}_s(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n, \mathcal{S})$, we know
    $$\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_\mathsf{t}) = \mathcal{S}_o^\mathsf{t}.$$
    Let $\mathbb{O}' = \mathsf{obsv}(\mathcal{E}')$. Since $\mathcal{E}' \leqslant \mathcal{E}$ and $\mathbb{O} = \mathsf{obsv}(\mathcal{E})$, we know $\mathbb{O}' \leqslant \mathbb{O}$. Thus
    $$(\mathsf{visible}(\mathbb{O}', \mathsf{t}) \downharpoonright ar_\mathsf{t}) = (\xi_\mathsf{t}|_{\mathsf{visible}(\mathbb{O}',\mathsf{t})})$$
    Since $\mathbb{W}_0 \xRightarrow{(\mathbb{O},(\mathcal{S}_c^1,\ldots,\mathcal{S}_c^n),(\varphi(\mathbb{S}_o^1),\ldots,\varphi(\mathbb{S}_o^n)))}{}^* \mathbb{W}$, we know
    $$\mathsf{aexec\_st}(\Gamma, \varphi(\mathcal{S}), (\xi_\mathsf{t}|_{\mathsf{visible}(\mathbb{O}',\mathsf{t})})) = \varphi(\mathcal{S}_o^\mathsf{t})$$
    Thus $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_\mathsf{t})) = \mathsf{aexecST}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar_\mathsf{t})$.
  - For any $\mathcal{E}' \leqslant \mathcal{E}$, for any e such that $\mathsf{last}(\mathcal{E}') = e$ and $\mathsf{is\_orig}_\mathsf{t}(e)$, we prove $\mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar_\mathsf{t})$.
    Let $\mathbb{O}' = \mathsf{obsv}(\mathcal{E}')$. Since $\mathbb{W}_0 \xRightarrow{(\mathbb{O},(\mathcal{S}_c^1,\ldots,\mathcal{S}_c^n),(\varphi(\mathbb{S}_o^1),\ldots,\varphi(\mathbb{S}_o^n)))}{}^* \mathbb{W}$, we know there exist $\mathbb{W}_1, \mathbb{W}_2, \mathbb{O}_1, \mathbb{l}, \mathbb{O}_2$ such that
    $$\mathbb{W}_0 \overset{\mathbb{O}_1}{\Longleftrightarrow}{}^* \mathbb{W}_1,\ \mathbb{W}_1 \overset{\mathbb{l}}{\Longleftrightarrow} \mathbb{W}_2,\ \mathbb{W}_2 \overset{\mathbb{O}_2}{\Longleftrightarrow}{}^* \mathbb{W},$$
    $$\mathbb{O} = \mathbb{O}_1{+\!+}[\mathbb{l}]{+\!+}\mathbb{O}_2,\ \mathbb{O}_1{+\!+}[\mathbb{l}] = \mathbb{O}',\ \mathbb{l} = \mathsf{obsv}(e).$$
    Suppose $\mathbb{W}_2 = (\sigma_2, \Sigma_2, \mathbb{M}_s'', \bowtie)$, and $\forall \mathsf{t}.\ \Sigma_2(\mathsf{t}) = (\Gamma, \varphi(\mathcal{S}), \xi_\mathsf{t}'')$. Thus
    $$\begin{aligned}&\mathsf{rval}(e) = \mathsf{rval}(\mathbb{l})\\ &= \mathsf{aexec\_rv}(\Gamma, \varphi(\mathcal{S}), \xi_\mathsf{t}'')\\ &= \mathsf{aexec\_rv}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathbb{O}_1{+\!+}[\mathbb{l}], \mathsf{t}) \downharpoonright ar_\mathsf{t})\\ &= \mathsf{aexec\_rv}(\Gamma, \varphi(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar_\mathsf{t})\end{aligned}$$

- Below we prove $\forall \mathrm{t}' \neq \mathrm{t}.\ \mathsf{RCoh}_{(\mathrm{t},\mathrm{t}')}((ar_{\mathrm{t}}, ar_{\mathrm{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$. That is, for any $\mathcal{E}'$, $\mathcal{E}''$, e_1 and e_2, if $\mathcal{E}' \leqslant \mathcal{E}$, $\mathcal{E}'' \leqslant \mathcal{E}$, $\{e_1, e_2\} \subseteq \textsf{nc-vis}(\mathcal{E}', \mathrm{t}, (\Gamma, \rhd))$, $\{e_1, e_2\} \subseteq \textsf{nc-vis}(\mathcal{E}'', \mathrm{t}', (\Gamma, \rhd))$ and $e_1 \bowtie_\Gamma e_2$, we want to prove $((e_1, e_2) \in ar_{\mathrm{t}} \cap ar_{\mathrm{t}'} \vee (e_2, e_1) \in ar_{\mathrm{t}} \cap ar_{\mathrm{t}'})$ and $(\mathsf{Concurrent}_{\mathcal{E}}(e_1, e_2) \wedge (e_1 \blacktriangleleft_\Gamma e_2) \implies (e_1, e_2) \in ar_{\mathrm{t}})$.
  Let $\mathbb{e}_1 = \mathsf{abs}(e_1)$ and $\mathbb{e}_2 = \mathsf{abs}(e_2)$. Since $\{e_1, e_2\} \subseteq \textsf{nc-vis}(\mathcal{E}', \mathrm{t}, (\Gamma, \rhd))$ and $\{e_1, e_2\} \subseteq \textsf{nc-vis}(\mathcal{E}'', \mathrm{t}', (\Gamma, \rhd))$, by the abstract operational semantics, we know there exist $\mathrm{t}_0 \in \{\mathrm{t}, \mathrm{t}'\}$ and $\mathbb{W}_1$, $\mathbb{W}_2$, $\mathbb{O}_1$, $\mathbb{l}$, $\mathbb{O}_2$ such that
  $$\mathbb{W}_0 \overset{\mathbb{O}_1}{\multimap\!\!\longrightarrow}{}^* \mathbb{W}_1,\ \mathbb{W}_1 \overset{\mathbb{l}}{\multimap\!\!\longrightarrow} \mathbb{W}_2,\ \mathbb{W}_2 \overset{\mathbb{O}_2}{\multimap\!\!\longrightarrow}{}^* \mathbb{W},$$
  $$\mathbb{O} = \mathbb{O}_1{+}{+}[\mathbb{l}]{+}{+}\mathbb{O}_2,\ \mathsf{tid}(\mathbb{l}) = \mathrm{t}_0,$$
  $$\mathbb{W}_2 = (\sigma_2, \Sigma_2, \mathbb{M}_s'', \mathbb{V}'', \bowtie, \blacktriangleleft),\ \forall \mathrm{t}.\ \Sigma_2(\mathrm{t}) = ((\Gamma, \rhd), \mathcal{S}_a, \xi_{\mathrm{t}}'', ms_{\mathrm{t}}'', \mathbb{er}_{\mathrm{t}}''),$$
  $$\{(\mathbb{e}_1, \mathbb{e}_2), (\mathbb{e}_2, \mathbb{e}_1)\} \cap \mathbb{er}_{\mathrm{t}}'' \neq \emptyset,\ \{(\mathbb{e}_1, \mathbb{e}_2), (\mathbb{e}_2, \mathbb{e}_1)\} \cap \mathbb{er}_{\mathrm{t}'}'' \neq \emptyset.$$
  So we know
  $$\textsf{AbsCoh-W}(\mathbb{er}_{\mathrm{t}}'', \mathbb{er}_{\mathrm{t}'}'', \mathbb{V}'', (\Gamma, \bowtie, \blacktriangleleft)).$$
  Since $e_1 \bowtie_\Gamma e_2$, we know
  $$((\mathbb{e}_1, \mathbb{e}_2) \in \mathbb{er}_{\mathrm{t}}'' \cap \mathbb{er}_{\mathrm{t}'}'' \vee (\mathbb{e}_2, \mathbb{e}_1) \in \mathbb{er}_{\mathrm{t}}'' \cap \mathbb{er}_{\mathrm{t}'}'') \text{ and}$$
  $$(\mathbb{e}_1.mid \notin \mathbb{V}''(\mathbb{e}_2.mid) \wedge \mathbb{e}_2.mid \notin \mathbb{V}''(\mathbb{e}_1.mid) \wedge (\mathbb{e}_1 \blacktriangleleft_\Gamma \mathbb{e}_2) \implies \mathbb{e}_1\ \mathbb{er}_{\mathrm{t}}''\ \mathbb{e}_2).$$
  Thus we know
  $$((e_1, e_2) \in ar_{\mathrm{t}} \cap ar_{\mathrm{t}'} \vee (e_2, e_1) \in ar_{\mathrm{t}} \cap ar_{\mathrm{t}'})$$
  If $\mathsf{Concurrent}_{\mathcal{E}}(e_1, e_2)$, from the abstract operational semantics, we know $\mathbb{e}_1.mid \notin \mathbb{V}''(\mathbb{e}_2.mid) \wedge \mathbb{e}_2.mid \notin \mathbb{V}''(\mathbb{e}_1.mid)$. Thus $(\mathbb{e}_1 \blacktriangleleft_\Gamma \mathbb{e}_2) \implies \mathbb{e}_1\ \mathbb{er}_{\mathrm{t}}''\ \mathbb{e}_2$. So we have $(e_1 \blacktriangleleft_\Gamma e_2) \implies (e_1, e_2) \in ar_{\mathrm{t}}$. So $\mathsf{RCoh}_{(\mathrm{t},\mathrm{t}')}((ar_{\mathrm{t}}, ar_{\mathrm{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$.

Thus we are done. □

C Proofs of the Convergence Lemmas

C.1 For ACC (Lemma 5)

Although we can prove Lemma 5 directly, here we take another proof path via the Abstraction Theorem. We first show that the abstract semantics in Fig. 17 inherently guarantees the convergence of the abstract object states (Lemma 17 below). Then we derive that the contextual refinement $\Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$ can ensure $\mathsf{Cv}_{\varphi}(\Pi)$, the convergence of the concrete object (Lemma 18 below). By the equivalence between $\mathsf{ACC}_{\varphi}(\Pi, (\Gamma, \bowtie))$ and $\Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$ (the Abstraction Theorem), we derive Lemma 5: $\mathsf{ACC}_{\varphi}(\Pi, (\Gamma, \bowtie))$ can ensure $\mathsf{Cv}_{\varphi}(\Pi)$ too.

Definition 16. $\mathsf{CvA}(\Gamma, \bowtie)$ iff for any $C_1, \ldots, C_n, \mathcal{S}, \mathbb{W}_0, \mathbb{W}, \mathbb{W}', \mathbb{O}, \mathbb{O}', \mathrm{t}, \mathrm{t}'$,

$$\begin{array}{l}(\textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \parallel \ldots \parallel C_n, \mathcal{S}) \overset{\mathsf{load}}{\looparrowright\!\!\longrightarrow} \mathbb{W}_0 \\ \wedge\ (\mathbb{W}_0 \overset{\mathbb{O}}{\looparrowright\!\!\longrightarrow}{}^{*} \mathbb{W}) \ \wedge\ (\mathbb{W} \overset{\mathbb{O}'}{\looparrowright\!\!\longrightarrow}{}^{*} \mathbb{W}') \ \wedge\ \mathsf{visible}(\mathbb{O}, \mathrm{t}) = \mathsf{visible}(\mathbb{O}{+\!+}\mathbb{O}', \mathrm{t}') \\ \Longrightarrow\ \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathbb{W}(\mathrm{t}).\xi) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathbb{W}'(\mathrm{t}').\xi)\end{array}$$

Here we use $\mathbb{W}(\mathrm{t}).\xi$ to represent ξ on the node t in $\mathbb{W}$, and $\mathsf{visible}(\mathbb{O}, \mathrm{t})$ is defined similarly as its concrete counterpart $\mathsf{visible}(\mathcal{E}, \mathrm{t})$ (see Fig. 16).

Lemma 17. If $\mathsf{nonComm}(\Gamma, \bowtie)$, then $\mathsf{CvA}(\Gamma, \bowtie)$.

Lemma 18 ($\sqsubseteq$ implies Cv)**.** If $\mathsf{nonComm}(\Gamma, \bowtie)$ and $\Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$, then $\mathsf{Cv}_{\varphi}(\Pi)$.

Proof of Lemma 17. We first unfold the definition of $\mathsf{CvA}(\Gamma, \bowtie)$: for any $C_1, \ldots, C_n, \mathcal{S}, \mathbb{W}_0, \mathbb{W}, \mathbb{W}', \mathbb{O}, \mathbb{O}'$, t and t′, suppose $(\textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \parallel \ldots \parallel C_n, \mathcal{S}) \overset{\mathsf{load}}{\looparrowright\!\!\longrightarrow} \mathbb{W}_0$, $\mathbb{W}_0 \overset{\mathbb{O}}{\looparrowright\!\!\longrightarrow}{}^{*} \mathbb{W}$, $\mathbb{W} \overset{\mathbb{O}'}{\looparrowright\!\!\longrightarrow}{}^{*} \mathbb{W}'$ and $\mathsf{visible}(\mathbb{O}, \mathrm{t}) = \mathsf{visible}(\mathbb{O}{+\!+}\mathbb{O}', \mathrm{t}')$, then we want to prove $\mathsf{aexecST}(\Gamma, \mathcal{S}, \mathbb{W}(\mathrm{t}).\xi) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathbb{W}'(\mathrm{t}').\xi)$.

Let $\xi'_{\mathrm{t}} = \mathbb{W}(\mathrm{t}).\xi$ and $\xi'_{\mathrm{t}'} = \mathbb{W}'(\mathrm{t}').\xi$. From $\mathsf{visible}(\mathbb{O}, \mathrm{t}) = \mathsf{visible}(\mathbb{O}{+\!+}\mathbb{O}', \mathrm{t}')$, by the abstract semantics, we know

$$\lfloor \xi'_{\mathrm{t}} \rfloor = \lfloor \xi'_{\mathrm{t}'} \rfloor.$$

Since $\mathbb{W}_0 \overset{\mathbb{O}{+\!+}\mathbb{O}'}{\looparrowright\!\!\longrightarrow}{}^{*} \mathbb{W}'$, from the abstract semantics, we know there exists ξ''_{t} such that $\xi'_{\mathrm{t}} \subseteq \xi''_{\mathrm{t}}$ and

$$\mathsf{AbsCoh}(\xi''_{\mathrm{t}}, \xi'_{\mathrm{t}'}, (\Gamma, \bowtie)).$$

Thus we know

$$\mathsf{AbsCoh}(\xi'_{\mathrm{t}}, \xi'_{\mathrm{t}'}, (\Gamma, \bowtie)).$$

Also, by the abstract semantics, we know $\forall (\mathit{mid}, (f, n)) \in \xi'_{\mathrm{t}}.\ (f, n) \in \mathit{dom}(\Gamma)$. Thus we know there exists $\mathcal{S}'$ such that $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi'_{\mathrm{t}}) = \mathcal{S}'$. Then we know $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi'_{\mathrm{t}'})$ by applying Lemma 19. □

Lemma 19. For any $\xi_1, \xi_2, \mathcal{S}$ and $\mathcal{S}'$, if $\mathsf{nonComm}(\Gamma, \bowtie)$, $\lfloor \xi_1 \rfloor = \lfloor \xi_2 \rfloor$, $\mathsf{AbsCoh}(\xi_1, \xi_2, (\Gamma, \bowtie))$ and $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_1) = \mathcal{S}'$, then $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_2)$.

Proof. Suppose the length of ξ_1 is n. By induction over n.

- $n = 0$. Thus $\xi_1 = \xi_2 = \epsilon$ and $\mathcal{S}' = \mathcal{S}$. Thus $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_2)$.
- $n = m + 1$. Suppose $\xi_1 = \mathbb{e}_1 :: \xi'_1$ and $\xi_2 = \mathbb{e}'_1 :: \xi'_2$.
  - $\mathbb{e}_1 = \mathbb{e}'_1$. Then we know
  $$\lfloor \xi'_1 \rfloor = \lfloor \xi'_2 \rfloor \text{ and } \mathsf{AbsCoh}(\xi'_1, \xi'_2, (\Gamma, \bowtie)).$$
  Let $\mathcal{S}'' = \mathsf{aexecST}(\Gamma, \mathcal{S}, [\mathbb{e}_1])$. Thus $\mathsf{aexecST}(\Gamma, \mathcal{S}'', \xi'_1) = \mathcal{S}'$. Then, by the induction hypothesis, we know
  $$\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}'', \xi'_2).$$
  Thus $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_2)$.
  - $\mathbb{e}_1 \neq \mathbb{e}'_1$. Suppose $\xi_1 = \mathbb{e}_1 :: \mathbb{e}_2 :: \ldots :: \mathbb{e}_n$ and $\xi_2 = \mathbb{e}'_1 :: \mathbb{e}'_2 :: \ldots :: \mathbb{e}'_n$.
  Since $\lfloor \xi_1 \rfloor = \lfloor \xi_2 \rfloor$, we know there exists $i > 1$ such that $e_1 = e'_i$.
  Let $\xi_3 = \mathbb{e}'_i :: \xi'_3$ and $\xi'_3 = \mathbb{e}'_1 :: \ldots :: \mathbb{e}'_{i-1} :: \mathbb{e}'_{i+1} :: \ldots :: \mathbb{e}'_n$.
  Below we first prove $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_3) = \mathcal{S}'$.
  Since $\lfloor \xi_1 \rfloor = \lfloor \xi_2 \rfloor$ and $\mathsf{AbsCoh}(\xi_1, \xi_2, (\Gamma, \bowtie))$, we know
  $$\lfloor \xi'_1 \rfloor = \lfloor \xi'_3 \rfloor \text{ and } \mathsf{AbsCoh}(\xi'_1, \xi'_3, (\Gamma, \bowtie)).$$
  Let $\mathcal{S}'' = \mathsf{aexecST}(\Gamma, \mathcal{S}, [\mathbb{e}_1])$. Thus $\mathsf{aexecST}(\Gamma, \mathcal{S}'', \xi'_1) = \mathcal{S}'$. Then, by the induction hypothesis, we know
  $$\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}'', \xi'_3).$$

Thus $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_3)$.
Next, we prove $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_2)$.
Since $\mathsf{AbsCoh}(\xi_1, \xi_2, (\Gamma, \bowtie))$, we know

$$\mathsf{AbsCoh}(\xi_3, \xi_2, (\Gamma, \bowtie)).$$

By Lemma 20, we know $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_2)$.

Thus we are done. □

Lemma 20. For any ξ_1, ξ_2, $\mathcal{S}$ and $\mathcal{S}'$, if $\mathsf{nonComm}(\Gamma, \bowtie)$, $\xi_1 = [\mathbb{e}_1]{+\!+}\xi_1'{+\!+}\xi_1''$, $\xi_2 = \xi_1'{+\!+}[\mathbb{e}_1]{+\!+}\xi_1''$, $\mathsf{AbsCoh}(\xi_1, \xi_2, (\Gamma, \bowtie))$ and $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_1) = \mathcal{S}'$, then $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_2)$.

Proof. Suppose the length of ξ_1' is n. By induction over n.

- $n = 0$. Trivial.
- $n = m + 1$. Suppose $\xi_1' = \xi_2'{+\!+}[\mathbb{e}_2]$. Thus $\xi_1 = [\mathbb{e}_1]{+\!+}\xi_2'{+\!+}[\mathbb{e}_2]{+\!+}\xi_1''$ and $\xi_2 = \xi_2'{+\!+}[\mathbb{e}_2]{+\!+}[\mathbb{e}_1]{+\!+}\xi_1''$.
  Let $\xi_3 = \xi_2'{+\!+}[\mathbb{e}_1]{+\!+}[\mathbb{e}_2]{+\!+}\xi_1''$.
  Below we first prove $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_3) = \mathcal{S}'$.
  Since $\mathsf{AbsCoh}(\xi_1, \xi_2, (\Gamma, \bowtie))$, we know
  $$\mathsf{AbsCoh}(\xi_1, \xi_3, (\Gamma, \bowtie)).$$
  Then, by the induction hypothesis, we know
  $$\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_3).$$
  Next, we prove $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_3) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_2)$.
  Let $\mathcal{S}_2 = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_2')$.
  So we only need to prove $\mathsf{aexecST}(\Gamma, \mathcal{S}_2, [\mathbb{e}_1]{+\!+}[\mathbb{e}_2]{+\!+}\xi_1'') = \mathsf{aexecST}(\Gamma, \mathcal{S}_2, [\mathbb{e}_2]{+\!+}[\mathbb{e}_1]{+\!+}\xi_1'')$.
  Since $\mathbb{e}_1 <_{\xi_1} \mathbb{e}_2$, $\mathbb{e}_2 <_{\xi_2} \mathbb{e}_1$ and $\lfloor \xi_1 \rfloor = \lfloor \xi_2 \rfloor$, by $\mathsf{AbsCoh}(\xi_1, \xi_2, (\Gamma, \bowtie))$, we know
  $$\neg(\Gamma \models \mathbb{e}_1 \bowtie \mathbb{e}_2)$$
  Since $\mathsf{nonComm}(\Gamma, \bowtie)$, we know
  $$\mathsf{aexecST}(\Gamma, \mathcal{S}_2, [\mathbb{e}_2]{+\!+}[\mathbb{e}_1]) = \mathsf{aexecST}(\Gamma, \mathcal{S}_2, [\mathbb{e}_1]{+\!+}[\mathbb{e}_2]).$$
  Thus $\mathsf{aexecST}(\Gamma, \mathcal{S}_2, [\mathbb{e}_1]{+\!+}[\mathbb{e}_2]{+\!+}\xi_1'') = \mathsf{aexecST}(\Gamma, \mathcal{S}_2, [\mathbb{e}_2]{+\!+}[\mathbb{e}_1]{+\!+}\xi_1'')$.

Thus we are done. □

Proof of Lemma 18. We first unfold the definition of $\mathsf{Cv}_\varphi(\Pi)$: for any $\mathcal{S}, \mathcal{S}_a, \mathcal{E}, \mathcal{E}', \mathcal{E}''$, t and t′, suppose $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\varphi(\mathcal{S}) = \mathcal{S}_a$, $\mathcal{E}' \leqslant \mathcal{E}$, $\mathcal{E}'' \leqslant \mathcal{E}$ and $\mathsf{visible}(\mathcal{E}', \mathsf{t}) = \mathsf{visible}(\mathcal{E}'', \mathsf{t}')$, then we want to prove $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}''|_{\mathsf{t}'}))$.

Without loss of generality, we can suppose $\mathcal{E}' \leqslant \mathcal{E}''$ (so there exists $\mathcal{E}'''$ such that $\mathcal{E}'{+\!+}\mathcal{E}''' = \mathcal{E}''$). From $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$ and $\mathcal{E}'' \leqslant \mathcal{E}$, we know there exist $C_1, \ldots, C_n$ and $\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n, \mathbb{S}_o^1, \ldots, \mathbb{S}_o^n$ such that

$$(\mathcal{E}'', (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\mathbb{S}_o^1, \ldots, \mathbb{S}_o^n)) \in \mathcal{T}_{\mathsf{s}}(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n, \mathcal{S}).$$

Suppose the length of $\mathcal{E}'$ is k, and the $(k+1)$-th state and the last state in $\mathbb{S}_o^i$ is $\mathcal{S}_i'$ and $\mathcal{S}_i''$ respectively. So we know

$$\forall i.\ \mathcal{S}_i' = \mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_i),\ \forall i.\ \mathcal{S}_i'' = \mathsf{exec\_st}(\mathcal{S}, \mathcal{E}''|_i)$$

From $\Pi \sqsubseteq_\varphi (\Gamma, \bowtie)$, we know

$$(\mathsf{obsv}(\mathcal{E}''), (\mathcal{S}_c^1, \ldots, \mathcal{S}_c^n), (\varphi(\mathbb{S}_o^1), \ldots, \varphi(\mathbb{S}_o^n))) \in \mathcal{T}_{\mathsf{s}}(\textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \,\|\, \ldots \,\|\, C_n, \mathcal{S}_a)$$

Let $\mathbb{O} = \mathsf{obsv}(\mathcal{E}')$ and $\mathbb{O}' = \mathsf{obsv}(\mathcal{E}''')$. So there exist $\mathbb{W}_0, \mathbb{W}, \mathbb{W}', \sigma_c, \Sigma$ such that

$$\begin{array}{l} ((\textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \,\|\, \ldots \,\|\, C_n, \mathcal{S}_a) \overset{\mathsf{load}}{\multimap\!\!\longrightarrow} \mathbb{W}_0) \ \wedge\ (\mathbb{W}_0 \overset{\mathbb{O}}{\multimap\!\!\longrightarrow}{}^* \mathbb{W}) \ \wedge\ (\mathbb{W} \overset{\mathbb{O}'}{\multimap\!\!\longrightarrow}{}^* \mathbb{W}') \\ \wedge\ \forall i.\ \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathbb{W}(i).\xi) = \varphi(\mathcal{S}_i') \\ \wedge\ \forall i.\ \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathbb{W}'(i).\xi) = \varphi(\mathcal{S}_i'') \end{array}$$

From $\mathsf{visible}(\mathcal{E}', \mathsf{t}) = \mathsf{visible}(\mathcal{E}'', \mathsf{t}')$, we know

$$\mathsf{visible}(\mathbb{O}, \mathsf{t}) = \mathsf{visible}(\mathbb{O}{+\!+}\mathbb{O}', \mathsf{t}').$$

From Lemma 17, we know $\mathsf{CvA}(\Gamma, \bowtie)$. Thus we know

$$\mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathbb{W}(\mathsf{t}).\xi) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathbb{W}'(\mathsf{t}').\xi)$$

Thus $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}''|_{\mathsf{t}'}))$. So we are done. □

C.2 For XACC

Below we prove that the new abstract semantics in Fig. 20 also inherently guarantees the convergence of the abstract object states (Lemma 22 below). Then we derive that the contextual refinement $\Pi \sqsubseteq_\varphi (\Gamma, \bowtie, \blacktriangleleft, \rhd)$ can ensure $\mathsf{CCv}_\varphi(\Pi)$, the convergence of the concrete object under the assumption of causal delivery (Lemma 23 below). By the equivalence between $\mathsf{XACC}_\varphi(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$ and $\Pi \sqsubseteq_\varphi (\Gamma, \bowtie, \blacktriangleleft, \rhd)$ (the Abstraction Theorem), we derive Lemma 24: $\mathsf{XACC}_\varphi(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$ can ensure $\mathsf{CCv}_\varphi(\Pi)$ too.

Definition 21. $\mathsf{CvA}(\Gamma, \bowtie, \blacktriangleleft, \rhd)$ iff for any $C_1, \ldots, C_n$, $\mathcal{S}$, $\mathbb{W}_0$, $\mathbb{W}$, $\mathbb{W}'$, $\mathbb{O}$, $\mathbb{O}'$, t, t',

$$
\begin{array}{l}
(\textbf{with } (\Gamma, \bowtie, \blacktriangleleft, \rhd) \textbf{ do } C_1 \parallel \ldots \parallel C_n, \mathcal{S}) \overset{\mathsf{load}}{\Longleftrightarrow} \mathbb{W}_0 \\
\wedge\ (\mathbb{W}_0 \overset{\mathbb{O}}{\Longleftrightarrow}{}^* \mathbb{W}) \ \wedge\ (\mathbb{W} \overset{\mathbb{O}'}{\Longleftrightarrow}{}^* \mathbb{W}') \ \wedge\ \mathsf{visible}(\mathbb{O}, \mathsf{t}) = \mathsf{visible}(\mathbb{O}{+\!+}\mathbb{O}', \mathsf{t}') \\
\Longrightarrow\ \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathbb{W}(\mathsf{t}).\xi) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathbb{W}'(\mathsf{t}').\xi)
\end{array}
$$

Definition 21 is the same as Definition 16, except here we use the new abstract semantics.

Lemma 22. If $\mathsf{nonComm}(\Gamma, \bowtie)$ and $\mathsf{cancel}(\rhd)$, then $\mathsf{CvA}(\Gamma, \bowtie, \blacktriangleleft, \rhd)$.

Lemma 23 ($\sqsubseteq$ implies Cv)**.** If $\mathsf{nonComm}(\Gamma, \bowtie)$, $\mathsf{cancel}(\rhd)$ and $\Pi \sqsubseteq_\varphi (\Gamma, \bowtie, \blacktriangleleft, \rhd)$, then $\mathsf{CCv}_\varphi(\Pi)$.

Lemma 24 (XACC implies Cv)**.** If $\mathsf{nonComm}(\Gamma, \bowtie)$, $\mathsf{cancel}(\rhd)$ and $\mathsf{XACC}_\varphi(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$, then $\mathsf{CCv}_\varphi(\Pi)$.

Proof of Lemma 22. We first unfold the definition of $\mathsf{CvA}(\Gamma, \bowtie)$: for any $C_1, \ldots, C_n$, $\mathcal{S}$, $\mathbb{W}_0$, $\mathbb{W}$, $\mathbb{W}'$, $\mathbb{O}$, $\mathbb{O}'$, t and t', suppose $(\textbf{with } (\Gamma, \bowtie, \blacktriangleleft, \rhd) \textbf{ do } C_1 \parallel \ldots \parallel C_n, \mathcal{S}) \overset{\mathsf{load}}{\Longleftrightarrow} \mathbb{W}_0$, $\mathbb{W}_0 \overset{\mathbb{O}}{\Longleftrightarrow}{}^* \mathbb{W}$, $\mathbb{W} \overset{\mathbb{O}'}{\Longleftrightarrow}{}^* \mathbb{W}'$ and $\mathsf{visible}(\mathbb{O}, \mathsf{t}) = \mathsf{visible}(\mathbb{O}{+\!+}\mathbb{O}', \mathsf{t}')$, then we want to prove $\mathsf{aexecST}(\Gamma, \mathcal{S}, \mathbb{W}(\mathsf{t}).\xi) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathbb{W}'(\mathsf{t}').\xi)$.

Suppose

$$
\begin{array}{c}
\mathbb{W} = (\sigma_c, \Sigma, \mathbb{M}_s, \mathbb{V}, \bowtie, \blacktriangleleft) \text{ and } \mathbb{W}' = (\sigma_c', \Sigma', \mathbb{M}_s', \mathbb{V}', \bowtie, \blacktriangleleft), \text{ where} \\
\forall \mathsf{t} \in [1..n].\ \Sigma(\mathsf{t}) = ((\Gamma, \rhd), \mathcal{S}_a, \xi_\mathsf{t}, ms_\mathsf{t}, \mathbb{er}_\mathsf{t}) \\
\forall \mathsf{t} \in [1..n].\ \Sigma'(\mathsf{t}) = ((\Gamma, \rhd), \mathcal{S}_a, \xi_\mathsf{t}', ms_\mathsf{t}', \mathbb{er}_\mathsf{t}')
\end{array}
$$

From $\mathsf{visible}(\mathbb{O}, \mathsf{t}) = \mathsf{visible}(\mathbb{O}{+\!+}\mathbb{O}', \mathsf{t}')$, by the abstract semantics, we know

$$\lfloor \xi_\mathsf{t} \rfloor = \lfloor \xi_{\mathsf{t}'}' \rfloor.$$

Below we first prove $\mathsf{AbsCoh}(\xi_\mathsf{t} \backslash ms_\mathsf{t}, \xi_{\mathsf{t}'}' \backslash ms_{\mathsf{t}'}', (\Gamma, \bowtie))$.

- That is, we want to prove: if $\mathbb{e}_1 <_{\xi_\mathsf{t} \backslash ms_\mathsf{t}} \mathbb{e}_2$ and $\mathbb{e}_2 <_{\xi_{\mathsf{t}'}' \backslash ms_{\mathsf{t}'}'} \mathbb{e}_1$, then $\neg(\mathbb{e}_1 \bowtie_\Gamma \mathbb{e}_2)$.
  Since $\mathbb{e}_1 <_{\xi_\mathsf{t} \backslash ms_\mathsf{t}} \mathbb{e}_2$ and $\mathbb{e}_2 <_{\xi_{\mathsf{t}'}' \backslash ms_{\mathsf{t}'}'} \mathbb{e}_1$, by Lemma 25, we know
  $$(\mathbb{e}_1, \mathbb{e}_2) \in \mathbb{er}_\mathsf{t} \text{ and } (\mathbb{e}_2, \mathbb{e}_1) \in \mathbb{er}_{\mathsf{t}'}'.$$
  From the abstract semantics, we know there exist $\mathbb{er}_\mathsf{t}''$ and $\mathbb{V}''$ such that $\mathbb{er}_\mathsf{t} \subseteq \mathbb{er}_\mathsf{t}''$ and
  $$\mathsf{AbsCoh\text{-}W}(\mathbb{er}_\mathsf{t}'', \mathbb{er}_{\mathsf{t}'}', \mathbb{V}'', (\Gamma, \bowtie, \blacktriangleleft))$$
  Also from the abstract semantics, we know neither $\mathbb{er}_\mathsf{t}''$ or $\mathbb{er}_{\mathsf{t}'}'$ is symmetric. Thus we know
  $$
  \begin{array}{c}
  (\mathbb{e}_1, \mathbb{e}_2) \in \mathbb{er}_\mathsf{t}'',\ (\mathbb{e}_2, \mathbb{e}_1) \notin \mathbb{er}_\mathsf{t}'', \\
  (\mathbb{e}_2, \mathbb{e}_1) \in \mathbb{er}_{\mathsf{t}'}',\ (\mathbb{e}_1, \mathbb{e}_2) \notin \mathbb{er}_{\mathsf{t}'}',
  \end{array}
  $$
  So, by the definition of AbsCoh-W, we know
  $$\neg(\mathbb{e}_1 \bowtie_\Gamma \mathbb{e}_2).$$
  Thus we have proved $\mathsf{AbsCoh}(\xi_\mathsf{t} \backslash ms_\mathsf{t}, \xi_{\mathsf{t}'}' \backslash ms_{\mathsf{t}'}', (\Gamma, \bowtie))$.

From the abstract semantics, we know

$$
\begin{array}{c}
ms_\mathsf{t} = \mathsf{get\text{-}all\text{-}ms}_{(\Gamma, \rhd)}(\mathbb{O}, \mathsf{t}) \\
ms_{\mathsf{t}'}' = \mathsf{get\text{-}all\text{-}ms}_{(\Gamma, \rhd)}(\mathbb{O}{+\!+}\mathbb{O}', \mathsf{t}')
\end{array}
$$

where $\mathsf{get\text{-}all\text{-}ms}_{(\Gamma, \rhd)}(\mathbb{O}, \mathsf{t})$ is defined similarly as $\mathsf{get\text{-}all\text{-}ms}_{(\Gamma, \rhd)}(\mathcal{E}, \mathsf{t})$ in Figure 21:

$$
\begin{array}{l}
\mathsf{get\text{-}ms}_{(\Gamma, \rhd)}(\mathbb{O}, \mathbb{o}) \overset{\text{def}}{=} \{\mathbb{o}' \mid (\mathbb{o}' \overset{\mathsf{vis}}{\longmapsto}_\mathbb{O} \mathbb{o}) \wedge (\mathbb{o}' \rhd_\Gamma \mathbb{o})\} \\
\mathsf{get\text{-}all\text{-}ms}_{(\Gamma, \rhd)}(\mathbb{O}, \mathsf{t}) \overset{\text{def}}{=} \bigcup \{\mathsf{get\text{-}ms}_{(\Gamma, \rhd)}(\mathbb{O}, \mathbb{o}) \mid \mathbb{o} \in \mathsf{visible}(\mathbb{O}, \mathsf{t})\}
\end{array}
$$

Then, since $\mathsf{visible}(\mathbb{O}, \mathsf{t}) = \mathsf{visible}(\mathbb{O}{+\!+}\mathbb{O}', \mathsf{t}')$, we know

$$ms_{\mathrm{t}} = ms'_{\mathrm{t}'}.$$

Since $\lfloor \xi_{\mathrm{t}} \rfloor = \lfloor \xi'_{\mathrm{t}'} \rfloor$, we know

$$\lfloor \xi_{\mathrm{t}} \backslash ms_{\mathrm{t}} \rfloor = \lfloor \xi'_{\mathrm{t}'} \backslash ms'_{\mathrm{t}'} \rfloor.$$

Then, by applying Lemma 19, we know

$$\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_{\mathrm{t}} \backslash ms_{\mathrm{t}}) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi'_{\mathrm{t}'} \backslash ms'_{\mathrm{t}'}).$$

By Lemma 26, we know

$$\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_{\mathrm{t}}) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_{\mathrm{t}} \backslash ms_{\mathrm{t}})$$
$$\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi'_{\mathrm{t}'}) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi'_{\mathrm{t}'} \backslash ms'_{\mathrm{t}'})$$

Thus $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_{\mathrm{t}}) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi'_{\mathrm{t}'})$. So we are done. □

Lemma 25. If $(\mathbb{P}, \mathcal{S}) \xrightarrow{\mathsf{load}} \mathbb{W}_0$, $\mathbb{W}_0 \xrightarrow{\mathbb{O}}{}^{m} \mathbb{W}$, $\mathbb{W} = (\sigma_c, \Sigma, \mathbb{M}_s, \mathbb{V}, \bowtie, \blacktriangleleft)$, $\Sigma(\mathrm{t}) = ((\Gamma, \rhd), \mathcal{S}_a, \xi_{\mathrm{t}}, ms_{\mathrm{t}}, \mathbb{er}_{\mathrm{t}})$, then $\prec_{\xi_{\mathrm{t}} \backslash ms_{\mathrm{t}}} \subseteq \mathbb{er}_{\mathrm{t}}$.

Proof. By induction over m. □

Lemma 26. If $(\textbf{with } (\Gamma, \bowtie, \blacktriangleleft, \rhd) \textbf{ do } C_1 \,\|\, \ldots \,\|\, C_n, \mathcal{S}) \xrightarrow{\mathsf{load}} \mathbb{W}_0$, $\mathbb{W}_0 \xrightarrow{\mathbb{O}}{}^{*} \mathbb{W}$, $\mathbb{W} \xrightarrow{\mathbb{O}_1}{}^{*} \mathbb{W}_1$, $\mathsf{cancel}(\rhd)$, $ms = \mathsf{get\text{-}all\text{-}ms}_{(\Gamma,\rhd)}(\mathbb{O}, \mathrm{t})$, $\xi = \mathbb{W}_1(\mathrm{t}).\xi$, then $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi \backslash ms)$.

Proof. By induction over the length m of $(\mathbb{O}|_{\mathrm{t}})$.

- $m = 0$. So $ms = \emptyset$. Thus we are done.
- $m = k + 1$. Let $\mathbb{o}' = \mathsf{last}(\mathbb{O}|_{\mathrm{t}})$. So there exist $\mathbb{o}$ and $\mathbb{O}'$ such that $\mathbb{o} \overset{\mathrm{t}}{\underset{\mathbb{O}}{\Rightarrow}} \mathbb{o}'$ and $\mathbb{O}' \mathbin{++} [\mathbb{o}'] \leqslant \mathbb{O}$.
  Let $ms' = \mathsf{get\text{-}all\text{-}ms}_{(\Gamma,\rhd)}(\mathbb{O}', \mathrm{t})$. Then, by the induction hypothesis, we know
  $$\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi \backslash ms').$$
  So we only need to prove $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi \backslash ms') = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi \backslash ms)$.
  Since $\mathsf{visible}(\mathbb{O}, \mathrm{t}) = \mathsf{visible}(\mathbb{O}', \mathrm{t}) \cup \{\mathbb{o}\}$, we know
  $$\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi \backslash ms) = \mathsf{aexecST}(\Gamma, \mathcal{S}, (\xi \backslash ms') \backslash \mathsf{get\text{-}ms}_{(\Gamma,\rhd)}(\mathbb{O}, \mathbb{o}))$$
  By the abstract semantics, we know
  $$\forall \mathbb{o}'' \in \mathsf{get\text{-}ms}_{(\Gamma,\rhd)}(\mathbb{O}, \mathbb{o}).\ \mathbb{o}'' \prec_{\xi} \mathbb{o}$$
  From $\mathsf{cancel}(\rhd)$, we know
  $$\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi \backslash ms') = \mathsf{aexecST}(\Gamma, \mathcal{S}, (\xi \backslash ms') \backslash \mathsf{get\text{-}ms}_{(\Gamma,\rhd)}(\mathbb{O}, \mathbb{o}))$$
  So $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi \backslash ms') = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi \backslash ms)$. Thus $\mathsf{aexecST}(\Gamma, \mathcal{S}, \xi) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi \backslash ms)$.

Thus we are done. □

Proof of Lemma 23. Similar to the proof of Lemma 18. □

D Compositionality of ACC/XACC

In this section, we consider the problem of compositionality of ACC (XACC), that is, whether the composition of multiple ACC (XACC) objects are also ACC (XACC). We prove Lemma 29 (Compositionality of ACC) and Lemma 30 (Compositionality of XACC) below. We also prove Lemma 31, saying that nonComm (see Def. 1) is compositional.

Suppose the concrete program P is **let** Π **in** $C_1 \,\|\, \ldots \,\|\, C_n$, and the whole object Π can be split into multiple small objects Π_1, $\ldots, \Pi_n$, where $\forall i \neq j.\ dom(\Pi_i) \cap dom(\Pi_j) = \emptyset$. Also, suppose the object state $\mathcal{S}$ can be split into disjoint $\mathcal{S}_1, \ldots, \mathcal{S}_n$. That is, $\Pi = \Pi_1 \uplus \ldots \uplus \Pi_m$ and $\mathcal{S} = \mathcal{S}_1 \uplus \ldots \uplus \mathcal{S}_m$. We can write $\Pi_i \subseteq \Pi$ and $\mathcal{S}_i \subseteq \mathcal{S}$.

For each single object Π_i, we define $\mathcal{E}|_{\Pi_i}$ to project the trace $\mathcal{E}$ to the events of operations in Π_i. That is,

$$\mathcal{E}|_{\Pi} \stackrel{\text{def}}{=} \begin{cases} \epsilon & \text{if } \mathcal{E} = \epsilon \\ e :: (\mathcal{E}'|_{\Pi}) & \text{if } \mathcal{E} = e :: \mathcal{E}' \wedge \mathsf{op}(e) \in dom(\Pi) \\ (\mathcal{E}'|_{\Pi}) & \text{if } \mathcal{E} = e :: \mathcal{E}' \wedge \mathsf{op}(e) \notin dom(\Pi) \end{cases}$$

We give each object Π_i a specification $(\Gamma_i, \bowtie_i)$. We assume each $\bowtie_i$ is a relation over actions of Γ_i, i.e.,

$$\bowtie_i \subseteq \mathsf{act}(\Gamma_i) \times \mathsf{act}(\Gamma_i), \quad \text{where } \mathsf{act}(\Gamma) = \{\alpha \mid \exists f, n.\ \mathsf{split}(\Gamma(f,n)) = (\_, \alpha)\}.$$

Also we assume $\forall i.\ (dom(\Gamma_i) = dom(\Pi_i)) \wedge \forall j \neq i.\ (\mathsf{act}(\Gamma_i) \cap \mathsf{act}(\Gamma_j) = \emptyset)$. Their abstract object states are disjoint too. Also, it is natural to assume that the operations in Γ_i do *not* conflict with the operations in Γ_j, since the object states of Γ_i and Γ_j are disjoint. So we define the composition of $\bowtie_i$ and $\bowtie_j$ simply as their disjoint union.

We also need each Π_i and Γ_i to have strong locality. We have defined $\mathsf{SLocality}(\Gamma)$ in Def. 40. We define $\mathsf{SLocality}(\Pi)$ as follows. Here $fv(\Pi)$ returns the free variables in Π and its effectors.

Definition 27. $\mathsf{SLocality}(\Pi)$ iff all the following holds:

1. for any f, n, n', δ, $\mathcal{S}$ and $\mathcal{S}_1$, if $\Pi(f,n)(\mathcal{S}) = (n', \delta)$ and $dom(\mathcal{S}) \cap dom(\mathcal{S}_1) = \emptyset$, then $\Pi(f,n)(\mathcal{S} \uplus \mathcal{S}_1) = (n', \delta)$.
2. for any f, n, δ, $\mathcal{S}$, $\mathcal{S}'$ and $\mathcal{S}_1$, if $\mathsf{valid}_{\Pi}(f,n,\delta)$, $\delta(\mathcal{S}) = \mathcal{S}'$ and $dom(\mathcal{S}) \cap dom(\mathcal{S}_1) = \emptyset$, then $\delta(\mathcal{S} \uplus \mathcal{S}_1) = \mathcal{S}' \uplus \mathcal{S}_1$.
3. for any $f, n, n', \delta, \mathcal{S}$ and $\mathcal{S}_1$, if $\Pi(f,n)(\mathcal{S} \uplus \mathcal{S}_1) = (n', \delta)$ and $fv(\Pi) \subseteq dom(\mathcal{S})$, then there exists $\mathcal{S}'$ such that $\mathcal{S}'' = \mathcal{S}' \uplus \mathcal{S}_1$ and $\Pi(f,n)(\mathcal{S}) = (n', \delta)$.
4. for any f, n, δ, $\mathcal{S}$, $\mathcal{S}''$ and $\mathcal{S}_1$, if $\mathsf{valid}_{\Pi}(f,n,\delta)$, $\delta(\mathcal{S} \uplus \mathcal{S}_1) = \mathcal{S}''$ and $fv(\Pi) \subseteq dom(\mathcal{S})$, then there exists $\mathcal{S}'$ such that $\mathcal{S}'' = \mathcal{S}' \uplus \mathcal{S}_1$ and $\delta(\mathcal{S}) = \mathcal{S}'$.

Definition 28. $\mathsf{well\text{-}disjoint}((\varphi_1, \Pi_1, \Gamma_1, \bowtie_1), (\varphi_2, \Pi_2, \Gamma_2, \bowtie_2))$ iff $\mathsf{SLocality}(\Pi_1)$, $\mathsf{SLocality}(\Pi_2)$, $\mathsf{SLocality}(\Gamma_1)$, $\mathsf{SLocality}(\Gamma_2)$, $dom(\varphi_1) \cap dom(\varphi_2) = \emptyset$, $range(\varphi_1) \cap range(\varphi_2) = \emptyset$, $dom(\Pi_1) \cap dom(\Pi_2) = \emptyset$, $dom(\Gamma_1) = dom(\Pi_1)$, $dom(\Gamma_2) = dom(\Pi_2)$, $\bowtie_1 \subseteq \mathsf{act}(\Gamma_1) \times \mathsf{act}(\Gamma_1)$, $\bowtie_2 \subseteq \mathsf{act}(\Gamma_2) \times \mathsf{act}(\Gamma_2)$, $\mathsf{act}(\Gamma_1) \cap \mathsf{act}(\Gamma_2) = \emptyset$.

Lemma 29. If $\mathsf{ACC}_{\varphi_1}(\Pi_1, (\Gamma_1, \bowtie_1))$, $\mathsf{ACC}_{\varphi_2}(\Pi_2, (\Gamma_2, \bowtie_2))$ and $\mathsf{well\text{-}disjoint}((\varphi_1, \Pi_1, \Gamma_1, \bowtie_1), (\varphi_2, \Pi_2, \Gamma_2, \bowtie_2))$, then $\mathsf{ACC}_{\varphi_1 \uplus \varphi_2}(\Pi_1 \uplus \Pi_2, (\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2))$.

Proof of Lemma 29. By unfolding the definition of ACC, we want to prove: $\forall \mathcal{S}, \mathcal{E}.\ \mathcal{E} \in \mathcal{T}(\Pi_1 \uplus \Pi_2, \mathcal{S}) \wedge \mathcal{S} \in dom(\varphi_1 \uplus \varphi_2) \implies \mathsf{ACT}_{\varphi_1 \uplus \varphi_2}(\mathcal{E}, \mathcal{S}, (\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2))$.

Since $\mathcal{S} \in dom(\varphi_1 \uplus \varphi_2)$, we know there exist $\mathcal{S}_1$ and $\mathcal{S}_2$ such that

$$\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2,\ \mathcal{S}_1 \in dom(\varphi_1) \text{ and } \mathcal{S}_2 \in dom(\varphi_2).$$

Let $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$ and $\mathcal{E}_2 = \mathcal{E}|_{\Pi_2}$. Since $\mathcal{E} \in \mathcal{T}(\Pi_1 \uplus \Pi_2, \mathcal{S})$, we know

$$\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$$

By Lemma 32, we know

$$\mathcal{E}_1 \in \mathcal{T}(\Pi_1, \mathcal{S}_1) \text{ and } \mathcal{E}_2 \in \mathcal{T}(\Pi_2, \mathcal{S}_2).$$

Then, from $\mathsf{ACC}_{\varphi_1}(\Pi_1, (\Gamma_1, \bowtie_1))$ and $\mathsf{ACC}_{\varphi_2}(\Pi_2, (\Gamma_2, \bowtie_2))$, we know

$$\mathsf{ACT}_{\varphi_1}(\mathcal{E}_1, \mathcal{S}_1, (\Gamma_1, \bowtie_1)) \text{ and } \mathsf{ACT}_{\varphi_2}(\mathcal{E}_2, \mathcal{S}_2, (\Gamma_2, \bowtie_2)).$$

By Lemma 33, we know

$$\mathsf{ACT}_{\varphi_1 \uplus \varphi_2}(\mathcal{E}, \mathcal{S}, (\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2)).$$

Thus we are done. □

Lemma 30 (Compositionality of XACC)**.** If

- $\mathsf{XACC}_{\varphi_1}(\Pi_1, (\Gamma_1, \bowtie_1, \blacktriangleleft_1, \rhd_1))$ and $\mathsf{XACC}_{\varphi_2}(\Pi_2, (\Gamma_2, \bowtie_2, \blacktriangleleft_2, \rhd_2))$,

- $\mathsf{well\text{-}disjoint}((\varphi_1, \Pi_1, \Gamma_1, \bowtie_1), (\varphi_2, \Pi_2, \Gamma_2, \bowtie_2))$

then $\mathsf{XACC}_{\varphi_1 \uplus \varphi_2}(\Pi_1 \uplus \Pi_2, (\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2, \blacktriangleleft_1 \uplus \blacktriangleleft_2, \rhd_1 \uplus \rhd_2))$.

Proof. By unfolding the definition of XACC, we want to prove:
$\forall \mathcal{S}, \mathcal{E}.\ \mathcal{E} \in \mathcal{T}(\Pi_1 \uplus \Pi_2, \mathcal{S}) \wedge \mathcal{S} \in \mathit{dom}(\varphi_1 \uplus \varphi_2) \wedge \mathsf{causalDelivery}(\mathcal{E}) \implies \mathsf{XACT}_{\varphi_1 \uplus \varphi_2}(\mathcal{E}, \mathcal{S}, (\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2, \blacktriangleleft_1 \uplus \blacktriangleleft_2, \rhd_1 \uplus \rhd_2))$.

Since $\mathcal{S} \in \mathit{dom}(\varphi_1 \uplus \varphi_2)$, we know there exist $\mathcal{S}_1$ and $\mathcal{S}_2$ such that

$$\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2, \mathcal{S}_1 \in \mathit{dom}(\varphi_1) \text{ and } \mathcal{S}_2 \in \mathit{dom}(\varphi_2).$$

Let $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$ and $\mathcal{E}_2 = \mathcal{E}|_{\Pi_2}$. Since $\mathcal{E} \in \mathcal{T}(\Pi_1 \uplus \Pi_2, \mathcal{S})$, we know

$$\mathsf{op}(\mathcal{E}) \subseteq \mathit{dom}(\Pi_1) \uplus \mathit{dom}(\Pi_2)$$

By Lemma 32, we know

$$\mathcal{E}_1 \in \mathcal{T}(\Pi_1, \mathcal{S}_1) \text{ and } \mathcal{E}_2 \in \mathcal{T}(\Pi_2, \mathcal{S}_2).$$

Since $\mathsf{causalDelivery}(\mathcal{E})$, we know

$$\mathsf{causalDelivery}(\mathcal{E}_1) \text{ and } \mathsf{causalDelivery}(\mathcal{E}_2).$$

Then, from $\mathsf{XACC}_{\varphi_1}(\Pi_1, (\Gamma_1, \bowtie_1, \blacktriangleleft_1, \rhd_1))$ and $\mathsf{XACC}_{\varphi_2}(\Pi_2, (\Gamma_2, \bowtie_2, \blacktriangleleft_2, \rhd_2))$, we know

$$\mathsf{XACT}_{\varphi_1}(\mathcal{E}_1, \mathcal{S}_1, (\Gamma_1, \bowtie_1, \blacktriangleleft_1, \rhd_1)) \text{ and } \mathsf{XACT}_{\varphi_2}(\mathcal{E}_2, \mathcal{S}_2, (\Gamma_2, \bowtie_2, \blacktriangleleft_2, \rhd_2)).$$

By Lemma 34, we know

$$\mathsf{XACT}_{\varphi_1 \uplus \varphi_2}(\mathcal{E}, \mathcal{S}, (\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2, \blacktriangleleft_1 \uplus \blacktriangleleft_2, \rhd_1 \uplus \rhd_2)).$$

Thus we are done. □

Lemma 31 (Compositionality of nonComm)**.** If

- $\mathsf{nonComm}(\Gamma_1, \bowtie_1)$ and $\mathsf{nonComm}(\Gamma_2, \bowtie_2)$,
- $\mathsf{SLocality}(\Gamma_1)$, $\mathsf{SLocality}(\Gamma_2)$, $\mathit{dom}(\Gamma_1) \cap \mathit{dom}(\Gamma_2) = \emptyset$, $\bowtie_1 \subseteq \mathsf{act}(\Gamma_1) \times \mathsf{act}(\Gamma_1)$, $\bowtie_2 \subseteq \mathsf{act}(\Gamma_2) \times \mathsf{act}(\Gamma_2)$, $\mathsf{act}(\Gamma_1) \cap \mathsf{act}(\Gamma_2) = \emptyset$,

then $\mathsf{nonComm}(\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2)$.

Proof. By unfolding the definition of nonComm, we want to prove: for any f_1, n_1, f_2, n_2, α_1 and α_2, if $\mathsf{split}((\Gamma_1 \uplus \Gamma_2)(f_1, n_1)) = (\_, \alpha_1)$, $\mathsf{split}((\Gamma_1 \uplus \Gamma_2)(f_2, n_2)) = (\_, \alpha_2)$ and $\neg(\alpha_1 (\bowtie_1 \uplus \bowtie_2) \alpha_2)$, then $\alpha_1 \mathbin{⨾} \alpha_2 = \alpha_2 \mathbin{⨾} \alpha_1$.

Since $\mathsf{split}((\Gamma_1 \uplus \Gamma_2)(f_1, n_1)) = (\_, \alpha_1)$ and $\mathsf{split}((\Gamma_1 \uplus \Gamma_2)(f_2, n_2)) = (\_, \alpha_2)$, we have four cases:

1. $\mathsf{split}(\Gamma_1(f_1, n_1)) = (\_, \alpha_1)$ and $\mathsf{split}(\Gamma_1(f_2, n_2)) = (\_, \alpha_2)$. Since $\neg(\alpha_1 (\bowtie_1 \uplus \bowtie_2) \alpha_2)$, we know
$$\neg(\alpha_1 \bowtie_1 \alpha_2).$$
From $\mathsf{nonComm}(\Gamma_1, \bowtie_1)$, we know $\alpha_1 \mathbin{⨾} \alpha_2 = \alpha_2 \mathbin{⨾} \alpha_1$.
2. $\mathsf{split}(\Gamma_2(f_1, n_1)) = (\_, \alpha_1)$ and $\mathsf{split}(\Gamma_2(f_2, n_2)) = (\_, \alpha_2)$. Similar to case (1).
3. $\mathsf{split}(\Gamma_1(f_1, n_1)) = (\_, \alpha_1)$ and $\mathsf{split}(\Gamma_2(f_2, n_2)) = (\_, \alpha_2)$. So, for any $\mathcal{S}$ such that $\mathit{fv}(\Gamma_1) \uplus \mathit{fv}(\Gamma_2) \subseteq \mathit{dom}(\mathcal{S})$, we know there exists $\mathcal{S}_1$ and $\mathcal{S}_2$ such that
$$\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2, \mathit{fv}(\Gamma_1) \subseteq \mathit{dom}(\mathcal{S}_1) \text{ and } \mathit{fv}(\Gamma_2) \subseteq \mathit{dom}(\mathcal{S}_2).$$
Since $\mathsf{SLocality}(\Gamma_1)$ and $\mathsf{SLocality}(\Gamma_2)$, we know
$$\begin{array}{ll} \alpha_1(\mathcal{S}_1 \uplus \mathcal{S}_2) = \alpha_1(\mathcal{S}_1) \uplus \mathcal{S}_2 & \alpha_1(\mathcal{S}_1 \uplus \alpha_2(\mathcal{S}_2)) = \alpha_1(\mathcal{S}_1) \uplus \alpha_2(\mathcal{S}_2) \\ \alpha_2(\mathcal{S}_2 \uplus \mathcal{S}_1) = \alpha_2(\mathcal{S}_2) \uplus \mathcal{S}_1 & \alpha_2(\mathcal{S}_2 \uplus \alpha_1(\mathcal{S}_1)) = \alpha_2(\mathcal{S}_2) \uplus \alpha_1(\mathcal{S}_1) \end{array}$$
So,
$$\begin{array}{ll} (\alpha_1 \mathbin{⨾} \alpha_2)(\mathcal{S}) & = \alpha_2(\alpha_1(\mathcal{S}_1 \uplus \mathcal{S}_2)) \\ & = \alpha_2(\alpha_1(\mathcal{S}_1) \uplus \mathcal{S}_2) \\ & = \alpha_2(\mathcal{S}_2) \uplus \alpha_1(\mathcal{S}_1) \\ & = \alpha_1(\alpha_2(\mathcal{S}_2 \uplus \mathcal{S}_1)\ = (\alpha_2 \mathbin{⨾} \alpha_1)(\mathcal{S}) \end{array}$$
4. $\mathsf{split}(\Gamma_2(f_1, n_1)) = (\_, \alpha_1)$ and $\mathsf{split}(\Gamma_1(f_2, n_2)) = (\_, \alpha_2)$. Similar to case (3).

Thus we are done. □

Lemma 32. If

- $\mathcal{E} \in \mathcal{T}(\Pi_1 \uplus \Pi_2, \mathcal{S}_1 \uplus \mathcal{S}_2)$, $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$, $fv(\Pi_1) \subseteq dom(\mathcal{S}_1)$,
- $\mathsf{SLocality}(\Pi_1)$,

then $\mathcal{E}_1 \in \mathcal{T}(\Pi_1, \mathcal{S}_1)$.

Proof. From the definition of $\mathcal{T}(\Pi_1, \mathcal{S}_1)$, we want to prove: there exist $C_1, \ldots, C_n$ such that $\mathcal{E}_1 \in \mathcal{T}(\mathbf{let}\ \Pi_1\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, \mathcal{S}_1)$.

Since $\mathcal{E} \in \mathcal{T}(\Pi_1 \uplus \Pi_2, \mathcal{S}_1 \uplus \mathcal{S}_2)$, we know there exist $C'_1, \ldots, C'_n$ such that

$$\mathcal{E} \in \mathcal{T}(\mathbf{let}\ \Pi_1 \uplus \Pi_2\ \mathbf{in}\ C'_1 \parallel \ldots \parallel C'_n, \mathcal{S}_1 \uplus \mathcal{S}_2).$$

For each i, first we construct $C''_i \overset{\text{def}}{=} \mathsf{code}_i(\mathcal{E}|_i)$. Here we define $\mathsf{code}_{\mathsf{t}}(\mathcal{E})$ as follows.

$$\mathsf{code}_{\mathsf{t}}(\mathcal{E}) \overset{\text{def}}{=} \begin{cases} \mathbf{skip} & \text{if } \mathcal{E} = \epsilon \\ (x := f(n)); \mathsf{code}_{\mathsf{t}}(\mathcal{E}') & \text{if } \mathcal{E} = e :: \mathcal{E}' \land e = (mid, \mathsf{t}, (f, n, n', \delta)) \\ \mathsf{code}_{\mathsf{t}}(\mathcal{E}') & \text{if } \mathcal{E} = e :: \mathcal{E}' \land \neg \mathsf{is\_orig}_{\mathsf{t}}(e) \end{cases}$$

Then we can prove

$$\mathcal{E} \in \mathcal{T}(\mathbf{let}\ \Pi_1 \uplus \Pi_2\ \mathbf{in}\ C''_1 \parallel \ldots \parallel C''_n, \mathcal{S}_1 \uplus \mathcal{S}_2).$$

Next, for each i, we construct $C_i \overset{\text{def}}{=} C''_i|_{\Pi_1}$. Here we define $C|_{\Pi}$ as follows.

$$C|_{\Pi} \overset{\text{def}}{=} \begin{cases} \mathbf{skip} & \text{if } C = \mathbf{skip} \\ (x := f(n)); (C'|_{\Pi}) & \text{if } C = ((x := f(n)); C') \land (f, n) \in dom(\Pi) \\ C'|_{\Pi} & \text{if } C = ((x := f(n)); C') \land (f, n) \notin dom(\Pi) \end{cases}$$

Since $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$, $fv(\Pi_1) \subseteq dom(\mathcal{S}_1)$ and $\mathsf{SLocality}(\Pi_1)$, we can prove

$$\mathcal{E}_1 \in \mathcal{T}(\mathbf{let}\ \Pi_1\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, \mathcal{S}_1).$$

Thus we are done. □

Lemma 33. If

- $\mathsf{ACT}_{\varphi_1}(\mathcal{E}_1, \mathcal{S}_1, (\Gamma_1, \bowtie_1))$ and $\mathsf{ACT}_{\varphi_2}(\mathcal{E}_2, \mathcal{S}_2, (\Gamma_2, \bowtie_2))$,
- $\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2$, $\mathcal{S}_1 \in dom(\varphi_1)$, $\mathcal{S}_2 \in dom(\varphi_2)$,
- $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$, $\mathcal{E}_2 = \mathcal{E}|_{\Pi_2}$, $\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$,
- $dom(\varphi_1) \cap dom(\varphi_2) = \emptyset$, $range(\varphi_1) \cap range(\varphi_2) = \emptyset$, $dom(\Gamma_1) = dom(\Pi_1)$, $dom(\Gamma_2) = dom(\Pi_2)$,
- $\bowtie_1 \subseteq \mathsf{act}(\Gamma_1) \times \mathsf{act}(\Gamma_1)$, $\bowtie_2 \subseteq \mathsf{act}(\Gamma_2) \times \mathsf{act}(\Gamma_2)$, $\mathsf{act}(\Gamma_1) \cap \mathsf{act}(\Gamma_2) = \emptyset$,
- $\mathsf{SLocality}(\Pi_1)$, $\mathsf{SLocality}(\Pi_2)$, $\mathsf{SLocality}(\Gamma_1)$, $\mathsf{SLocality}(\Gamma_2)$,

then $\mathsf{ACT}_{\varphi_1 \uplus \varphi_2}(\mathcal{E}, \mathcal{S}, (\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2))$.

Proof. From $\mathsf{ACT}_{\varphi_1}(\mathcal{E}_1, \mathcal{S}_1, (\Gamma_1, \bowtie_1))$, we know there exists $ar_1, \ldots, ar_n$ such that

$$\begin{aligned} &\forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_1, \mathsf{t})}(ar_{\mathsf{t}}) \land (\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}_1} \subseteq ar_{\mathsf{t}}) \land \mathsf{ExecRelated}_{\varphi_1}(\mathsf{t}, (\mathcal{E}_1, \mathcal{S}_1), (\Gamma_1, ar_{\mathsf{t}})) \\ &\land \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{Coh}(ar_{\mathsf{t}}, ar_{\mathsf{t}'}, (\Gamma_1, \bowtie_1)) \end{aligned}$$

From $\mathsf{ACT}_{\varphi_2}(\mathcal{E}_2, \mathcal{S}_2, (\Gamma_2, \bowtie_2))$, we know there exists $ar'_1, \ldots, ar'_n$ such that

$$\begin{aligned} &\forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_2, \mathsf{t})}(ar'_{\mathsf{t}}) \land (\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}_2} \subseteq ar'_{\mathsf{t}}) \land \mathsf{ExecRelated}_{\varphi_2}(\mathsf{t}, (\mathcal{E}_2, \mathcal{S}_2), (\Gamma_2, ar'_{\mathsf{t}})) \\ &\land \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{Coh}(ar'_{\mathsf{t}}, ar'_{\mathsf{t}'}, (\Gamma_2, \bowtie_2)) \end{aligned}$$

From Lemma 35, we know

$$\forall \mathsf{t}.\ \mathsf{partialOrder}((ar_{\mathsf{t}} \cup ar'_{\mathsf{t}} \cup \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}})^+).$$

Thus we know there exist $ar''_1, \ldots, ar''_n$ such that

$$\forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}, \mathsf{t})}(ar''_{\mathsf{t}}) \land \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \subseteq ar''_{\mathsf{t}} \land ar_{\mathsf{t}} \subseteq ar''_{\mathsf{t}} \land ar'_{\mathsf{t}} \subseteq ar''_{\mathsf{t}}$$

From Lemma 36, we know

$$\mathsf{ExecRelated}_{\varphi_1 \uplus \varphi_2}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma_1 \uplus \Gamma_2, ar''_{\mathsf{t}})).$$

Take t and t' such that $\mathsf{t} \neq \mathsf{t}'$. Below we prove $\mathsf{Coh}(ar''_{\mathsf{t}}, ar''_{\mathsf{t}'}, (\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2))$. That is, for any e_1 and e_2, if $e_1\ ar''_{\mathsf{t}}\ e_2$ and $e_2\ ar''_{\mathsf{t}'}\ e_1$, we need to prove $\neg(\Gamma_1 \uplus \Gamma_2 \models e_1\ (\bowtie_1 \uplus \bowtie_2)\ e_2)$. Since $e_1\ ar''_{\mathsf{t}}\ e_2$ and $e_2\ ar''_{\mathsf{t}'}\ e_1$, we know

$$\{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}, \mathsf{t}) \cap \mathsf{visible}(\mathcal{E}, \mathsf{t}').$$

Since $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$ and $\mathcal{E}_2 = \mathcal{E}|_{\Pi_2}$ and $\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$, we know

$$\forall \mathsf{t}.\ \mathsf{visible}(\mathcal{E}, \mathsf{t}) = \mathsf{visible}(\mathcal{E}_1, \mathsf{t}) \cup \mathsf{visible}(\mathcal{E}_2, \mathsf{t}).$$

Since $\mathsf{visible}(\mathcal{E}, \mathsf{t}) = \mathsf{visible}(\mathcal{E}_1, \mathsf{t}) \cup \mathsf{visible}(\mathcal{E}_2, \mathsf{t})$, we have four cases:

1. $\{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}_1, \mathsf{t})$. So $\{\mathsf{op}(e_1), \mathsf{op}(e_2)\} \subseteq dom(\Pi_1)$. Thus
$$\{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}_1, \mathsf{t}').$$
Since $e_1\ ar''_{\mathsf{t}}\ e_2$, $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t})}(ar_{\mathsf{t}})$ and $ar_{\mathsf{t}'} \subseteq ar''_{\mathsf{t}'}$, we know
$$e_1\ ar_{\mathsf{t}}\ e_2.$$
Since $e_2\ ar''_{\mathsf{t}'}\ e_1$, $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t}')}(ar_{\mathsf{t}'})$ and $ar_{\mathsf{t}'} \subseteq ar''_{\mathsf{t}'}$, we know
$$e_2\ ar_{\mathsf{t}'}\ e_1.$$
From $\mathsf{Coh}(ar_{\mathsf{t}}, ar_{\mathsf{t}'}, (\Gamma_1, \bowtie_1))$, we know
$$\neg(\Gamma_1 \models e_1\ \bowtie_1\ e_2).$$
Since $\{\mathsf{op}(e_1), \mathsf{op}(e_2)\} \subseteq dom(\Pi_1) = dom(\Gamma_1)$ and $\bowtie_2 \subseteq \mathsf{act}(\Gamma_2) \times \mathsf{act}(\Gamma_2)$ and $\mathsf{act}(\Gamma_1) \cap \mathsf{act}(\Gamma_2) = \emptyset$, we know
$$\neg(\Gamma_1 \uplus \Gamma_2 \models e_1\ (\bowtie_1 \uplus \bowtie_2)\ e_2).$$
2. $\{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}_2, \mathsf{t})$. Similar to case (1).
3. $e_1 \in \mathsf{visible}(\mathcal{E}_1, \mathsf{t})$ and $e_2 \in \mathsf{visible}(\mathcal{E}_2, \mathsf{t})$. So
$$\mathsf{op}(e_1) \in dom(\Gamma_1) \text{ and } \mathsf{op}(e_2) \in dom(\Gamma_2).$$
Since $\bowtie_1 \subseteq \mathsf{act}(\Gamma_1) \times \mathsf{act}(\Gamma_1)$ and $\bowtie_2 \subseteq \mathsf{act}(\Gamma_2) \times \mathsf{act}(\Gamma_2)$ and $\mathsf{act}(\Gamma_1) \cap \mathsf{act}(\Gamma_2) = \emptyset$, we know
$$\neg(\Gamma_1 \uplus \Gamma_2 \models e_1\ (\bowtie_1 \uplus \bowtie_2)\ e_2).$$
4. $e_1 \in \mathsf{visible}(\mathcal{E}_2, \mathsf{t})$ and $e_2 \in \mathsf{visible}(\mathcal{E}_1, \mathsf{t})$. Similar to case (3).

Thus we are done. □

Lemma 34. If

- $\mathsf{XACT}_{\varphi_1}(\mathcal{E}_1, \mathcal{S}_1, (\Gamma_1, \bowtie_1, \blacktriangleleft_1, \rhd_1))$ and $\mathsf{XACT}_{\varphi_2}(\mathcal{E}_2, \mathcal{S}_2, (\Gamma_2, \bowtie_2, \blacktriangleleft_2, \rhd_2))$,
- $\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2$, $\mathcal{S}_1 \in dom(\varphi_1)$, $\mathcal{S}_2 \in dom(\varphi_2)$,
- $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$, $\mathcal{E}_2 = \mathcal{E}|_{\Pi_2}$, $\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$,
- $dom(\varphi_1) \cap dom(\varphi_2) = \emptyset$, $range(\varphi_1) \cap range(\varphi_2) = \emptyset$, $dom(\Gamma_1) = dom(\Pi_1)$, $dom(\Gamma_2) = dom(\Pi_2)$,
- $\bowtie_1 \subseteq \mathsf{act}(\Gamma_1) \times \mathsf{act}(\Gamma_1)$, $\bowtie_2 \subseteq \mathsf{act}(\Gamma_2) \times \mathsf{act}(\Gamma_2)$, $\mathsf{act}(\Gamma_1) \cap \mathsf{act}(\Gamma_2) = \emptyset$,
- $\mathsf{SLocality}(\Pi_1)$, $\mathsf{SLocality}(\Pi_2)$, $\mathsf{SLocality}(\Gamma_1)$, $\mathsf{SLocality}(\Gamma_2)$,
- $\blacktriangleleft_1 \subseteq \bowtie_1$, $\rhd_1 \subseteq \bowtie_1$, $\blacktriangleleft_2 \subseteq \bowtie_2$, $\rhd_2 \subseteq \bowtie_2$,

then $\mathsf{XACT}_{\varphi_1 \uplus \varphi_2}(\mathcal{E}, \mathcal{S}, (\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2, \blacktriangleleft_1 \uplus \blacktriangleleft_2, \rhd_1 \uplus \rhd_2))$.

Proof. From $\mathsf{XACT}_{\varphi_1}(\mathcal{E}_1, \mathcal{S}_1, (\Gamma_1, \bowtie_1, \blacktriangleleft_1, \rhd_1))$, we know there exists $ar_1, \ldots, ar_n$ such that

$$\begin{aligned}&\forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t})}(ar_{\mathsf{t}}) \wedge (\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}_1}\ \subseteq ar_{\mathsf{t}}) \wedge \mathsf{PresvCancel}(ar_{\mathsf{t}}, \mathsf{t}, \mathcal{E}_1, (\Gamma_1, \rhd_1))\\ &\wedge \mathsf{ExecRelated}_{\varphi_1}(\mathsf{t}, (\mathcal{E}_1, \mathcal{S}_1), (\Gamma_1, ar_{\mathsf{t}})) \wedge \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((ar_{\mathsf{t}}, ar_{\mathsf{t}'}), \mathcal{E}_1, (\Gamma_1, \bowtie_1, \blacktriangleleft_1, \rhd_1))\end{aligned}$$

From $\mathsf{XACT}_{\varphi_2}(\mathcal{E}_2, \mathcal{S}_2, (\Gamma_2, \bowtie_2, \blacktriangleleft_1, \rhd_1))$, we know there exists $ar'_1, \ldots, ar'_n$ such that

$$\begin{aligned}&\forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_2,\mathsf{t})}(ar'_{\mathsf{t}}) \wedge (\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}_2}\ \subseteq ar'_{\mathsf{t}}) \wedge \mathsf{PresvCancel}(ar'_{\mathsf{t}}, \mathsf{t}, \mathcal{E}_2, (\Gamma_2, \rhd_2))\\ &\wedge \mathsf{ExecRelated}_{\varphi_2}(\mathsf{t}, (\mathcal{E}_2, \mathcal{S}_2), (\Gamma_2, ar'_{\mathsf{t}})) \wedge \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((ar'_{\mathsf{t}}, ar'_{\mathsf{t}'}), \mathcal{E}_2, (\Gamma_2, \bowtie_2, \blacktriangleleft_2, \rhd_2))\end{aligned}$$

From Lemma 35, we know

$$\forall \mathsf{t}.\ \mathsf{partialOrder}((ar_{\mathsf{t}} \cup ar'_{\mathsf{t}} \cup \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}\ )^+).$$

Thus we know there exist $ar''_1, \ldots, ar''_n$ such that

$$\forall \mathsf{t}.\ \mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(ar''_{\mathsf{t}}) \wedge \ \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}\ \subseteq ar''_{\mathsf{t}} \ \wedge\ ar_{\mathsf{t}} \subseteq ar''_{\mathsf{t}} \ \wedge\ ar'_{\mathsf{t}} \subseteq ar''_{\mathsf{t}}$$

Since $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$ and $\mathcal{E}_2 = \mathcal{E}|_{\Pi_2}$ and $\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$, we know

$$\forall \mathsf{t}.\ \mathsf{visible}(\mathcal{E}, \mathsf{t}) = \mathsf{visible}(\mathcal{E}_1, \mathsf{t}) \uplus \mathsf{visible}(\mathcal{E}_2, \mathsf{t}).$$

Since $dom(\Gamma_1) = dom(\Pi_1)$, $dom(\Gamma_2) = dom(\Pi_2)$, $\bowtie_1 \subseteq \mathsf{act}(\Gamma_1) \times \mathsf{act}(\Gamma_1)$, $\bowtie_2 \subseteq \mathsf{act}(\Gamma_2) \times \mathsf{act}(\Gamma_2)$, $\mathsf{act}(\Gamma_1) \cap \mathsf{act}(\Gamma_2) = \emptyset$, $\rhd_1 \subseteq \bowtie_1$ and $\rhd_2 \subseteq \bowtie_2$, we have

$$\left(\xmapsto{\mathsf{vis}}_{\mathcal{E}} \cap (\rhd_1 \uplus \rhd_2)_{\Gamma_1 \uplus \Gamma_2}\right)|_{\mathsf{visible}(\mathcal{E},\mathsf{t})} \subseteq \left(\xmapsto{\mathsf{vis}}_{\mathcal{E}_1} \cap (\rhd_1)_{\Gamma_1}\right)|_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t})} \uplus \left(\xmapsto{\mathsf{vis}}_{\mathcal{E}_2} \cap (\rhd_2)_{\Gamma_2}\right)|_{\mathsf{visible}(\mathcal{E}_2,\mathsf{t})}$$

From $\mathsf{PresvCancel}(ar_{\mathsf{t}}, \mathsf{t}, \mathcal{E}_1, (\Gamma_1, \rhd_1))$ and $\mathsf{PresvCancel}(ar'_{\mathsf{t}}, \mathsf{t}, \mathcal{E}_2, (\Gamma_2, \rhd_2))$, we know

$$\left(\xmapsto{\mathsf{vis}}_{\mathcal{E}_1} \cap (\rhd_1)_{\Gamma_1}\right)|_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t})} \subseteq ar_{\mathsf{t}},\quad \left(\xmapsto{\mathsf{vis}}_{\mathcal{E}_2} \cap (\rhd_2)_{\Gamma_2}\right)|_{\mathsf{visible}(\mathcal{E}_2,\mathsf{t})} \subseteq ar'_{\mathsf{t}}$$

Since $ar_{\mathsf{t}} \subseteq ar''_{\mathsf{t}}$ and $ar'_{\mathsf{t}} \subseteq ar''_{\mathsf{t}}$, we know

$$\left(\xmapsto{\mathsf{vis}}_{\mathcal{E}} \cap (\rhd_1 \uplus \rhd_2)_{\Gamma_1 \uplus \Gamma_2}\right)|_{\mathsf{visible}(\mathcal{E},\mathsf{t})} \subseteq ar''_{\mathsf{t}}$$

So $\mathsf{PresvCancel}(ar''_{\mathsf{t}}, \mathsf{t}, \mathcal{E}, (\Gamma_1 \uplus \Gamma_2, \rhd_1 \uplus \rhd_2))$ holds.

From Lemma 36, we know

$$\mathsf{ExecRelated}_{\varphi_1 \uplus \varphi_2}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma_1 \uplus \Gamma_2, ar''_{\mathsf{t}})).$$

Take t and t′ such that $\mathsf{t} \neq \mathsf{t}'$. Below we prove $\mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((ar''_{\mathsf{t}}, ar''_{\mathsf{t}'}), \mathcal{E}, (\Gamma_1 \uplus \Gamma_2, \bowtie_1 \uplus \bowtie_2, \blacktriangleleft_1 \uplus \blacktriangleleft_2, \rhd_1 \uplus \rhd_2))$. That is, for any $\mathcal{E}'$, $\mathcal{E}''$, e_1 and e_2, if $\mathcal{E}' \leqslant \mathcal{E}$, $\mathcal{E}'' \leqslant \mathcal{E}$, $\{e_1, e_2\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}', \mathsf{t}, (\Gamma_1 \uplus \Gamma_2, \rhd_1 \uplus \rhd_2))$, $\{e_1, e_2\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}'', \mathsf{t}', (\Gamma_1 \uplus \Gamma_2, \rhd_1 \uplus \rhd_2))$ and $e_1(\bowtie_1 \uplus \bowtie_2)_{\Gamma_1 \uplus \Gamma_2} e_2$, we want to prove

$$((e_1, e_2) \in ar''_{\mathsf{t}} \cap ar''_{\mathsf{t}'} \vee (e_2, e_1) \in ar''_{\mathsf{t}} \cap ar''_{\mathsf{t}'}) \text{ and } (\mathsf{Concurrent}_{\mathcal{E}}(e_1, e_2) \wedge (e_1(\blacktriangleleft_1 \uplus \blacktriangleleft_2)_{\Gamma_1 \uplus \Gamma_2} e_2) \implies (e_1, e_2) \in ar''_{\mathsf{t}}).$$

Let $\mathcal{E}'_1 = \mathcal{E}'|_{\Pi_1}$, $\mathcal{E}'_2 = \mathcal{E}'|_{\Pi_2}$, $\mathcal{E}''_1 = \mathcal{E}''|_{\Pi_1}$ and $\mathcal{E}''_2 = \mathcal{E}''|_{\Pi_2}$. Since $\mathcal{E}' \leqslant \mathcal{E}$ and $\mathcal{E}'' \leqslant \mathcal{E}$, we know

$$\mathcal{E}'_1 \leqslant \mathcal{E}_1, \mathcal{E}''_1 \leqslant \mathcal{E}_1, \mathcal{E}'_2 \leqslant \mathcal{E}_2, \mathcal{E}''_2 \leqslant \mathcal{E}_2.$$

Since $\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$, we know

$$\mathsf{visible}(\mathcal{E}', \mathsf{t}) = \mathsf{visible}(\mathcal{E}'_1, \mathsf{t}) \cup \mathsf{visible}(\mathcal{E}'_2, \mathsf{t}),$$
$$\mathsf{visible}(\mathcal{E}'', \mathsf{t}') = \mathsf{visible}(\mathcal{E}''_1, \mathsf{t}') \cup \mathsf{visible}(\mathcal{E}''_2, \mathsf{t}').$$

Since $\{e_1, e_2\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}', \mathsf{t}, (\Gamma_1 \uplus \Gamma_2, \rhd_1 \uplus \rhd_2))$ and $\{e_1, e_2\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}'', \mathsf{t}', (\Gamma_1 \uplus \Gamma_2, \rhd_1 \uplus \rhd_2))$, we know

$$\{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}', \mathsf{t}) \cap \mathsf{visible}(\mathcal{E}'', \mathsf{t}').$$

Since $\mathsf{visible}(\mathcal{E}', \mathsf{t}) = \mathsf{visible}(\mathcal{E}'_1, \mathsf{t}) \cup \mathsf{visible}(\mathcal{E}'_2, \mathsf{t})$, we have four cases:

1. $\{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}'_1, \mathsf{t})$. So $\{\mathsf{op}(e_1), \mathsf{op}(e_2)\} \subseteq dom(\Pi_1)$. Thus
$$\{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}''_1, \mathsf{t}').$$
Thus we know
$$\{e_1, e_2\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}'_1, \mathsf{t}, (\Gamma_1, \rhd_1)) \text{ and } \{e_1, e_2\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}''_1, \mathsf{t}', (\Gamma_1, \rhd_1)).$$
Also we know
$$e_1(\bowtie_1)_{\Gamma_1} e_2.$$
From $\mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((ar_{\mathsf{t}}, ar_{\mathsf{t}'}), \mathcal{E}_1, (\Gamma_1, \bowtie_1, \blacktriangleleft_1, \rhd_1))$, we know
$$((e_1, e_2) \in ar_{\mathsf{t}} \cap ar_{\mathsf{t}'} \vee (e_2, e_1) \in ar_{\mathsf{t}} \cap ar_{\mathsf{t}'}) \text{ and } (\mathsf{Concurrent}_{\mathcal{E}}(e_1, e_2) \wedge (e_1(\blacktriangleleft_1)_{\Gamma_1} e_2) \implies (e_1, e_2) \in ar_{\mathsf{t}}).$$
Thus we know
$$((e_1, e_2) \in ar''_{\mathsf{t}} \cap ar''_{\mathsf{t}'} \vee (e_2, e_1) \in ar''_{\mathsf{t}} \cap ar''_{\mathsf{t}'}) \text{ and } (\mathsf{Concurrent}_{\mathcal{E}}(e_1, e_2) \wedge (e_1(\blacktriangleleft_1 \uplus \blacktriangleleft_2)_{\Gamma_1 \uplus \Gamma_2} e_2) \implies (e_1, e_2) \in ar''_{\mathsf{t}}).$$
2. $\{e_1, e_2\} \subseteq \mathsf{visible}(\mathcal{E}_2, \mathsf{t})$. Similar to case (1).
3. $e_1 \in \mathsf{visible}(\mathcal{E}_1, \mathsf{t})$ and $e_2 \in \mathsf{visible}(\mathcal{E}_2, \mathsf{t})$. So
$$\mathsf{op}(e_1) \in dom(\Gamma_1) \text{ and } \mathsf{op}(e_2) \in dom(\Gamma_2).$$
Since $\bowtie_1 \subseteq \mathsf{act}(\Gamma_1) \times \mathsf{act}(\Gamma_1)$ and $\bowtie_2 \subseteq \mathsf{act}(\Gamma_2) \times \mathsf{act}(\Gamma_2)$ and $\mathsf{act}(\Gamma_1) \cap \mathsf{act}(\Gamma_2) = \emptyset$, we know
$$\neg(e_1(\bowtie_1 \uplus \bowtie_2)_{\Gamma_1 \uplus \Gamma_2} e_2).$$
So this case is impossible.

4. $e_1 \in \mathsf{visible}(\mathcal{E}_2, \mathsf{t})$ and $e_2 \in \mathsf{visible}(\mathcal{E}_1, \mathsf{t})$. Similar to case (3).

Thus we are done. □

Lemma 35. If

- $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}, \mathcal{E}_2 = \mathcal{E}|_{\Pi_2}, \mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$,
- $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t})}(ar_1)$, $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_2,\mathsf{t})}(ar_2)$, $\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}_1} \subseteq ar_1$, $\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}_2} \subseteq ar_2$,

then $\mathsf{partialOrder}((ar_1 \cup ar_2 \cup \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}})^+)$.

Proof. Let $rel = (ar_1 \cup ar_2 \cup \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}})$. We know $\mathsf{transitive}(rel^+)$. Below we prove $\mathsf{irreflexive}(rel^+)$. So we only need to prove: $\neg\mathsf{cyclic}(rel)$.

By contradiction. Suppose there exist $n, e_1, \ldots, e_n$ such that $\forall i \in [1..n-1]. (e_i, e_{i+1}) \in rel$ and $(e_n, e_1) \in rel$. Without loss of generality, we can suppose n is the length of the smallest cycle. We analyze the following cases.

- $n = 1$. Since $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t})}(ar_1)$ and $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_2,\mathsf{t})}(ar_2)$, we know this case is impossible.
- $n > 1$.
  Since $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t})}(ar_1)$, $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_2,\mathsf{t})}(ar_2)$, $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$, $\mathcal{E}_2 = \mathcal{E}|_{\Pi_2}$ and $dom(\Pi_1) \cap dom(\Pi_2) = \emptyset$, we know
  $$\neg((\forall i \in [1..n-1]. (e_i, e_{i+1}) \in (ar_1 \cup ar_2)) \land ((e_n, e_1) \in (ar_1 \cup ar_2)))$$
  So, without loss of generality, we can assume that $(e_1, e_2) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$. We analyze the different cases of $(e_n, e_1) \in ar$:
  - $(e_n, e_1) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$. Thus $(e_n, e_2) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$. So we have a smaller cycle $e_2, \ldots, e_n, e_2$. Thus we get a contradiction.
  - $(e_n, e_1) \in ar_1$. So
    $$\{\mathsf{op}(e_n), \mathsf{op}(e_1)\} \subseteq dom(\Pi_1).$$
    Since $e_2 \in \mathcal{E}$ and $\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$, we know $\mathsf{op}(e_2) \in dom(\Pi_1) \uplus dom(\Pi_2)$.
    - $\mathsf{op}(e_2) \in dom(\Pi_1)$.
      Since $(e_1, e_2) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$, we know
      $$(e_1, e_2) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}_1}.$$
      Since $\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}_1} \subseteq ar_1$, we know
      $$(e_1, e_2) \in ar_1.$$
      Since $(e_n, e_1) \in ar_1$ and $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t})}(ar_1)$, we know
      $$(e_n, e_2) \in ar_1.$$
      So we have a smaller cycle $e_2, \ldots, e_n, e_2$. Thus we get a contradiction.
    - $\mathsf{op}(e_2) \in dom(\Pi_2)$. Since $\mathsf{op}(e_n) \in dom(\Pi_1)$, we know $e_2 \neq e_n$. So $n > 2$. We analyze the different cases of $(e_2, e_3) \in ar$:
      - $(e_2, e_3) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$. Since $(e_1, e_2) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$, we know $(e_1, e_3) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$. So we have a smaller cycle $e_1, e_3, \ldots, e_n, e_1$. Thus we get a contradiction.
      - $(e_2, e_3) \in ar_1$. Since $\mathsf{op}(e_2) \in dom(\Pi_2)$, we know this case is impossible.
      - $(e_2, e_3) \in ar_2$. So $\mathsf{op}(e_3) \in dom(\Pi_2)$. Since $\mathsf{op}(e_n) \in dom(\Pi_1)$, we know $e_3 \neq e_n$. So $n > 3$. We analyze the different cases of $(e_3, e_4) \in ar$:
        - $(e_3, e_4) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$. Since $(e_1, e_2) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$, we know $(e_2, e_4) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$ or $(e_4, e_2) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$.
          - $(e_2, e_4) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$. So we have a smaller cycle $e_1, e_2, e_4, \ldots, e_n, e_1$. Thus we get a contradiction.
          - $(e_4, e_2) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$. So we have a smaller cycle e_2, e_3, e_4, e_2. Thus we get a contradiction.
        - $(e_3, e_4) \in ar_1$. Since $\mathsf{op}(e_3) \in dom(\Pi_2)$, we know this case is impossible.
        - $(e_3, e_4) \in ar_2$. Since $(e_2, e_3) \in ar_2$ and $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_2,\mathsf{t})}(ar_2)$, we know
          $$(e_2, e_4) \in ar_2.$$
        So we have a smaller cycle $e_1, e_2, e_4, \ldots, e_n, e_1$. Thus we get a contradiction.
  - $(e_n, e_1) \in ar_2$. Similar to the previous case.

Thus we are done. □

Lemma 36. If

- $\mathsf{ExecRelated}_{\varphi_1}(\mathsf{t}, (\mathcal{E}_1, \mathcal{S}_1), (\Gamma_1, ar_1))$ and $\mathsf{ExecRelated}_{\varphi_2}(\mathsf{t}, (\mathcal{E}_2, \mathcal{S}_2), (\Gamma_2, ar_2))$,
- $\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2$, $\mathcal{S}_1 \in dom(\varphi_1)$, $\mathcal{S}_2 \in dom(\varphi_2)$,
- $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$, $\mathcal{E}_2 = \mathcal{E}|_{\Pi_2}$, $\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$,
- $dom(\varphi_1) \cap dom(\varphi_2) = \emptyset$, $range(\varphi_1) \cap range(\varphi_2) = \emptyset$, $dom(\Gamma_1) = dom(\Pi_1)$, $dom(\Gamma_2) = dom(\Pi_2)$,
- $\mathsf{SLocality}(\Pi_1)$, $\mathsf{SLocality}(\Pi_2)$, $\mathsf{SLocality}(\Gamma_1)$, $\mathsf{SLocality}(\Gamma_2)$,
- $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t})}(ar_1)$, $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_2,\mathsf{t})}(ar_2)$, $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(ar)$, $\overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_{\mathcal{E}} \subseteq ar$, $ar_1 \subseteq ar$, $ar_2 \subseteq ar$,

then $\mathsf{ExecRelated}_{\varphi_1 \uplus \varphi_2}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma_1 \uplus \Gamma_2, ar))$.

Proof. We want to prove: for any $\mathcal{E}'$, if $\mathcal{E}' \leqslant \mathcal{E}$, then

$$\begin{array}{c}(\varphi_1 \uplus \varphi_2)(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma_1 \uplus \Gamma_2, (\varphi_1 \uplus \varphi_2)(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar) \text{ and} \\ \forall e.\ \mathsf{last}(\mathcal{E}') = e \wedge \mathsf{is\_orig}_{\mathsf{t}}(e) \implies \mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma_1 \uplus \Gamma_2, (\varphi_1 \uplus \varphi_2)(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar).\end{array}$$

Let $\mathcal{E}'_1 = \mathcal{E}'|_{\Pi_1}$ and $\mathcal{E}'_2 = \mathcal{E}'|_{\Pi_2}$. Since $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$, $\mathcal{E}_2 = \mathcal{E}|_{\Pi_2}$ and $\mathcal{E}' \leqslant \mathcal{E}$, we know

$$\mathcal{E}'_1 \leqslant \mathcal{E}_1 \text{ and } \mathcal{E}'_2 \leqslant \mathcal{E}_2.$$

Then, from $\mathsf{ExecRelated}_{\varphi_1}(\mathsf{t}, (\mathcal{E}_1, \mathcal{S}_1), (\Gamma_1, ar_1))$ and $\mathsf{ExecRelated}_{\varphi_2}(\mathsf{t}, (\mathcal{E}_2, \mathcal{S}_2), (\Gamma_2, ar_2))$, we know

$$\begin{array}{r}\varphi_1(\mathsf{exec\_st}(\mathcal{S}_1, \mathcal{E}'_1|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma_1, \varphi_1(\mathcal{S}_1), \mathsf{visible}(\mathcal{E}'_1, \mathsf{t}) \downharpoonright ar_1), \\ \forall e.\ \mathsf{last}(\mathcal{E}'_1) = e \wedge \mathsf{is\_orig}_{\mathsf{t}}(e) \implies \mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma_1, \varphi_1(\mathcal{S}_1), \mathsf{visible}(\mathcal{E}'_1, \mathsf{t}) \downharpoonright ar_1), \\ \varphi_2(\mathsf{exec\_st}(\mathcal{S}_2, \mathcal{E}'_2|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma_2, \varphi_2(\mathcal{S}_2), \mathsf{visible}(\mathcal{E}'_2, \mathsf{t}) \downharpoonright ar_2), \\ \forall e.\ \mathsf{last}(\mathcal{E}'_2) = e \wedge \mathsf{is\_orig}_{\mathsf{t}}(e) \implies \mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma_2, \varphi_2(\mathcal{S}_2), \mathsf{visible}(\mathcal{E}'_2, \mathsf{t}) \downharpoonright ar_2).\end{array}$$

Since $\mathcal{E}'_1 = \mathcal{E}'|_{\Pi_1}$ and $\mathcal{E}'_2 = \mathcal{E}'|_{\Pi_2}$, we know

$$\mathcal{E}'_1|_{\mathsf{t}} = (\mathcal{E}'|_{\mathsf{t}})|_{\Pi_1} \text{ and } \mathcal{E}'_2|_{\mathsf{t}} = (\mathcal{E}'|_{\mathsf{t}})|_{\Pi_2}.$$

By Lemma 37, we know

$$\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}}) = \mathsf{exec\_st}(\mathcal{S}_1, \mathcal{E}'_1|_{\mathsf{t}}) \uplus \mathsf{exec\_st}(\mathcal{S}_2, \mathcal{E}'_2|_{\mathsf{t}}).$$

Since $dom(\varphi_1) \cap dom(\varphi_2) = \emptyset$, we know

$$(\varphi_1 \uplus \varphi_2)(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \varphi_1(\mathsf{exec\_st}(\mathcal{S}_1, \mathcal{E}'_1|_{\mathsf{t}})) \uplus \varphi_2(\mathsf{exec\_st}(\mathcal{S}_2, \mathcal{E}'_2|_{\mathsf{t}})).$$

Since $\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2$, we know

$$(\varphi_1 \uplus \varphi_2)(\mathcal{S}) = \varphi_1(\mathcal{S}_1) \uplus \varphi_2(\mathcal{S}_2)$$

Since $\mathcal{E}'_1 = \mathcal{E}'|_{\Pi_1}$, $\mathcal{E}'_2 = \mathcal{E}'|_{\Pi_2}$, $dom(\Gamma_1) = dom(\Pi_1)$ and $dom(\Gamma_2) = dom(\Pi_2)$, we know

$$\mathcal{E}'_1 = \mathcal{E}'|_{\Gamma_1} \text{ and } \mathcal{E}'_2 = \mathcal{E}'|_{\Gamma_2}.$$

Then, since $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_1,\mathsf{t})}(ar_1)$, $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}_2,\mathsf{t})}(ar_2)$, $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(ar)$, $ar_1 \subseteq ar$, $ar_2 \subseteq ar$ and $\mathcal{E}' \leqslant \mathcal{E}$, we know

$$\mathsf{visible}(\mathcal{E}'_1, \mathsf{t}) \downharpoonright ar_1 = (\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar)|_{\Gamma_1} \text{ and } \mathsf{visible}(\mathcal{E}'_2, \mathsf{t}) \downharpoonright ar_2 = (\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar)|_{\Gamma_2}.$$

By Lemma 38, we know

$$\begin{array}{l}\mathsf{aexecST}(\Gamma_1 \uplus \Gamma_2, \varphi_1(\mathcal{S}_1) \uplus \varphi_2(\mathcal{S}_2), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar) \\ = \mathsf{aexecST}(\Gamma_1, \varphi_1(\mathcal{S}_1), \mathsf{visible}(\mathcal{E}'_1, \mathsf{t}) \downharpoonright ar_1) \uplus \mathsf{aexecST}(\Gamma_2, \varphi_2(\mathcal{S}_2), \mathsf{visible}(\mathcal{E}'_2, \mathsf{t}) \downharpoonright ar_2)\end{array}$$

Thus we know

$$(\varphi_1 \uplus \varphi_2)(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma_1 \uplus \Gamma_2, (\varphi_1 \uplus \varphi_2)(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar).$$

Next, we prove $\forall e.\ \mathsf{last}(\mathcal{E}') = e \wedge \mathsf{is\_orig}_{\mathsf{t}}(e) \implies \mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma_1 \uplus \Gamma_2, (\varphi_1 \uplus \varphi_2)(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar)$. Since $\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$, we have two cases:

- $\mathsf{op}(e) \in dom(\Pi_1)$. Since $\mathcal{E}'_1 = \mathcal{E}'|_{\Pi_1}$ and $\mathsf{last}(\mathcal{E}') = e$, we know $\mathsf{last}(\mathcal{E}'_1) = e$. Thus
$$\mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma_1, \varphi_1(\mathcal{S}_1), \mathsf{visible}(\mathcal{E}'_1, \mathsf{t}) \downharpoonright ar_1).$$

Since $\xrightarrow[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \subseteq ar$, $\mathsf{last}(\mathcal{E}') = e$, and $\mathsf{is\_orig}_{\mathsf{t}}(e)$, we know

$$\mathsf{last}(\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar) = e \text{ and } \mathsf{last}(\mathsf{visible}(\mathcal{E}_1', \mathsf{t}) \downharpoonright ar_1) = e.$$

Suppose $(\mathsf{visible}(\mathcal{E}_1', \mathsf{t}) \downharpoonright ar_1) = \mathcal{E}_1''{+}{+}[e]$ and $(\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar) = \mathcal{E}''{+}{+}[e]$. Then there exist $\mathcal{S}_1''$ and n' such that

$$\mathsf{aexecST}(\Gamma_1, \varphi_1(\mathcal{S}_1), \mathcal{E}_1'') = \mathcal{S}_1'', \quad \Gamma_1(\mathsf{op}(e))(\mathcal{S}_1'') = (n', \_) \text{ and } \mathsf{aexecRV}(\Gamma_1, \varphi_1(\mathcal{S}_1), \mathsf{visible}(\mathcal{E}_1', \mathsf{t}) \downharpoonright ar_1) = n'$$

Since $\mathsf{visible}(\mathcal{E}_1', \mathsf{t}) \downharpoonright ar_1 = (\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar)|_{\Gamma_1}$ and $\mathsf{visible}(\mathcal{E}_2', \mathsf{t}) \downharpoonright ar_2 = (\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar)|_{\Gamma_2}$, we know

$$\mathcal{E}''|_{\Gamma_1} = \mathcal{E}_1'' \text{ and } \mathcal{E}''|_{\Gamma_2} = \mathsf{visible}(\mathcal{E}_2', \mathsf{t}) \downharpoonright ar_2.$$

By Lemma 38, we know

$$\begin{aligned}
&\mathsf{aexecST}(\Gamma_1 \uplus \Gamma_2, \varphi_1(\mathcal{S}_1) \uplus \varphi_2(\mathcal{S}_2), \mathcal{E}'') \\
&= \mathsf{aexecST}(\Gamma_1, \varphi_1(\mathcal{S}_1), \mathcal{E}_1'') \uplus \mathsf{aexecST}(\Gamma_2, \varphi_2(\mathcal{S}_2), \mathsf{visible}(\mathcal{E}_2', \mathsf{t}) \downharpoonright ar_2) \\
&= \mathcal{S}_1'' \uplus \mathsf{aexecST}(\Gamma_2, \varphi_2(\mathcal{S}_2), \mathsf{visible}(\mathcal{E}_2', \mathsf{t}) \downharpoonright ar_2)
\end{aligned}$$

Since $\Gamma_1(\mathsf{op}(e))(\mathcal{S}_1'') = (n', \_)$, from $\mathsf{SLocality}(\Gamma_1)$, we know

$$\Gamma_1(\mathsf{op}(e))(\mathcal{S}_1'' \uplus \mathsf{aexecST}(\Gamma_2, \varphi_2(\mathcal{S}_2), \mathsf{visible}(\mathcal{E}_2', \mathsf{t}) \downharpoonright ar_2)) = (n', \_).$$

Thus we know

$$\begin{aligned}
&\mathsf{aexecRV}(\Gamma_1 \uplus \Gamma_2, (\varphi_1 \uplus \varphi_2)(\mathcal{S}), \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar) \\
&= \mathsf{aexecRV}(\Gamma_1 \uplus \Gamma_2, \varphi_1(\mathcal{S}_1) \uplus \varphi_2(\mathcal{S}_2), \mathcal{E}''{+}{+}[e]) \\
&= n' = \mathsf{rval}(e)
\end{aligned}$$

- $\mathsf{op}(e) \in dom(\Pi_2)$. Similar to the previous case.

Thus we are done. □

Lemma 37. If

- $\mathsf{exec\_st}(\mathcal{S}_1, \mathcal{E}_1) = \mathcal{S}_1'$, $\mathsf{exec\_st}(\mathcal{S}_2, \mathcal{E}_2) = \mathcal{S}_2'$,
- $\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2$, $\mathcal{E}_1 = \mathcal{E}|_{\Pi_1}$, $\mathcal{E}_2 = \mathcal{E}|_{\Pi_2}$, $\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$,
- $\mathsf{SLocality}(\Pi_1)$, $\mathsf{SLocality}(\Pi_2)$,

then $\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}) = \mathcal{S}_1' \uplus \mathcal{S}_2'$.

Proof. By induction over the length n of $\mathcal{E}$.

- $n = 0$. So $\mathcal{E}_1 = \mathcal{E}_2 = \mathcal{E} = \epsilon$. So $\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}) = \mathcal{S}_1' \uplus \mathcal{S}_2'$.
- $n = k + 1$. Suppose $\mathcal{E} = e :: \mathcal{E}'$.
  Since $\mathsf{op}(\mathcal{E}) \subseteq dom(\Pi_1) \uplus dom(\Pi_2)$, we have two cases:
  - $\mathsf{op}(e) \in dom(\Pi_1)$.
    Let $\mathcal{E}_1' = \mathcal{E}'|_{\Pi_1}$. So we know
    $$\mathcal{E}_1 = e :: \mathcal{E}_1' \text{ and } \mathcal{E}_2 = \mathcal{E}'|_{\Pi_2}.$$
    Since $\mathsf{exec\_st}(\mathcal{S}_1, \mathcal{E}_1) = \mathcal{S}_1'$, we know there exists $\mathcal{S}_1''$ such that
    $$\mathsf{exec\_st}(\mathcal{S}_1, [e]) = \mathcal{S}_1'' \text{ and } \mathsf{exec\_st}(\mathcal{S}_1'', \mathcal{E}_1') = \mathcal{S}_1'.$$
    Since $\mathsf{SLocality}(\Pi_1)$, we know
    $$\mathsf{exec\_st}(\mathcal{S}_1 \uplus \mathcal{S}_2, [e]) = \mathcal{S}_1'' \uplus \mathcal{S}_2.$$
    Also, by the induction hypothesis, we know
    $$\mathsf{exec\_st}(\mathcal{S}_1'' \uplus \mathcal{S}_2, \mathcal{E}') = \mathcal{S}_1' \uplus \mathcal{S}_2'.$$
    So we know $\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}) = \mathcal{S}_1' \uplus \mathcal{S}_2'$.
  - $\mathsf{op}(e) \in dom(\Pi_2)$. Similar to the previous case.

Thus we are done. □

Lemma 38. If

- $\mathsf{aexecST}(\Gamma_1, \mathcal{S}_1, \mathcal{E}_1) = \mathcal{S}_1'$, $\mathsf{aexecST}(\Gamma_2, \mathcal{S}_2, \mathcal{E}_2) = \mathcal{S}_2'$,
- $\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2$, $\mathcal{E}_1 = \mathcal{E}|_{\Gamma_1}$, $\mathcal{E}_2 = \mathcal{E}|_{\Gamma_2}$, $\mathsf{op}(\mathcal{E}) \subseteq dom(\Gamma_1) \uplus dom(\Gamma_2)$,
- $\mathsf{SLocality}(\Gamma_1)$, $\mathsf{SLocality}(\Gamma_2)$,

then $\mathsf{aexecST}(\Gamma_1 \uplus \Gamma_2, \mathcal{S}, \mathcal{E}) = \mathcal{S}_1' \uplus \mathcal{S}_2'$.

Proof. By induction over the length n of $\mathcal{E}$.

- $n = 0$. So $\mathcal{E}_1 = \mathcal{E}_2 = \mathcal{E} = \epsilon$. So $\mathsf{aexecST}(\Gamma_1 \uplus \Gamma_2, \mathcal{S}, \mathcal{E}) = \mathcal{S}_1' \uplus \mathcal{S}_2'$.
- $n = k + 1$. Suppose $\mathcal{E} = e :: \mathcal{E}'$.
  Since $\mathsf{op}(\mathcal{E}) \subseteq dom(\Gamma_1) \uplus dom(\Gamma_2)$, we have two cases:

- $\mathsf{op}(e) \in dom(\Gamma_1)$.
  Let $\mathcal{E}_1' = \mathcal{E}'|_{\Gamma_1}$. So we know
  $$\mathcal{E}_1 = e :: \mathcal{E}_1' \text{ and } \mathcal{E}_2 = \mathcal{E}'|_{\Gamma_2}.$$
  Since $\mathsf{aexecST}(\Gamma_1, \mathcal{S}_1, \mathcal{E}_1) = \mathcal{S}_1'$, we know there exists $\mathcal{S}_1''$ such that
  $$\mathsf{aexecST}(\Gamma_1, \mathcal{S}_1, [e]) = \mathcal{S}_1'' \text{ and } \mathsf{aexecST}(\Gamma_1, \mathcal{S}_1'', \mathcal{E}_1') = \mathcal{S}_1'.$$
  Since $\mathsf{SLocality}(\Gamma_1)$, we know
  $$\mathsf{aexecST}(\Gamma_1, \mathcal{S}_1 \uplus \mathcal{S}_2, [e]) = \mathcal{S}_1'' \uplus \mathcal{S}_2.$$
  So
  $$\mathsf{aexecST}(\Gamma_1 \uplus \Gamma_2, \mathcal{S}_1 \uplus \mathcal{S}_2, [e]) = \mathcal{S}_1'' \uplus \mathcal{S}_2.$$
  Also, by the induction hypothesis, we know
  $$\mathsf{aexecST}(\Gamma_1 \uplus \Gamma_2, \mathcal{S}_1'' \uplus \mathcal{S}_2, \mathcal{E}') = \mathcal{S}_1' \uplus \mathcal{S}_2'.$$
  So we know $\mathsf{aexecST}(\Gamma_1 \uplus \Gamma_2, \mathcal{S}, \mathcal{E}) = \mathcal{S}_1' \uplus \mathcal{S}_2'$.
- $\mathsf{op}(e) \in dom(\Gamma_2)$. Similar to the previous case.

Thus we are done. □

$$
\begin{array}{llll}
(\mathit{DoneFlag}) & d ::= \mathbf{prd} \mid \mathbf{cmt} & (\mathit{Trace}) & T ::= \epsilon \mid (i, \mathrm{t}, \alpha, d) :: T \\
(\mathit{ActSet}) & \mathcal{A} \in \mathit{Nat} \rightharpoonup (\mathit{NodeID} \times \mathit{Action} \times \mathit{DoneFlag}) & (\mathit{ActOrd}) & \eta \in \mathscr{P}(\mathit{Nat} \times \mathit{Nat})
\end{array}
$$

$$\mathcal{A}, \eta \models T \quad \text{iff} \quad \lfloor T \rfloor = \mathcal{A} \land (\forall i, j.\ (i, j) \in \eta \land \{i, j\} \subseteq \mathit{dom}(\mathcal{A}) \implies i <_T j)$$

$$
\mathrm{exec}(\mathcal{S}, T) \overset{\mathrm{def}}{=} \begin{cases} \mathrm{exec}(\alpha(\mathcal{S}), T') & \text{if } T = (i, \mathrm{t}, \alpha, \mathbf{cmt}) :: T' \\ \mathrm{exec}(\mathcal{S}, T') & \text{if } T = (i, \mathrm{t}, \alpha, \mathbf{prd}) :: T' \\ \mathcal{S} & \text{if } T = \epsilon \end{cases}
$$

(a) the action model

$$
\begin{array}{lll}
(\mathcal{S}, \mathcal{A}, \eta) \models \mathcal{P} & \text{iff} & \forall T, \mathcal{S}'.\ (\mathcal{A}, \eta \models T) \land (\mathcal{S}' = \mathrm{exec}(\mathcal{S}, T)) \implies \mathcal{S}' \models_{\text{HOARE}} \mathcal{P} \\
(\mathcal{S}, \mathcal{A}, \eta) \models \mathsf{emp} & \text{iff} & \mathcal{A} = \emptyset \\
(\mathcal{S}, \mathcal{A}, \eta) \models \mathsf{Id} & \text{iff} & \forall i \in \mathit{dom}(\mathcal{A}).\ \mathcal{A}(i) = (\_, \alpha_{\mathsf{Id}}, \_) \\
(\mathcal{S}, \mathcal{A}, \eta) \models [\alpha]^i_{\mathrm{t}} & \text{iff} & \mathcal{A} = \{i \rightsquigarrow (\mathrm{t}, \alpha, \_)\} \\
(\mathcal{S}, \mathcal{A}, \eta) \models \boxed{\alpha}^i_{\mathrm{t}} & \text{iff} & \mathcal{A} = \{i \rightsquigarrow (\mathrm{t}, \alpha, \mathbf{cmt})\} \\
(\mathcal{S}, \mathcal{A}, \eta) \models p \sqcup q & \text{iff} & \exists \mathcal{A}_1, \mathcal{A}_2, \eta_1, \eta_2.\ (\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2) \land (\eta = \eta_1 \cup \eta_2) \\
 & & \land\ ((\mathcal{S}, \mathcal{A}_1, \eta_1) \models p) \land ((\mathcal{S}, \mathcal{A}_2, \eta_2) \models q) \\
(\mathcal{S}, \mathcal{A}, \eta) \models p \ltimes [\alpha]^i_{\mathrm{t}} & \text{iff} & \exists \mathcal{A}', \eta'.\ ((\mathcal{S}, \mathcal{A}', \eta') \models p) \land \mathcal{A} = \mathcal{A}' \uplus \{i \rightsquigarrow (\mathrm{t}, \alpha, \_)\} \\
 & & \land\ \eta = \eta' \uplus \{(j, i) \mid j \in \mathit{dom}(\mathcal{A}')\} \\
(\mathcal{S}, \mathcal{A}, \eta) \models p \ltimes \boxed{\alpha}^i_{\mathrm{t}} & \text{iff} & \exists \mathcal{A}', \eta'.\ ((\mathcal{S}, \mathcal{A}', \eta') \models p) \land \mathcal{A} = \mathcal{A}' \uplus \{i \rightsquigarrow (\mathrm{t}, \alpha, \mathbf{cmt})\} \\
 & & \land\ \eta = \eta' \uplus \{(j, i) \mid j \in \mathit{dom}(\mathcal{A}')\} \\
(\mathcal{S}, \mathcal{A}, \eta) \models (p, \bowtie) \ltimes [\alpha]^i_{\mathrm{t}} & \text{iff} & \exists \mathcal{A}', \eta'.\ ((\mathcal{S}, \mathcal{A}', \eta') \models p) \land \mathcal{A} = \mathcal{A}' \uplus \{i \rightsquigarrow (\mathrm{t}, \alpha, \_)\} \\
 & & \land\ \eta = \eta' \uplus \{(j, i) \mid \exists \alpha'.\ \mathcal{A}'(j) = (\_, \alpha', \mathbf{cmt}) \land \alpha \bowtie \alpha'\} \\
(\mathcal{S}, \mathcal{A}, \eta) \models (p, \bowtie) \ltimes \boxed{\alpha}^i_{\mathrm{t}} & \text{iff} & \exists \mathcal{A}', \eta'.\ ((\mathcal{S}, \mathcal{A}', \eta') \models p) \land \mathcal{A} = \mathcal{A}' \uplus \{i \rightsquigarrow (\mathrm{t}, \alpha, \mathbf{cmt})\} \\
 & & \land\ \eta = \eta' \uplus \{(j, i) \mid \exists \alpha'.\ \mathcal{A}'(j) = (\_, \alpha', \mathbf{cmt}) \land \alpha \bowtie \alpha'\} \\
(\mathcal{S}, \mathcal{A}, \eta) \models p \Rrightarrow q & \text{iff} & (\mathcal{S}, \mathcal{A}, \eta) \models p \implies \\
 & & \forall \mathcal{A}'.\ \mathcal{A}' = \{(i, (\mathrm{t}, \alpha, \mathbf{cmt})) \mid \mathcal{A}(i) = (\mathrm{t}, \alpha, \_)\} \implies (\mathcal{S}, \mathcal{A}', \eta) \models q \\
(\mathcal{S}, \mathcal{A}, \eta) \models p \Rightarrow q & \text{iff} & ((\mathcal{S}, \mathcal{A}, \eta) \models p) \implies \forall T.\ (\mathcal{A}, \eta \models T) \implies \exists \eta'.\ (\mathcal{A}, \eta' \models T) \land (\mathcal{S}, \mathcal{A}, \eta') \models q \land (\eta \subseteq \eta') \\
 & & ⟐[\alpha]^i_{\mathrm{t}} \overset{\mathrm{def}}{=} [\alpha]^i_{\mathrm{t}} \sqcup \mathsf{true} \qquad ⟐\boxed{\alpha}^i_{\mathrm{t}} \overset{\mathrm{def}}{=} \boxed{\alpha}^i_{\mathrm{t}} \sqcup \mathsf{true}
\end{array}
$$

(b) semantics of action assertions p and q

$$
\begin{array}{l}
((\mathcal{S}, \mathcal{A}, \eta), (i', \mathrm{t}', \alpha')) \models \mathsf{Emp} \qquad \text{never holds} \\
((\mathcal{S}, \mathcal{A}, \eta), (i', \mathrm{t}', \alpha')) \models p \rightsquigarrow [\alpha]^i_{\mathrm{t}} \ \text{ iff } \ ((\mathcal{S}, \mathcal{A}, \eta) \models p) \land i \notin \mathit{dom}(\mathcal{A}) \land i = i' \land \mathrm{t} = \mathrm{t}' \land \alpha = \alpha' \\
\qquad \mathsf{IId} \overset{\mathrm{def}}{=} \exists i, \mathrm{t}.\ \mathsf{true} \rightsquigarrow [\alpha_{\mathsf{Id}}]^i_{\mathrm{t}} \qquad \text{(here } \alpha_{\mathsf{Id}} \text{ is the identity action)}
\end{array}
$$

(c) semantics of rely/guarantee assertions R and G

$$p \overset{\mu}{\twoheadrightarrow} n' \ \text{ iff } \ \forall \mathcal{S}, \mathcal{A}, \eta, T, \mathcal{S}'.\ ((\mathcal{S}, \mathcal{A}, \eta) \models p) \land (\mathcal{A}, \eta \models T) \land (\mathcal{S}' = \mathrm{exec}(\mathcal{S}, T)) \implies \mu(\mathcal{S}') = n'$$

$$\text{cmt-closed}(p) \ \text{ iff } \ \forall \mathcal{S}, \mathcal{A}, \eta, i, \mathrm{t}, \alpha.\ (\mathcal{S}, \mathcal{A} \uplus \{i \rightsquigarrow (\mathrm{t}, \alpha, \mathbf{prd})\}, \eta) \models p \implies (\mathcal{S}, \mathcal{A} \uplus \{i \rightsquigarrow (\mathrm{t}, \alpha, \mathbf{cmt})\}, \eta) \models p$$

(d) auxiliary definitions used in inference rules

Figure 22. Semantics of assertions.

E Program Logic for Client Verification: Assertion Semantics and Logic Soundness Proofs

E.1 Semantics of Assertions

We define the syntax of the assertions in Fig. 10 and their semantics in Fig. 22. The action assertions p and q specify the set of actions $\mathcal{A}$ and their ordering η in the current thread's view.

Actions and their ordering. In order to distinguish actions that originate from different program points, we assign a unique ID (a natural number) to each action α. As defined in Fig. 22(a), the action set $\mathcal{A}$ maps each action ID i to a triple (t, α, d). Here t specifies the *origin* node of the request α. The flag d indicates whether α is predicted (**prd**) or committed (**cmt**) in the view of the current thread t_c. As we explained before, at some program point t_c may know that the request of α has been issued but not arrived yet, in which case we say t_c *predicts* the future receipt of α. Later when it actually receives α, we think α is *committed* in t_c's view. The action ordering η is a relation over action IDs.

As shown in Fig. 22, p and q are assertions over $(\mathcal{S}, \mathcal{A}, \eta)$. $[\alpha]^i_{\mathrm{t}}$ and $\boxed{\alpha}^i_{\mathrm{t}}$ describe singleton action sets. The former says the action α (with ID i) has been issued from its origin t, but we do not care whether it's on the way or it as been arrived at the current node, while the latter says the current node has received such an α. To simplify the presentation, we may omit the superscript action ID in an assertion when it is clear from the context what the action denotes. We write emp for an empty action set. The assertion $p \sqcup q$ allows us to merge two action sets without enforcing new ordering. For instance, $[\texttt{addAfter(a,b)}]_{\mathsf{t}_1} \sqcup \boxed{\texttt{remove(e)}}_{\mathsf{t}_2}$ says addAfter(a,b) and remove(e) can be ordered either way.

We use $p \ltimes [\alpha]^i_{\mathsf{t}}$, $p \ltimes \boxed{\alpha}^i_{\mathsf{t}}$, $(p, \bowtie) \ltimes [\alpha]^i_{\mathsf{t}}$ and $(p, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}}$ to add a new action α and some new orders about α. The assertion $p \ltimes [\alpha]^i_{\mathsf{t}}$ requires α to be ordered after all the actions in p, while $(p, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}}$ enforces the ordering between α (with action ID i and origin t) and only the actions that have been committed and conflict ($\bowtie$) with α. For instance, for the RGA object, if p is $[\texttt{addAfter(a,b)}]^1_{\mathsf{t}_1} \sqcup \boxed{\texttt{remove(e)}}^2_{\mathsf{t}_2}$, then the following $\mathcal{A}_1$ and η_1 satisfy $(p, \bowtie) \ltimes \boxed{\texttt{addAfter(a,c)}}^3_{\mathsf{t}}$ but does *not* satisfy $p \ltimes [\texttt{addAfter(a,c)}]^3_{\mathsf{t}}$:

$$\mathcal{A}_1 = \{1 \rightsquigarrow (\mathsf{t}_1, \texttt{addAfter(a,b)}, \mathbf{prd}), 2 \rightsquigarrow (\mathsf{t}_2, \texttt{remove(e)}, \mathbf{cmt}), 3 \rightsquigarrow (\mathsf{t}, \texttt{addAfter(a,c)}, \mathbf{cmt})\}, \eta_1 = \emptyset \qquad \text{(E.1)}$$

The assertion $(p, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}}$ is introduced when the current thread t calls the operation of α at the status p (see the CALL rule in Fig. 11). If α' in p is **prd**, which means t has not received α', we do not care about the orders of α' and α. If α' does not conflict with α, from nonComm$(\Gamma, \bowtie)$, we know applying α and α' in any order will have the same effects, so the ordering is still unimportant. As a result, we only enforce the orders that α is after the committed and conflicting actions of p. $(p, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}}$ also requires α be committed since the assertion is used only at the origin node of α. We define some useful shorthand at the bottom of Fig. 22(b). We have the following equivalences/implications:

$$\begin{array}{l} \boxed{\alpha}^i_{\mathsf{t}} \Leftrightarrow (\mathsf{emp} \sqcup \boxed{\alpha}^i_{\mathsf{t}}) \Leftrightarrow (\mathsf{emp} \ltimes \boxed{\alpha}^i_{\mathsf{t}}) \Leftrightarrow ((\mathsf{emp}, \bowtie) \ltimes \boxed{\alpha}^i_{\mathsf{t}}) \\ [\alpha]^i_{\mathsf{t}} \Leftrightarrow (\mathsf{emp} \sqcup [\alpha]^i_{\mathsf{t}}) \Leftrightarrow (\mathsf{emp} \ltimes [\alpha]^i_{\mathsf{t}}) \Leftrightarrow ((\mathsf{emp}, \bowtie) \ltimes [\alpha]^i_{\mathsf{t}}) \\ [\alpha]^i_{\mathsf{t}} \Rightarrow (\diamond [\alpha]^i_{\mathsf{t}}) \quad \boxed{\alpha}^i_{\mathsf{t}} \Rightarrow (\diamond \boxed{\alpha}^i_{\mathsf{t}}) \quad (p \ltimes [\alpha]^i_{\mathsf{t}}) \Rightarrow (p \sqcup [\alpha]^i_{\mathsf{t}}) \end{array} \qquad \text{(E.2)}$$

We lift Hoare-logic state assertions $\mathcal{P}$ to action assertions. $(\mathcal{S}, \mathcal{A}, \eta) \models \mathcal{P}$ requires $\mathcal{P}$ hold over any final states resulting from executing any traces T that respect $(\mathcal{A}, \eta)$. We define traces in Fig. 22(a), which are sequences of actions $(i, \mathsf{t}, \alpha, d)$. We say a trace T respects $\mathcal{A}$ and η, written as $\mathcal{A}, \eta \models T$, if $\mathcal{A}$'s actions constitute T and their orderings in η are all contained in T. Here $\lfloor T \rfloor$ turns the trace to a set of actions, and $i <_T j$ says that in T the action with ID i is before the action with ID j. As a consequence, if η contains cycles (e.g., $\eta = \{(1, 2), (2, 1)\}$), no trace T can satisfy $\mathcal{A}, \eta \models T$. (But as we will see soon, the way we add orderings by our logic would *not introduce* cycles to the action model.) When executing T from the initial state $\mathcal{S}$, written as exec$(\mathcal{S}, T)$, we only execute committed actions since predicted actions are not received in the current view. For instance, suppose in $\mathcal{S}$ the RGA sequence s is ae (i.e., $\mathcal{S}(\mathsf{s}) = \mathsf{ae}$). Then $(\mathcal{S}, \mathcal{A}_1, \eta_1) \models (\mathsf{s} = \mathsf{ac})$ holds (where $\mathcal{A}_1$ and η_1 are defined in (E.1)). Also $(\mathcal{S}, \mathcal{A}_2, \eta_2) \models (\mathsf{s} = \mathsf{abce} \vee \mathsf{s} = \mathsf{acbe})$ holds for the following $\mathcal{A}_2$ and η_2.

$$\mathcal{A}_2 = \{1 \rightsquigarrow (\mathsf{t}_1, \texttt{addAfter(a,b)}, \mathbf{cmt}), 3 \rightsquigarrow (\mathsf{t}, \texttt{addAfter(a,c)}, \mathbf{cmt})\}, \quad \eta_2 = \emptyset$$

The assertion $p \Rrightarrow q$ says, q holds after committing all the actions in p. It is used when the whole client program terminates (see the PAR rule in Fig. 11).

Rely/guarantee assertions and stability. Following rely-guarantee reasoning, R and G specify the interface between a thread and its environment. They are binary relations between a $(\mathcal{S}, \mathcal{A}, \eta)$ triple (reflecting the current status) and a newly issued action $(i', \mathsf{t}', \alpha')$. The guarantee G specifies the invocations of object actions made by the thread itself. The rely R specifies the thread's expectations of the object actions that originate from its environment. As defined in Fig. 22(c), the only primitive rely/guarantee assertion $p \rightsquigarrow [\alpha]^i_{\mathsf{t}}$ says that t invokes the action α when p holds. Usually we use it to specify the prerequisite for t to issue the request α. We define IId to represent the invocation of an identity action (e.g., read operations). It specifies stuttering steps.

Threads can cooperate if the rely condition of a thread t is implied by the guarantee of the other t′. The assertion p at each program point of t must be stable under its rely R, i.e., it is resistant to interference from the environment. We define the stability check Sta$(p, R, \bowtie)$ in Def. 39. It says that, given the current knowledge $(\mathcal{S}, \mathcal{A}, \eta)$ satisfying p, if as specified in R the prerequisite $(\mathcal{S}, \mathcal{A}', \eta')$ for the invocation of α on t′ is met (i.e., $\mathcal{A}' \subsetneq \mathcal{A}$ and $\eta' \subseteq \eta$), then p must still hold after the invocation of α, that is $(\mathcal{S}, \mathcal{A} \uplus \{i \rightsquigarrow (\mathsf{t}', \alpha, \_)\}, \eta'') \models p$. There are several details we should note.

- For an action to be invoked on certain node t′, its prerequisite actions in $\mathcal{A}'$ must have all arrived at t′, although they may not have all arrived at the current node yet. Therefore, we use $\mathcal{A}' \subsetneq \mathcal{A}$ to say that p is aware of the invocations of the prerequisite actions in $\mathcal{A}'$.
- Although p needs to hold after the invocation of α, p does not have to know whether α has arrived at the current node. Thus we use the underscore in $(\mathcal{S}, \mathcal{A} \uplus \{i \rightsquigarrow (\mathsf{t}', \alpha, \_)\}, \eta'') \models p$.
- By predicting α, p expands its knowledge about action ordering from η to η''. For those α' in $\mathcal{A}'$ that are prerequisite of α and are also in conflict with α (i.e., $\alpha' \bowtie \alpha$), α' should be ordered before α on all nodes, since we require all nodes to observe the same ordering of conflicting actions. Therefore η'' extends η with the new knowledge about the ordering.

$$\frac{\begin{array}{c}[\![E]\!]_{\mathcal{S}_c} = n \qquad \mathsf{split}(\Gamma(f,n)) = (\mu,\alpha) \\ \eta \subseteq \eta_1 \qquad \mathcal{A},\eta_1 \models T \qquad \forall T,\mathcal{S}'.\ (\mathcal{A},\eta_1 \models T) \wedge (\mathcal{S}' = \mathrm{exec}(\mathcal{S},T)) \implies \mu(\mathcal{S}') = n' \\ i \notin dom(\mathcal{A}) \qquad \mathcal{A}' = \mathcal{A} \uplus \{i \rightsquigarrow (\mathrm{t},\alpha,\mathbf{cmt})\} \\ \eta' = \eta_1 \uplus \{(j,i) \mid \exists\alpha'.\ \mathcal{A}(j) = (\_,\alpha',\mathbf{cmt}) \wedge \alpha \bowtie \alpha'\}\end{array}}{((x := f(E),\mathcal{S}_c),(\mathcal{S},\mathcal{A},\eta),(\Gamma,\bowtie)) \overset{\text{CALL}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((\mathbf{skip},\mathcal{S}_c\{x \rightsquigarrow n'\}),(\mathcal{S},\mathcal{A}',\eta'),(\Gamma,\bowtie))}\ (\text{CALL})$$

$$\frac{\begin{array}{c}((\mathcal{S},\mathcal{A}',\eta'),(i,\mathrm{t}',\alpha)) \models' R \qquad \mathcal{A}' \subsetneq \mathcal{A} \qquad \eta' \subseteq \eta \\ \mathcal{A}'' = \mathcal{A} \uplus \{i \rightsquigarrow (\mathrm{t}',\alpha,\mathbf{prd})\} \qquad \eta'' = \eta \uplus \{(j,i) \mid \exists\alpha'.\ \mathcal{A}'(j) = (\_,\alpha',\mathbf{cmt}) \wedge \alpha \bowtie \alpha'\}\end{array}}{((C,\mathcal{S}_c),(\mathcal{S},\mathcal{A},\eta),(\Gamma,\bowtie)) \overset{\text{PRD}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((C,\mathcal{S}_c),(\mathcal{S},\mathcal{A}'',\eta''),(\Gamma,\bowtie))}\ (\text{PRD-}R)$$

$$\frac{\mathcal{A}(i) = (\mathrm{t}',\alpha,\mathbf{prd}) \qquad \mathcal{A}' = \mathcal{A}\{i \rightsquigarrow (\mathrm{t}',\alpha,\mathbf{cmt})\} \qquad \neg(R \Rightarrow \mathsf{Emp})}{((C,\mathcal{S}_c),(\mathcal{S},\mathcal{A},\eta),(\Gamma,\bowtie)) \overset{\text{CMT}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((C,\mathcal{S}_c),(\mathcal{S},\mathcal{A}',\eta),(\Gamma,\bowtie))}\ (\text{CMT})$$

$$\frac{[\![E]\!]_{\mathcal{S}_c} = n}{((x := E,\mathcal{S}_c),(\mathcal{S},\mathcal{A},\eta),(\Gamma,\bowtie)) \overset{\text{LC}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((\mathbf{skip},\mathcal{S}_c\{x \rightsquigarrow n\}),(\mathcal{S},\mathcal{A},\eta),(\Gamma,\bowtie))}\ (\text{LOCAL})$$

Figure 23. Local R-semantics for clients.

Definition 39. $\mathsf{Sta}(p,R,\bowtie)$ iff

$$(\mathcal{S},\mathcal{A},\eta) \models p \wedge \forall\mathcal{A}',\eta',(i,\mathrm{t}',\alpha).\ (\mathcal{A}' \subsetneq \mathcal{A}) \wedge (\eta' \subseteq \eta) \wedge ((\mathcal{S},\mathcal{A}',\eta'),(i,\mathrm{t}',\alpha)) \models' R$$
$$\implies (\mathcal{S},\mathcal{A} \uplus \{i \rightsquigarrow (\mathrm{t}',\alpha,\_)\},\eta'') \models p$$

where $\eta'' = \eta \uplus \{(j,i) \mid \exists\alpha'.\ \mathcal{A}'(j) = (\_,\alpha',\mathbf{cmt}) \wedge \alpha \bowtie \alpha'\}$
and $\mathcal{A}' \subsetneq \mathcal{A}$ iff $\forall i,\mathrm{t},\alpha.\ \mathcal{A}'(i) = (\mathrm{t},\alpha,\mathbf{cmt}) \implies \mathcal{A}(i) = (\mathrm{t},\alpha,\_)$
and $((\mathcal{S},\mathcal{A}',\eta'),(i,\mathrm{t}',\alpha)) \models' R$ iff $\forall T.\ (\mathcal{A}',\eta' \models T) \implies \exists\eta_0.\ (\mathcal{A}',\eta_0 \models T) \wedge (\eta' \subseteq \eta_0) \wedge ((\mathcal{S},\mathcal{A}',\eta_0),(i,\mathrm{t}',\alpha)) \models R$

E.2 Judgment Semantics and Soundness Theorems

In this section, for any program $\mathbb{P} = \mathbf{with}\ (\Gamma,\bowtie)\ \mathbf{do}\ C_1 \parallel \ldots \parallel C_n$, we assume $\forall i.\ fv(\Gamma) \cap fv(C_i) = \emptyset \wedge \forall j \neq i.\ fv(C_i) \cap fv(C_j) = \emptyset$. We also assume that Γ has strong locality, i.e., $\mathsf{SLocality}(\Gamma)$ (which is useful in proving soundness of the PAR rule).

Definition 40 (Strong Locality). $\mathsf{SLocality}(\Gamma)$ iff both the following holds:

1. for any $f, n, n', \mathcal{S}, \mathcal{S}'$ and $\mathcal{S}_1$, if $\Gamma(f,n)(\mathcal{S}) = (n',\mathcal{S}')$ and $dom(\mathcal{S}) \cap dom(\mathcal{S}_1) = \emptyset$, then $\Gamma(f,n)(\mathcal{S} \uplus \mathcal{S}_1) = (n',\mathcal{S}' \uplus \mathcal{S}_1)$.
2. for any f, n, n', $\mathcal{S}$, $\mathcal{S}''$ and $\mathcal{S}_1$, if $\Gamma(f,n)(\mathcal{S} \uplus \mathcal{S}_1) = (n',\mathcal{S}'')$ and $fv(\Gamma) \subseteq dom(\mathcal{S})$, then there exists $\mathcal{S}'$ such that $\mathcal{S}'' = \mathcal{S}' \uplus \mathcal{S}_1$ and $\Gamma(f,n)(\mathcal{S}) = (n',\mathcal{S}')$.

Definition 41 define the semantics for the local judgment $R,G;\Gamma,\bowtie\ \vdash_{\mathrm{t}} \{p\}C\{q\}$, following standard rely-guarantee. We define the local R-semantics in Fig. 23.

Definition 41 (Local Judgment Semantics). $R,G;\Gamma,\bowtie\ \models_{\mathrm{t}} \{p\}C\{q\}$ iff
for any $\mathcal{S}$, $\mathcal{A}$, η and $\mathcal{S}_c$, if $(\mathcal{S} \uplus \mathcal{S}_c,\mathcal{A},\eta) \models p$, $fv(\Gamma) \subseteq dom(\mathcal{S})$ and $fv(C) \subseteq dom(\mathcal{S}_c)$, then the following are true:

1. for any $\mathcal{A}'$, η' and $\mathcal{S}_c'$, if $((C,\mathcal{S}_c),(\mathcal{S},\mathcal{A},\eta),(\Gamma,\bowtie)) \overset{R}{\hookrightarrow}{}^{*}_{\mathrm{t}} ((\mathbf{skip},\mathcal{S}_c'),(\mathcal{S},\mathcal{A}',\eta'),(\Gamma,\bowtie))$, then $\forall T'.\ (\mathcal{A}',\eta' \models T') \implies \exists\eta''.\ (\mathcal{A}',\eta'' \models T') \wedge (\eta' \subseteq \eta'') \wedge (\mathcal{S}_c' \uplus \mathcal{S},\mathcal{A}',\eta'') \models q$.
2. for any n, $((C,\mathcal{S}_c),(\mathcal{S},\mathcal{A},\eta),(\Gamma,\bowtie),R)$ $\mathsf{guarantees}^n_{\mathrm{t}}$ G.

Definition 42 (Guarantees).

- $((C,\mathcal{S}_c),(\mathcal{S},\mathcal{A},\eta),(\Gamma,\bowtie),R)$ $\mathsf{guarantees}^0_{\mathrm{t}}$ G always holds.
- $((C,\mathcal{S}_c),(\mathcal{S},\mathcal{A},\eta),(\Gamma,\bowtie),R)$ $\mathsf{guarantees}^{n+1}_{\mathrm{t}}$ G iff

  for any C', $\mathcal{S}_c'$, $\mathcal{A}'$, η', if $((C,\mathcal{S}_c),(\mathcal{S},\mathcal{A},\eta),(\Gamma,\bowtie)) \overset{\widehat{\mathbb{I}}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((C',\mathcal{S}_c'),(\mathcal{S},\mathcal{A}',\eta'),(\Gamma,\bowtie))$, then

  1. $((C',\mathcal{S}_c'),(\mathcal{S},\mathcal{A}',\eta'),(\Gamma,\bowtie),R)$ $\mathsf{guarantees}^n_{\mathrm{t}}$ G; and

$$\begin{tikzcd}
\text{Proof by induction} \arrow[d, Rightarrow] & \\
\text{Theorem 44(1): If } R, G; \Gamma, \bowtie \vdash_{\mathrm{t}} \{p\}C\{q\}, \text{ then } R, G; \Gamma, \bowtie \models_{\mathrm{t}} \{p\}C\{q\}. \arrow[d, Rightarrow, "\textsc{par}\text{-sound}"] & \\
\text{Lemma 48: If } \vdash \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}, \text{ then } \models^{\mathsf{prd}} \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}. \arrow[d, Rightarrow] & \circ\!\!\longrightarrow \text{ implies } \longmapsto . \arrow[dl, Rightarrow] \\
\text{Theorem 44(2): If } \vdash \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\} \text{ and } \mathrm{nonComm}(\Gamma, \bowtie), \text{ then } \models \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}. &
\end{tikzcd}$$

Figure 24. Logic soundness proof.

2. if $\widehat{\mathbb{I}} = \textsc{call}$, then
   there exist i, α such that $\mathcal{A}' = \mathcal{A} \uplus \{i \rightsquigarrow (\mathrm{t}, \alpha, \mathbf{cmt})\}$, and
   $\forall T.\ (\mathcal{A}, \eta \models T) \implies \exists \eta_0.\ (\mathcal{A}, \eta_0 \models T) \wedge (\eta \subseteq \eta_0) \wedge ((\mathcal{S}, \mathcal{A}, \eta_0), (i, \mathrm{t}, \alpha)) \models G$.

Definition 43 defines the semantics for the global judgment $\vdash \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}$ based on the the abstract semantics in Figure 17.

Definition 43 (Abstract Global Judgment Semantics). $\models \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}$ iff
for any $\mathcal{S}$, $\mathbb{O}$ and $\mathcal{S}_1, \ldots, \mathcal{S}_n, \mathcal{S}'_1, \ldots, \mathcal{S}'_n$, if $\mathcal{S} \models_{\textsc{hoare}} \mathcal{P}$ and $(\mathbb{P}, \mathcal{S}) \overset{\mathbb{O}}{\circ\!\!\longrightarrow}{}^{*} (\mathbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}'_1, \ldots, \mathcal{S}'_n))$, then $\mathcal{S}'_1 = \ldots = \mathcal{S}'_n$ and $(\mathcal{S}_1 \uplus \ldots \uplus \mathcal{S}_n \uplus \mathcal{S}'_1) \models_{\textsc{hoare}} \mathcal{Q}$.

Theorem 44 (Logic Soundness).
1. If $R, G; \Gamma, \bowtie \vdash_{\mathrm{t}} \{p\}C\{q\}$, then $R, G; \Gamma, \bowtie \models_{\mathrm{t}} \{p\}C\{q\}$.
2. If $\vdash \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}$ and $\mathrm{nonComm}(\Gamma, \bowtie)$, then $\models \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}$.

Finally, our logic plus contextual refinement (or ACC, or its proof method) ensure correctness of the whole system.

Definition 45 (Concrete Global Judgment Semantics). $\models_{\varphi} \{\mathcal{P}\}P\{\mathcal{Q}\}$ iff
for any $\mathcal{S}$, $\mathcal{E}$ and $\mathcal{S}_1, \ldots, \mathcal{S}_n, \mathcal{S}'_1, \ldots, \mathcal{S}'_n$, if $\mathcal{S} \models_{\textsc{hoare}} \mathcal{P}$ and $(P, \mathcal{S}) \overset{\mathcal{E}}{\longmapsto}{}^{*} (\mathbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}'_1, \ldots, \mathcal{S}'_n))$, then $\varphi(\mathcal{S}'_1) = \ldots = \varphi(\mathcal{S}'_n)$ and $(\mathcal{S}_1 \uplus \ldots \uplus \mathcal{S}_n \uplus \mathcal{S}'_1) \models_{\textsc{hoare}} \mathcal{Q}$.

Theorem 46 (Whole System Correctness).
If $\vdash \{\mathcal{P}\}\mathbf{with}\ (\Gamma, \bowtie)\ \mathbf{do}\ C_1 \| \ldots \| C_n\{\mathcal{Q}\}$, $\Pi \sqsubseteq_{\varphi} (\Gamma, \bowtie)$, $\mathrm{nonComm}(\Gamma, \bowtie)$, $\mathcal{P}' \Rightarrow \varphi^{-1}[\mathcal{P}]$ and $fv(\mathcal{Q}) \subseteq \bigcup_{\mathrm{t}} fv(C_{\mathrm{t}})$, then $\models_{\varphi} \{\mathcal{P}'\}\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \| \ldots \| C_n\{\mathcal{Q}\}$.

E.3 Proof Structure

Theorem 44 is proved in the way in Figure 24. We introduce an intermediate global semantics in Figure 25, and define the corresponding global judgment semantics in Definition 47. We prove Lemma 48 as the direct soundness for our logic rules, from which we derive Theorem 44(2) and (3).

Definition 47 (Predict-Commit Global Judgment Semantics). $\models^{\mathsf{prd}} \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}$ iff
for any $\mathcal{S}, \mathcal{S}_1, \ldots, \mathcal{S}_n, \mathcal{S}_0, \mathcal{A}', \eta'$, if
- $\mathcal{S} \models_{\textsc{hoare}} \mathcal{P}$, and
- $(\mathbb{P}, \mathcal{S}) \longmapsto^{*} (\mathbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}_0, \mathcal{A}', \eta'))$,

then $\forall \mathcal{S}'_0.\ (\exists T.\ (\mathcal{A}', \eta' \models T) \wedge \mathrm{exec}(\mathcal{S}_0, T) = \mathcal{S}'_0) \implies (\mathcal{S}_1 \uplus \ldots \uplus \mathcal{S}_n \uplus \mathcal{S}'_0) \models_{\textsc{hoare}} \mathcal{Q}$.

Lemma 48. If $\vdash \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}$, then $\models^{\mathsf{prd}} \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}$.

Proof of Lemma 48. From the derivation of $\vdash \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}$, and from Theorem 44(1) and Lemma 56, we are done. □

Proof of Theorem 44(1). By induction over the derivation of $R, G; \Gamma, \bowtie \vdash_{\mathrm{t}} \{p\}C\{q\}$, and from Lemma 49, Lemma 50, and Lemma 55. □

$$
\begin{array}{llll}
(\mathit{ActIDSet}) & \mathit{ms} & \in & \mathscr{P}(\mathit{Nat}) \\
(\mathit{AbsTrState}) & \Omega & ::= & \{\mathrm{t}_1 \leadsto (\mathcal{S}, \mathcal{A}_1, \eta_1, \mathit{ms}_1), \ldots, \mathrm{t}_n \leadsto (\mathcal{S}, \mathcal{A}_n, \eta_n, \mathit{ms}_n)\}
\end{array}
$$

$$
\frac{\mathit{fv}(\Gamma) \subseteq \mathit{dom}(\mathcal{S}) \qquad \forall \mathrm{t} \in [1..n].\ \sigma_c(\mathrm{t}) = (C_\mathrm{t}, \emptyset) \qquad \forall \mathrm{t} \in [1..n].\ \Omega(\mathrm{t}) = (\mathcal{S}, \emptyset, \emptyset, \emptyset)}{(\textbf{with } (\Gamma, \bowtie) \textbf{ do } C_1 \parallel \ldots \parallel C_n, \mathcal{S}) \longmapsto (\sigma_c, \Omega, (\Gamma, \bowtie))}\ (\textsc{load})
$$

$$
\frac{\begin{array}{c} \mathit{dom}(\sigma_c) = [1..n] \qquad \forall \mathrm{t}.\ \sigma_c(\mathrm{t}) = (\textbf{skip}, \mathcal{S}_\mathrm{t}) \\ \forall \mathrm{t}.\ \Omega(\mathrm{t}) = (\mathcal{S}, \mathcal{A}, \eta, \mathit{ms}) \qquad \forall i.\ \mathcal{A}(i) = (\_, \_, \textbf{cmt}) \end{array}}{(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto (\textbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}, \mathcal{A}, \eta))}\ (\textsc{end})
$$

$$
\frac{\begin{array}{c} \sigma_c(\mathrm{t}) = (x := f(E), \mathcal{S}_c) \qquad [\![E]\!]_{\mathcal{S}_c} = n \qquad \mathsf{split}(\Gamma(f, n)) = (\mu, \alpha) \\ \Omega(\mathrm{t}) = (\mathcal{S}, \mathcal{A}, \eta, \mathit{ms}) \qquad \eta \subseteq \eta_1 \qquad \mathcal{A}, \eta_1 \models T \qquad \forall T, \mathcal{S}'.\ (\mathcal{A}, \eta_1 \models T) \wedge (\mathcal{S}' = \mathrm{exec}(\mathcal{S}, T)) \implies \mu(\mathcal{S}') = n' \\ \Omega'(\mathrm{t}) = (\mathcal{S}, \mathcal{A}', \eta', \mathit{ms} \cup \mathit{ms}'') \qquad i \notin \mathit{dom}(\mathcal{A}) \qquad \mathcal{A}' = \mathcal{A} \uplus \{i \leadsto (\mathrm{t}, \alpha, \textbf{cmt})\} \\ \mathit{ms}'' = \{j \mid \exists \alpha'.\ \mathcal{A}(j) = (\_, \alpha', \textbf{cmt}) \wedge \alpha \bowtie \alpha'\} \qquad \eta' = \eta_1 \uplus \{(j, i) \mid j \in \mathit{ms}''\} \\ \forall \mathrm{t}' \neq \mathrm{t}: \quad \Omega(\mathrm{t}') = (\mathcal{S}, \mathcal{A}_{\mathrm{t}'}, \eta_{\mathrm{t}'}, \mathit{ms}) \qquad \mathcal{A}'_{\mathrm{t}'} = \mathcal{A}_{\mathrm{t}'} \uplus \{i \leadsto (\mathrm{t}, \alpha, \textbf{prd})\} \\ \Omega'(\mathrm{t}') = (\mathcal{S}, \mathcal{A}'_{\mathrm{t}'}, \eta'_{\mathrm{t}'}, \mathit{ms} \cup \mathit{ms}'') \qquad \eta'_{\mathrm{t}'} = \eta_{\mathrm{t}'} \uplus \{(j, i) \mid j \in \mathit{ms}''\} \end{array}}{(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto (\sigma_c\{\mathrm{t} \leadsto (\textbf{skip}, \mathcal{S}_c\{x \leadsto n'\})\}, \Omega', (\Gamma, \bowtie))}\ (\textsc{call\&prd})
$$

$$
\frac{\Omega(\mathrm{t}) = (\mathcal{S}, \mathcal{A}, \eta, \mathit{ms}) \qquad \mathcal{A}(i) = (\mathrm{t}', \alpha, \textbf{prd}) \qquad \mathcal{A}' = \mathcal{A}\{i \leadsto (\mathrm{t}', \alpha, \textbf{cmt})\}}{(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto (\sigma_c, \Omega\{\mathrm{t} \leadsto (\mathcal{S}, \mathcal{A}', \eta, \mathit{ms})\}, (\Gamma, \bowtie))}\ (\textsc{cmt})
$$

$$
\frac{\sigma_c(\mathrm{t}) = (x := E, \mathcal{S}_c) \qquad [\![E]\!]_{\mathcal{S}_c} = n \qquad C' = \textbf{skip} \qquad \mathcal{S}'_c = \mathcal{S}_c\{x \leadsto n\}}{(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto (\sigma_c\{\mathrm{t} \leadsto (C', \mathcal{S}'_c)\}, \Omega, (\Gamma, \bowtie))}\ (\textsc{local})
$$

Figure 25. Predict-commit semantics for clients.

E.4 Soundness Proofs for Local Rules and PAR Rule

Lemma 49 (CALL-sound). If

1. $p \Rightarrow E = n$, $\mathsf{split}(\Gamma(f, n)) = (\mu, \alpha)$, $p \overset{\mu}{\twoheadrightarrow} n'$, $x = n' \wedge \exists v.\ p[v/x] \Rightarrow q$,
2. $q \leadsto [\alpha]^i_\mathrm{t} \Rightarrow G$, $\mathit{fv}(G) \subseteq \mathit{fv}(\Gamma)$,

then $\mathsf{Emp}, G; \Gamma, \bowtie \models_\mathrm{t} \{p\} x := f(E) \{(q, \bowtie) \ltimes \boxed{\alpha}^i_\mathrm{t}\}$.

Proof. For any $\mathcal{S}$, $\mathcal{A}$, η and $\mathcal{S}_c$, if $(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}, \eta) \models p$, $\mathit{fv}(\Gamma) \subseteq \mathit{dom}(\mathcal{S})$ and $\{x\} \cup \mathit{fv}(E) \subseteq \mathit{dom}(\mathcal{S}_c)$, we want to prove the following:

(1) for any n, if $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\mathsf{Emp}}{\hookrightarrow}{}^n_\mathrm{t} ((\textbf{skip}, \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, then $\forall T'.\ (\mathcal{A}', \eta' \models T') \implies \exists \eta''.\ (\mathcal{A}', \eta'' \models T') \wedge (\eta' \subseteq \eta'') \wedge (\mathcal{S}'_c \uplus \mathcal{S}, \mathcal{A}', \eta'') \models (q, \bowtie) \ltimes \boxed{\alpha}^i_\mathrm{t}$.

(2) for any n, $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), \mathsf{Emp})$ $\mathsf{guarantees}^n_\mathrm{t}$ G.

Since $(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}, \eta) \models p$, $\mathit{fv}(\Gamma) \subseteq \mathit{dom}(\mathcal{S})$ and $\{x\} \cup \mathit{fv}(E) \subseteq \mathit{dom}(\mathcal{S}_c)$, and since $p \Rightarrow E = n$, we know

$$[\![E]\!]_{\mathcal{S}_c} = n.$$

From $p \overset{\mu}{\twoheadrightarrow} n'$, we know

$$\forall T, \mathcal{S}''.\ (\mathcal{A}, \eta \models T) \wedge (\mathcal{S}'' = \mathrm{exec}(\mathcal{S} \uplus \mathcal{S}_c, T)) \implies \mu(\mathcal{S}'') = n'$$

From $\mathit{fv}(\Gamma) \subseteq \mathit{dom}(\mathcal{S})$ and $\mathsf{SLocality}(\Gamma)$, we know

$$\forall T, \mathcal{S}'.\ (\mathcal{A}, \eta \models T) \wedge (\mathcal{S}' = \mathrm{exec}(\mathcal{S}, T)) \implies \mu(\mathcal{S}') = n'$$

Since $(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}, \eta) \models p$, we know

$$(\mathcal{S} \uplus \mathcal{S}_c\{x \rightsquigarrow n'\}, \mathcal{A}, \eta) \models x = n' \land \exists v.\, p[v/x]$$

From $x = n' \land \exists v.\, p[v/x] \Rightarrow q$, we know

$$(\mathcal{S} \uplus \mathcal{S}_c\{x \rightsquigarrow n'\}, \mathcal{A}, \eta) \models q$$

Let $\mathcal{A}' = \mathcal{A} \uplus \{i \rightsquigarrow (\mathrm{t}, \alpha, \mathbf{cmt})\}$ and $\eta' = \eta \uplus \{(j, i) \mid \exists \alpha'.\, \mathcal{A}(j) = (\_, \alpha', \mathbf{cmt}) \land \alpha \bowtie \alpha'\}$. Then

$$(\mathcal{S} \uplus \mathcal{S}_c\{x \rightsquigarrow n'\}, \mathcal{A}', \eta') \models (q, \bowtie) \ltimes \boxed{\alpha}_{\mathrm{t}}^{i}$$

Also we know

$$((\mathcal{S} \uplus \mathcal{S}_c\{x \rightsquigarrow n'\}, \mathcal{A}, \eta), (i, \mathrm{t}, \alpha)) \models q \rightsquigarrow [\alpha]_{\mathrm{t}}^{i}$$

Since $q \rightsquigarrow [\alpha]_{\mathrm{t}}^{i} \Rightarrow G$, we know

$$((\mathcal{S} \uplus \mathcal{S}_c\{x \rightsquigarrow n'\}, \mathcal{A}, \eta), (i, \mathrm{t}, \alpha)) \models G$$

Since $fv(G) \subseteq fv(\Gamma)$ and $fv(\Gamma) \subseteq dom(\mathcal{S})$, we know

$$((\mathcal{S}, \mathcal{A}, \eta), (i, \mathrm{t}, \alpha)) \models G$$

1. For any $\mathcal{A}', \eta', \mathcal{S}_c'$, if $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow{\mathsf{Emp}}{}_{\mathrm{t}}^{*} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, from the semantics, we know
$$((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow[R]{\text{CALL}}{}_{\mathrm{t}} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie)),$$
Thus $(\mathcal{S} \uplus \mathcal{S}_c', \mathcal{A}', \eta') \models (q, \bowtie) \ltimes \boxed{\alpha}_{\mathrm{t}}^{i}$.
2. Also we know for any n, $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), \mathsf{Emp})$ $\mathsf{guarantees}_{\mathrm{t}}^{n}$ G.

Thus we are done. □

Lemma 50 (CALL-R-sound). If

1. $\mathsf{Emp}, G; \Gamma, \bowtie \models_{\mathrm{t}} \{p\} x := f(E) \{q\}$,
2. $\mathsf{Sta}(\{p, q\}, R, \bowtie)$,
3. $\text{cmt-closed}(\{p, q\})$,
4. $fv(R) \subseteq fv(\Gamma)$,

then $R, G; \Gamma, \bowtie \models_{\mathrm{t}} \{p\} x := f(E) \{q\}$.

Proof. For any $\mathcal{S}$, $\mathcal{A}$, η and $\mathcal{S}_c$, if $(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}, \eta) \models p$, $fv(\Gamma) \subseteq dom(\mathcal{S})$ and $\{x\} \cup fv(E) \subseteq dom(\mathcal{S}_c)$, we want to prove the following:

(1) for any n, if $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow{R}{}_{\mathrm{t}}^{n} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, then $\forall T'.\, (\mathcal{A}', \eta' \models T') \implies \exists \eta''.\, (\mathcal{A}', \eta'' \models T') \land (\eta' \subseteq \eta'') \land (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta'') \models q$.
(2) for any n, $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), R)$ $\mathsf{guarantees}_{\mathrm{t}}^{n}$ G.

From $\mathsf{Emp}, G; \Gamma, \bowtie \models_{\mathrm{t}} \{p\} x := f(E) \{q\}$, we know

(a) for any $\mathcal{A}', \eta', \mathcal{S}_c'$, if $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow{\mathsf{Emp}}{}_{\mathrm{t}}^{*} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, then $\forall T'.\, (\mathcal{A}', \eta' \models T') \implies \exists \eta''.\, (\mathcal{A}', \eta'' \models T') \land (\eta' \subseteq \eta'') \land (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta'') \models q$.
(b) for any n, $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), \mathsf{Emp})$ $\mathsf{guarantees}_{\mathrm{t}}^{n}$ G.

For (1), by induction over n. The base case $n = 0$ is trivial. Suppose $n = k + 1$.

Since $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow{R}{}_{\mathrm{t}}^{n} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, we have three cases:

- there exist $\mathcal{A}''$ and η'' such that
  $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow[R]{\text{CALL}}{}_{\mathrm{t}} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie))$ and
  $((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie)) \xhookrightarrow{R}{}_{\mathrm{t}}^{k} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$.
  From (a), we know
  $$\forall T''.\, (\mathcal{A}'', \eta'' \models T'') \implies \exists \eta_1''.\, (\mathcal{A}'', \eta_1'' \models T'') \land (\eta'' \subseteq \eta_1'') \land (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}'', \eta_1'') \models q.$$
  Since $\mathsf{Sta}(q, R, \bowtie)$ and $\text{cmt-closed}(q)$ and $fv(R) \subseteq fv(\Gamma)$, we know
  $$\forall T'.\, (\mathcal{A}', \eta' \models T') \implies \exists \eta_1'.\, (\mathcal{A}', \eta_1' \models T') \land (\eta' \subseteq \eta_1') \land (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta_1') \models q.$$

- there exist $\mathcal{A}''$ and η'' such that
  $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow[R]{\text{PRD}}_{\text{t}} ((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie))$ and
  $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie)) \xhookrightarrow{R}{}^{k}_{\text{t}} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$.
  Since $\text{Sta}(p, R, \bowtie)$, we know
  $$(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}'', \eta'') \models p.$$
  By the induction hypothesis, we are done.
- there exist $\mathcal{A}''$ and η'' such that
  $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow[R]{\text{CMT}}_{\text{t}} ((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie))$ and
  $((x := f(E), \mathcal{S}_c), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie)) \xhookrightarrow{R}{}^{k}_{\text{t}} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$.
  Since cmt-closed(p), we know
  $$(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}'', \eta'') \models p.$$
  By the induction hypothesis, we are done.

For (2), by induction over n and from (b). Thus we are done. □

Lemma 51 (CSQ-sound). If

1. $R', G'; \Gamma, \bowtie \models_{\text{t}} \{p'\}C\{q'\}$,
2. $p \Rightarrow p'$, $R \Rightarrow R'$, $q' \Rightarrow q$, $G' \Rightarrow G$,

then $R, G; \Gamma, \bowtie \models_{\text{t}} \{p\}C\{q\}$.

Proof. For any $\mathcal{S}$, $\mathcal{A}$, η and $\mathcal{S}_c$, if $(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}, \eta) \models p$, $fv(\Gamma) \subseteq dom(\mathcal{S})$ and $fv(C) \subseteq dom(\mathcal{S}_c)$, we want to prove the following:

(1) for any $\mathcal{A}'$, η', $\mathcal{S}_c'$, if $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow{R}{}^{*}_{\text{t}} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, then
$\forall T'. (\mathcal{A}', \eta' \models T') \implies \exists \eta''. (\mathcal{A}', \eta'' \models T') \wedge (\eta' \subseteq \eta'') \wedge (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta'') \models q$.
(2) for any n, $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), R)$ guarantees$^{n}_{\text{t}}$ G.

For (1), from the operational semantics, we know

$$dom(\mathcal{A}) \subseteq dom(\mathcal{A}'),\ \eta \subseteq \eta'$$
$$\forall i, \text{t}, \alpha, d.\ \mathcal{A}(i) = (\text{t}, \alpha, d) \implies \exists d'.\ \mathcal{A}'(i) = (\text{t}, \alpha, d') \wedge (d = \mathbf{cmt} \implies d' = \mathbf{cmt})$$

Since $\mathcal{A}', \eta' \models T'$, let $T = \text{proj}(T', \mathcal{A})$, then we know $\mathcal{A}, \eta \models T$. Here we define

$$\text{proj}(T, \mathcal{A}) \overset{\text{def}}{=} \begin{cases} \epsilon & \text{if } T = \epsilon \\ (i, \text{t}, \alpha, d') :: T' & \text{if } T = (i, \text{t}, \alpha, d) :: T' \wedge \mathcal{A}(i) = (\text{t}, \alpha, d') \\ T' & \text{if } T = (i, \text{t}, \alpha, d) :: T' \wedge i \notin dom(\mathcal{A}) \end{cases}$$

Since $p \Rightarrow p'$, we know there exists η_1 such that

$$\mathcal{A}, \eta_1 \models T,\ \eta \subseteq \eta_1,\ (\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}, \eta_1) \models p'.$$

From $R', G'; \Gamma, \bowtie \models_{\text{t}} \{p'\}C\{q'\}$, we know

(a) for any $\mathcal{A}'$, η_1', $\mathcal{S}_c'$, if $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie)) \xhookrightarrow{R'}{}^{*}_{\text{t}} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta_1'), (\Gamma, \bowtie))$, then
$\forall T'. (\mathcal{A}', \eta_1' \models T') \implies \exists \eta_1''. (\mathcal{A}', \eta_1'' \models T') \wedge (\eta_1' \subseteq \eta_1'') \wedge (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta_1'') \models q'$.

Since $R \Rightarrow R'$ and $q' \Rightarrow q$, from (a), we know

(b) for any $\mathcal{A}'$, η_1', $\mathcal{S}_c'$, if $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie)) \xhookrightarrow{R}{}^{*}_{\text{t}} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta_1'), (\Gamma, \bowtie))$, then
$\forall T'. (\mathcal{A}', \eta_1' \models T') \implies \exists \eta_1''. (\mathcal{A}', \eta_1'' \models T') \wedge (\eta_1' \subseteq \eta_1'') \wedge (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta_1'') \models q$.

Suppose $\eta_1 = \eta \uplus \eta_2$. Since $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow{R}{}^{*}_{\text{t}} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, by Lemma 53, we know

$$((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie)) \xhookrightarrow{R}{}^{*}_{\text{t}} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta' \cup \eta_2), (\Gamma, \bowtie)) \quad \text{and} \quad \mathcal{A}', \eta' \cup \eta_2 \models T'.$$

From (c), we know $\exists \eta''. (\mathcal{A}', \eta'' \models T') \wedge (\eta' \subseteq \eta'') \wedge (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta'') \models q$. So (1) is done.

For (2), from $(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}, \eta) \models p$, $p \Rightarrow p'$ and $R', G'; \Gamma, \bowtie \models_{\text{t}} \{p'\}C\{q'\}$, we know

$$\forall T. (\mathcal{A}, \eta \models T) \implies \exists \eta_1. (\mathcal{A}, \eta_1 \models T) \wedge (\eta \subseteq \eta_1) \wedge \forall n. ((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie), R') \text{ guarantees}^{n}_{\text{t}}\ G'$$

Since $R \Rightarrow R'$ and $G' \Rightarrow G$, we know

$$\forall T.\ (\mathcal{A}, \eta \models T) \implies \exists \eta_1.\ (\mathcal{A}, \eta_1 \models T) \wedge (\eta \subseteq \eta_1) \wedge \forall n.\ ((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^n_{\mathsf{t}}\ G$$

By Lemma 52, we know

$$\forall n.\ ((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^n_{\mathsf{t}}\ G.$$

Thus we are done. □

Lemma 52. For all n, if $\forall T.\ (\mathcal{A}, \eta \models T) \implies \exists \eta_1.\ (\mathcal{A}, \eta_1 \models T) \wedge (\eta \subseteq \eta_1) \wedge ((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^n_{\mathsf{t}}\ G$, then $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^n_{\mathsf{t}}\ G$.

Proof. By induction over n.

- $n = 0$. Trivial.
- $n = k + 1$. We want to prove: for any $C', \mathcal{S}'_c, \mathcal{A}', \eta'$, if $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{l}}}{\underset{R}{\hookrightarrow}}_{\mathsf{t}} ((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, then
  (1) $((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^k_{\mathsf{t}}\ G$; and
  (2) if $\widehat{\mathbb{l}} = \textsc{call}$, then there exist i, α such that $\mathcal{A}' = \mathcal{A} \uplus \{i \rightsquigarrow (\mathsf{t}, \alpha, \mathbf{cmt})\}$, and
  $\forall T.\ (\mathcal{A}, \eta \models T) \implies \exists \eta_0.\ (\mathcal{A}, \eta_0 \models T) \wedge (\eta \subseteq \eta_0) \wedge ((\mathcal{S}, \mathcal{A}, \eta_0), (i, \mathsf{t}, \alpha)) \models G$.
  - For (1). For any T' such that $\mathcal{A}', \eta' \models T'$, let $T = \mathsf{proj}(T', \mathcal{A})$, then we know $\mathcal{A}, \eta \models T$. By the premise, we know there exists η_1 such that
  $$(\mathcal{A}, \eta_1 \models T) \wedge (\eta \subseteq \eta_1) \wedge ((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^n_{\mathsf{t}}\ G.$$
  Suppose $\eta_1 = \eta \uplus \eta_2$. Since $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{l}}}{\underset{R}{\hookrightarrow}}_{\mathsf{t}} ((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, by Lemma 53, we know
  $$((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{l}}}{\underset{R}{\hookrightarrow}}_{\mathsf{t}} ((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta' \cup \eta_2), (\Gamma, \bowtie)) \quad \text{and} \quad \mathcal{A}', \eta' \cup \eta_2 \models T'.$$
  Let $\eta'_1 = \eta' \cup \eta_2$. From $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^n_{\mathsf{t}}\ G$, we know
  (a) $((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta'_1), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^k_{\mathsf{t}}\ G$.
  So we have proved $\forall T'.\ (\mathcal{A}', \eta' \models T') \implies \exists \eta'_1.\ (\mathcal{A}', \eta'_1 \models T') \wedge (\eta' \subseteq \eta'_1) \wedge ((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta'_1), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^k_{\mathsf{t}}\ G$.
  By the induction hypothesis, we know
  $$((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^k_{\mathsf{t}}\ G.$$
  So (1) holds.
  - For (2). Suppose $\widehat{\mathbb{l}} = \textsc{call}$. If $\mathcal{A}, \eta \models T$, from the premise, we know there exists η_1 such that
  $$(\mathcal{A}, \eta_1 \models T) \wedge (\eta \subseteq \eta_1) \wedge ((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^n_{\mathsf{t}}\ G.$$
  Suppose $\eta_1 = \eta \uplus \eta_2$. Since $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{l}}}{\underset{R}{\hookrightarrow}}_{\mathsf{t}} ((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, by the operational semantics, we know
  $$((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{l}}}{\underset{R}{\hookrightarrow}}_{\mathsf{t}} ((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta' \cup \eta_2), (\Gamma, \bowtie)).$$
  Let $\eta'_1 = \eta' \cup \eta_2$. From $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie), R)\ \mathsf{guarantees}^n_{\mathsf{t}}\ G$, we know
  (b) if $\widehat{\mathbb{l}} = \textsc{call}$, then there exist i, α such that $\mathcal{A}' = \mathcal{A} \uplus \{i \rightsquigarrow (\mathsf{t}, \alpha, \mathbf{cmt})\}$, and
  $\forall T.\ (\mathcal{A}, \eta_1 \models T) \implies \exists \eta_0.\ (\mathcal{A}, \eta_0 \models T) \wedge (\eta_1 \subseteq \eta_0) \wedge ((\mathcal{S}, \mathcal{A}, \eta_0), (i, \mathsf{t}, \alpha)) \models G$.
  From (b), since $\mathcal{A}, \eta_1 \models T$ and $\eta \subseteq \eta_1$, we know (2) holds.

Thus we are done. □

Lemma 53. If

1. $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{l}}}{\underset{R}{\hookrightarrow}}_{\mathsf{t}} ((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$,
2. $\eta_1 = \eta \uplus \eta_2$, $(\mathcal{A}', \eta' \models T')$, $T = \mathsf{proj}(T', \mathcal{A})$, $(\mathcal{A}, \eta \models T)$, $(\mathcal{A}, \eta_1 \models T)$,

then $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{l}}}{\underset{R}{\hookrightarrow}}_{\mathsf{t}} ((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta' \cup \eta_2), (\Gamma, \bowtie))$ and $\mathcal{A}', \eta' \cup \eta_2 \models T'$.

Proof. By case analysis over the transition step.

- $\widehat{\mathbb{l}} = \textsc{call}$. By inversion over $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{l}}}{\underset{R}{\hookrightarrow}}_{\mathsf{t}} ((C', \mathcal{S}'_c), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, we know

$$\begin{gathered} [\![E]\!]_{\mathcal{S}_c} = n \qquad \mathsf{split}(\Gamma(f, n)) = (\mu, \alpha) \\ \eta \subseteq \eta_0 \qquad \mathcal{A}, \eta_0 \models T_0 \qquad \forall T, \mathcal{S}'.\ (\mathcal{A}, \eta_0 \models T) \wedge (\mathcal{S}' = \mathsf{exec}(\mathcal{S}, T)) \implies \mu(\mathcal{S}') = n' \\ i \notin dom(\mathcal{A}) \qquad \mathcal{A}' = \mathcal{A} \uplus \{i \rightsquigarrow (\mathsf{t}, \alpha, \mathbf{cmt})\} \\ \eta' = \eta_0 \uplus \{(j, i) \mid \exists \alpha'.\ \mathcal{A}(j) = (\_, \alpha', \mathbf{cmt}) \wedge \alpha \bowtie \alpha'\} \end{gathered}$$

Let $\eta_0' = \eta_0 \cup \eta_2$. Since $\eta \subseteq \eta_0$ and $\eta_1 = \eta \uplus \eta_2$, we know $\eta_1 \subseteq \eta_0'$.
Since $\mathcal{A}', \eta' \models T'$ and $T = \mathrm{proj}(T', \mathcal{A})$, we know $\mathcal{A}, \eta_0 \models T$. Since $\mathcal{A}, \eta_1 \models T$, we know $\mathcal{A}, \eta_0' \models T$.
Since $\eta_0' = \eta_0 \cup \eta_2$, we know $\forall T.\ (\mathcal{A}, \eta_0' \models T) \implies (\mathcal{A}, \eta_0 \models T)$. Thus

$$\forall T, \mathcal{S}'.\ (\mathcal{A}, \eta_0' \models T) \wedge (\mathcal{S}' = \mathrm{exec}(\mathcal{S}, T)) \implies \mu(\mathcal{S}') = n'.$$

Thus $((C, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{l}}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((C', \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta' \cup \eta_2), (\Gamma, \bowtie))$.
Since $\eta_1 = \eta \uplus \eta_2$, $(\mathcal{A}', \eta' \models T')$, $T = \mathrm{proj}(T', \mathcal{A})$, $(\mathcal{A}, \eta_1 \models T)$, we know $\mathcal{A}', \eta' \cup \eta_2 \models T'$.

- Other steps. Similar to the previous case.

Thus we are done. □

Lemma 54 (SEQ-sound)**.** If

1. $R, G; \Gamma, \bowtie \models_{\mathrm{t}} \{p\}C_1\{q\}$,
2. $R, G; \Gamma, \bowtie \models_{\mathrm{t}} \{q\}C_2\{q'\}$,

then $R, G; \Gamma, \bowtie \models_{\mathrm{t}} \{p\}C_1; C_2\{q'\}$.

Proof. For any $\mathcal{S}$, $\mathcal{A}$, η and $\mathcal{S}_c$, if $(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}, \eta) \models p$, $fv(\Gamma) \subseteq dom(\mathcal{S})$ and $fv(C) \subseteq dom(\mathcal{S}_c)$, we want to prove the following:

(1) for any $\mathcal{A}'$, η', $\mathcal{S}_c'$, if $((C_1; C_2, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow{R}_{\mathrm{t}}^{*} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, then $\forall T'.\ (\mathcal{A}', \eta' \models T') \implies \exists \eta''.\ (\mathcal{A}', \eta'' \models T') \wedge (\eta' \subseteq \eta'') \wedge (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta'') \models q$.
(2) for any n, $((C_1; C_2, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), R)$ guarantees$_{\mathrm{t}}^{n}$ G.

For (1), from the operational semantics, we know there exist $\mathcal{S}_c''$, $\mathcal{A}''$ and η'' such that

$$((C_1, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow{R}_{\mathrm{t}}^{*} ((\mathbf{skip}, \mathcal{S}_c''), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie))$$
$$((C_2, \mathcal{S}_c''), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie)) \xhookrightarrow{R}_{\mathrm{t}}^{*} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$$

Since $\mathcal{A}', \eta' \models T'$, let $T'' = \mathrm{proj}(T', \mathcal{A}'')$, then we know $\mathcal{A}'', \eta'' \models T''$. From $R, G; \Gamma, \bowtie \models_{\mathrm{t}} \{p\}C_1\{q\}$, we know

(a) for any $\mathcal{A}''$, η'', $\mathcal{S}_c''$, if $((C_1, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \xhookrightarrow{R}_{\mathrm{t}}^{*} ((\mathbf{skip}, \mathcal{S}_c''), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie))$, then $\forall T''.\ (\mathcal{A}'', \eta'' \models T'') \implies \exists \eta_1''.\ (\mathcal{A}'', \eta_1'' \models T'') \wedge (\eta'' \subseteq \eta_1'') \wedge (\mathcal{S}_c'' \uplus \mathcal{S}, \mathcal{A}'', \eta_1'') \models q$.
(b) for any n, $((C_1, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), R)$ guarantees$_{\mathrm{t}}^{n}$ G.

Thus there exists η_1'' such that

$$(\mathcal{A}'', \eta_1'' \models T'') \wedge (\eta'' \subseteq \eta_1'') \wedge (\mathcal{S}_c'' \uplus \mathcal{S}, \mathcal{A}'', \eta_1'') \models q.$$

Suppose $\eta_1'' = \eta'' \uplus \eta_2$. Since $((C_2, \mathcal{S}_c''), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie)) \xhookrightarrow{R}_{\mathrm{t}}^{*} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, by Lemma 53, we know

$$((C_2, \mathcal{S}_c''), (\mathcal{S}, \mathcal{A}'', \eta_1''), (\Gamma, \bowtie)) \xhookrightarrow{R}_{\mathrm{t}}^{*} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta' \cup \eta_2), (\Gamma, \bowtie)) \quad \text{and} \quad \mathcal{A}', \eta' \cup \eta_2 \models T'.$$

From $R, G; \Gamma, \bowtie \models_{\mathrm{t}} \{q\}C_2\{q'\}$, we know

(c) for any $\mathcal{A}'$, η_1', $\mathcal{S}_c'$, if $((C_2, \mathcal{S}_c''), (\mathcal{S}, \mathcal{A}'', \eta_1''), (\Gamma, \bowtie)) \xhookrightarrow{R}_{\mathrm{t}}^{*} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta_1'), (\Gamma, \bowtie))$, then $\forall T'.\ (\mathcal{A}', \eta_1' \models T') \implies \exists \eta_2'.\ (\mathcal{A}', \eta_2' \models T') \wedge (\eta_1' \subseteq \eta_2') \wedge (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta_2') \models q'$.

Thus there exists η_2' such that

$$(\mathcal{A}', \eta_2' \models T') \wedge (\eta' \cup \eta_2 \subseteq \eta_2') \wedge (\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta_2') \models q'.$$

So (1) is done.

For (2), by induction over n.

- $n = 0$. Trivial.
- $n = k + 1$. We want to prove: for any C', $\mathcal{S}_c'$, $\mathcal{A}'$, η', if $((C_1; C_2, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{l}}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((C', \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, then
  (1) $((C', \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie), R)$ guarantees$_{\mathrm{t}}^{k}$ G; and
  (2) if $\widehat{\mathbb{l}} = \textsc{call}$, then there exist i, α such that $\mathcal{A}' = \mathcal{A} \uplus \{i \rightsquigarrow (\mathrm{t}, \alpha, \mathbf{cmt})\}$, and there exists η_0 such that $\eta \subseteq \eta_0$ and $((\mathcal{S}, \mathcal{A}, \eta_0), (i, \mathrm{t}, \alpha)) \models G$.

  We have two cases on C_1.

- $C_1 \neq \mathbf{skip}$. Then there exists C_1' such that $C' = (C_1'; C_2)$ and $((C_1, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\widehat{\mathbb{I}}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((C_1', \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$. From (b) and by the induction hypothesis, we are done.
- $C_1 = \mathbf{skip}$. Then $C' = C_2$, $\mathcal{S}_c' = \mathcal{S}_c$, $\mathcal{A}' = \mathcal{A}$ and $\eta' = \eta$. From (a), we know
$$\forall T.\ (\mathcal{A}, \eta \models T) \implies \exists \eta_1.\ (\mathcal{A}, \eta_1 \models T) \wedge (\eta \subseteq \eta_1) \wedge (\mathcal{S}_c \uplus \mathcal{S}, \mathcal{A}, \eta_1) \models q.$$
From $R, G; \Gamma, \bowtie \models_{\mathrm{t}} \{q\} C_2 \{q'\}$, we know
$$\forall T.\ (\mathcal{A}, \eta \models T) \implies \exists \eta_1.\ (\mathcal{A}, \eta_1 \models T) \wedge (\eta \subseteq \eta_1) \wedge \forall m.\ ((C_2, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta_1), (\Gamma, \bowtie), R) \text{ guarantees}_{\mathrm{t}}^m\ G.$$
By Lemma 52, we know
$$((C_2, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), R) \text{ guarantees}_{\mathrm{t}}^k\ G.$$

Thus we are done. □

Lemma 55 (LOCAL-sound)**.** If

1. $\mathsf{Sta}(p, R, \bowtie)$,
2. $\text{cmt-closed}(p)$,

then $R, G; \Gamma, \bowtie \models_{\mathrm{t}} \{p\} x := E \{\exists v.\ x = E[v/x] \wedge p[v/x]\}$.

Proof. For any $\mathcal{S}$, $\mathcal{A}$, η and $\mathcal{S}_c$, if $(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}, \eta) \models p$, $fv(\Gamma) \subseteq dom(\mathcal{S})$ and $\{x\} \cup fv(E) \subseteq dom(\mathcal{S}_c)$, we want to prove the following:

(1) for any n, if $((x := E, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{R}{\hookrightarrow}{}_{\mathrm{t}}^{n} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, then $(\mathcal{S}_c' \uplus \mathcal{S}, \mathcal{A}', \eta') \models q$.
(2) for any n, $((x := E, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie), R)$ guarantees$_{\mathrm{t}}^n$ G.

For (1), by induction over n. The base case $n = 0$ is trivial. Suppose $n = k + 1$.

Since $((x := E, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{R}{\hookrightarrow}{}_{\mathrm{t}}^{n} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$, we have three cases:

- $((x := E, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\text{LC}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie))$ and
$((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{R}{\hookrightarrow}{}_{\mathrm{t}}^{k} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$.
From the semantics, we know
$$[\![E]\!]_{\mathcal{S}_c} = n \text{ and } \mathcal{S}_c' = \mathcal{S}_c\{x \rightsquigarrow n\}.$$
Since $(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}, \eta) \models p$, we know
$$(\mathcal{S} \uplus \mathcal{S}_c', \mathcal{A}, \eta) \models \exists v.\ x = E[v/x] \wedge p[v/x].$$
Since $\mathsf{Sta}(p, R, \bowtie)$ and $\text{cmt-closed}(p)$, we know
$$\mathsf{Sta}((\exists v.\ x = E[v/x] \wedge p[v/x]), R, \bowtie) \text{ and cmt-closed}(\exists v.\ x = E[v/x] \wedge p[v/x]).$$
Thus
$$(\mathcal{S} \uplus \mathcal{S}_c', \mathcal{A}', \eta') \models \exists v.\ x = E[v/x] \wedge p[v/x].$$
- there exist $\mathcal{A}''$ and η'' such that
$((x := E, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\text{PRD}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((x := E, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie))$ and
$((x := E, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie)) \overset{R}{\hookrightarrow}{}_{\mathrm{t}}^{k} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$.
Since $\mathsf{Sta}(p, R, \bowtie)$, we know
$$(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}'', \eta'') \models p.$$
By the induction hypothesis, we are done.
- there exist $\mathcal{A}''$ and η'' such that
$((x := E, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}, \eta), (\Gamma, \bowtie)) \overset{\text{CMT}}{\underset{R}{\hookrightarrow}}_{\mathrm{t}} ((x := E, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie))$ and
$((x := E, \mathcal{S}_c), (\mathcal{S}, \mathcal{A}'', \eta''), (\Gamma, \bowtie)) \overset{R}{\hookrightarrow}{}_{\mathrm{t}}^{k} ((\mathbf{skip}, \mathcal{S}_c'), (\mathcal{S}, \mathcal{A}', \eta'), (\Gamma, \bowtie))$.
Since $\text{cmt-closed}(p)$, we know
$$(\mathcal{S} \uplus \mathcal{S}_c, \mathcal{A}'', \eta'') \models p.$$
By the induction hypothesis, we are done.

(2) is trivial. Thus we are done. □

Lemma 56 (PAR-sound)**.** If for any $\mathrm{t} \in [1..n]$ we have

1. $R_{\mathrm{t}}, G_{\mathrm{t}}; \Gamma, \bowtie \vdash_{\mathrm{t}} \{\mathcal{P} \wedge \mathsf{emp}\} C_{\mathrm{t}} \{q_{\mathrm{t}}\}$,

2. $(\bigvee_{t' \neq t} G_{t'}) \Rightarrow R_t$,
3. $q_t \Rrightarrow Q_t$,
4. $fv(\{G_t, R_t\}) \subseteq fv(\Gamma)$,

then $\models^{\mathsf{prd}} \{\mathcal{P}\}\mathbf{with}\ (\Gamma, \bowtie)\ \mathbf{do}\ C_1 \parallel \ldots \parallel C_n \{\bigwedge_t Q_t\}$.

Proof. Let $\mathbb{P} = \mathbf{with}\ (\Gamma, \bowtie)\ \mathbf{do}\ C_1 \parallel \ldots \parallel C_n$.

For any $\mathcal{S}, \mathcal{S}'_1, \ldots, \mathcal{S}'_n, \mathcal{S}_0, \mathcal{A}', \eta'$, if $\mathcal{S} \models_{\text{HOARE}} \mathcal{P}$ and $(\mathbb{P}, \mathcal{S}) \longmapsto^* (\mathbf{end}, (\mathcal{S}'_1, \ldots, \mathcal{S}'_n), (\mathcal{S}_0, \mathcal{A}', \eta'))$, we want to prove

$$\forall \mathcal{S}'_0.\ (\exists T.\ (\mathcal{A}', \eta' \models T) \land \mathsf{exec}(\mathcal{S}_0, T) = \mathcal{S}'_0) \implies (\mathcal{S}'_1 \uplus \ldots \uplus \mathcal{S}'_n \uplus \mathcal{S}'_0) \models_{\text{HOARE}} \bigwedge_t Q_t.$$

Since $\mathcal{S} \models_{\text{HOARE}} \mathcal{P}$, we know

$$(\mathcal{S}, \emptyset, \emptyset) \models \mathcal{P} \land \mathsf{emp}.$$

For any t, since $R_t, G_t; \Gamma, \bowtie \vdash_t \{\mathcal{P} \land \mathsf{emp}\} C_t \{q_t\}$, we know

(a) if $((C_t, \emptyset), (\mathcal{S}, \emptyset, \emptyset), (\Gamma, \bowtie)) \xhookrightarrow{R_t}{}^*_t ((\mathbf{skip}, \mathcal{S}^c_t), (\mathcal{S}, \mathcal{A}'_t, \eta'_t), (\Gamma, \bowtie))$, then
$\forall T'.\ (\mathcal{A}'_t, \eta'_t \models T') \implies \exists \eta''_t.\ (\mathcal{A}'_t, \eta''_t \models T') \land (\eta'_t \subseteq \eta''_t) \land (\mathcal{S}^c_t \uplus \mathcal{S}, \mathcal{A}'_t, \eta''_t) \models q_t$.
(b) for any k, $((C_t, \emptyset), (\mathcal{S}, \emptyset, \emptyset), (\Gamma, \bowtie), R_t)$ $\mathsf{guarantees}^k_t$ G_t.

Since $(\mathbb{P}, \mathcal{S}) \longmapsto^* (\mathbf{end}, (\mathcal{S}'_1, \ldots, \mathcal{S}'_n), (\mathcal{S}_0, \mathcal{A}', \eta'))$, from the semantics, we know there exists m such that

$$\begin{array}{l}(\mathbb{P}, \mathcal{S}) \longmapsto (\sigma_c, \Omega, (\Gamma, \bowtie)), \quad (\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto^m (\sigma'_c, \Omega', (\Gamma, \bowtie)) \\ (\sigma'_c, \Omega', (\Gamma, \bowtie)) \longmapsto (\mathbf{end}, (\mathcal{S}'_1, \ldots, \mathcal{S}'_n), (\mathcal{S}, \mathcal{A}', \eta')), \quad \mathcal{S}_0 = \mathcal{S} \\ fv(\Gamma) \subseteq dom(\mathcal{S}), \qquad \forall t \in [1..n].\ \sigma_c(t) = (C_t, \emptyset) \land \Omega(t) = (\mathcal{S}, \emptyset, \emptyset, \emptyset) \\ \forall t.\ \sigma'_c(t) = (\mathbf{skip}, \mathcal{S}'_t) \land \Omega'(t) = (\mathcal{S}, \mathcal{A}', \eta', ms'), \quad \forall i.\ \mathcal{A}'(i) = (\_, \_, \mathbf{cmt})\end{array}$$

By induction over m.

- $m = 0$. Then we know $C_1 = \ldots = C_n = \mathbf{skip}$. For any t, from (a), we know
$$\forall T'.\ (\mathcal{A}', \eta' \models T') \implies \exists \eta''.\ (\mathcal{A}', \eta'' \models T') \land (\eta' \subseteq \eta'') \land (\mathcal{S}'_t \uplus \mathcal{S}, \mathcal{A}', \eta'') \models q_t.$$
Since $q_t \Rrightarrow Q_t$ and $\forall i.\ \mathcal{A}'(i) = (\_, \_, \mathbf{cmt})$, we know
$$\forall T'.\ (\mathcal{A}', \eta' \models T') \implies \exists \eta''.\ (\mathcal{A}', \eta'' \models T') \land (\eta' \subseteq \eta'') \land (\mathcal{S}'_t \uplus \mathcal{S}, \mathcal{A}', \eta'') \models Q_t.$$
From $\mathsf{SLocality}(\Gamma)$, we know
$$\forall T, \mathcal{S}'.\ (\mathcal{A}', \eta' \models T) \land (\mathcal{S}' = \mathsf{exec}(\mathcal{S}, T)) \implies \mathcal{S}'_t \uplus \mathcal{S}' \models_{\text{HOARE}} Q_t.$$
Thus we know
$$\forall \mathcal{S}'.\ (\exists T.\ (\mathcal{A}', \eta' \models T) \land \mathsf{exec}(\mathcal{S}, T) = \mathcal{S}') \implies (\mathcal{S}'_1 \uplus \ldots \uplus \mathcal{S}'_n \uplus \mathcal{S}') \models_{\text{HOARE}} \bigwedge_t Q_t.$$
- $m = k + 1$. Thus there exist σ''_c and Ω'' such that
$$(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto (\sigma''_c, \Omega'', (\Gamma, \bowtie)), \qquad (\sigma''_c, \Omega'', (\Gamma, \bowtie)) \longmapsto^k (\sigma'_c, \Omega', (\Gamma, \bowtie))$$
By case analysis over $(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto (\sigma''_c, \Omega'', (\Gamma, \bowtie))$.
  1. It is a (CALL&PRD) step. From the semantics, we know there exists t such that
$$\begin{array}{c}\sigma_c(t) = (C_t, \mathcal{S}_t), \quad \Omega(t) = (\mathcal{S}, \mathcal{A}_t, \eta, ms), \\ \sigma''_c(t) = (C''_t, \mathcal{S}''_t), \quad \Omega''(t) = (\mathcal{S}, \mathcal{A}''_t, \eta'', ms''), \\ [\![E]\!]_{\mathcal{S}_t} = n,\ \mathsf{split}(\Gamma(f, n)) = (\mu, \alpha),\ \mathcal{A}_t, \eta \models T,\ \mathcal{S}' = \mathsf{exec}(\mathcal{S}, T),\ \mu(\mathcal{S}') = n', \\ ms'' = ms \cup ms_0,\ i \notin dom(\mathcal{A}_t),\ \mathcal{A}''_t = \mathcal{A}_t \uplus \{i \rightsquigarrow (t, \alpha, \mathbf{cmt})\}, \\ ms_0 = \{j \mid \exists \alpha'.\ \mathcal{A}_t(j) = (\_, \alpha', \mathbf{cmt}) \land \alpha \bowtie \alpha'\},\ \eta'' = \eta \uplus \{(j, i) \mid j \in ms_0\}, \\ ((C_t, \mathcal{S}_t), (\mathcal{S}, \mathcal{A}_t, \eta), (\Gamma, \bowtie)) \xhookrightarrow[R_t]{\text{CALL}}_t ((C''_t, \mathcal{S}''_t), (\mathcal{S}, \mathcal{A}''_t, \eta''), (\Gamma, \bowtie))\end{array}$$
From (b), we know
$$((\mathcal{S}, \mathcal{A}_t, \eta), (i, t, \alpha)) \models' G_t$$
For any $t' \neq t$, since $G_t \Rightarrow R_{t'}$, we know
$$((\mathcal{S}, \mathcal{A}_t, \eta), (i, t, \alpha)) \models' R_{t'}$$
Suppose
$$\begin{array}{c}\sigma_c(t') = (C_{t'}, \mathcal{S}_{t'}), \quad \Omega(t') = (\mathcal{S}, \mathcal{A}_{t'}, \eta, ms), \\ \sigma''_c(t') = (C''_{t'}, \mathcal{S}''_{t'}), \quad \Omega''(t') = (\mathcal{S}, \mathcal{A}''_{t'}, \eta'', ms''),\end{array}$$
From the semantics, we know
$$\mathcal{A}_t \subsetneq \mathcal{A}_{t'}.$$
Thus we know

$$((C_{t'}, \mathcal{S}_{t'}), (\mathcal{S}, \mathcal{A}_{t'}, \eta), (\Gamma, \bowtie)) \xhookrightarrow[R_{t'}]{\text{PRD}}{}_{t'} ((C_{t'}, \mathcal{S}_{t'}), (\mathcal{S}, \mathcal{A}''_{t'}, \eta''), (\Gamma, \bowtie)).$$

By the induction hypothesis, we are done.

2. It is a (CMT) step. From the semantics we know there exists t such that
$$\Omega(t) = (\mathcal{S}, \mathcal{A}_t, \eta, ms),\ \mathcal{A}_t(i) = (t', \alpha, \mathbf{prd}),$$
$$\mathcal{A}''_t = \mathcal{A}_t\{i \rightsquigarrow (t', \alpha, \mathbf{cmt})\},\ \Omega''(t) = (\mathcal{S}, \mathcal{A}''_t, \eta, ms)$$
Also from the semantics and (b) we know $\neg(R_t \Rightarrow \mathsf{Emp})$. Thus
$$((C_t, \mathcal{S}_t), (\mathcal{S}, \mathcal{A}_t, \eta), (\Gamma, \bowtie)) \xhookrightarrow[R_t]{\text{CMT}}{}_t ((C_t, \mathcal{S}_t), (\mathcal{S}, \mathcal{A}''_t, \eta), (\Gamma, \bowtie)).$$
By the induction hypothesis, we are done.

3. It is a (LOCAL) step. Similar to the above case.

Thus we are done. □

E.5 Final Soundness Proofs for Clients with ACC Objects

Proof of Theorem 46(1). Let $P = \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n$ and $\mathbb{P} = \mathbf{with}\ (\Gamma, \bowtie)\ \mathbf{do}\ C_1 \parallel \ldots \parallel C_n$.

For any $\mathcal{S}$, $\mathcal{E}$ and $\mathcal{S}_1, \ldots, \mathcal{S}_n, \mathcal{S}'_1, \ldots, \mathcal{S}'_n$, if $\mathcal{S} \models_{\text{HOARE}} \mathcal{P}'$ and $(P, \mathcal{S}) \overset{\mathcal{E}}{\longmapsto}{}^* (\mathbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}'_1, \ldots, \mathcal{S}'_n))$, since $\mathcal{P}' \Rightarrow \varphi^{-1}[\mathcal{P}]$, we know there exists $\mathcal{S}_a$ such that

$$(\varphi(\mathcal{S}) = \mathcal{S}_a) \wedge (\mathcal{S}_a \models \mathcal{P}).$$

From $\Pi \sqsubseteq_\varphi (\Gamma, \bowtie)$, we know

$$(\mathbb{P}, \mathcal{S}_a) \overset{\mathbb{O}}{\multimap}{}^* (\mathbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\varphi(\mathcal{S}'_1), \ldots, \varphi(\mathcal{S}'_n))).$$

where $\mathbb{O} = \text{obsv}(\mathcal{E})$. From Theorem 44(2), we know $\models \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}$. Thus

$$\varphi(\mathcal{S}'_1) = \ldots = \varphi(\mathcal{S}'_n) \text{ and } (\mathcal{S}_1 \uplus \ldots \uplus \mathcal{S}_n \uplus \varphi(\mathcal{S}'_1)) \models_{\text{HOARE}} \mathcal{Q}.$$

From the semantics we know $dom(\varphi(\mathcal{S}'_1)) \subseteq fv(\Gamma)$. Since $\forall i.\ fv(\Gamma) \cap fv(C_i) = \emptyset$, we know $dom(\varphi(\mathcal{S}'_1)) \cap (\bigcup_t fv(C_t)) = \emptyset$. Since $fv(\mathcal{Q}) \subseteq \bigcup_t fv(C_t)$, we know $dom(\varphi(\mathcal{S}'_1)) \cap fv(\mathcal{Q}) = \emptyset$. Thus

$$(\mathcal{S}_1 \uplus \ldots \uplus \mathcal{S}_n) \models_{\text{HOARE}} \mathcal{Q}.$$

Thus $(\mathcal{S}_1 \uplus \ldots \uplus \mathcal{S}_n \uplus \mathcal{S}'_1) \models_{\text{HOARE}} \mathcal{Q}$. Thus we are done. □

Proof of Theorem 44(2). For any $\mathcal{S}$, $\mathbb{O}$ and $\mathcal{S}_1, \ldots, \mathcal{S}_n, \mathcal{S}'_1, \ldots, \mathcal{S}'_n$, if $\mathcal{S} \models_{\text{HOARE}} \mathcal{P}$ and $(\mathbb{P}, \mathcal{S}) \overset{\mathbb{O}}{\multimap}{}^* (\mathbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}'_1, \ldots, \mathcal{S}'_n))$, from Lemma 57, we know there exist $\mathcal{S}_0$, $\mathcal{A}'$, η' such that

- $(\mathbb{P}, \mathcal{S}) \longmapsto\!\!\triangleright_t^* (\mathbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}_0, \mathcal{A}', \eta'))$, and
- $\forall t.\ \exists T.\ (\mathcal{A}', \eta' \models T) \wedge \mathsf{exec}(\mathcal{S}_0, T) = \mathcal{S}'_t$.

From Lemma 48, we know $\models^{\mathsf{prd}} \{\mathcal{P}\}\mathbb{P}\{\mathcal{Q}\}$. Thus we know

$$\forall \mathcal{S}'_0.\ (\exists T.\ (\mathcal{A}', \eta' \models T) \wedge \mathsf{exec}(\mathcal{S}_0, T) = \mathcal{S}'_0) \implies (\mathcal{S}_1 \uplus \ldots \uplus \mathcal{S}_n \uplus \mathcal{S}'_0) \models_{\text{HOARE}} \mathcal{Q}.$$

From Lemma 17, we know $\mathsf{CvA}(\Gamma, \bowtie)$. Thus we know

$$\mathcal{S}'_1 = \ldots = \mathcal{S}'_n.$$

As a result we know

$$(\mathcal{S}_1 \uplus \ldots \uplus \mathcal{S}_n \uplus \mathcal{S}'_1) \models_{\text{HOARE}} \mathcal{Q}.$$

Thus we are done. □

Lemma 57 ($\multimap$ implies $\longmapsto\!\!\triangleright$). If $(\mathbb{P}, \mathcal{S}) \overset{\mathbb{O}}{\multimap}{}^* (\mathbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}'_1, \ldots, \mathcal{S}'_n))$, then there exist $\mathcal{S}_0$, $\mathcal{A}'$, η' such that

- $(\mathbb{P}, \mathcal{S}) \longmapsto\!\!\triangleright_t^* (\mathbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}_0, \mathcal{A}', \eta'))$, and
- $\forall t.\ \exists T.\ (\mathcal{A}', \eta' \models T) \wedge \mathsf{exec}(\mathcal{S}_0, T) = \mathcal{S}'_t$.

Proof. From $(\mathbb{P}, \mathcal{S}) \overset{\mathbb{O}}{\multimap}{}^* (\mathbf{end}, (\mathcal{S}_1, \ldots, \mathcal{S}_n), (\mathcal{S}'_1, \ldots, \mathcal{S}'_n))$, we know there exist σ_c, Σ, σ'_c, Σ', $\mathbb{M}'$ such that

$$
\begin{array}{c}
(\mathbb{P}, \mathcal{S}) \xrightarrow{\text{load}} (\sigma_c, \Sigma, \emptyset, \bowtie), \qquad (\sigma_c, \Sigma, \emptyset, \bowtie) \longmapsto^* (\sigma_c', \Sigma', \mathbb{M}', \bowtie), \\
(\sigma_c', \Sigma', \mathbb{M}', \bowtie) \longmapsto (\mathbf{end}, (\mathcal{S}_1, \dots, \mathcal{S}_n), (\mathcal{S}_1', \dots, \mathcal{S}_n')) \\
\forall \mathrm{t} \in [1..n].\ \sigma_c'(\mathrm{t}) = (\mathbf{skip}, \mathcal{S}_\mathrm{t}) \qquad \forall \mathrm{t} \in [1..n].\ \Sigma'(\mathrm{t}) = (\Gamma, \mathcal{S}, \xi_\mathrm{t}') \\
\forall \mathrm{t} \in [1..n].\ dom(\mathbb{M}') = dom(\xi_\mathrm{t}') \qquad \forall \mathrm{t} \in [1..n].\ \mathcal{S}_\mathrm{t}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \xi_\mathrm{t}')
\end{array}
$$

Let $\Omega = \lambda \mathrm{t} \in [1..n].\ (\mathcal{S}, \emptyset, \emptyset, \emptyset)$. So

$$(\mathbb{P}, \mathcal{S}) \longmapsto (\sigma_c, \Omega, (\Gamma, \bowtie)).$$

By Lemma 58, we know there exist Ω', $\mathcal{A}'$, η', ms' such that

$$
\begin{array}{c}
(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto^* (\sigma_c', \Omega', (\Gamma, \bowtie)) \qquad (\sigma_c', \Omega', (\Gamma, \bowtie)) \longmapsto (\mathbf{end}, (\mathcal{S}_1, \dots, \mathcal{S}_n), (\mathcal{S}, \mathcal{A}', \eta')) \\
\Omega' = \lambda \mathrm{t} \in [1..n].\ (\mathcal{S}, \mathcal{A}', \eta', ms') \qquad \mathcal{S}_0 = \mathcal{S} \\
dom(\mathcal{A}') = dom(\mathbb{M}') \qquad \forall i.\ \mathcal{A}'(i) = (\mathbb{M}'(i).\mathrm{t}, \mathsf{split}(\Gamma(\mathbb{M}'(i))).\alpha, \mathbf{cmt}) \\
\forall \mathrm{t} \in [1..n].\ \forall i, j.\ (i, j) \in \eta' \wedge \{i, j\} \subseteq dom(\xi_\mathrm{t}') \implies i \prec_{\xi_\mathrm{t}'} j
\end{array}
$$

For any t, let $T_\mathrm{t} = \mathsf{tr}(\mathcal{A}', \xi_\mathrm{t}')$. Here we define

$$
\mathsf{tr}(\mathcal{A}, \xi) \overset{\text{def}}{=} \begin{cases} \epsilon & \text{if } \xi = \epsilon \\ (mid, \mathcal{A}(mid)) :: \mathsf{tr}(\mathcal{A}, \xi') & \text{if } \xi = (mid, (f, n)) :: \xi' \wedge mid \in dom(\mathcal{A}) \\ \mathsf{tr}(\mathcal{A}, \xi') & \text{if } \xi = (mid, (f, n)) :: \xi' \wedge mid \notin dom(\mathcal{A}) \end{cases}
$$

Since $dom(\mathcal{A}') = dom(\mathbb{M}') = dom(\xi_\mathrm{t}')$, we know $\lfloor T_\mathrm{t} \rfloor = \mathcal{A}'$. So

$$(\mathcal{A}', \eta' \models T_\mathrm{t}) \wedge \mathsf{exec}(\mathcal{S}, T_\mathrm{t}) = \mathcal{S}_\mathrm{t}'$$

Thus we are done. □

Lemma 58. If

- $(\sigma_c, \Sigma, \mathbb{M}, \bowtie) \longmapsto^m (\sigma_c', \Sigma', \mathbb{M}', \bowtie)$, $(\sigma_c', \Sigma', \mathbb{M}', \bowtie) \longmapsto (\mathbf{end}, (\mathcal{S}_1, \dots, \mathcal{S}_n), (\mathcal{S}_1', \dots, \mathcal{S}_n'))$,
- $\Sigma = \lambda \mathrm{t} \in [1..n].\ (\Gamma, \mathcal{S}, \xi_\mathrm{t})$, $\Sigma' = \lambda \mathrm{t} \in [1..n].\ (\Gamma, \mathcal{S}, \xi_\mathrm{t}')$,
- $\Omega = \lambda \mathrm{t} \in [1..n].\ (\mathcal{S}, \mathcal{A}_\mathrm{t}, \eta, ms)$,
- for any t: $dom(\mathcal{A}_\mathrm{t}) = dom(\mathbb{M})$, $\forall i \in dom(\xi_\mathrm{t}).\ \mathcal{A}_\mathrm{t}(i) = (\mathbb{M}(i).\mathrm{t}, \mathsf{split}(\Gamma(\mathbb{M}(i))).\alpha, \mathbf{cmt})$,
  $\forall i \in dom(\mathbb{M}) - dom(\xi_\mathrm{t}).\ \mathcal{A}_\mathrm{t}(i) = (\mathbb{M}(i).\mathrm{t}, \mathsf{split}(\Gamma(\mathbb{M}(i))).\alpha, \mathbf{prd})$,
- for any t: $\forall i, j.\ (i, j) \in \eta \wedge \{i, j\} \subseteq dom(\xi_\mathrm{t}) \implies i \prec_{\xi_\mathrm{t}} j$,
  $\forall i, j.\ (i, j) \in \eta \wedge \{i, j\} \subseteq dom(\xi_\mathrm{t}') \implies i \prec_{\xi_\mathrm{t}'} j$,
- for any t: $dom(\mathbb{M}') = dom(\xi_\mathrm{t}')$,

then there exist Ω', $\mathcal{A}_1', \dots, \mathcal{A}_n'$, η', ms' such that

- $(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto^* (\sigma_c', \Omega', (\Gamma, \bowtie))$,
- $\Omega' = \lambda \mathrm{t} \in [1..n].\ (\mathcal{S}, \mathcal{A}_\mathrm{t}', \eta', ms')$,
- for any t: $dom(\mathcal{A}_\mathrm{t}') = dom(\mathbb{M}')$, $\forall i \in dom(\xi_\mathrm{t}').\ \mathcal{A}_\mathrm{t}'(i) = (\mathbb{M}'(i).\mathrm{t}, \mathsf{split}(\Gamma(\mathbb{M}'(i))).\alpha, \mathbf{cmt})$,
- for any t: $\forall i, j.\ (i, j) \in \eta' \wedge \{i, j\} \subseteq dom(\xi_\mathrm{t}') \implies i \prec_{\xi_\mathrm{t}'} j$.

Proof. By induction over m.

- $m = 0$. Let $\Omega' = \Omega$, $\eta' = \eta$, $ms' = ms$ and for any t, let $\mathcal{A}_\mathrm{t}' = \mathcal{A}_\mathrm{t}$. We get the conclusion trivially.
- $m = k + 1$. So there exist σ_c'', Σ'', $\mathbb{M}''$, $\xi_1'', \dots, \xi_n''$ such that

$$
\begin{array}{c}
(\sigma_c, \Sigma, \mathbb{M}, \bowtie) \longmapsto (\sigma_c'', \Sigma'', \mathbb{M}'', \bowtie), \qquad (\sigma_c'', \Sigma'', \mathbb{M}'', \bowtie) \longmapsto^k (\sigma_c', \Sigma', \mathbb{M}', \bowtie), \\
\Sigma'' = \lambda \mathrm{t} \in [1..n].\ (\Gamma, \mathcal{S}, \xi_\mathrm{t}'').
\end{array}
$$

  By the operational semantics of $\longmapsto$, we know there exists t such that

$$
\begin{array}{c}
\sigma_c(\mathrm{t}) = (C_\mathrm{t}, \mathcal{S}_\mathrm{t}), \qquad \Sigma(\mathrm{t}) = (\Gamma, \mathcal{S}, \xi_\mathrm{t}), \\
((C_\mathrm{t}, \mathcal{S}_\mathrm{t}), (\Gamma, \mathcal{S}, \xi_\mathrm{t}), \mathbb{M}) \xrightarrow{\mathbb{I}}_\mathrm{t} ((C_\mathrm{t}'', \mathcal{S}_\mathrm{t}''), (\Gamma, \mathcal{S}, \xi_\mathrm{t}''), \mathbb{M}'') \\
\sigma_c'' = \sigma_c\{\mathrm{t} \rightsquigarrow (C_\mathrm{t}'', \mathcal{S}_\mathrm{t}'')\}, \qquad \Sigma'' = \Sigma\{\mathrm{t} \rightsquigarrow (\Gamma, \mathcal{S}, \xi_\mathrm{t}'')\}, \qquad \forall \mathrm{t}' \neq \mathrm{t}.\ \xi_{\mathrm{t}'}'' = \xi_{\mathrm{t}'} \\
\forall \mathrm{t}' \neq \mathrm{t}.\ \mathsf{AbsCoh}(\xi_\mathrm{t}'', \xi_{\mathrm{t}'}'', (\Gamma, \bowtie))
\end{array}
$$

  We consider different cases of $\xrightarrow{\mathbb{I}}_\mathrm{t}$:

  1. It is a call step. So

$$C_{\mathrm{t}} = (x := f(E)),\quad C''_{\mathrm{t}} = \mathbf{skip},\quad \mathcal{S}''_{\mathrm{t}} = \mathcal{S}_{\mathrm{t}}\{x \rightsquigarrow n'\},$$
$$[\![E]\!]_{\mathcal{S}_{\mathrm{t}}} = n,\quad mid \notin dom(\mathbb{M}),\quad \mathbb{M}'' = \mathbb{M} \uplus \{mid \rightsquigarrow (\mathrm{t}, f, n)\},$$
$$\xi''_{\mathrm{t}} = \xi_{\mathrm{t}}{+}{+}[(mid, (\mathrm{t}, f, n))],\quad \mathsf{aexecRV}(\Gamma, \mathcal{S}, \xi''_{\mathrm{t}}) = n'$$

Suppose $\mathsf{split}(\Gamma(f, n)) = (\mu, \alpha)$. Let $T = \mathsf{tr}(\mathcal{A}_{\mathrm{t}}, \xi'_{\mathrm{t}})$ where tr is defined above in the proof of Lemma 57. Thus $\lfloor T \rfloor = \mathcal{A}_{\mathrm{t}}$. Also from the semantics, we know $dom(\mathcal{A}_{\mathrm{t}}) = dom(\mathbb{M}) \subseteq dom(\mathbb{M}') = dom(\xi'_{\mathrm{t}})$. Since $\forall i, j.\ (i, j) \in \eta \wedge \{i, j\} \subseteq dom(\xi'_{\mathrm{t}}) \implies i <_{\xi'_{\mathrm{t}}} j$, we know $\forall i, j.\ (i, j) \in \eta \wedge \{i, j\} \subseteq dom(\mathcal{A}_{\mathrm{t}}) \implies i <_T j$. Thus we know

$$\mathcal{A}_{\mathrm{t}}, \eta \models T$$

Since $\xi''_{\mathrm{t}} = \xi_{\mathrm{t}}{+}{+}[(mid, (\mathrm{t}, f, n))]$ and $\mathsf{aexecRV}(\Gamma, \mathcal{S}, \xi''_{\mathrm{t}}) = n'$, we know there exists $\mathcal{S}'$ such that

$$\mathcal{S}' = \mathsf{exec}(\mathcal{S}, T),\quad \mu(\mathcal{S}') = n'$$

Since $mid \notin dom(\mathbb{M})$, we know

$$mid \notin dom(\mathcal{A}_{\mathrm{t}})$$

Let

$$ms'' = \{j \mid \exists \alpha'.\ \mathcal{A}_{\mathrm{t}}(j) = (\_, \alpha', \mathbf{cmt}) \wedge \alpha \bowtie \alpha'\} \qquad \eta'' = \eta \uplus \{(j, mid) \mid j \in ms''\}$$
$$\mathcal{A}''_{\mathrm{t}} = \mathcal{A}_{\mathrm{t}} \uplus \{mid \rightsquigarrow (\mathrm{t}, \alpha, \mathbf{cmt})\} \qquad \forall \mathrm{t}' \neq \mathrm{t}.\ \mathcal{A}''_{\mathrm{t}'} = \mathcal{A}_{\mathrm{t}'} \uplus \{mid \rightsquigarrow (\mathrm{t}, \alpha, \mathbf{prd})\}$$
$$\Omega''(\mathrm{t}) = (\mathcal{S}, \mathcal{A}''_{\mathrm{t}}, \eta'', ms \cup ms'') \qquad \forall \mathrm{t}' \neq \mathrm{t}.\ \Omega''(\mathrm{t}') = (\mathcal{S}, \mathcal{A}''_{\mathrm{t}'}, \eta'', ms \cup ms'')$$

Thus we know

$$(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto (\sigma''_c, \Omega'', (\Gamma, \bowtie))$$

Also we know, for any t':

$$dom(\mathcal{A}''_{\mathrm{t}'}) = dom(\mathbb{M}''),$$
$$\forall i \in dom(\xi''_{\mathrm{t}'}).\ \mathcal{A}''_{\mathrm{t}'}(i) = (\mathbb{M}''(i).\mathrm{t}, \mathsf{split}(\Gamma(\mathbb{M}''(i))).\alpha, \mathbf{cmt}),$$
$$\forall i \in dom(\mathbb{M}'') - dom(\xi''_{\mathrm{t}'}).\ \mathcal{A}''_{\mathrm{t}'}(i) = (\mathbb{M}''(i).\mathrm{t}, \mathsf{split}(\Gamma(\mathbb{M}''(i))).\alpha, \mathbf{prd}),$$
$$\forall i, j.\ (i, j) \in \eta'' \wedge \{i, j\} \subseteq dom(\xi''_{\mathrm{t}'}) \implies i <_{\xi''_{\mathrm{t}'}} j$$

Also, by the semantics, we know $\forall \mathrm{t}' \neq \mathrm{t}.\ \mathsf{AbsCoh}(\xi''_{\mathrm{t}}, \xi'_{\mathrm{t}'}, (\Gamma, \bowtie))$. So

$$\forall \mathrm{t}'.\ \forall j.\ j \in ms'' \implies j <_{\xi'_{\mathrm{t}'}} mid$$

Thus we know, for any t':

$$\forall i, j.\ (i, j) \in \eta'' \wedge \{i, j\} \subseteq dom(\xi'_{\mathrm{t}'}) \implies i <_{\xi'_{\mathrm{t}'}} j$$

So, by the induction hypothesis, we are done.

2. It is a receive step. So

$$\mathbb{M}(mid) = (f, n),\quad mid \notin dom(\xi_{\mathrm{t}}),\quad \xi_{\mathrm{t}} = \xi'{+}{+}\xi'',\quad \xi''_{\mathrm{t}} = \xi'{+}{+}[(mid, (f, n))]{+}{+}\xi'',$$
$$C_{\mathrm{t}} = C''_{\mathrm{t}},\quad \mathcal{S}_{\mathrm{t}} = \mathcal{S}''_{\mathrm{t}},\quad \mathbb{M} = \mathbb{M}''$$

Suppose $\mathsf{split}(\Gamma(f, n)) = (\mu, \alpha)$. Since $\mathbb{M}(mid) = (f, n)$ and $mid \notin dom(\xi_{\mathrm{t}})$, we know there exists t_0 such that

$$\mathcal{A}_{\mathrm{t}}(mid) = (\mathrm{t}_0, \alpha, \mathbf{prd})$$

Let

$$\mathcal{A}''_{\mathrm{t}} = \mathcal{A}_{\mathrm{t}}\{mid \rightsquigarrow (\mathrm{t}_0, \alpha, \mathbf{cmt})\} \qquad \forall \mathrm{t}' \neq \mathrm{t}.\ \mathcal{A}''_{\mathrm{t}'} = \mathcal{A}_{\mathrm{t}'} \qquad \eta'' = \eta$$
$$\Omega''(\mathrm{t}) = (\mathcal{S}, \mathcal{A}''_{\mathrm{t}}, \eta, ms) \qquad \forall \mathrm{t}' \neq \mathrm{t}.\ \Omega''(\mathrm{t}') = \Omega(\mathrm{t}')$$

Thus we know

$$(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto (\sigma''_c, \Omega'', (\Gamma, \bowtie))$$

Also we know, for any t':

$$dom(\mathcal{A}''_{\mathrm{t}'}) = dom(\mathbb{M}''),$$
$$\forall i \in dom(\xi''_{\mathrm{t}'}).\ \mathcal{A}''_{\mathrm{t}'}(i) = (\mathbb{M}''(i).\mathrm{t}, \mathsf{split}(\Gamma(\mathbb{M}''(i))).\alpha, \mathbf{cmt}),$$
$$\forall i \in dom(\mathbb{M}'') - dom(\xi''_{\mathrm{t}'}).\ \mathcal{A}''_{\mathrm{t}'}(i) = (\mathbb{M}''(i).\mathrm{t}, \mathsf{split}(\Gamma(\mathbb{M}''(i))).\alpha, \mathbf{prd})$$

Since $\forall i, j.\ (i, j) \in \eta \wedge \{i, j\} \subseteq dom(\xi'_{\mathrm{t}}) \implies i <_{\xi'_{\mathrm{t}}} j$, we know, for any t':

$$\forall i, j.\ (i, j) \in \eta'' \wedge \{i, j\} \subseteq dom(\xi''_{\mathrm{t}'}) \implies i <_{\xi''_{\mathrm{t}'}} j$$
$$\forall i, j.\ (i, j) \in \eta'' \wedge \{i, j\} \subseteq dom(\xi'_{\mathrm{t}'}) \implies i <_{\xi'_{\mathrm{t}'}} j$$

So, by the induction hypothesis, we are done.

3. It is a local step. So

$$C_{\mathrm{t}} = (x := E),\quad C''_{\mathrm{t}} = \mathbf{skip},\quad \mathcal{S}''_{\mathrm{t}} = \mathcal{S}_{\mathrm{t}}\{x \rightsquigarrow n\},\quad [\![E]\!]_{\mathcal{S}_{\mathrm{t}}} = n,\quad \xi''_{\mathrm{t}} = \xi_{\mathrm{t}},\quad \mathbb{M}'' = \mathbb{M}$$

Let $\Omega'' = \Omega$, $\eta'' = \eta$ and $\forall \mathrm{t}'.\ \mathcal{A}''_{\mathrm{t}'} = \mathcal{A}_{\mathrm{t}'}$. Thus we know

$$(\sigma_c, \Omega, (\Gamma, \bowtie)) \longmapsto (\sigma''_c, \Omega'', (\Gamma, \bowtie))$$

Also we know, for any t':

$$dom(\mathcal{A}''_{t'}) = dom(\mathbb{M}''),$$
$$\forall i \in dom(\xi''_{t'}).\ \mathcal{A}''_{t'}(i) = (\mathbb{M}''(i).\mathrm{t}, \mathrm{split}(\Gamma(\mathbb{M}''(i))).\alpha, \mathbf{cmt}),$$
$$\forall i \in dom(\mathbb{M}'') - dom(\xi''_{t'}).\ \mathcal{A}''_{t'}(i) = (\mathbb{M}''(i).\mathrm{t}, \mathrm{split}(\Gamma(\mathbb{M}''(i))).\alpha, \mathbf{prd})$$
$$\forall i, j.\ (i, j) \in \eta'' \land \{i, j\} \subseteq dom(\xi''_{t'}) \implies i <_{\xi''_{t'}} j$$
$$\forall i, j.\ (i, j) \in \eta'' \land \{i, j\} \subseteq dom(\xi'_{t'}) \implies i <_{\xi'_{t'}} j$$

So, by the induction hypothesis, we are done.

Thus we are done. □

F Examples of Client Verification

We first present the full proof for the client program of RGA in Fig. 12. We also verify more client examples: three clients of RGA and two clients of registers.

F.1 RGA Client in Fig. 12

$$p_a \overset{\text{def}}{=} (\mathsf{s = a}) \wedge \mathsf{Id} \qquad \alpha_b \overset{\text{def}}{=} \texttt{addAfter(a,b)} \qquad \alpha_c \overset{\text{def}}{=} \texttt{addAfter(a,c)} \qquad \alpha_d \overset{\text{def}}{=} \texttt{addAfter(c,d)}$$

$$\begin{aligned}
G_{\mathsf{t}_1} &\overset{\text{def}}{=} (\mathsf{true} \rightsquigarrow [\alpha_b]^1_{\mathsf{t}_1}) \vee \mathsf{IId} & R_{\mathsf{t}_1} &\overset{\text{def}}{=} G_{\mathsf{t}_2} \vee G_{\mathsf{t}_3}\\
G_{\mathsf{t}_2} &\overset{\text{def}}{=} ((\diamondminus \boxed{\alpha_b}^1_{\mathsf{t}_1}) \rightsquigarrow [\alpha_c]^2_{\mathsf{t}_2}) \vee \mathsf{IId} & R_{\mathsf{t}_2} &\overset{\text{def}}{=} G_{\mathsf{t}_1} \vee G_{\mathsf{t}_3}\\
G_{\mathsf{t}_3} &\overset{\text{def}}{=} ((\diamondminus \boxed{\alpha_c}^2_{\mathsf{t}_2}) \rightsquigarrow [\alpha_d]^3_{\mathsf{t}_3}) \vee \mathsf{IId} & R_{\mathsf{t}_3} &\overset{\text{def}}{=} G_{\mathsf{t}_1} \vee G_{\mathsf{t}_2}
\end{aligned}$$

$\{\mathsf{s = a}\}$

$\{p_a\}$
```
addAfter(a, b);
```
$\left\{\begin{array}{l} p_a \sqcup \boxed{\alpha_b}^1_{\mathsf{t}_1} \\ \vee\ p_a \sqcup (\boxed{\alpha_b}^1_{\mathsf{t}_1} \ltimes [\alpha_c]^2_{\mathsf{t}_2}) \\ \vee\ p_a \sqcup (\boxed{\alpha_b}^1_{\mathsf{t}_1} \ltimes [\alpha_c]^2_{\mathsf{t}_2} \ltimes [\alpha_d]^3_{\mathsf{t}_3}) \end{array}\right\}$

$\|$

$\{p_a \vee p_a \sqcup [\alpha_b]^1_{\mathsf{t}_1}\}$
```
u := read();
if (b ∈ u)
```
$\quad\{p_a \sqcup \boxed{\alpha_b}^1_{\mathsf{t}_1}\}$
```
  addAfter(a, c);
```
$\quad\left\{\begin{array}{l} p_a \sqcup (\boxed{\alpha_b}^1_{\mathsf{t}_1} \ltimes \boxed{\alpha_c}^2_{\mathsf{t}_2}) \\ \vee\ p_a \sqcup (\boxed{\alpha_b}^1_{\mathsf{t}_1} \ltimes \boxed{\alpha_c}^2_{\mathsf{t}_2} \ltimes [\alpha_d]^3_{\mathsf{t}_3}) \end{array}\right\}$
```
x := read();
```
$\{\mathsf{d \in x \Rightarrow\ s = x = acdb}\}$

$\|$

$\left\{\begin{array}{l} p_a \vee p_a \sqcup [\alpha_b]^1_{\mathsf{t}_1} \\ \vee\ p_a \sqcup ([\alpha_b]^1_{\mathsf{t}_1} \ltimes [\alpha_c]^2_{\mathsf{t}_2}) \end{array}\right\}$
```
v := read();
if (c ∈ v)
```
$\quad\{p_a \sqcup ([\alpha_b]^1_{\mathsf{t}_1} \ltimes \boxed{\alpha_c}^2_{\mathsf{t}_2})\}$
```
  addAfter(c, d);
```
$\quad\{p_a \sqcup ([\alpha_b]^1_{\mathsf{t}_1} \ltimes \boxed{\alpha_c}^2_{\mathsf{t}_2} \ltimes \boxed{\alpha_d}^3_{\mathsf{t}_3})\}$
```
y := read();
```
$\{\mathsf{s = acdb\ \Rightarrow\ y = s \vee y = acd}\}$

$\{\mathsf{d \in x \Rightarrow (s = x = acdb) \wedge (y = x \vee y = acd)}\}$

Figure 26. Proof of RGA Client in Fig. 12.

Figure 26 shows the proof sketch of the RGA Client discussed in Fig. 12. By the definition of $\bowtie$ in the RGA specification, we know both $\alpha_b \bowtie \alpha_c$ and $\alpha_c \bowtie \alpha_d$ hold, but $\alpha_b \bowtie \alpha_d$ does not hold. The proof follows our logic rules.

F.2 RGA Client 1

$\{\mathsf{s = a}\}$

$\{p_a \vee (p_a \sqcup [\alpha_c]^2_{\mathsf{t}_2})\}$
```
addAfter(a, b);
```
$\{(p_a \sqcup \boxed{\alpha_b}^1_{\mathsf{t}_1}) \vee (p_a \sqcup \boxed{\alpha_b}^1_{\mathsf{t}_1} \sqcup [\alpha_c]^2_{\mathsf{t}_2})\}$
```
x := read();
```
$\{\mathsf{x = acb \Rightarrow s = acb}\}$

$\|$

$\{p_a \vee (p_a \sqcup [\alpha_b]^1_{\mathsf{t}_1})\}$
```
addAfter(a, c);
```
$\{(p_a \sqcup \boxed{\alpha_c}^2_{\mathsf{t}_2}) \vee (p_a \sqcup \boxed{\alpha_c}^2_{\mathsf{t}_2} \sqcup [\alpha_b]^1_{\mathsf{t}_1})\}$
```
y := read();
```
$\{\mathsf{s = acb \Rightarrow y = ac \vee y = acb}\}$

$\{\mathsf{x = acb \Rightarrow y = ac \vee y = acb}\}$

$$p_a \overset{\text{def}}{=} (\mathsf{s = a}) \wedge \mathsf{Id} \qquad \alpha_b \overset{\text{def}}{=} \texttt{addAfter(a,b)} \qquad \alpha_c \overset{\text{def}}{=} \texttt{addAfter(a,c)}$$

$$\begin{aligned}
G_{\mathsf{t}_1} &\overset{\text{def}}{=} (\mathsf{true} \rightsquigarrow [\alpha_b]^1_{\mathsf{t}_1}) \vee \mathsf{IId} & R_{\mathsf{t}_1} &\overset{\text{def}}{=} G_{\mathsf{t}_2}\\
G_{\mathsf{t}_2} &\overset{\text{def}}{=} (\mathsf{true} \rightsquigarrow [\alpha_c]^2_{\mathsf{t}_2}) \vee \mathsf{IId} & R_{\mathsf{t}_2} &\overset{\text{def}}{=} G_{\mathsf{t}_1}
\end{aligned}$$

Figure 27. Proof of RGA Client 1.

Figure 27 shows the proof of a client of RGA. Suppose initially the RGA list s is a. We hope to prove a kind of convergence property of the client threads' observations. That is, the right thread t_2 must agree with the left thread t_1 on the order of the operations. So, when the program terminates, if x reads out acb, then y must read out either acb or ac.

We verify the program using our program logic. We first define the rely/guarantee conditions at the bottom of Figure 27. G_{t_1} says that t_1 guarantees the invocation of α_b unconditionally. G_{t_2} is similar. Here we use IId to represent the invocation of an identity action (e.g., read operations). It specifies stuttering steps.

By the PAR rule, we only need to verify each thread independently. For thread t_1, we first stabilize $(s = a)$ under R_{t_1}, resulting in the assertion $p_a \vee (p_a \sqcup [\alpha_c]^2_{t_2})$. Here the definition of p_a allows identity actions only, as specified by Id (defined in Fig. 22 in Sec. 7). After performing `addAfter(a,b)`, the action set must contain $\boxed{\alpha_b}^1_{t_1}$. Then, after `x:=read()`, we know if x reads out acb, then the list object s must be acb.

The verification of t_2 is similar. The only interesting case is the post-condition after `y:=read()`. If at the end the list object s is acb, we know α_b must be ordered before α_c, so it must be the case $p_a \ltimes [\alpha_b]^1_{t_1} \ltimes \boxed{\alpha_c}^2_{t_2}$ at the time of the read. Thus y must read out acb or ac.

F.3 RGA Client 2

```
                              {s = a}
{p_a}                           || {p_a ∨ p_a ⊔ [α_b]^1_t1 ∨ p_a ⊔ ([α_b]^1_t1 ⋉ [α_c]^2_t1)}
addAfter(a, b);                 || x := read();
{p_a ⊔ ☐α_b^1_t1}               || { x ≠ a ⇒
addAfter(a, c);                 ||     x = ab ∧ (p_a ⊔ ☐α_b^1_t1 ∨ p_a ⊔ (☐α_b^1_t1 ⋉ [α_c]^2_t1))
{p_a ⊔ (☐α_b^1_t1 ⋉ ☐α_c^2_t1)} ||     x = ac ∧ p_a ⊔ ([α_b]^1_t1 ⋉ ☐α_c^2_t1)
                                ||     x = acb ∧ p_a ⊔ (☐α_b^1_t1 ⋉ ☐α_c^2_t1) }
                                || y := read();
                                || { x ≠ a ⇒
                                ||     (x = ab ∨ x = ac ∨ x = acb) ∧ (y = x ∨ y = acb) }
         {x ≠ a ⇒ (x = ab ∨ x = ac ∨ x = acb) ∧ (y = x ∨ y = acb)}
```

$$\{s = a\}$$

Left thread:

$\{p_a\}$
`addAfter(a, b);`
$\{p_a \sqcup \boxed{\alpha_b}^1_{t_1}\}$
`addAfter(a, c);`
$\{p_a \sqcup (\boxed{\alpha_b}^1_{t_1} \ltimes \boxed{\alpha_c}^2_{t_1})\}$

Right thread:

$\{p_a \vee p_a \sqcup [\alpha_b]^1_{t_1} \vee p_a \sqcup ([\alpha_b]^1_{t_1} \ltimes [\alpha_c]^2_{t_1})\}$
`x := read();`
$\left\{\begin{array}{l} x \neq a \Longrightarrow \\ \quad x = ab \wedge (p_a \sqcup \boxed{\alpha_b}^1_{t_1} \vee p_a \sqcup (\boxed{\alpha_b}^1_{t_1} \ltimes [\alpha_c]^2_{t_1})) \\ \quad x = ac \wedge p_a \sqcup ([\alpha_b]^1_{t_1} \ltimes \boxed{\alpha_c}^2_{t_1}) \\ \quad x = acb \wedge p_a \sqcup (\boxed{\alpha_b}^1_{t_1} \ltimes \boxed{\alpha_c}^2_{t_1}) \end{array}\right\}$
`y := read();`
$\left\{\begin{array}{l} x \neq a \Longrightarrow \\ \quad (x = ab \vee x = ac \vee x = acb) \wedge (y = x \vee y = acb) \end{array}\right\}$

$$\{x \neq a \Longrightarrow (x = ab \vee x = ac \vee x = acb) \wedge (y = x \vee y = acb)\}$$

$$p_a \overset{\text{def}}{=} (s = a) \wedge \mathrm{Id} \qquad \alpha_b \overset{\text{def}}{=} \texttt{addAfter(a,b)} \qquad \alpha_c \overset{\text{def}}{=} \texttt{addAfter(a,c)}$$

$$G_{t_1} \overset{\text{def}}{=} (\mathrm{true} \rightsquigarrow [\alpha_b]^1_{t_1}) \vee ((\Diamond \boxed{\alpha_b}^1_{t_1}) \rightsquigarrow [\alpha_c]^2_{t_1}) \vee \mathrm{IId} \qquad R_{t_1} \overset{\text{def}}{=} G_{t_2}$$

$$G_{t_2} \overset{\text{def}}{=} \mathrm{IId} \qquad R_{t_2} \overset{\text{def}}{=} G_{t_1}$$

Figure 28. Proof of RGA Client 2.

Figure 29 shows the proof of a client of RGA. Suppose initially the RGA list s is a. We hope to verify the results of the two reads are sensible. The post-condition of the whole program says, if x is not a, then x must read among {ab, ac, acb} and y must be the same as x or get acb. It shows that 1) the right thread t_2 cannot observe abc, and 2) the results of the two consecutive reads must be consistent (i.e., either they are equal, or the latter one observes more operations than the earlier one). Note it is possible that x get ac because we do not assume causal delivery.

We verify the program using our program logic. We first define the rely/guarantee conditions at the bottom of Figure 29. G_{t_1} says that t_1 guarantees the invocation of α_b unconditionally, and the invocation of α_c after it invokes α_b. G_{t_2} is simply IId since t_2 invokes only read operations.

By the PAR rule, we only need to verify each thread independently. For thread t_1, we first stabilize $(s = a)$ under R_{t_1}, resulting in the assertion p_a which allows identity actions. After performing `addAfter(a,b)`, the action set must contain $\boxed{\alpha_b}^1_{t_1}$. Next, after `addAfter(a,c)`, the action set must contain both $\boxed{\alpha_b}^1_{t_1}$ and $\boxed{\alpha_c}^2_{t_1}$, and $\boxed{\alpha_c}^2_{t_1}$ is after $\boxed{\alpha_b}^1_{t_1}$.

For thread t_2, we first stabilize $(s = a)$ under R_{t_2}, resulting in the assertion which consist of three disjunctive branches: p_a, $p_a \sqcup [\alpha_b]^1_{t_1}$ and $p_a \sqcup ([\alpha_b]^1_{t_1} \ltimes [\alpha_c]^2_{t_1})$. Note that in the third branch, t_2 has the knowledge that α_b must be ordered before

$$p \overset{\text{def}}{=} (\mathsf{s} = \mathsf{ae}) \wedge \mathsf{emp} \qquad \alpha_b \overset{\text{def}}{=} \texttt{addAfter(a,b)} \qquad \alpha_c \overset{\text{def}}{=} \texttt{addAfter(a,c)} \qquad \alpha_r \overset{\text{def}}{=} \texttt{remove(e)}$$

$$G_{\mathsf{t}_1} \overset{\text{def}}{=} (\mathsf{true} \rightsquigarrow [\alpha_b]^1_{\mathsf{t}_1}) \qquad G_{\mathsf{t}_2} \overset{\text{def}}{=} ((\Diamond \boxed{\alpha_b}^1_{\mathsf{t}_1}) \rightsquigarrow [\alpha_r]^2_{\mathsf{t}_2})$$

$$G_{\mathsf{t}_3} \overset{\text{def}}{=} ((\Diamond \boxed{\alpha_r}^2_{\mathsf{t}_2}) \rightsquigarrow [\alpha_c]^3_{\mathsf{t}_3}) \qquad R_{\mathsf{t}_2} \overset{\text{def}}{=} G_{\mathsf{t}_1} \vee G_{\mathsf{t}_3} \qquad R_{\mathsf{t}_1} \overset{\text{def}}{=} G_{\mathsf{t}_2} \vee G_{\mathsf{t}_3} \qquad R_{\mathsf{t}_3} \overset{\text{def}}{=} G_{\mathsf{t}_1} \vee G_{\mathsf{t}_2}$$

$\{\mathsf{s} = \mathsf{ae}\}$

```
$\{p\}$
addAfter(a, b);
$\{\mathsf{true}\}$
```

$\|$

```
$\{p \ \vee \ p \sqcup [\alpha_b]^1_{\mathsf{t}_1}\}$
u := read();
if (b ∈ u)
  $\{p \sqcup \boxed{\alpha_b}^1_{\mathsf{t}_1}\}$
  remove(e);
  $\{p \sqcup \boxed{\alpha_b}^1_{\mathsf{t}_1} \sqcup \boxed{\alpha_r}^2_{\mathsf{t}_2} \ \vee \ p \sqcup \boxed{\alpha_b}^1_{\mathsf{t}_1} \sqcup \boxed{\alpha_r}^2_{\mathsf{t}_2} \sqcup [\alpha_c]^3_{\mathsf{t}_3}\}$
x := read();
$\{\mathsf{c} \in \mathsf{x} \Rightarrow \mathsf{s} = \mathsf{x} = \mathsf{acb} \vee \mathsf{s} = \mathsf{x} = \mathsf{abc}\}$
```

$\|$

```
$\{p \ \vee \ p \sqcup [\alpha_b]^1_{\mathsf{t}_1} \ \vee p \sqcup [\alpha_b]^1_{\mathsf{t}_1} \sqcup [\alpha_r]^2_{\mathsf{t}_2}\}$    (1)
v := read();
if (e ∉ v)
  $\{p \sqcup [\alpha_b]^1_{\mathsf{t}_1} \sqcup \boxed{\alpha_r}^2_{\mathsf{t}_2}\}$    (2)
  addAfter(a, c);
  $\{p \sqcup [\alpha_b]^1_{\mathsf{t}_1} \sqcup \boxed{\alpha_r}^2_{\mathsf{t}_2} \sqcup \boxed{\alpha_c}^3_{\mathsf{t}_3}\}$    (3)
y := read();
$\{(\mathsf{s} = \mathsf{acb} \vee \mathsf{s} = \mathsf{abc}) \Rightarrow (\mathsf{y} = \mathsf{s} \vee \mathsf{y} = \mathsf{ac})\}$    (4)
```

$\{\mathsf{c} \in \mathsf{x} \Rightarrow (\mathsf{s} = \mathsf{x}) \wedge (\mathsf{x} = \mathsf{abc} \vee \mathsf{x} = \mathsf{acb}) \wedge (\mathsf{y} = \mathsf{x} \vee \mathsf{y} = \mathsf{ac})\}$

Figure 29. Proof of RGA Client 3.

α_c, because we require all nodes to observe the same ordering of the conflicting operations α_b and α_c. After `x:=read()`, we analyze each branch and get the value of `x`. Finally, after `y:=read()`, we know the post-condition holds.

F.4 RGA Client 3

Fig. 29 gives the proof of the last example of RGA client. We first define the rely/guarantee conditions of each thread. G_{t_1} says that the thread t_1 guarantees the invocation of α_b unconditionally. G_{t_2} says that t_2 calls α_r after it receives (commits) α_b. Similarly, G_{t_3} says that t_3 calls α_c after it commits α_r.

By the PAR rule, we only need to verify each thread independently. For thread t_3, we first stabilize p under R_{t_3}, resulting in the assertion (1) in Fig. 12. After reading out the removal of e, we can discard the branches where α_r is not committed. So we get the assertion (2). Then, t_3 calls `addAfter(a,c)`. By the CALL rule, the immediate post-condition is $(p \sqcup [\alpha_b]^1_{\mathsf{t}_1} \sqcup \boxed{\alpha_r}^2_{\mathsf{t}_2}, \bowtie) \ltimes \boxed{\alpha_c}^3_{\mathsf{t}_3}$. Using the CSQ rule, we weaken it to the assertion (3), which is stable and cmt-closed. Finally we get the assertion (4). It has the branch $\mathsf{y} = \mathsf{ac}$ because it is possible that t_3 has not yet committed α_b by the read.

F.5 Register Client 1

$$\{\mathsf{s} = 0\}$$

$$\begin{array}{l||l}
\{p_a \vee (p_a \sqcup [\alpha_2]^2_{\mathsf{t}_2})\} & \{p_a \vee (p_a \sqcup [\alpha_1]^1_{\mathsf{t}_1})\} \\
\mathtt{write(1);} & \mathtt{write(2);} \\
\left\{(p_a \sqcup \boxed{\alpha_1}^1_{\mathsf{t}_1}) \vee (p_a \sqcup \boxed{\alpha_1}^1_{\mathsf{t}_1} \sqcup [\alpha_2]^2_{\mathsf{t}_2})\right\} & \left\{(p_a \sqcup \boxed{\alpha_2}^2_{\mathsf{t}_2}) \vee (p_a \sqcup \boxed{\alpha_2}^2_{\mathsf{t}_2} \sqcup [\alpha_1]^1_{\mathsf{t}_1})\right\} \\
\mathtt{x\ :=\ read();} & \mathtt{y\ :=\ read();} \\
\left\{\mathsf{x} = 2 \Rightarrow p_a \sqcup (\boxed{\alpha_1}^1_{\mathsf{t}_1} \ltimes \boxed{\alpha_2}^2_{\mathsf{t}_2})\right\} & \left\{\mathsf{y} \neq 2 \Rightarrow p_a \sqcup (\boxed{\alpha_2}^2_{\mathsf{t}_2} \ltimes \boxed{\alpha_1}^1_{\mathsf{t}_1})\right\} \\
\Rrightarrow & \Rrightarrow \\
\{\mathsf{x} = 2 \Rightarrow \mathsf{s} = 2\} & \{\mathsf{s} = 2 \Rightarrow \mathsf{y} = 2\}
\end{array}$$

$$\{\mathsf{x} = 2 \Rightarrow \mathsf{y} = 2 \wedge \mathsf{s} = 2\}$$

$$p_a \overset{\text{def}}{=} (\mathsf{s} = 0) \wedge \mathsf{Id} \qquad \alpha_1 \overset{\text{def}}{=} \mathtt{write(1)} \qquad \alpha_2 \overset{\text{def}}{=} \mathtt{write(2)}$$

$$\begin{array}{llll}
G_{\mathsf{t}_1} \overset{\text{def}}{=} (\mathsf{true} \rightsquigarrow [\alpha_1]^1_{\mathsf{t}_1}) \vee \mathsf{IId} & & R_{\mathsf{t}_1} \overset{\text{def}}{=} G_{\mathsf{t}_2} \\
G_{\mathsf{t}_2} \overset{\text{def}}{=} (\mathsf{true} \rightsquigarrow [\alpha_2]^2_{\mathsf{t}_2}) \vee \mathsf{IId} & & R_{\mathsf{t}_2} \overset{\text{def}}{=} G_{\mathsf{t}_1}
\end{array}$$

Figure 30. Proof of Register Client 1.

Figure 30 shows the proof of a client of a register. Suppose initially the register s contains 0. We hope to prove the post-condition $(\mathsf{x} = 2 \Rightarrow \mathsf{y} = 2 \wedge \mathsf{s} = 2)$ holds, which shows a kind of convergence property of the client threads' observations. Intuitively, if x reads out 2, then the left thread t_1 must see `write(2)` from the right thread after its own `write(1)`. The right thread t_2 must observe the same ordering, so y must read out 2 too and the final register s also contains 2.

To verify the program, we first define the rely/guarantee conditions at the bottom of Figure 30. G_{t_1} says that t_1 guarantees the invocation of $[\alpha_1]^1_{\mathsf{t}_1}$ unconditionally. G_{t_2} is similar.

By the PAR rule, we only need to verify each thread independently. For thread t_1, we first stabilize $(\mathsf{s} = 0)$ under R_{t_1}, resulting in the assertion $p_a \vee (p_a \sqcup [\alpha_2]^2_{\mathsf{t}_2})$. After performing `write(1)`, the action set must contain $\boxed{\alpha_1}^1_{\mathsf{t}_1}$. Then, after `x:=read()`, if $\mathsf{x} = 2$, we know t_1 must have received (committed) $[\alpha_2]^2_{\mathsf{t}_2}$ and ordered it after its own $\boxed{\alpha_1}^1_{\mathsf{t}_1}$, which implies $\mathsf{s} = 2$. The verification of t_2 is similar. By conjoining the post-conditions of the two threads, we derive $(\mathsf{x} = 2 \Rightarrow \mathsf{y} = 2 \wedge \mathsf{s} = 2)$.

F.6 Register Client 2

$$\{\mathsf{s} = 0\}$$

$$\begin{array}{l||l||l}
 & & \left\{\begin{array}{l} p_a \vee p_a \sqcup [\alpha_1]^1_{\mathsf{t}_1} \vee p_a \sqcup [\alpha_2]^2_{\mathsf{t}_2} \\ \quad \vee p_a \sqcup [\alpha_1]^1_{\mathsf{t}_1} \sqcup [\alpha_2]^2_{\mathsf{t}_2} \end{array}\right\} \\
 & & \texttt{x := read();} \\
\{\mathsf{true}\} & \{\mathsf{true}\} & \left\{\begin{array}{l} \mathsf{x} = 1 \Rightarrow \\ \quad p_a \sqcup \boxed{\alpha_1}^1_{\mathsf{t}_1} \vee p_a \sqcup \boxed{\alpha_1}^1_{\mathsf{t}_1} \sqcup [\alpha_2]^2_{\mathsf{t}_2} \end{array}\right\} \\
\texttt{write(1);} & \texttt{write(2);} & \texttt{y := read();} \\
\{\mathsf{true}\} & \{\mathsf{true}\} & \left\{\begin{array}{l} \mathsf{x} = 1 \wedge \mathsf{y} = 2 \Rightarrow \\ \quad p_a \sqcup (\boxed{\alpha_1}^1_{\mathsf{t}_1} \ltimes \boxed{\alpha_2}^2_{\mathsf{t}_2}) \end{array}\right\} \\
 & & \texttt{z := read();} \\
 & & \{\mathsf{x} = 1 \wedge \mathsf{y} = 2 \Rightarrow \mathsf{z} = 2\}
\end{array}$$

$$\{\mathsf{x} = 1 \wedge \mathsf{y} = 2 \Rightarrow \mathsf{z} = 2\}$$

$$p_a \overset{\text{def}}{=} (\mathsf{s} = 0) \wedge \mathsf{Id} \qquad \alpha_1 \overset{\text{def}}{=} \texttt{write(1)} \qquad \alpha_2 \overset{\text{def}}{=} \texttt{write(2)}$$

$$\begin{array}{ll}
G_{\mathsf{t}_1} \overset{\text{def}}{=} (\mathsf{true} \rightsquigarrow [\alpha_1]^1_{\mathsf{t}_1}) \vee \mathsf{IId} & R_{\mathsf{t}_1} \overset{\text{def}}{=} G_{\mathsf{t}_2} \vee G_{\mathsf{t}_3} \\
G_{\mathsf{t}_2} \overset{\text{def}}{=} (\mathsf{true} \rightsquigarrow [\alpha_2]^2_{\mathsf{t}_2}) \vee \mathsf{IId} & R_{\mathsf{t}_2} \overset{\text{def}}{=} G_{\mathsf{t}_1} \vee G_{\mathsf{t}_3} \\
G_{\mathsf{t}_3} \overset{\text{def}}{=} \mathsf{IId} & R_{\mathsf{t}_3} \overset{\text{def}}{=} G_{\mathsf{t}_1} \vee G_{\mathsf{t}_2}
\end{array}$$

Figure 31. Proof of Register Client 2.

Figure 31 shows the proof of a client of a register. We first define the rely/guarantee conditions at the bottom of Figure 31. G_{t_1} says that t_1 guarantees the invocation of $[\alpha_1]^1_{\mathsf{t}_1}$ unconditionally. G_{t_2} says that t_2 guarantees the invocation of $[\alpha_2]^2_{\mathsf{t}_2}$ unconditionally. G_{t_3} is simply IId since t_3 invokes only read operations.

For the right thread t_3, if $\mathsf{x} = 1 \wedge \mathsf{y} = 2$, we know t_3 must have received (committed) $[\alpha_1]^1_{\mathsf{t}_1}$ and $[\alpha_2]^2_{\mathsf{t}_2}$ and ordered $[\alpha_2]^2_{\mathsf{t}_2}$ after $[\alpha_1]^1_{\mathsf{t}_1}$ (as shown in the assertion after `y:=read()`), so it must get $\mathsf{z} = 2$ after the final `z:=read()`.

$$
\begin{array}{lcll}
\rightarrowtail & \in & \mathscr{P}(\mathit{Effector} \times \mathit{Effector}) & \text{(the time-stamp order)} \\
\mathcal{V} & \in & \mathit{LocalState} \to \mathscr{P}(\mathit{Effector}) & \text{(the view function)}
\end{array}
$$

$$
\begin{array}{rl}
\mathsf{followTS}(\mathcal{S}, \delta, \rightarrowtail, \mathcal{V}) & \text{iff } \forall \delta'.\ \delta' \in \mathcal{V}(\mathcal{S}) \implies \neg(\delta \rightarrowtail \delta') \\
\mathsf{valid}_\Pi(f, n, \delta) & \text{iff } \exists \mathcal{S}.\ \Pi(f, n)(\mathcal{S}) = (\_, \delta) \\
\mathsf{genAt}_\Pi(\mathcal{S}, \delta) & \text{iff } \exists f, n.\ \Pi(f, n)(\mathcal{S}) = (\_, \delta)
\end{array}
$$

Figure 32. Auxiliary Definitions for CRDTs with Time-Stamps.

G Proof Method for ACC and Soundness

G.1 Formalization of the Proof Method

We give a proof method for verifying ACC of CRDT algorithms. As we have explained, ACC captures both SEC and functional correctness. CRDT algorithms use commutative effectors to achieve SEC. On functional correctness, they usually apply a specific strategy to resolve conflicts, such as time-stamps, so that executing the effectors in any order can correspond to the same sequence of abstract operations ordered by the strategy. To guide the verification, we ask users to specify the conflict-resolution strategy $\rightarrowtail$ (called the time-stamp order), which is a *partial order* between effectors.

Besides, we hope our proof method is local in that the reasoning of each execution step relies on the current local state on the node only, without referring to the execution traces. To this end, we introduce a "view" function $\mathcal{V}$ mapping each local state $\mathcal{S}$ to a set of effectors that must have been applied before reaching $\mathcal{S}$. The function $\mathcal{V}$ is application-dependent and needs to be provided by programmers, just like $\rightarrowtail$. For the RGA algorithm, $\mathcal{V}$ can be defined as follows:

$$
\begin{array}{rl}
\mathcal{V}(\mathcal{S}) \stackrel{\text{def}}{=} & \{\delta \mid \exists \mathtt{a}, \mathtt{i}, \mathtt{b}.\ ((\mathtt{a}, \mathtt{i}, \mathtt{b}) \in \mathcal{S}(\mathtt{N})) \\
& \wedge (\delta = \mathtt{AddAfter(a,i,b)}) \vee \exists \mathtt{a}.\ (\mathtt{a} \in \mathcal{S}(\mathtt{T})) \wedge (\delta = \mathtt{Rmv(a)})\}
\end{array}
$$

The types of $\rightarrowtail$ and $\mathcal{V}$ are given at the top of Fig. 32. The main proof obligations of our method are formulated as effComm, sameRVal and lockStep-S below. We assume that all the δ-s mentioned in the conditions are generated from some valid operation calls of Π. In certain cases we need to specify extra well-formedness of initial states. Then we introduce the state mapping ψ for this purpose. It is stronger than φ and is applied only to the initial state.

The condition sameRVal requires that the corresponding operations in Π and Γ executed at φ-related states should return the same value. The condition lockStep-TS specifies the state correspondence if the effectors are applied in the order of their time-stamps. We define followTS (see Fig. 32) to characterize the executions where effectors are applied in $\rightarrowtail$ order. In $\mathsf{followTS}(\mathcal{S}, \delta, \rightarrowtail, \mathcal{V})$, $\mathcal{S}$ is supposed to be the state over which δ applies. It says, δ does *not* have a time-stamp smaller (following the order $\rightarrowtail$) than earlier effectors δ' that can be seen at $\mathcal{S}$. In other words, executing δ at state $\mathcal{S}$ does not violate $\rightarrowtail$. Fig. 32 also defines $\mathsf{valid}_\Pi(f, n, \delta)$ and $\mathsf{genAt}_\Pi(\mathcal{S}, \delta)$ used below.

Definition 59 (Commutativity of Effectors). $\mathsf{effComm}_\varphi$ iff $\forall \delta, \delta'.\ \mathsf{commute}_\varphi(\delta, \delta')$, where $\mathsf{commute}_\varphi(\delta, \delta')$ iff $\forall \mathcal{S}, \mathcal{S}'.\ \delta(\delta'(\mathcal{S})) = \mathcal{S}' \implies \varphi(\delta'(\delta(\mathcal{S}))) = \varphi(\mathcal{S}')$.

Definition 60 (Same Return Values). $\mathsf{sameRVal}_\varphi(\Pi, \Gamma)$ iff

$$
\begin{array}{c}
\forall f, n, n', \mathcal{S}, \mathcal{S}_a.\ \varphi(\mathcal{S}) = \mathcal{S}_a \wedge \Pi(f, n)(\mathcal{S}) = (n', \_) \\
\implies \Gamma(f, n)(\mathcal{S}_a) = (n', \_)
\end{array}
$$

Definition 61 (φ-Preservation). $\mathsf{lockStep\text{-}TS}_\varphi(\Pi, \Gamma, \rightarrowtail, \mathcal{V})$ iff $\forall f, n, \delta, \mathcal{S}, \mathcal{S}', \mathcal{S}_a$,

$$
\begin{array}{c}
\mathsf{valid}_\Pi(f, n, \delta) \wedge \varphi(\mathcal{S}) = \mathcal{S}_a \wedge \delta(\mathcal{S}) = \mathcal{S}' \wedge \mathsf{followTS}(\mathcal{S}, \delta, \rightarrowtail, \mathcal{V}) \\
\implies \exists \mathcal{S}'_a.\ \varphi(\mathcal{S}') = \mathcal{S}'_a \wedge \Gamma(f, n)(\mathcal{S}_a) = (\_, \mathcal{S}'_a)
\end{array}
$$

We also need to ensure that the user-specified $\rightarrowtail$ and $\mathcal{V}$ make sense. We define a set of conditions for well-formedness check in wfV and wfTS below.

Definition 62 (Well-formed $\mathcal{V}$). $\mathsf{wfV}_\psi(\mathcal{V})$ iff the following hold:

1. No effectors can be seen at the initial state.
$$\forall \mathcal{S}.\ \mathcal{S} \in \mathit{dom}(\psi) \implies \mathcal{V}(\mathcal{S}) = \emptyset$$
2. $\mathcal{V}$ cannot increase arbitrarily:
$$\forall \delta, \mathcal{S}, \mathcal{S}'.\ \delta(\mathcal{S}) = \mathcal{S}' \implies \mathcal{V}(\mathcal{S}') \subseteq (\mathcal{V}(\mathcal{S}) \cup \{\delta\})$$

Definition 63 (Well-formed $\rightarrowtail$). $\mathsf{wfTS}_\Pi(\rightarrowtail, \mathcal{V}, (\Gamma, \bowtie))$ iff the following hold:

1. $\rightarrowtail$ is consistent with the visibility order on the current node, in the sense that when δ is generated for a client request at state $\mathcal{S}$, it cannot have a smaller time-stamp than earlier effectors seen at $\mathcal{S}$. That is, $\forall \mathcal{S}, \delta.\ \mathsf{genAt}_\Pi(\mathcal{S}, \delta) \implies \mathsf{followTS}(\mathcal{S}, \delta, \rightarrowtail, \mathcal{V})$.

$e_1 \rightarrowtail e_2$ iff $\mathsf{eff}(e_1) \rightarrowtail \mathsf{eff}(e_2)$

$\mathsf{conflict\text{-}ts}(\rightarrowtail, (\Gamma, \bowtie))$ iff
$\quad \forall f_1, n_1, \delta_1, f_2, n_2, \delta_2.\ (\Gamma \models (f_1, n_1) \bowtie (f_2, n_2)) \wedge \mathsf{valid}_\Pi(f_1, n_1, \delta_1) \wedge \mathsf{valid}_\Pi(f_2, n_2, \delta_2)$
$\quad \Longrightarrow \delta_1 \rightarrowtail \delta_2 \vee \delta_2 \rightarrowtail \delta_1$

$\mathsf{effGenFollowTS}_\Pi(\rightarrowtail, \mathcal{V})$ iff $\forall \mathcal{S}, \delta.\ \mathsf{genAt}_\Pi(\mathcal{S}, \delta) \Longrightarrow \mathsf{followTS}(\mathcal{S}, \delta, \rightarrowtail, \mathcal{V})$

$\mathsf{wfTSV}(\rightarrowtail, \mathcal{V})$ iff
$\quad (\forall \delta, \mathcal{S}, \mathcal{S}'.\ (\delta(\mathcal{S}) = \mathcal{S}') \Longrightarrow \forall \delta'.\ \delta' \in (\mathcal{V}(\mathcal{S}) - \mathcal{V}(\mathcal{S}')) \Longrightarrow (\delta' \rightarrowtail \delta) \wedge (\delta \in \mathcal{V}(\mathcal{S}')))$
$\quad \wedge\ (\forall \delta, \mathcal{S}, \mathcal{S}'.\ (\delta(\mathcal{S}) = \mathcal{S}') \wedge \delta \in \mathsf{image}(\rightarrowtail) \wedge \delta \notin \mathcal{V}(\mathcal{S}') \Longrightarrow \exists \delta'.\ (\delta \rightarrowtail \delta') \wedge (\delta' \in \mathcal{V}(\mathcal{S}')))$

$\mathsf{vts}(\mathsf{t}, \mathcal{E}, \rightarrowtail) \overset{\text{def}}{=} (\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \cup \mathsf{TS}_{\mathcal{E}}^{\rightarrowtail})^+$

$\mathsf{TS}_{\mathcal{E}}^{\rightarrowtail}(e, e')$ iff $e \rightarrowtail e' \wedge \{e, e'\} \subseteq \mathsf{orig}(\mathcal{E})$

$\mathsf{RValRelated}(\mathsf{t}, \mathcal{E}, (\Gamma, \mathcal{S}_a, ar))$ iff
$\quad \forall \mathcal{E}', e.\ \mathcal{E}' \leqslant \mathcal{E} \wedge \mathsf{last}(\mathcal{E}') = e \wedge \mathsf{is\_orig}_\mathsf{t}(e) \Longrightarrow \mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar)$

$\mathsf{StRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathcal{S}_a, ar))$ iff
$\quad \forall \mathcal{E}'.\ \mathcal{E}' \leqslant \mathcal{E} \Longrightarrow \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_\mathsf{t})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar)$

$\mathsf{execFollowTS}(\mathcal{S}, \mathcal{E}, \rightarrowtail, \mathcal{V})$ iff
$\quad \forall \mathcal{E}'.e, \mathcal{S}'.\ (\mathcal{E}' {+}{+} [e] \leqslant \mathcal{E}) \wedge \mathsf{exec\_st}(\mathcal{S}, \mathcal{E}') = \mathcal{S}' \Longrightarrow \mathsf{followTS}(\mathcal{S}', \mathsf{eff}(e), \rightarrowtail, \mathcal{V})$

$\mathsf{cyclic}(rel)$ iff $\exists n, e_1, \ldots, e_n.\ (\forall i \in [1..n-1].\ (e_i, e_{i+1}) \in rel) \wedge (e_n, e_1) \in rel$

Figure 33. Auxiliary Definitions for the Soundness Proof of the Proof Method with Time-Stamps.

2. Any effector δ' which disappears after applying δ must have a smaller time-stamp than δ in $\rightarrowtail$, and δ should be seen at the resulting state. That is,
$$\forall \delta, \mathcal{S}, \mathcal{S}'.\ (\delta(\mathcal{S}) = \mathcal{S}') \wedge \forall \delta'.\ \delta' \in (\mathcal{V}(\mathcal{S}) - \mathcal{V}(\mathcal{S}')) \Longrightarrow (\delta' \rightarrowtail \delta) \wedge (\delta \in \mathcal{V}(\mathcal{S}'))$$
3. If a time-stamped effector δ is not seen after applied, then one must see some δ' with a higher time-stamp. Here $\mathsf{image}(\rightarrowtail) \overset{\text{def}}{=} \{\delta \mid \exists \delta'.\ \delta' \rightarrowtail \delta\}$ denotes the set of time-stamped effectors.
$$\forall \delta, \mathcal{S}, \mathcal{S}'.\ (\delta(\mathcal{S}) = \mathcal{S}') \wedge \delta \in \mathsf{image}(\rightarrowtail) \wedge \delta \notin \mathcal{V}(\mathcal{S}') \Longrightarrow \exists \delta'.\ (\delta \rightarrowtail \delta') \wedge (\delta' \in \mathcal{V}(\mathcal{S}'))$$
4. Conflicting operations must be ordered by $\rightarrowtail$. That is,
$$\forall f_1, n_1, \delta_1, f_2, n_2, \delta_2.\ \mathsf{valid}_\Pi(f_1, n_1, \delta_1) \wedge \mathsf{valid}_\Pi(f_2, n_2, \delta_2) \wedge ((f_1, n_1) \bowtie_\Gamma (f_2, n_2)) \Longrightarrow \delta_1 \rightarrowtail \delta_2 \vee \delta_2 \rightarrowtail \delta_1$$

Definition 64 (Proof Obligations). $\mathsf{CRDT\text{-}TS}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie))$ iff there exist $\rightarrowtail$ and $\mathcal{V}$ such that
$$\mathsf{effComm}_\varphi \wedge \mathsf{sameRVal}_\varphi(\Pi, \Gamma) \wedge \mathsf{lockStep\text{-}TS}_\varphi(\Pi, \Gamma, \rightarrowtail, \mathcal{V}) \wedge \mathsf{wfV}_\psi(\mathcal{V}) \wedge \mathsf{wfTS}_\Pi(\rightarrowtail, \mathcal{V}, (\Gamma, \bowtie)).$$

G.2 Soundness of the Proof Method

Proof of Theorem 8. By applying Lemma 69 and Lemma 68. □

Definition 65 (Eventual Delivery). $\mathsf{eventualDelivery}(\mathcal{E})$ iff
$$\forall e, \mathsf{t}.\ e \in \mathcal{E} \wedge \mathsf{is\_orig}_\mathsf{t}(e) \Longrightarrow \forall \mathsf{t}' \neq \mathsf{t}.\ \exists e'.\ e \xrightarrow{\mathsf{t}'}_{\mathcal{E}} e'$$

Lemma 66. If $\mathsf{eventualDelivery}(\mathcal{E})$, then $\forall \mathsf{t}.\ \mathsf{visible}(\mathcal{E}, \mathsf{t}) = \mathsf{orig}(\mathcal{E})$.

Definition 67 (E-ACC)**.** $\mathsf{E\text{-}ACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie))$, iff

$$\begin{array}{l}\forall \mathcal{S}, \mathcal{S}_a, \mathcal{E}.\ \mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S}) \wedge \mathsf{eventualDelivery}(\mathcal{E}) \wedge \psi(\mathcal{S}) = \mathcal{S}_a \\ \qquad \Longrightarrow\ \mathsf{ACT}_{\varphi}(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie))\end{array}$$

Lemma 68 (E-ACC implies ACC)**.** If $\mathsf{E\text{-}ACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie))$, then $\mathsf{ACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie))$.

Proof. For any $\mathcal{S}$, $\mathcal{S}_a$ and $\mathcal{E}$, if $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$ and $\psi(\mathcal{S}) = \mathcal{S}_a$, we know there exist $\mathcal{E}'$ and $\mathcal{E}''$ such that

$$\mathcal{E}' = \mathcal{E}{+}{+}\mathcal{E}'', \forall e \in \mathcal{E}''.\ \mathsf{is\_recv}(e), \mathcal{E}' \in \mathcal{T}(\Pi, \mathcal{S}) \text{ and } \mathsf{eventualDelivery}(\mathcal{E}').$$

By $\mathsf{E\text{-}ACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie))$, we know

$$\mathsf{ACT}_{\varphi}(\mathcal{E}', \mathcal{S}, (\Gamma, \bowtie)).$$

From $\mathsf{ACT}_{\varphi}(\mathcal{E}', \mathcal{S}, (\Gamma, \bowtie))$, we know there exist $ar'_1, \ldots, ar'_n$ such that, for any t, we have

$$\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E}',\mathsf{t})}(ar'_{\mathsf{t}}),\ \underset{\mathsf{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}'} \subseteq ar'_{\mathsf{t}}, \mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}', \mathcal{S}), (\Gamma, ar'_{\mathsf{t}})), \forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{Coh}(ar'_{\mathsf{t}}, ar'_{\mathsf{t}'}, (\Gamma, \bowtie)).$$

Since $\mathcal{E}' = \mathcal{E}{+}{+}\mathcal{E}''$ and $\forall e \in \mathcal{E}''.\ \mathsf{is\_recv}(e)$, we know

$$\mathsf{orig}(\mathcal{E}') = \mathsf{orig}(\mathcal{E}).$$

Let $ar_{\mathsf{t}} = ar'_{\mathsf{t}}|_{\mathsf{visible}(\mathcal{E},\mathsf{t})}$. From $\underset{\mathsf{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}'} \subseteq ar'_{\mathsf{t}}$, we know

$$\underset{\mathsf{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}} \subseteq ar_{\mathsf{t}}.$$

From $\mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}', \mathcal{S}), (\Gamma, ar'_{\mathsf{t}}))$, we know

$$\mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathsf{t}})).$$

For any $\mathsf{t}' \neq \mathsf{t}$, from $\mathsf{Coh}(ar'_{\mathsf{t}}, ar'_{\mathsf{t}'}, (\Gamma, \bowtie))$, we know

$$\mathsf{Coh}(ar_{\mathsf{t}}, ar_{\mathsf{t}'}, (\Gamma, \bowtie)).$$

Thus $\mathsf{ACT}_{\varphi}(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie))$. Thus we are done. □

Lemma 69 (CRDT-TS implies E-ACC)**.**
If $\mathsf{CRDT\text{-}TS}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie))$ and $\psi \Rightarrow \varphi$, then $\mathsf{E\text{-}ACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie))$.

Proof. For any $\mathcal{S}$, $\mathcal{S}_a$ and $\mathcal{E}$, suppose $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\mathsf{eventualDelivery}(\mathcal{E})$ and $\psi(\mathcal{S}) = \mathcal{S}_a$.
By $\mathsf{CRDT\text{-}TS}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie))$, we know there exist $\rightarrowtail$ and $\mathcal{V}$ such that

$$\mathsf{effComm}_{\varphi}, \mathsf{sameRVal}_{\varphi}(\Pi, \Gamma), \mathsf{lockStep\text{-}TS}_{\varphi}(\Pi, \Gamma, \rightarrowtail, \mathcal{V}), \mathsf{wfV}_{\psi}(\mathcal{V}), \mathsf{wfTS}_{\Pi}(\rightarrowtail, \mathcal{V}, (\Gamma, \bowtie)).$$

Below we prove $\mathsf{ACT}_{\varphi}(\mathcal{E}, \mathcal{S}, ((\Gamma, \bowtie), \mathcal{S}_a))$. For any t, we first define $\mathsf{vts}(\mathsf{t}, \mathcal{E}, \rightarrowtail)$ in Figure 33. By Lemma 71, we know

$$\mathsf{partialOrder}(\mathsf{vts}(\mathsf{t}, \mathcal{E}, \rightarrowtail)).$$

So there exists ar_{t} such that $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar_{\mathsf{t}})$ and $\mathsf{vts}(\mathsf{t}, \mathcal{E}, \rightarrowtail) \subseteq ar_{\mathsf{t}}$. By Lemma 66, we know $\mathsf{totalOrder}_{\mathsf{visible}(\mathcal{E},\mathsf{t})}(ar_{\mathsf{t}})$.
Also,

$$\underset{\mathsf{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}} \subseteq ar_{\mathsf{t}} \text{ and } \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}} \subseteq ar_{\mathsf{t}}.$$

- Below we prove $\mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathsf{t}}))$.
  We first prove $\mathsf{StRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathcal{S}_a, ar_{\mathsf{t}}))$ by applying Lemma 75. Then, by Lemma 74, we know $\mathsf{RValRelated}(\mathsf{t}, \mathcal{E}, (\Gamma, \mathcal{S}_a, ar_{\mathsf{t}}))$.
  Thus $\mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathsf{t}}))$.
- We prove $\forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{Coh}(ar_{\mathsf{t}}, ar_{\mathsf{t}'}, (\Gamma, \bowtie))$ by Lemma 70.

Thus we are done. □

Lemma 70 (Coherence)**.** For any ar, ar', t, t′ and $\mathcal{E}$, if

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$,
2. $\mathsf{conflict\text{-}ts}(\rightarrowtail, (\Gamma, \bowtie))$,
3. $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar)$, $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar')$, $\mathsf{TS}^{\rightarrowtail}_{\mathcal{E}} \subseteq ar$, $\mathsf{TS}^{\rightarrowtail}_{\mathcal{E}} \subseteq ar'$,

then $\mathsf{Coh}(ar, ar', (\Gamma, \bowtie))$.

Proof. For any e_0 and e_1, if $e_0\ ar\ e_1$ and $e_1\ ar'\ e_0$, since $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar)$ and $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar')$, we know

$$\{e_0, e_1\} \subseteq \mathsf{orig}(\mathcal{E}),\ (e_1, e_0) \notin ar \text{ and } (e_0, e_1) \notin ar'.$$

Since $\mathsf{TS}^{\rightarrowtail}_{\mathcal{E}} \subseteq ar$ and $\mathsf{TS}^{\rightarrowtail}_{\mathcal{E}} \subseteq ar'$, we know

$$(e_1, e_0) \notin \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}} \text{ and } (e_0, e_1) \notin \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}}.$$

Thus we know

$$\neg(e_1 \rightarrowtail e_0) \text{ and } \neg(e_0 \rightarrowtail e_1).$$

Since $\mathsf{conflict\text{-}ts}(\rightarrowtail, (\Gamma, \bowtie))$, we know

$$\neg(e_0 \bowtie_\Gamma e_1).$$

Thus we are done. □

Lemma 71 (vts is a partial order)**.** If

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\mathsf{eventualDelivery}(\mathcal{E})$,
2. $\mathsf{partialOrder}(\rightarrowtail)$, $\mathsf{effGenFollowTS}_\Pi(\rightarrowtail, \mathcal{V})$, $\mathsf{wfTSV}(\rightarrowtail, \mathcal{V})$,

then $\mathsf{partialOrder}(\mathsf{vts}(\mathsf{t}, \mathcal{E}, \rightarrowtail))$.

Proof. By the definition of vts, we know $\mathsf{transitive}(\mathsf{vts}(\mathsf{t}, \mathcal{E}, \rightarrowtail))$.

Below we prove $\mathsf{irreflexive}(\mathsf{vts}(\mathsf{t}, \mathcal{E}, \rightarrowtail))$. So we only need to prove: $\neg\mathsf{cyclic}(\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \cup \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}})$.

By contradiction. Suppose there exist $n, e_1, \ldots, e_n$ such that $\forall i \in [1..n{-}1].\ (e_i, e_{i+1}) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \cup \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}}$ and $(e_n, e_1) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \cup \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}}$. Without loss of generality, we can suppose n is the length of the smallest cycle. We analyze the following two cases:

- $n = 1$. We know it is impossible from $\mathsf{partialOrder}(\rightarrowtail)$.
- $n > 1$.
  Since $\mathsf{eventualDelivery}(\mathcal{E})$, we know
  $$\{e_1, \ldots, e_n\} \subseteq \mathsf{visible}(\mathcal{E}, \mathsf{t}).$$
  Without loss of generality, we can suppose e_n is the last event among $e_1, \ldots, e_n$ that t applies, that is, $\forall i \in [1..n - 1].\ e_i \Subset^{\mathsf{t}}_{\mathcal{E}} e_n$.
  Since $(e_n, e_1) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \cup \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}}$, we know
  $$(e_n, e_1) \in \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}}.$$
  Thus we know
  $$\mathsf{eff}(e_n) \rightarrowtail \mathsf{eff}(e_1).$$
  Next we do case analysis of $(e_{n-1}, e_n) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \cup \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}}$.
  - $(e_{n-1}, e_n) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$.
    Thus we know $\mathsf{is\_orig}_{\mathsf{t}}(e)$. Thus $(e_1, e_n) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$. Since $\mathsf{effGenFollowTS}_\Pi(\rightarrowtail, \mathcal{V})$ and $\mathsf{wfTSV}(\rightarrowtail, \mathcal{V})$, by Lemma 72, we know
    $$\neg(\mathsf{eff}(e_n) \rightarrowtail \mathsf{eff}(e_1)).$$
    Then we reach a contradiction.
  - $(e_{n-1}, e_n) \in \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}}$.
    Since $\mathsf{partialOrder}(\rightarrowtail)$, we know $\mathsf{eff}(e_{n-1}) \rightarrowtail \mathsf{eff}(e_1)$. Thus we know $(e_{n-1}, e_1) \in \mathsf{TS}^{\rightarrowtail}_{\mathcal{E}}$. Thus we have constructed a cycle of length $n - 1$: $e_1, \ldots, e_{n-1}, e_1$. It contradicts the assumption that n is the length of the smallest cycle.

Thus we are done. □

Lemma 72 ($\rightarrowtail$ and $\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$ do not conflict)**.** If

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$,
2. $\mathsf{partialOrder}(\rightarrowtail)$, $\mathsf{effGenFollowTS}_\Pi(\rightarrowtail, \mathcal{V})$, $\mathsf{wfTSV}(\rightarrowtail, \mathcal{V})$,

3. $e_1 \overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_{\mathcal{E}} e_2$,

then $\neg(\mathsf{eff}(e_2) \rightarrowtail \mathsf{eff}(e_1))$.

Proof. By contradiction. Suppose $\mathsf{eff}(e_2) \rightarrowtail \mathsf{eff}(e_1)$.

Suppose $\mathsf{eff}(e_1) = \delta_1$, $\mathsf{eff}(e_2) = \delta_2$ and $\mathsf{op}(e_2) = (f, n)$. By the operational semantics, we know there exist $\mathcal{E}_2$ and $\mathcal{S}_2$ such that

$$\Pi(f,n)(\mathcal{S}_2) = (\_, \delta_2),\ \mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_2) = \mathcal{S}_2,\ \mathcal{E}_2{+}{+}[e_2] \leqslant (\mathcal{E}|_{\mathsf{t}}).$$

Since $e_1 \overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_{\mathcal{E}} e_2$, we know there exist e_1', $\mathcal{E}_0$, $\mathcal{E}_1$, $\mathcal{S}_1$ and $\mathcal{S}_1'$ such that

$$e_1 \overset{\mathsf{t}}{\underset{\mathcal{E}}{\Rightarrow}} e_1',\ \mathcal{E}_2 = \mathcal{E}_0{+}{+}[e_1']{+}{+}\mathcal{E}_1,\ \mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_0) = \mathcal{S}_1,\ \delta_1(\mathcal{S}_1) = \mathcal{S}_1',\ \mathsf{exec\_st}(\mathcal{S}_1', \mathcal{E}_1) = \mathcal{S}_2.$$

We do case analysis on whether $\delta_1 \in \mathcal{V}(\mathcal{S}_1')$.

- $\delta_1 \in \mathcal{V}(\mathcal{S}_1')$. By Lemma 73, we know $\neg(\delta_2 \rightarrowtail \delta_1)$.
- $\delta_1 \notin \mathcal{V}(\mathcal{S}_1')$. From $\mathsf{wfTSV}(\rightarrowtail, \mathcal{V})$, we know there exist δ' such that
$$\delta_1 \rightarrowtail \delta' \text{ and } \delta' \in \mathcal{V}(\mathcal{S}_1').$$
By Lemma 73, we know
$$\neg(\delta_2 \rightarrowtail \delta').$$
Since $\mathsf{partialOrder}(\rightarrowtail)$, we know $\neg(\delta_2 \rightarrowtail \delta_1)$.

Thus we are done. □

Lemma 73. If

1. $\delta_1 \in \mathcal{V}(\mathcal{S}_1)$, $\mathsf{exec\_st}(\mathcal{S}_1, \mathcal{E}) = \mathcal{S}_2$, $\mathsf{genAt}_\Pi(\mathcal{S}_2, \delta_2)$,
2. $\mathsf{partialOrder}(\rightarrowtail)$, $\mathsf{effGenFollowTS}_\Pi(\rightarrowtail, \mathcal{V})$, $\mathsf{wfTSV}(\rightarrowtail, \mathcal{V})$,

then $\neg(\delta_2 \rightarrowtail \delta_1)$.

Proof. By induction over the length of $\mathcal{E}$.

- $|\mathcal{E}| = 0$. Thus $\mathcal{S}_1 = \mathcal{S}_2$. From $\mathsf{effGenFollowTS}_\Pi(\rightarrowtail, \mathcal{V})$, we know $\neg(\delta_2 \rightarrowtail \delta_1)$.
- $|\mathcal{E}| > 0$. We do case analysis on whether $\delta_1 \in \mathcal{V}(\mathcal{S}_2)$:
  - $\delta_1 \in \mathcal{V}(\mathcal{S}_2)$.
    From $\mathsf{effGenFollowTS}_\Pi(\rightarrowtail, \mathcal{V})$, we know $\neg(\delta_2 \rightarrowtail \delta_1)$.
  - $\delta_1 \notin \mathcal{V}(\mathcal{S}_2)$.
    Since $\delta_1 \in \mathcal{V}(\mathcal{S}_1)$, we know there exist $\mathcal{E}_3$, $\mathcal{E}_3'$, e_3, δ_3, $\mathcal{S}_3$, $\mathcal{S}_3'$ such that
    $$\mathcal{E} = \mathcal{E}_3{+}{+}[e_3]{+}{+}\mathcal{E}_3',\ \mathsf{exec\_st}(\mathcal{S}_1, \mathcal{E}_3) = \mathcal{S}_3,\ \mathsf{eff}(e_3) = \delta_3,\ \delta_3(\mathcal{S}_3) = \mathcal{S}_3',\ \mathsf{exec\_st}(\mathcal{S}_3', \mathcal{E}_3') = \mathcal{S}_2,$$
    $$\delta_1 \in \mathcal{V}(\mathcal{S}_3),\ \delta_1 \notin \mathcal{V}(\mathcal{S}_3').$$
    From $\mathsf{wfTSV}(\rightarrowtail, \mathcal{V})$, we know
    $$\delta_1 \rightarrowtail \delta_3 \text{ and } \delta_3 \in \mathcal{V}(\mathcal{S}_3').$$
    Since $|\mathcal{E}_3'| < |\mathcal{E}|$, by the induction hypothesis, we know
    $$\neg(\delta_2 \rightarrowtail \delta_3).$$
    Since $\mathsf{partialOrder}(\rightarrowtail)$, we know $\neg(\delta_2 \rightarrowtail \delta_1)$.

Thus we are done. □

Lemma 74 (Return Value Related)**.** If

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$,
2. $\mathsf{StRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathcal{S}_a, ar))$,
3. $\overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_{\mathcal{E}} \subseteq ar$,
4. $\mathsf{sameRVal}_\varphi(\Pi, \Gamma)$,

then $\mathsf{RValRelated}(\mathsf{t}, \mathcal{E}, (\Gamma, \mathcal{S}_a, ar))$.

Proof. For any $\mathcal{E}'$ and e, if $(\mathcal{E}'{+}{+}e) \leqslant \mathcal{E}$ and $\mathsf{is\_orig}_{\mathsf{t}}(e)$, we want to prove
$\mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}'{+}{+}e, \mathsf{t}) \downharpoonright ar)$.

Suppose $e = (mid, \mathsf{t}, (f, n, n', \delta))$. Since $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$ and $\mathsf{is\_orig}_{\mathsf{t}}(e)$, we know there exists $\mathcal{S}'$ such that

$$\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}}) = \mathcal{S}' \text{ and } \Pi(f, n)(\mathcal{S}') = (n', \delta).$$

From $\mathsf{StRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathcal{S}_a, ar))$, since $\mathcal{E}' \leqslant \mathcal{E}$, we know

$$\varphi(\mathcal{S}') = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar).$$

Let $\mathcal{S}_a' = \varphi(\mathcal{S}')$. Thus $\mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar) = \mathcal{S}_a'$. From $\mathsf{sameRVal}_\varphi(\Pi, \Gamma)$, we know

$$\Gamma(f, n)(\mathcal{S}_a') = (n', \_).$$

Since $\mathsf{is\_orig}_{\mathsf{t}}(e)$, we know $\mathsf{visible}(\mathcal{E}'{+}{+}e, \mathsf{t}) = \mathsf{visible}(\mathcal{E}', \mathsf{t}) \cup \{e\}$. Since $\overset{\mathsf{vis}}{\underset{\mathsf{t}}{\longmapsto}}_{\mathcal{E}} \subseteq ar$, we know

$$\forall e' \in \mathsf{visible}(\mathcal{E}', \mathsf{t}).\ (e', e) \in ar.$$

Thus

$$\mathsf{aexecRV}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}'{+}{+}e, \mathsf{t}) \downharpoonright ar) = n'.$$

Thus $\mathsf{rval}(e) = \mathsf{aexecRV}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}'{+}{+}e, \mathsf{t}) \downharpoonright ar)$. So we are done. □

Lemma 75 (tStRelated)**.** If

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\mathsf{eventualDelivery}(\mathcal{E})$, $\psi(\mathcal{S}) = \mathcal{S}_a$, $\psi \Rightarrow \varphi$,
2. $\mathsf{lockStep\text{-}TS}_\varphi(\Pi, \Gamma, \rightarrowtail, \mathcal{V})$, $\mathsf{effComm}_\varphi$, $\mathsf{wfV}_\psi(\mathcal{V})$,
3. $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar)$, $\mathsf{TS}^{\rightarrowtail}_{\mathcal{E}} \subseteq ar$,

then $\mathsf{StRelated}_\varphi(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar))$.

Proof. For any $\mathcal{E}'$, if $\mathcal{E}' \leqslant \mathcal{E}$, from Lemma 76, we know

$$\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar).$$

From Lemma 77 and Lemma 78, we know

$$\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar).$$

Thus $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar)$. So we are done. □

Lemma 76. If

1. $\mathsf{effComm}_\varphi$,
2. $\lfloor \mathcal{E}_1 \rfloor = \lfloor \mathcal{E}_2 \rfloor$,
3. $\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_1) = \mathcal{S}_1'$,

then $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_1)) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_2))$.

Proof. By induction over the length of $\mathcal{E}_1$.

- The length is 0. Thus $\mathcal{E}_1 = \mathcal{E}_2 = \epsilon$. The case is trivial.
- The length is 1. Thus $\mathcal{E}_1 = \mathcal{E}_2$. The case is trivial.
- The length is $n + 1$ where $n \geq 1$. Suppose $\forall i.\ \mathcal{E}_1(i) = e_i \wedge \mathcal{E}_2(i) = e_i'$. Since $\lfloor \mathcal{E}_1 \rfloor = \lfloor \mathcal{E}_2 \rfloor$, we know there are two cases:
  1. $e_{n+1} = e_{n+1}'$.
     Suppose $\mathcal{E}_1 = \mathcal{E}_1'{+}{+}[e_{n+1}]$ and $\mathcal{E}_2 = \mathcal{E}_2'{+}{+}[e_{n+1}]$. Then we know
     $$\lfloor \mathcal{E}_1' \rfloor = \lfloor \mathcal{E}_2' \rfloor,\ \ |\mathcal{E}_1'| = |\mathcal{E}_2'| = n.$$
     By the induction hypothesis, we know
     $$\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_1')) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_2')).$$
     Thus we know $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_1)) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_2))$.
  2. There exists i such that $1 \leq i \leq n$ and $e_{n+1} = e_i'$.
     Let $\mathcal{E}_3 = e_1' \dots e_{i-1}' e_{i+1}' \dots e_{n+1}' e_i' = \mathcal{E}_3'{+}{+}[e_i']$. Then we know
     $$\lfloor \mathcal{E}_1' \rfloor = \lfloor \mathcal{E}_3' \rfloor,\ \ |\mathcal{E}_1'| = |\mathcal{E}_3'| = n.$$
     By the induction hypothesis, we know
     $$\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_1')) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_3')).$$
     Thus we know $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_1)) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_3))$.
     Below we prove $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_3)) = \varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_2))$. Let
     $$\mathcal{E}_3'' = e_1' \dots e_{i-1}',\ \ \mathcal{E}_3''' = e_{i+1}' \dots e_{n+1}' e_i' \ \text{ and } \mathcal{E}_2''' = e_i' e_{i+1}' \dots e_{n+1}'.$$
     Then

$$\mathcal{E}_3 = \mathcal{E}_3''{+}{+}\mathcal{E}_3''' \text{ and } \mathcal{E}_2 = \mathcal{E}_3''{+}{+}\mathcal{E}_2'''.$$

Suppose

$$\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_3'') = \mathcal{S}'.$$

Thus we only need to prove $\varphi(\mathsf{exec\_st}(\mathcal{S}', \mathcal{E}_3''')) = \varphi(\mathsf{exec\_st}(\mathcal{S}', \mathcal{E}_2'''))$.
Let $k = |\mathcal{E}_3'''|$. We know $k = n + 2 - i \geq 2$. By induction over k.

- $k = 2$. So our goal is to prove $\varphi(\mathsf{exec\_st}(\mathcal{S}', e'_{n+1}e'_i)) = \varphi(\mathsf{exec\_st}(\mathcal{S}', e'_ie'_{n+1}))$.
  From $\mathsf{effComm}_\varphi$, we know
  $$\mathsf{commute}_\varphi(\mathsf{eff}(e'_i), \mathsf{eff}(e'_{n+1})).$$
  Thus $\varphi(\mathsf{exec\_st}(\mathcal{S}', e'_{n+1}e'_i)) = \varphi(\mathsf{exec\_st}(\mathcal{S}', e'_ie'_{n+1}))$.
- $k = k' + 1$.
  First, as the $k = 2$ case, from $\mathsf{effComm}_\varphi$, we know
  $$\forall \mathcal{S}''.\ \varphi(\mathsf{exec\_st}(\mathcal{S}'', e'_{n+1}e'_i)) = \varphi(\mathsf{exec\_st}(\mathcal{S}'', e'_ie'_{n+1})).$$
  Then we know
  $$\varphi(\mathsf{exec\_st}(\mathcal{S}', e'_{i+1} \dots e'_ne'_{n+1}e'_i)) = \varphi(\mathsf{exec\_st}(\mathcal{S}', e'_{i+1} \dots e'_ne'_ie'_{n+1}))\ . \tag{G.1}$$
  Next, by the induction hypothesis, we know
  $$\varphi(\mathsf{exec\_st}(\mathcal{S}', e'_{i+1} \dots e'_ne'_i)) = \varphi(\mathsf{exec\_st}(\mathcal{S}', e'_ie'_{i+1} \dots e'_n)).$$
  Then we know
  $$\varphi(\mathsf{exec\_st}(\mathcal{S}', e'_{i+1} \dots e'_ne'_ie'_{n+1})) = \varphi(\mathsf{exec\_st}(\mathcal{S}', e'_ie'_{i+1} \dots e'_ne'_{n+1}))\ . \tag{G.2}$$
  By (G.1) and (G.2), we know
  $$\varphi(\mathsf{exec\_st}(\mathcal{S}', e'_{i+1} \dots e'_ne'_{n+1}e'_i)) = \varphi(\mathsf{exec\_st}(\mathcal{S}', e'_ie'_{i+1} \dots e'_ne'_{n+1})).$$

Thus we are done. □

Lemma 77. If

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\mathcal{S} \in dom(\psi)$, $\mathcal{E}' \leqslant \mathcal{E}$,
2. $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar)$, $\mathsf{TS}_{\mathcal{E}}^{\rightarrowtail} \subseteq ar$,
3. $\mathsf{wfV}_\psi(\mathcal{V})$,

then $\mathsf{execFollowTS}(\mathcal{S}, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar, \rightarrowtail, \mathcal{V})$.

Proof. By unfolding the definition of execFollowTS in Figure 33, we want to prove:

$$\begin{array}{l}\forall \mathcal{E}''.e, \mathcal{S}'.\ (\mathcal{E}''{+}{+}[e] \leqslant (\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar)) \wedge \mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'') = \mathcal{S}' \\ \Longrightarrow \mathsf{followTS}(\mathcal{S}', \mathsf{eff}(e), \rightarrowtail, \mathcal{V})\end{array}$$

Suppose $\mathsf{eff}(e) = \delta$. By unfolding followTS, we want to prove:

$$\forall \delta'.\ \delta' \in \mathcal{V}(\mathcal{S}') \Longrightarrow \neg(\delta \rightarrowtail \delta')$$

By contradiction. Suppose there exists δ' such that $\delta' \in \mathcal{V}(\mathcal{S}')$ and $\delta \rightarrowtail \delta'$.
From $\mathsf{wfV}_\psi(\mathcal{V})$, we can prove:

$$\exists e'.\ (e' \in \mathcal{E}'') \wedge \mathsf{eff}(e') = \delta'.$$

Thus we know $(e', e) \in ar$. Also we know

$$\{e, e'\} \subseteq \mathsf{orig}(\mathcal{E}).$$

Since $\mathsf{TS}_{\mathcal{E}}^{\rightarrowtail} \subseteq ar$, we know

$$(e, e') \in ar.$$

So we get a contradiction. □

Lemma 78. If

1. $\mathcal{E}_0 \in \mathcal{T}(\Pi, \mathcal{S}_0)$, $\lfloor \mathcal{E} \rfloor \subseteq \mathsf{orig}(\mathcal{E}_0)$, $\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}) = \mathcal{S}'$,
2. $\mathsf{execFollowTS}(\mathcal{S}, \mathcal{E}, \rightarrowtail, \mathcal{V})$, $\psi(\mathcal{S}) = \mathcal{S}_a$,
3. $\psi \Rightarrow \varphi$, $\mathsf{lockStep\text{-}TS}_\varphi(\Pi, \Gamma, \rightarrowtail, \mathcal{V})$,

then $\varphi(\mathcal{S}') = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathcal{E})$.

Proof. By induction over the length n of $\mathcal{E}$.

- $n = 0$. Trivial.
- $n = m + 1$. Suppose $\mathcal{E} = \mathcal{E}'{+}{+}[e]$. Since $\mathcal{E}_0 \in \mathcal{T}(\Pi, \mathcal{S}_0)$ and $\lfloor \mathcal{E} \rfloor \subseteq \mathsf{orig}(\mathcal{E}_0)$, we can suppose $e = (mid, \mathsf{t}, (f, n, n', \delta))$. So $\mathsf{valid}_\Pi(f, n, \delta)$.
  Since $\mathsf{execFollowTS}(\mathcal{S}, \mathcal{E}, \rightarrowtail, \mathcal{V})$, we know $\mathsf{execFollowTS}(\mathcal{S}, \mathcal{E}', \rightarrowtail, \mathcal{V})$. By the induction hypothesis, we know
  $$\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}')) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathcal{E}').$$
  Suppose $\mathcal{S}'' = \mathsf{exec\_st}(\mathcal{S}, \mathcal{E}')$ and $\mathcal{S}_a'' = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathcal{E}')$. So $\varphi(\mathcal{S}'') = \mathcal{S}_a''$.
  Since $\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}) = \mathcal{S}'$, we know
  $$\delta(\mathcal{S}'') = \mathcal{S}'.$$
  Since $\mathsf{execFollowTS}(\mathcal{S}, \mathcal{E}, \rightarrowtail, \mathcal{V})$, we know
  $$\mathsf{followTS}(\mathcal{S}'', \delta, \rightarrowtail, \mathcal{V}).$$
  From $\mathsf{lockStep\text{-}TS}_\varphi(\Pi, \Gamma, \rightarrowtail, \mathcal{V})$, we know there exists $\mathcal{S}_a'$ such that
  $$\varphi(\mathcal{S}') = \mathcal{S}_a' \text{ and } \Gamma(f, n)(\mathcal{S}_a'') = (\_, \mathcal{S}_a').$$
  Thus $\varphi(\mathcal{S}') = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathcal{E})$.

Thus we are done. □

```
var current;

operation read(){
  return current;
  gen_eff IdEff;
}

operation inc(){
  return;
  gen_eff Inc();
}

effector Inc(){
  current := current + 1;
}

operation dec(){
  return;
  gen_eff Dec();
}

effector Dec(){
  current := current - 1;
}
```

Figure 34. The Replicated Counters

H Examples of CRDT Verification

By applying our proof method, we have verified *nine* CRDT algorithms taken from [19]. They are the replicated counter, the grow-only set, the last-writer-wins (LWW) register, the LWW-element set, the 2P set, the replicated growable array (RGA), the continuous sequence, the add-wins set and the remove-wins set. The first seven are verified using CRDT-TS. The add-wins set and the remove-wins set are verified in Section I.

Before giving the proofs of these algorithms, we first present a specific instantiation of CRDT-TS, called CRDT-S. With CRDT-S we can already verify replicated counter and the grow-only set.

CRDT-S: a special case of CRDT-TS. When no operations are conflicting, we can verify the CRDT algorithms by letting $\rightarrowtail$ be $\emptyset$ and letting $\mathcal{V}$ be $\lambda\mathcal{S}.\ \emptyset$. Then CRDT-TS is reduced to the following CRDT-S.

$$\begin{array}{l}\mathsf{no\text{-}conflict}(\Gamma, \bowtie)\ \text{ iff}\\ \quad \forall f_1, n_1, f_2, n_2.\ (f_1, n_1) \in dom(\Gamma) \wedge (f_2, n_2) \in dom(\Gamma) \implies \neg(\Gamma \models (f_1, n_1) \bowtie (f_2, n_2))\end{array}$$

Definition 79 (Simple Lock-Step φ-Preservation)**.** $\mathsf{lockStep\text{-}S}_\varphi(\Pi, \Gamma)$ iff

$$\begin{array}{l}\forall f, n, \delta.\ \mathsf{valid}_\Pi(f, n, \delta)\\ \implies \forall \mathcal{S}, \mathcal{S}', \mathcal{S}_a.\ \varphi(\mathcal{S}) = \mathcal{S}_a \wedge \delta(\mathcal{S}) = \mathcal{S}'\\ \implies \exists \mathcal{S}'_a.\ \varphi(\mathcal{S}') = \mathcal{S}'_a \wedge \Gamma(f, n)(\mathcal{S}_a) = (\_, \mathcal{S}'_a)\end{array}$$

Definition 80 (Simple CRDTs)**.** $\mathsf{CRDT\text{-}S}_{\psi,\varphi}(\Pi, \Gamma)$ iff

$$(\psi \Rightarrow \varphi) \wedge \mathsf{effComm}_\varphi \wedge \mathsf{sameRVal}_\varphi(\Pi, \Gamma) \wedge \mathsf{lockStep\text{-}S}_\varphi(\Pi, \Gamma)$$

Theorem 81. If $\mathsf{CRDT\text{-}S}_{\psi,\varphi}(\Pi, \Gamma)$ and $\mathsf{no\text{-}conflict}(\Gamma, \bowtie)$, then $\mathsf{ACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie))$.

Proof. We first prove $\mathsf{CRDT\text{-}TS}_{\psi,\varphi}(\Pi, \Gamma)$ by letting $\rightarrowtail$ be $\emptyset$ and letting $\mathcal{V}$ be $\lambda\mathcal{S}.\ \emptyset$. By Theorem 8, we are done. □

H.1 The Replicated Counters

Fig. 34 shows the implementation Π_{counter} of the replicated counter with both `inc` and `dec` operations. The specification Γ_{counter} is the same as the one for sequential counters, i.e.,

`INC(){x:=x+1}` and `DEC(){x:=x-1}`.

The conflicting relation $\bowtie$ for counters is empty, so $\mathsf{no\text{-}conflict}(\Gamma_{\text{counter}}, \bowtie)$ holds. We also let both φ and ψ relate $\mathcal{S}$ and $\mathcal{S}_a$ when $\mathcal{S}(\mathtt{current}) = \mathcal{S}_a(\mathtt{x})$ holds.

We can prove all the conditions in $\mathsf{CRDT\text{-}S}_{\psi,\varphi}(\Pi_{\text{counter}}, \Gamma_{\text{counter}})$. Then, by Theorem 81, we get $\mathsf{ACC}_{\psi,\varphi}(\Pi_{\text{counter}}, (\Gamma_{\text{counter}}, \bowtie))$.

H.2 The Grow-Only Sets

Fig. 35 shows the implementation Π_{add} of the grow-only set with `add` and `lookup` operations. The specification Γ_{add} is the same as the one for sequential sets in Fig. 36:

`LOOKUP(e){ return e ∈ S; }`   `ADD(e){ S:=S ∪ {e}; }`

The conflicting relation $\bowtie$ for Γ_{add} is empty, so $\mathsf{no\text{-}conflict}(\Gamma_{\text{add}}, \bowtie)$ holds. We also let both φ and ψ relate $\mathcal{S}$ and $\mathcal{S}_a$ when $\mathcal{S}(\mathtt{A}) = \mathcal{S}_a(\mathtt{S})$ holds.

We can prove all the conditions in $\mathsf{CRDT\text{-}S}_{\psi,\varphi}(\Pi_{\text{add}}, \Gamma_{\text{add}})$. Then, we get $\mathsf{ACC}_{\psi,\varphi}(\Pi_{\text{add}}, (\Gamma_{\text{add}}, \bowtie))$ by Theorem 81.

H.3 The Last-Writer-Wins (LWW) Register

Fig. 37 shows the LWW register implementation Π_{reg}. The specification Γ_{reg} is the same as the one for sequential registers:

```
var A := ∅;                    operation add(e){
                                 return;
operation lookup(e){             gen_eff Add(e);
  return (e ∈ A);              }
  gen_eff IdEff;
}                              effector Add(e){
                                 A := A ∪ {e};
                               }
```

Figure 35. The Grow-Only Sets

```
LOOKUP(e){              ADD(e){              REMOVE(e){
  return (e ∈ S);         S := S ∪ {e};        S := S - {e};
}                       }                    }
```

$$\alpha \bowtie \alpha' \quad \text{iff} \quad \exists \texttt{a}.\ \alpha = \texttt{add(a)} \wedge \alpha' = \texttt{rmv(a)} \vee \alpha = \texttt{rmv(a)} \wedge \alpha' = \texttt{add(a)}$$

Figure 36. The Specification for Sets

```
var x := 0, ts := (0,cid);   operation write(v){          effector Write(v, i){
                               local i;                     if (i > ts) {
operation read(){              i := (ts.fst+1, cid);          x := v;
  return x;                    return;                        ts := i;
  gen_eff IdEff;               gen_eff Write(v, i);         }
}                            }                            }
```

Figure 37. The Last-Writer-Wins Registers

```
READ(){ return x; }        WRITE(v){ x := v; }
```

The conflicting relation $\bowtie$ for Γ_{reg} relates `write` operations:

$$\alpha \bowtie \alpha' \text{ iff } \exists \texttt{v}, \texttt{v'}.\ \alpha = \texttt{write(v)} \wedge \alpha' = \texttt{write(v')}$$

We let φ relate states $\mathcal{S}$ and $\mathcal{S}_a$ where the values of x are the same. The initial state mapping ψ is stronger than φ. It additionally requires that ts in $\mathcal{S}$ contains the initial (smallest) time-stamp $(0, \texttt{cid})$.

$$\begin{array}{lll} \varphi(\mathcal{S}) = \mathcal{S}_a & \text{iff} & \mathcal{S}(\texttt{x}) = \mathcal{S}_a(\texttt{x}) \\ \psi(\mathcal{S}) = \mathcal{S}_a & \text{iff} & (\mathcal{S}(\texttt{x}) = \mathcal{S}_a(\texttt{x})) \wedge (\mathcal{S}(\texttt{ts}) = (0, \texttt{cid})) \end{array}$$

To verify the algorithm, we ask users to provide $\rightarrowtail$ and $\mathcal{V}$. They can be defined as follows.

$$\begin{array}{lll} \delta \rightarrowtail \delta' & \text{iff} & \exists \texttt{v}, \texttt{i}, \texttt{v'}, \texttt{i'}.\ (\delta = \texttt{Write(v,i)}) \wedge (\delta' = \texttt{Write(v',i')}) \wedge (\texttt{i} < \texttt{i'}) \\ \mathcal{V}(\mathcal{S}) & \stackrel{\text{def}}{=} & \{\delta \mid \exists \texttt{v}, \texttt{i}.\ (\mathcal{S}(\texttt{x}) = \texttt{v}) \wedge (\mathcal{S}(\texttt{ts}) = \texttt{i} > (0,0)) \wedge (\delta = \texttt{Write(v,i)})\} \end{array}$$

Here $\rightarrowtail$ orders two `Write` effectors using their time-stamps, and $\mathcal{V}(\mathcal{S})$ returns the most recent `Write` which leads to $\mathcal{S}$. We can prove all the conditions in CRDT-TS$_{\psi,\varphi}(\Pi_{\text{reg}}, (\Gamma_{\text{reg}}, \bowtie))$. By Theorem 8, we get ACC$_{\psi,\varphi}(\Pi_{\text{reg}}, (\Gamma_{\text{reg}}, \bowtie))$.

H.4 The LWW-Element Sets

Fig. 38 shows the implementation Π_{LWWES} of the LWW-element set with add, remove and lookup operations. Its specification Γ_{set} is shown in Fig. 36, which is the same for the sequential sets. The conflicting relation $\bowtie$ for Γ_{set} is shown at the bottom of Fig. 36, which relates add and rmv on the same element. We let φ relate states $\mathcal{S}$ and $\mathcal{S}_a$ such that $\mathcal{S}_a(\texttt{S})$ contains all the elements that can be looked up in $\mathcal{S}$. We let ψ relate the initial states $\mathcal{S}$ and $\mathcal{S}_a$ where A, R and S at the two levels are all empty.

$$\begin{array}{lll} \varphi(\mathcal{S}) = \mathcal{S}_a & \text{iff} & \mathcal{S}_a(\texttt{S}) = \{\texttt{e} \mid \exists i.\ (\texttt{e}, i) \in \mathcal{S}(\texttt{A}) \wedge \forall i' > i.\ (\texttt{e}, i') \notin \mathcal{S}(\texttt{R})\} \\ \psi(\mathcal{S}) = \mathcal{S}_a & \text{iff} & \mathcal{S}(\texttt{A}) = \mathcal{S}(\texttt{R}) = \mathcal{S}_a(\texttt{S}) = \emptyset \end{array}$$

To verify the algorithm, we ask users to provide $\rightarrowtail$ and $\mathcal{V}$. They can be defined as follows.

$$\begin{array}{lll} \delta \rightarrowtail \delta' & \text{iff} & \exists \texttt{i}, \texttt{i'}.\ (\delta = \_(\_, \texttt{i})) \wedge (\delta' = \_(\_, \texttt{i'})) \wedge (\texttt{i} < \texttt{i'}) \\ \mathcal{V}(\mathcal{S}) & \stackrel{\text{def}}{=} & \{\delta \mid \exists \texttt{e}, \texttt{i}.\ ((\texttt{e},\texttt{i}) \in \mathcal{S}(\texttt{A})) \wedge (\delta = \texttt{Add(e,i)}) \vee ((\texttt{e},\texttt{i}) \in \mathcal{S}(\texttt{R})) \wedge (\delta = \texttt{Rmv(e,i)})\} \end{array}$$

```
var A := ∅, R := ∅;
var ts := (0, cid);

operation lookup(e){
  return (∃i. (e, i) ∈ A ∧ ∀i' > i. (e, i') ∉ R);
  gen_eff IdEff;
}

operation add(e){
  local i;
  i := (ts.fst + 1, cid);
  return;
  gen_eff Add(e, i);
}

effector Add(e, i){
  A := A ∪ {(e, i)};
  if (ts < i) ts := i;
}

operation remove(e){
  assume (lookup(e));
  local i;
  i := (ts.fst + 1, cid);
  return;
  gen_eff Rmv(e, ts);
}

effector Rmv(e, i){
  R := R ∪ {(e, i)};
  if (ts < i) ts := i;
}
```

Figure 38. The LWW-Element Sets

```
 1 var A := ∅, R := ∅;

 2 operation lookup(e){
 3   return (e ∈ A && e ∉ R);
 4   gen_eff IdEff;
 5 }

 6 operation add(e){
 7   assume (e ∉ R);
 8   return;
 9   gen_eff Add(e);
10 }

11 effector Add(e){
12   A := A ∪ {e};
13 }

14 operation remove(e){
15   assume (lookup(e));
16   return;
17   gen_eff Rmv(e);
18 }

19 effector Rmv(e){
20   R := R ∪ {e};
21 }
```

Figure 39. The 2P-Set

Here $\rightarrowtail$ orders two effectors (Add or Rmv) using their time-stamps, and $\mathcal{V}(\mathcal{S})$ returns all the applied Add and Rmv which leads to $\mathcal{S}$. We can prove all the conditions in CRDT-TS$_{\psi,\varphi}(\Pi_{\text{LWWES}}, (\Gamma_{\text{set}}, \bowtie))$. By Theorem 8, we get ACC$_{\psi,\varphi}(\Pi_{\text{LWWES}}, (\Gamma_{\text{set}}, \bowtie))$.

H.5 The 2P-Set

Fig. 39 shows the implementation Π_{2PSet} of the 2P set with add, remove and lookup operations. The specification Γ_{set} and the conflicting relation $\bowtie$ are shown in Fig. 36. We let φ relate states $\mathcal{S}$ and $\mathcal{S}_a$ such that $\mathcal{S}_a(\mathsf{S})$ contains all the elements that can be looked up in $\mathcal{S}$. We let ψ relate the initial states $\mathcal{S}$ and $\mathcal{S}_a$ where A, R and S at the two levels are all empty.

$$\begin{aligned} \varphi(\mathcal{S}) = \mathcal{S}_a \quad &\text{iff} \quad \mathcal{S}(\mathsf{A}) - \mathcal{S}(\mathsf{R}) = \mathcal{S}_a(\mathsf{S}) \\ \psi(\mathcal{S}) = \mathcal{S}_a \quad &\text{iff} \quad \mathcal{S}(\mathsf{A}) = \mathcal{S}(\mathsf{R}) = \mathcal{S}_a(\mathsf{S}) = \emptyset \end{aligned}$$

The 2P-set algorithm assumes that an element is never added again after it is removed [19]. So at line 7 in Fig. 39, we assume that an add(e) can only happen if remove(e) has not been applied. Then we can follow the time-stamp pattern of CRDTs to verify the algorithm.

To verify the algorithm, we ask users to provide $\rightarrowtail$ and $\mathcal{V}$. They can be defined as follows.

```
var N := ∅, T := ∅;

operation read(){
  return orderedSeq(N,T);
  gen_eff IdEff;
}

operation addAfter(a, b){
  assume(a = ∘ ∨
    a ≠ ∘ ∧ (_,_,a) ∈ N ∧ a ∉ T);
  local i, j, k;
  i := getTagReal(a, N);
  k := getNextTagReal(i, N);
  j := allocateRealBetween(i, k);
  return;
  gen_eff AddAfter(a, (j, cid), b);
}

effector AddAfter(a, tag, b){
  N := N ∪ {(a, tag, b)};
}

operation remove(a){
  assume((_,_,a) ∈ N
    ∧ a ∉ T ∧ a ≠ ∘);
  return;
  gen_eff Rmv(a);
}

effector Rmv(a){
  T := T ∪ {a};
}
```

Figure 40. The Continuous Sequence

$$\delta \rightarrowtail \delta' \quad \text{iff} \quad \exists \mathsf{e}.\ (\delta = \mathsf{Add(e)}) \wedge (\delta' = \mathsf{Rmv(e)})$$
$$\mathcal{V}(\mathcal{S}) \overset{\text{def}}{=} \{\delta \mid \exists \mathsf{e}.\ (\mathsf{e} \in \mathcal{S}(\mathsf{A})) \wedge (\delta = \mathsf{Add(e)}) \vee (\mathsf{e} \in \mathcal{S}(\mathsf{R})) \wedge (\delta = \mathsf{Rmv(e)})\}$$

Here $\rightarrowtail$ orders Add before the corresponding Rmv, and $\mathcal{V}(\mathcal{S})$ returns all the applied Add and Rmv which leads to $\mathcal{S}$. We can prove all the conditions in CRDT-TS$_{\psi,\varphi}(\Pi_{\text{2PSet}}, (\Gamma_{\text{set}}, \bowtie))$. By Theorem 8, we get ACC$_{\psi,\varphi}(\Pi_{\text{2PSet}}, (\Gamma_{\text{set}}, \bowtie))$.

H.6 The Replicated Growable Array (RGA)

We verify the RGA algorithm Π_{RGA} in Fig. 2[2]. We prove that the RGA algorithm Π_{RGA} is ACC with respect to the following list specification Γ_{list}, which uses L for the list:

$$\Gamma_{\text{list}}(\mathsf{READ})(\mathcal{S}_a) \overset{\text{def}}{=} \mathcal{S}_a(\mathsf{L})$$
$$\Gamma_{\text{list}}(\mathsf{ADDAFTER}, (\mathsf{a},\mathsf{b}))(\mathcal{S}_a) \overset{\text{def}}{=} \begin{cases} \mathcal{S}_a\{\mathsf{L} \rightsquigarrow l'{+}{+}[\mathsf{a}]{+}{+}[\mathsf{b}]{+}{+}l''\}, & \text{if } \mathcal{S}_a(\mathsf{L}) = l'{+}{+}[\mathsf{a}]{+}{+}l'' \\ \mathcal{S}_a, & \text{if } \mathsf{a} \notin \mathcal{S}_a(\mathsf{L}) \end{cases}$$
$$\Gamma_{\text{list}}(\mathsf{RMV}, \mathsf{a})(\mathcal{S}_a) \overset{\text{def}}{=} \begin{cases} \mathcal{S}_a\{\mathsf{L} \rightsquigarrow l'{+}{+}l''\}, & \text{if } \mathcal{S}_a(\mathsf{L}) = l'{+}{+}[\mathsf{a}]{+}{+}l'' \\ \mathcal{S}_a, & \text{if } \mathsf{a} \notin \mathcal{S}_a(\mathsf{L}) \end{cases}$$

The conflicting relation $\bowtie$ for Γ_{list} is shown in Sec. 4.

The initial state relation ψ relates empty lists:

$$\psi(\mathcal{S}) = \mathcal{S}_a \quad \text{iff} \quad \mathcal{S}(\mathsf{N}) = \emptyset \wedge \mathcal{S}(\mathsf{T}) = \emptyset \wedge \mathcal{S}(\mathsf{ts}) = (0,\_) \wedge \mathcal{S}_a(\mathsf{L}) = \epsilon$$
$$\varphi(\mathcal{S}) = \mathcal{S}_a \quad \text{iff} \quad (\mathsf{traverse}(\mathcal{S}(\mathsf{N}),\mathcal{S}(\mathsf{T})) = \mathcal{S}_a(\mathsf{L})) \wedge (\mathcal{S}(\mathsf{ts}) = \mathsf{max\_ts}(\mathcal{S}(\mathsf{N})))$$

Here we max_ts to get the maximal time-stamp associated with a node in N. Then, φ not only maps the concrete time-stamped tree to the abstract list, but also ensures that ts is always the newest time-stamp. The latter is the key to proving that a newly generated AddAfter effector always has the greatest time-stamp, and hence satisfies followTS (see the first item in Def. 63).

We instantiate $\mathcal{V}$ and $\rightarrowtail$ as follows.

$$\delta \rightarrowtail \delta' \quad \text{iff} \quad \exists \mathsf{a},\mathsf{i},\mathsf{b},\mathsf{a}',\mathsf{i}',\mathsf{b}'.\ (\delta = \mathsf{AddAfter(a,i,b)}) \wedge ((\delta' = \mathsf{AddAfter(a',i',b')}) \wedge (\mathsf{i} < \mathsf{i}') \vee (\delta' = \mathsf{Rmv(a)}) \vee (\delta' = \mathsf{Rmv(b)}))$$
$$\mathcal{V}(\mathcal{S}) \overset{\text{def}}{=} \{\delta \mid \exists \mathsf{a},\mathsf{i},\mathsf{b}.\ ((\mathsf{a},\mathsf{i},\mathsf{b}) \in \mathcal{S}(\mathsf{N})) \wedge (\delta = \mathsf{AddAfter(a,i,b)}) \vee \exists \mathsf{a}.\ (\mathsf{a} \in \mathcal{S}(\mathsf{T})) \wedge (\delta = \mathsf{Rmv(a)})\}$$

Here $\delta \rightarrowtail \delta'$ holds between AddAfter effectors which have time-stamps. Note Rmv(a) (or Rmv(b)) conflicts with AddAfter(a,_,b), so we also let $\delta \rightarrowtail \delta'$ hold between an AddAfter and a Rmv.

We prove all the conditions in CRDT-TS$_{\psi,\varphi}(\Pi_{\text{RGA}}, (\Gamma_{\text{list}}, \bowtie))$. Then we get ACC$_{\psi,\varphi}(\Pi_{\text{RGA}}, (\Gamma_{\text{list}}, \bowtie))$ by Theorem 8.

H.7 The Continuous Sequence

The idea of the continuous sequence algorithm [19] is to assign elements unique identifiers in a dense identifier space such as the reals. So a unique identifier can always be allocated between any two given identifiers.

Our code is shown in Figure 40. We revise the original code in [19] in the following way:

[2]The assume statement uses blocking semantics.

- We provide the operation `addAfter` instead of the operation `addBetween`, so that we can use the same specification as RGA.
- Our `remove` operation does not directly remove the element from the set `N`. Instead, we use a tombstone set `T` to record the elements that are removed. We do this change for two reasons.
  - First, the algorithm requires that the identifiers allocated for elements should be unique. To ensure the uniqueness, we should find some way to remember the identifiers that have been allocated (no matter whether the elements are removed or not). A natural way might be using a tombstone set `T` for remove, and keeping all the elements (and their identifiers) in the set `N`.
  - Second, the original algorithm assumes causal delivery. By using the tombstone set `T` for remove, we no longer need to assume causal delivery.

The `read` operation calls the function `orderedSeq(N, T)` (line 3 in Figure 40). It uses the unique identifiers to order the elements in `N` but not in `T`, and returns the ordered sequence of elements.

The algorithm requires an `addAfter(a, b)` operation to generate a unique identifier `tag` for `b`, and the effector should add `(b, tag)` to `N`. In our implementation in Figure 40, we allocate the appropriate real number `j` and let the identifier `tag` be a pair `(j, cid)` to ensure its uniqueness. Here `getTagReal(a, N)` (line 10) returns the real number in the identifier of `a` in `N`, `getNextTagReal(i, N)` (line 11) returns the smallest real number that is greater than `i` in `N`, and `allocateRealBetween(i, k)` (line 12) returns an arbitrary real number between `i` and `k`.

Our effector `AddAfter` actually adds `(a, tag, b)` to `N` (see line 17). That is, each added element `b` is also associated with the element `a` after which the client calls `addAfter` to add `b`. (We call this element `a` the "intended preceding" element of `b`.) This information is not useful in the algorithm itself, but helps us specify $\mathcal{V}$. It can be viewed as ghost state and is introduced for verification purpose only.

We prove that the continuous sequence algorithm Π_{cont} is ACC with respect to the following list specification Γ_{list}, which uses `L` for the list. It is the same specification as for RGA.

$$\begin{array}{rcl}
\Gamma_{\text{list}}(\mathsf{READD})(\mathcal{S}_a) & \overset{\text{def}}{=} & \mathcal{S}_a(\mathsf{L}) \\
\Gamma_{\text{list}}(\mathsf{ADDAFTER}, (\mathsf{a},\mathsf{b}))(\mathcal{S}_a) & \overset{\text{def}}{=} & \mathcal{S}_a\{\mathsf{L} \leadsto l'{+}{+}[\mathsf{a}]{+}{+}[\mathsf{b}]{+}{+}l''\}\,, \quad \text{if } \mathcal{S}_a(\mathsf{L}) = l'{+}{+}[\mathsf{a}]{+}{+}l'' \\
\Gamma_{\text{list}}(\mathsf{RMV}, \mathsf{a})(\mathcal{S}_a) & \overset{\text{def}}{=} & \mathcal{S}_a\{\mathsf{L} \leadsto l'{+}{+}l''\}\,, \quad \text{if } \mathcal{S}_a(\mathsf{L}) = l'{+}{+}[\mathsf{a}]{+}{+}l''
\end{array}$$

The conflicting relation $\bowtie$ for Γ_{list} is shown in Sec. 4.

The initial state relation ψ relates empty lists:

$$\begin{array}{lcl}
\psi(\mathcal{S}) = \mathcal{S}_a & \text{iff} & \mathcal{S}(\mathsf{N}) = \emptyset \land \mathcal{S}(\mathsf{T}) = \emptyset \land \mathcal{S}_a(\mathsf{L}) = \epsilon \\
\varphi(\mathcal{S}) = \mathcal{S}_a & \text{iff} & (\mathsf{orderedSeq}(\mathcal{S}(\mathsf{N}), \mathcal{S}(\mathsf{T})) = \mathcal{S}_a(\mathsf{L}))
\end{array}$$

To verify the algorithm, we ask users to provide $\rightarrowtail$ and $\mathcal{V}$. They can be defined as follows.

$$\begin{array}{rcl}
\delta \rightarrowtail \delta' & \text{iff} & \exists \mathsf{a}, \mathsf{tag}, \mathsf{b}.\ (\delta = \mathtt{AddAfter(a,tag,b)}) \\
& & \quad \land (\exists \mathtt{tag'}, \mathtt{b'}.\ (\delta' = \mathtt{AddAfter(a,tag',b')}) \land (\mathtt{tag} > \mathtt{tag'}) \lor (\delta' = \mathtt{Rmv(b)})) \\
\mathcal{V}(\mathcal{S}) & \overset{\text{def}}{=} & \{\delta \mid \exists \mathsf{a}, \mathsf{tag}, \mathsf{b}.\ ((\mathsf{a}, \mathsf{tag}, \mathsf{b}) \in \mathcal{S}(\mathsf{N})) \land (\delta = \mathtt{AddAfter(a,tag,b)}) \\
& & \quad \lor \exists \mathsf{a}.\ (\mathsf{a} \in \mathcal{S}(\mathsf{T})) \land (\delta = \mathtt{Rmv(a)})\}
\end{array}$$

The definition of $\delta \rightarrowtail \delta'$ may be interesting. It holds between any two `AddAfter` effectors δ and δ', which have the same "intended preceding" element a. Then `AddAfter(a,tag,b)` $\rightarrowtail$ `AddAfter(a,tag',b')`, (which means `addAfter(a,b)` should be applied before `addAfter(a,b')` in abstract executions), if the unique identifier `tag` for `b` is greater than `tag'` for `b'`.

We can prove that a newly generated `AddAfter(a, tag, b)` effector always has a smaller real number identifier than any other `AddAfter(a, tag', b')` that has been applied before, i.e., `tag < tag'` must hold, and hence satisfies followTS.

We also let $\delta \rightarrowtail \delta'$ hold between an `AddAfter` and a `Rmv`, as for the RGA algorithm.

We prove all the conditions in CRDT-TS$_{\psi,\varphi}(\Pi_{\text{cont}}, (\Gamma_{\text{list}}, \bowtie))$. So we get ACC$_{\psi,\varphi}(\Pi_{\text{cont}}, (\Gamma_{\text{list}}, \bowtie))$ by Theorem 8.

$$\begin{array}{rcll}
\text{✂} & \in & \mathscr{P}(\mathit{Effector} \times \mathit{Effector}) & \text{(the canceled-by order)} \\
\mathbf{+} & \in & \mathscr{P}(\mathit{Effector}) & \text{(the winner set)} \\
\mathbf{-} & \in & \mathscr{P}(\mathit{Effector}) & \text{(the loser set)}
\end{array}$$

$$\delta' \Join_{\Pi,\Gamma} \delta \ \text{ iff } \ \forall f, n, f', n'.\ \mathsf{valid}_{\Pi}(f, n, \delta) \wedge \mathsf{valid}_{\Pi}(f', n', \delta') \implies (f, n) \Join_{\Gamma} (f', n')$$

$$\delta' \blacktriangleleft_{\Pi,\Gamma} \delta \ \text{ iff } \ \forall f, n, f', n'.\ \mathsf{valid}_{\Pi}(f, n, \delta) \wedge \mathsf{valid}_{\Pi}(f', n', \delta') \implies (f, n) \blacktriangleleft_{\Gamma} (f', n')$$

$$\delta' \rhd_{\Pi,\Gamma} \delta \ \text{ iff } \ \forall f, n, f', n'.\ \mathsf{valid}_{\Pi}(f, n, \delta) \wedge \mathsf{valid}_{\Pi}(f', n', \delta') \implies (f, n) \rhd_{\Gamma} (f', n')$$

$$\mathsf{loseAt}_{\Pi}(\delta, \mathcal{S}, \mathcal{V}, \mathbf{+}, \mathbf{-}, \text{✂}, (\Gamma, \Join)) \ \text{ iff } \ \delta \in \mathbf{-} \wedge \exists \delta'.\ \delta' \in \mathcal{V}(\mathcal{S}) \wedge \delta' \in \mathbf{+} \wedge (\delta' \Join_{\Pi,\Gamma} \delta) \wedge \neg(\delta' \,\text{✂}\, \delta)$$

Figure 41. Auxiliary Definitions for CRDTs with the Cancel-Win Pattern.

I Verifying Add-Wins Sets and Remove-Wins Sets

As we explained, algorithms like add-wins sets and remove-wins sets have more relaxed behaviors and cannot satisfy ACC. Their correctness XACC relies on causal delivery and the cancellation property of abstract operations. In this section we explain our proof method for XACC, and apply the proof method to verify add-wins sets and remove-wins sets.

Our proof method is based on the following properties of the add-wins sets and remove-wins sets: ($\Join = (\rhd \cup \rhd^{-1})$) and $\mathsf{cancel}(\rhd^{-1})$.

I.1 Proof Method

We ask users to provide ✂, $\mathcal{V}$, $\mathbf{+}$ and $\mathbf{-}$ to facilitate the proofs. Fig. 41 shows the types of ✂, $\mathbf{+}$ and $\mathbf{-}$.

We introduce ✂ as the concrete implementation of $\rhd$. It also specifies the particular order between non-commutative *effectors* in add-wins sets and remove-wins sets. Unlike RGA, not all effectors are commutative now. If `(e,i)` $\in$ `R` holds, `Add(e,i)` (①) and `Rmv(R)` (②) are *not* commutative. But in this case, ① must happen before ②. Under causal delivery, all the nodes execute the two effectors in the same order ①②, so the algorithm can still ensure SEC. Intuitively, when we map the concrete executions to the abstract level, we should execute the corresponding abstract operations in the same particular order. We introduce ✂ (a partial order between effectors) to specify the particular order between non-commutative effectors.

As in CRDT-TS in Sec. 8, we ask users to provide the view function $\mathcal{V}$. In the add-wins sets, we let an add win over a concurrent remove only if the add can be "seen" ($\mathcal{V}$) from the state at which the remove applies.

As the concrete implementation of the strategy "X wins Y", we ask users to provide two disjoint sets $\mathbf{+}$ and $\mathbf{-}$ of effectors, where the effectors in $\mathbf{+}$ could win over the effectors in $\mathbf{-}$, but not the other way round. For the add-wins set, $\mathbf{+}$ includes all the `Add` effectors, while $\mathbf{-}$ includes all the `Rmv` effectors. For the remove-wins set, $\mathbf{+}$ is the set of `Rmv` effectors, while $\mathbf{-}$ is the set of `Add` effectors.

Our main proof obligations are $\mathsf{sameRVal}_{\varphi}(\Pi, \Gamma)$ and $\mathsf{step\text{-}CW}_{\varphi}(\Pi, (\Gamma, \Join), \mathcal{V}, \mathbf{+}, \mathbf{-}, \text{✂})$, formulated below (except that sameRVal is in Sec. 8). We also need a set of conditions, uniqView, wfC and wfWL, to check well-formedness of the user-specified $\mathcal{V}$, ✂, $\mathbf{+}$ and $\mathbf{-}$.

Definition 82 (φ-Preservation for CRDT-CW). $\mathsf{step\text{-}CW}_{\varphi}(\Pi, (\Gamma, \Join), \mathcal{V}, \mathbf{+}, \mathbf{-}, \text{✂})$ iff

1. If δ loses at $\mathcal{S}$, then it has no effect at the abstract level.
$$\forall f, n, \delta, \mathcal{S}, \mathcal{S}', \mathcal{S}_a.\ \mathsf{valid}_{\Pi}(f, n, \delta) \wedge \varphi(\mathcal{S}) = \mathcal{S}_a \wedge \delta(\mathcal{S}) = \mathcal{S}' \wedge \mathsf{loseAt}_{\Pi}(\delta, \mathcal{S}, \mathcal{V}, \mathbf{+}, \mathbf{-}, \text{✂}, (\Gamma, \Join))$$
$$\implies \varphi(\mathcal{S}') = \mathcal{S}_a$$
2. If δ does not lose at $\mathcal{S}$, then it corresponds to executing the related abstract operation.
$$\forall f, n, \delta, \mathcal{S}, \mathcal{S}', \mathcal{S}_a.\ \mathsf{valid}_{\Pi}(f, n, \delta) \wedge \varphi(\mathcal{S}) = \mathcal{S}_a \wedge \delta(\mathcal{S}) = \mathcal{S}' \wedge \neg\mathsf{loseAt}_{\Pi}(\delta, \mathcal{S}, \mathcal{V}, \mathbf{+}, \mathbf{-}, \text{✂}, (\Gamma, \Join))$$
$$\implies \exists \mathcal{S}'_a.\ \varphi(\mathcal{S}') = \mathcal{S}'_a \wedge \Gamma(f, n)(\mathcal{S}_a) = (\_, \mathcal{S}'_a)$$

Here $\mathsf{loseAt}_{\Pi}(\delta, \mathcal{S}, \mathcal{V}, \mathbf{+}, \mathbf{-}, \text{✂}, (\Gamma, \Join))$ is defined in Fig. 41, which says δ is won by some effector at state $\mathcal{S}$. In the definition, we lift $\Join$ to effectors.

Definition 83 (Unique $\mathcal{V}$). $\mathsf{uniqView}_{\psi,\Pi}(\mathcal{V})$ iff (here we write $\delta \in \mathcal{V}$ for $\exists \mathcal{S}.\ \delta \in \mathcal{V}(\mathcal{S})$)

1. $\mathsf{wfV}_{\psi}(\mathcal{V})$ (see Def. 62) holds.
2. A "seeable" effector is generated at a unique state. That is,
$$\forall \delta, \mathcal{S}, \mathcal{S}'.\ (\delta \in \mathcal{V}) \wedge \mathsf{genAt}_{\Pi}(\mathcal{S}, \delta) \wedge \mathsf{genAt}_{\Pi}(\mathcal{S}', \delta) \implies \mathcal{S} = \mathcal{S}'$$
3. The state generating a "seeable" effector cannot appear twice. That is, there exist an irreflexive relation $\prec$ over states, such that
$$\forall \delta, \mathcal{S}, \mathcal{S}'.\ (\delta \in \mathcal{V}) \wedge \mathsf{genAt}_{\Pi}(\mathcal{S}, \delta) \wedge (\delta(\mathcal{S}) = \mathcal{S}') \implies \mathcal{S} \prec \mathcal{S}'$$
$$\forall \mathcal{S}, \mathcal{S}', \mathcal{S}'', \delta.\ (\mathcal{S} \prec \mathcal{S}') \wedge (\delta(\mathcal{S}') = \mathcal{S}'') \implies \mathcal{S} \prec \mathcal{S}''$$

```
var S := ∅;
var u := (0,cid);

operation lookup(e){
  return (∃w.(e,w) ∈ S);
  gen_eff IdEff;
}

operation add(e){
  return;
  gen_eff Add(e, u);
}

effector Add(e, i){
  S := S ∪ {(e, i)};
  if (i.snd = cid)
    u:=(i.fst+1,cid);
}

operation remove(e){
  assume (lookup(e));
  local R;
  R := {(e,w)|(e,w) ∈ S};
  return;
  gen_eff Rmv(R);
}

effector Rmv(R){
  S := S - R;
}
```

Figure 42. The Add-Wins Set

```
LOOKUP(e){              ADD(e){              REMOVE(e){
  return (e ∈ S);         S := S ∪ {e};        S := S - {e};
}                       }                    }
```

$$\begin{aligned} \alpha \bowtie \alpha' &\quad \text{iff} \quad \exists \mathsf{a}.\ \alpha = \mathsf{add(a)} \wedge \alpha' = \mathsf{rmv(a)}\ \vee\ \alpha = \mathsf{rmv(a)} \wedge \alpha' = \mathsf{add(a)} \\ \alpha \blacktriangleleft \alpha' &\quad \text{iff} \quad \exists \mathsf{a}.\ \alpha = \mathsf{rmv(a)} \wedge \alpha' = \mathsf{add(a)} \\ \alpha \triangleright \alpha' &\quad \text{iff} \quad \exists \mathsf{a}.\ \alpha = \mathsf{add(a)} \wedge \alpha' = \mathsf{rmv(a)} \end{aligned}$$

Figure 43. The Specification for Add-Wins Sets

For add-wins sets, this condition captures the fact that the tags for add operations are uniquely generated.

Definition 84 (Well-Formed $\mathrel{\succ\!\!\circ}$). $\mathsf{wfC}_\Pi(\mathrel{\succ\!\!\circ}, \mathcal{V}, (\Gamma, \triangleright))$ iff the following hold:

1. If $\delta' \mathrel{\succ\!\!\circ} \delta$, then δ' must be seen at the state generating δ.
$$\forall \mathcal{S}, \delta.\ \mathsf{genAt}_\Pi(\mathcal{S}, \delta) \implies \forall \delta'.\ (\delta' \mathrel{\succ\!\!\circ} \delta) \implies \delta' \in \mathcal{V}(\mathcal{S})$$
2. $\delta' \mathrel{\succ\!\!\circ} \delta$ if and only if δ' disappears after a step of δ.
$$\forall \delta, \delta', \mathcal{S}, \mathcal{S}'.\ (\delta(\mathcal{S}) = \mathcal{S}') \wedge (\delta' \in \mathcal{V}(\mathcal{S})) \implies (\delta' \mathrel{\succ\!\!\circ} \delta \iff \delta' \notin \mathcal{V}(\mathcal{S}'))$$
3. $\mathrel{\succ\!\!\circ}$ corresponds to the abstract canceled-by relation: $\forall \delta_1, \delta_2.\ (\delta_1 \mathrel{\succ\!\!\circ} \delta_2) \implies (\delta_1 \triangleright_{\Pi,\Gamma} \delta_2)$.

Definition 85 (Well-Formed $+$ and $-$). $\mathsf{wfWL}_\Pi(+, -, \mathcal{V}, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$ iff the following hold:

1. Conflicts are between $+$ and $-$: $\forall \delta_1, \delta_2.\ (\delta_1 \bowtie_{\Pi,\Gamma} \delta_2) \implies (\delta_1 \in + \wedge \delta_2 \in -) \vee (\delta_2 \in + \wedge \delta_1 \in -)$.
2. The won-by relation $\blacktriangleleft$ is between $-$ and $+$: $\forall \delta_1, \delta_2.\ (\delta_1 \blacktriangleleft_{\Pi,\Gamma} \delta_2) \implies (\delta_1 \in - \wedge \delta_2 \in +)$.
3. The canceled-by relation $\triangleright$ is between $+$ and $-$: $\forall \delta_1, \delta_2.\ (\delta_1 \triangleright_{\Pi,\Gamma} \delta_2) \implies (\delta_1 \in + \wedge \delta_2 \in -)$.
4. A $+$ effector is canceled by a $-$ effector on the same element. That is,
$$\forall \delta_1, \delta_2.\ (\delta_1\ (\bowtie_{\Pi,\Gamma})^+\ \delta_2) \wedge (\delta_1 \in +) \wedge (\delta_2 \in -) \implies (\delta_1 \triangleright_{\Pi,\Gamma} \delta_2)$$
5. Winners can always be seen after applied: $\forall \delta, \mathcal{S}, \mathcal{S}'.\ (\delta \in +) \wedge (\delta(\mathcal{S}) = \mathcal{S}') \implies \delta \in \mathcal{V}(\mathcal{S}')$.
6. If δ is generated at $\mathcal{S}$, then δ does not lose at $\mathcal{S}$. That is,
$$\forall \delta, \mathcal{S}.\ \mathsf{genAt}_\Pi(\mathcal{S}, \delta) \implies \neg \mathsf{loseAt}_\Pi(\delta, \mathcal{S}, \mathcal{V}, +, -, \mathrel{\succ\!\!\circ}, (\Gamma, \bowtie))$$
7. $+$ and $-$ are disjoint: $+ \cap - = \emptyset$.

Definition 86 (CRDTs with Cancel-Win). $\mathsf{CRDT\text{-}CW}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$ iff

$$\begin{aligned} &\exists +, -, \mathrel{\succ\!\!\circ}, \mathcal{V}.\ \mathsf{sameRVal}_\varphi(\Pi, \Gamma) \wedge \mathsf{step\text{-}CW}_\varphi(\Pi, (\Gamma, \bowtie), \mathcal{V}, +, -, \mathrel{\succ\!\!\circ}) \\ &\wedge \mathsf{uniqView}_{\psi,\Pi}(\mathcal{V}) \wedge \mathsf{wfC}_\Pi(\mathrel{\succ\!\!\circ}, \mathcal{V}, (\Gamma, \triangleright)) \wedge \mathsf{wfWL}_\Pi(+, -, \mathcal{V}, (\Gamma, \bowtie, \blacktriangleleft, \triangleright)) \end{aligned}$$

Theorem 87. Suppose $\mathsf{nonComm}(\Gamma, \bowtie)$, $(\bowtie = (\triangleright \cup \triangleright^{-1}))$, $\mathsf{cancel}(\triangleright)$ and $\mathsf{cancel}(\triangleright^{-1})$. Then, $\mathsf{CRDT\text{-}CW}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright)) \implies \mathsf{XACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$.

I.2 Applying the Proof Method to Add-Wins Sets and Remove-Wins Sets

I.2.1 The Add-Wins Set Below we verify the add-wins sets [19] Π_{awset} in Fig. 42. Its specification $(\Gamma_{\text{set}}, \bowtie, \blacktriangleleft, \triangleright)$ is shown in Fig. 43. We let ψ relate the initial states $\mathcal{S}$ and $\mathcal{S}_a$ where the sets at the two levels are both empty.

$$\begin{array}{lll} \psi(\mathcal{S}) = \mathcal{S}_a & \text{iff} & \mathcal{S}(\mathsf{S}) = \emptyset \wedge \mathcal{S}_a(\mathsf{S}) = \emptyset \\ \varphi(\mathcal{S}) = \mathcal{S}_a & \text{iff} & \lfloor \mathcal{S}(\mathsf{S}) \rfloor = \mathcal{S}_a(\mathsf{S}) \end{array}$$

Here $\lfloor \mathsf{S} \rfloor$ returns a set consisting of elements which are projected from the tagged elements in the add-wins set S.

To verify the algorithm, we first define $\rightarrowtail$, $\mathcal{V}$, $+$ and $-$ as follows.

$$\begin{array}{rcl} \delta \rightarrowtail \delta' & \text{iff} & \exists \mathsf{e}, \mathsf{i}, \mathsf{R}.\ (\delta = \mathtt{Add(e,i)}) \wedge (\delta' = \mathtt{Rmv(R)}) \wedge ((\mathsf{e}, \mathsf{i}) \in \mathsf{R}) \wedge (\lfloor \mathsf{R} \rfloor = \{\mathsf{e}\}) \\ \mathcal{V}(\mathcal{S}) & \overset{\text{def}}{=} & \{\delta \mid \exists \mathsf{e}, \mathsf{i}.\ ((\mathsf{e}, \mathsf{i}) \in \mathcal{S}(\mathsf{S})) \wedge (\delta = \mathtt{Add(e,i)})\} \\ + & \overset{\text{def}}{=} & \{\delta \mid \exists \mathsf{e}, \mathsf{i}.\ \delta = \mathtt{Add(e,i)}\} \\ - & \overset{\text{def}}{=} & \{\delta \mid \exists \mathsf{R}.\ \delta = \mathtt{Rmv(R)}\} \end{array}$$

The effector δ' cancels δ, i.e., $\delta \rightarrowtail \delta'$, if δ and δ' are Add and Rmv of the same element respectively, and δ is visible to δ'. An effector δ can be seen by $\mathcal{V}$ at the state $\mathcal{S}$, i.e., $\delta \in \mathcal{V}(\mathcal{S})$, if δ is an `Add(e, i)` and `(e, i)` is in the set in $\mathcal{S}$. The sets $+$ and $-$ contain Add and Rmv effectors respectively.

We can prove all the conditions in $\text{CRDT-CW}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$. By Theorem 87, we get $\text{XACC}_{\psi,\varphi}(\Pi_{\text{awset}}, (\Gamma_{\text{set}}, \bowtie, \blacktriangleleft, \triangleright))$.

```
var S := ∅;
var u := (0, cid);

operation lookup(e){
  return (∃w. (e, true, w) ∈ S ∧ ¬(∃w'. (e, false, w') ∈ S));
  gen_eff IdEff;
}

operation add(e){
  local R;
  R := {elem | elem = (e, _, _) ∧ elem ∈ S};
  return;
  gen_eff Add(e, u, R);
}

effector Add(e, i, R){  // Assume causal delivery
  S := S - R;
  if (¬(∃w'. (e, false, w') ∈ S))
    S := S ∪ {(e, true, i)};
  if (i.snd = cid) u := (i.fst + 1, cid); // set to the next fresh tag
}

operation remove(e){
  assume (lookup(e));
  return;
  gen_eff Rmv(e, u);
}

effector Rmv(e, i){  // Assume causal delivery
  S := S ∪ {(e, false, i)};
  if (i.snd = cid) u := (i.fst + 1, cid); // set to the next fresh tag
}
```

Figure 44. The Remove-Wins Set

I.2.2 The Remove-Wins Set Figure 44 shows the remove-wins set implementation Π_{rwset}. Every element `elem` is in the form of `(e, b, u)`, where `b` is boolean.

Its specification Γ_{set} and conflicting relation $\bowtie$ are the same as the ones for the add-wins set (see Fig. 43), but its $\blacktriangleleft$ and $\triangleright$ are the reverse:

$$\alpha \blacktriangleleft \alpha' \quad \text{iff} \quad \exists \mathtt{a}.\ \alpha = \mathtt{add(a)} \wedge \alpha' = \mathtt{rmv(a)}$$

$$\alpha \triangleright \alpha' \quad \text{iff} \quad \exists \mathtt{a}.\ \alpha = \mathtt{rmv(a)} \wedge \alpha' = \mathtt{add(a)}$$

We let φ relates $\mathcal{S}$ and $\mathcal{S}_a$ such that $\mathcal{S}_a(\mathtt{S})$ contains all the elements that can be looked up from $\mathcal{S}$. And ψ still relates the initial states $\mathcal{S}$ and $\mathcal{S}_a$ where the sets at the two levels are both empty.

$$\begin{array}{lll} \psi(\mathcal{S}) = \mathcal{S}_a & \text{iff} & \mathcal{S}(\mathtt{S}) = \emptyset \wedge \mathcal{S}_a(\mathtt{S}) = \emptyset \\ \varphi(\mathcal{S}) = \mathcal{S}_a & \text{iff} & \mathcal{S}_a(\mathtt{S}) = \{\mathtt{e} \mid \exists \mathtt{w}.\ (\mathtt{e,\ true,\ w}) \in \mathcal{S}(\mathtt{S}) \wedge \neg(\exists \mathtt{w'}.\ (\mathtt{e,\ false,\ w'}) \in \mathcal{S}(\mathtt{S}))\} \end{array}$$

To verify the algorithm, we first define $\rtimes$, $\mathcal{V}$, $+$ and $-$ as follows.

$$\begin{array}{lll} \delta \rtimes \delta' & \text{iff} & \exists \mathtt{e, i, i', R'}.\ (\delta = \mathtt{Rmv(e,\ i)}) \wedge (\delta' = \mathtt{Add(e,\ i',\ R')}) \\ & & \wedge\ ((\mathtt{e,\ false,\ i}) \in \mathtt{R'}) \\ \mathcal{V}(\mathcal{S}) & \overset{\text{def}}{=} & \{\delta \mid \exists \mathtt{e, i}.\ (\mathtt{e,\ false,\ i}) \in \mathcal{S}(\mathtt{S}) \wedge \delta = \mathtt{Rmv(e,\ i)}\} \\ + & \overset{\text{def}}{=} & \{\delta \mid \exists \mathtt{e, i}.\ \delta = \mathtt{Rmv(e,\ i)}\} \\ - & \overset{\text{def}}{=} & \{\delta \mid \exists \mathtt{e, i, R}.\ \delta = \mathtt{Add(e,\ i,\ R)}\} \end{array}$$

For the remove-wins set, we let an Add effector cancels a Rmv effector, and $\mathcal{V}$ gives the Rmv effectors visible in the state. The sets $+$ and $-$ contain Rmv and Add effectors respectively.

We can prove all the conditions in $\text{CRDT-CW}_{\psi,\varphi}(\Pi_{\text{rwset}}, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$. By Theorem 87, we get $\text{XACC}_{\psi,\varphi}(\Pi_{\text{rwset}}, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$.

I.3 Soundness of the Proof Method

Proof of Theorem 87. By applying Lemma 89 and Lemma 90. □

Definition 88. $\text{E-XACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$, iff

$$\begin{array}{l} \forall \mathcal{S}, \mathcal{S}_a, \mathcal{E}.\ \mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S}) \wedge \text{eventualDelivery}(\mathcal{E}) \wedge \text{causalDelivery}(\mathcal{E}) \wedge \psi(\mathcal{S}) = \mathcal{S}_a \\ \quad \Longrightarrow\ \text{XACT}_{\varphi}(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie, \blacktriangleleft, \triangleright)) \end{array}$$

Lemma 89 (E-XACC implies XACC). If $\text{E-XACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$, then $\text{XACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$.

Proof. For any $\mathcal{S}$, $\mathcal{S}_a$ and $\mathcal{E}$, if $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, causalDelivery$(\mathcal{E})$ and $\psi(\mathcal{S}) = \mathcal{S}_a$, we know there exist $\mathcal{E}'$ and $\mathcal{E}''$ such that

$$\mathcal{E}' = \mathcal{E}{+\!+}\mathcal{E}'',\ \forall e \in \mathcal{E}''.\ \text{is\_recv}(e),$$
$$\mathcal{E}' \in \mathcal{T}(\Pi, \mathcal{S}),\ \text{causalDelivery}(\mathcal{E}') \text{ and eventualDelivery}(\mathcal{E}').$$

By $\text{E-XACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$, we know

$$\text{XACT}_{\varphi}(\mathcal{E}', \mathcal{S}, (\Gamma, \bowtie, \blacktriangleleft, \triangleright)).$$

From $\text{XACT}_{\varphi}(\mathcal{E}', \mathcal{S}, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$, we know there exist $ar'_1, \ldots, ar'_n$ such that, for any t, we have

$$\begin{array}{c} \text{totalOrder}_{\text{visible}(\mathcal{E}',\mathrm{t})}(ar'_{\mathrm{t}}),\ \underset{\mathrm{t}}{\overset{\text{vis}}{\longmapsto}}_{\mathcal{E}'} \subseteq ar'_{\mathrm{t}},\ \text{PresvCancel}(ar'_{\mathrm{t}}, \mathrm{t}, \mathcal{E}', (\Gamma, \triangleright)),\ \text{ExecRelated}_{\varphi}(\mathrm{t}, (\mathcal{E}', \mathcal{S}), (\Gamma, ar'_{\mathrm{t}})), \\ \forall \mathrm{t}' \neq \mathrm{t}.\ \text{RCoh}_{(\mathrm{t},\mathrm{t}')}((ar'_{\mathrm{t}}, ar'_{\mathrm{t}'}), \mathcal{E}', (\Gamma, \bowtie, \blacktriangleleft, \triangleright)). \end{array}$$

Since $\mathcal{E}' = \mathcal{E}{+\!+}\mathcal{E}''$ and $\forall e \in \mathcal{E}''.\ \text{is\_recv}(e)$, we know

$$\text{visible}(\mathcal{E}, \mathrm{t}) \subseteq \text{visible}(\mathcal{E}', \mathrm{t}).$$

Let $ar_{\mathrm{t}} = ar'_{\mathrm{t}}|_{\text{visible}(\mathcal{E},\mathrm{t})}$. From $\underset{\mathrm{t}}{\overset{\text{vis}}{\longmapsto}}_{\mathcal{E}'} \subseteq ar'_{\mathrm{t}}$, we know

$$\underset{\mathrm{t}}{\overset{\text{vis}}{\longmapsto}}_{\mathcal{E}} \subseteq ar_{\mathrm{t}}.$$

From $\text{PresvCancel}(ar'_{\mathrm{t}}, \mathrm{t}, \mathcal{E}', (\Gamma, \triangleright))$, we know

$$\text{PresvCancel}(ar_{\mathrm{t}}, \mathrm{t}, \mathcal{E}, (\Gamma, \triangleright)).$$

From $\text{ExecRelated}_{\varphi}(\mathrm{t}, (\mathcal{E}', \mathcal{S}), (\Gamma, ar'_{\mathrm{t}}))$, we know

$$\text{ExecRelated}_{\varphi}(\mathrm{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathrm{t}})).$$

For any $\mathrm{t}' \neq \mathrm{t}$, from $\text{RCoh}_{(\mathrm{t},\mathrm{t}')}((ar'_{\mathrm{t}}, ar'_{\mathrm{t}'}), \mathcal{E}', (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$, we know

$$e_1 \succ\!\!\circ\, e_2 \text{ iff } \mathsf{eff}(e_1) \succ\!\!\circ\, \mathsf{eff}(e_2) \qquad e \in + \text{ iff } \mathsf{eff}(e) \in + \qquad e \in - \text{ iff } \mathsf{eff}(e) \in -$$

$$\mathsf{wfCGen}_{\Pi}(\succ\!\!\circ, \mathcal{V}) \text{ iff } \forall \mathcal{S}, \delta.\ \mathsf{genAt}_{\Pi}(\mathcal{S}, \delta) \implies \forall \delta'.\ (\delta' \succ\!\!\circ\, \delta) \implies \delta' \in \mathcal{V}(\mathcal{S})$$

$$\begin{array}{l} \mathsf{uniqSeeT}_{\Pi,\psi}(\mathcal{V}) \text{ iff} \\ \quad \forall \mathcal{S}, \mathcal{E}.\ \mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S}) \wedge (\mathcal{S} \models \psi) \\ \quad \implies \forall e_1, e_2.\ (e_1 \in \mathcal{E}) \wedge (e_2 \in \mathcal{E}) \wedge (\mathsf{eff}(e_1) = \mathsf{eff}(e_2)) \wedge (\mathsf{eff}(e_1) \in \mathcal{V}) \\ \quad \implies \mathsf{msgid}(e_1) = \mathsf{msgid}(e_2) \end{array}$$

$$\mathsf{loserWinnerDisj}(+, -) \text{ iff } + \cap - = \emptyset$$

$$\begin{array}{l} \mathsf{conflictWL}(+, -, (\Pi, \Gamma, \bowtie)) \text{ iff} \\ \quad \forall \delta_1, \delta_2.\ (\delta_1 \bowtie_{\Pi,\Gamma} \delta_2) \implies (\delta_1 \in + \wedge \delta_2 \in -) \vee (\delta_2 \in + \wedge \delta_1 \in -) \end{array}$$

$$\mathsf{wlConflict}(+, -, (\Pi, \Gamma, \bowtie, \triangleright)) \text{ iff } \forall \delta_1, \delta_2.\ (\delta_1 (\bowtie_{\Pi,\Gamma})^+ \delta_2) \wedge (\delta_1 \in +) \wedge (\delta_2 \in -) \implies (\delta_1 \triangleright_{\Pi,\Gamma} \delta_2)$$

$$\begin{array}{l} \mathsf{genNotLose}(+, -, \mathcal{V}, \succ\!\!\circ, (\Pi, \Gamma, \bowtie)) \text{ iff} \\ \quad \forall \delta, \mathcal{S}.\ \mathsf{genAt}_{\Pi}(\mathcal{S}, \delta) \implies \neg \mathsf{loseAt}_{\Pi}(\delta, \mathcal{S}, \mathcal{V}, +, -, \succ\!\!\circ, (\Gamma, \bowtie)) \end{array}$$

$$\mathsf{notVCancelee}(\succ\!\!\circ, \mathcal{V}) \text{ iff } \forall \delta, \mathcal{S}, \mathcal{S}'.\ (\delta(\mathcal{S}) = \mathcal{S}') \implies \forall \delta' \in (\mathcal{V}(\mathcal{S}) - \mathcal{V}(\mathcal{S}')) \implies \delta' \succ\!\!\circ\, \delta$$

$$\mathsf{canceleeNotV}(\succ\!\!\circ, \mathcal{V}) \text{ iff } \forall \delta, \delta', \mathcal{S}, \mathcal{S}'.\ (\delta(\mathcal{S}) = \mathcal{S}') \wedge \delta' \in \mathcal{V}(\mathcal{S}) \wedge \delta' \succ\!\!\circ\, \delta \implies \delta' \notin \mathcal{V}(\mathcal{S}')$$

$$\begin{array}{l} \mathsf{loserGenCancelWinner}(\succ\!\!\circ, +, -, \mathcal{V}, (\Pi, \Gamma, \bowtie)) \text{ iff} \\ \quad \forall \delta, \delta', \mathcal{S}, \mathcal{S}'.\ \delta \in - \wedge \mathsf{genAt}_{\Pi}(\mathcal{S}, \delta) \wedge \delta' \in \mathcal{V}(\mathcal{S}) \wedge \delta' \in + \wedge (\delta' \bowtie_{\Pi,\Gamma} \delta) \implies \delta' \succ\!\!\circ\, \delta \end{array}$$

$$\mathsf{winnerSee}(+, \mathcal{V}) \text{ iff } \forall \delta, \mathcal{S}, \mathcal{S}'.\ (\delta \in +) \wedge (\delta(\mathcal{S}) = \mathcal{S}') \implies \delta \in \mathcal{V}(\mathcal{S}')$$

$$\mathsf{cancelAbsCancel}(\succ\!\!\circ, (\Pi, \Gamma, \triangleright)) \text{ iff } \forall \delta_1, \delta_2.\ (\delta_1 \succ\!\!\circ\, \delta_2) \implies (\delta_1 \triangleright_{\Pi,\Gamma} \delta_2)$$

$$\mathsf{abswonbyWL}(+, -, (\Pi, \Gamma, \blacktriangleleft)) \text{ iff } \forall \delta_1, \delta_2.\ (\delta_1 \blacktriangleleft_{\Pi,\Gamma} \delta_2) \implies (\delta_1 \in -) \wedge (\delta_2 \in +)$$

$$\begin{array}{l} \mathsf{ccCoh}(\mathcal{E}, \mathcal{E}', (\Gamma, \bowtie, \triangleright)) \text{ iff} \\ \quad \forall e_0, e_1.\ (e_0 <_{\mathcal{E}} e_1) \wedge (e_1 <_{\mathcal{E}'} e_0) \\ \quad \implies \neg(\Gamma \models e_0 \bowtie e_1) \vee \exists i \in \{0, 1\}.\ \exists e.\ (\Gamma \models e_i \triangleright e) \wedge (e_i <_{\mathcal{E}} e) \wedge (e_i <_{\mathcal{E}'} e) \end{array}$$

$$\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \succ\!\!\circ, +, -, (\Gamma, \bowtie, \triangleright)) \overset{\text{def}}{=} (\xrightarrow[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \cup (\xrightarrow{\mathsf{vis}}_{\mathcal{E}} \cap \triangleright_{\Gamma}) \cup \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\triangleright})^+$$

$$\begin{array}{l} \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\triangleright}(e, e') \text{ iff} \\ \quad (\Gamma \models e \bowtie e') \wedge \{e, e'\} \subseteq \mathsf{orig}(\mathcal{E}) \wedge e \in - \wedge e' \in + \\ \quad \wedge \neg \mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\triangleright}(e, e') \wedge \neg \mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\triangleright}(e', e) \end{array}$$

$$\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\triangleright}(e, e') \text{ iff } \exists e''.\ (\Gamma \models e \triangleright e'') \wedge (e \xrightarrow{\mathsf{vis}}_{\mathcal{E}} e'') \wedge (e'' <^{\mathsf{t}}_{\mathcal{E}} e' \vee e'' = e')$$

Figure 45. Auxiliary Definitions for the Soundness Proof of the Proof Method with Cancel-Win.

$$\mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((ar_{\mathsf{t}}, ar_{\mathsf{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \triangleright)).$$

Thus $\mathsf{XACT}_{\varphi}(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$. Thus we are done. □

Lemma 90 (CRDT-CW implies E-XACC)**.** Suppose $\mathsf{nonComm}(\Gamma, \bowtie)$, $(\bowtie = (\triangleright \cup \triangleright^{-1}))$, $\mathsf{cancel}(\triangleright)$ and $\mathsf{cancel}(\triangleright^{-1})$. Then, $\mathsf{CRDT\text{-}CW}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright)) \implies \mathsf{E\text{-}XACC}_{\psi,\varphi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \triangleright))$.

Proof. For any $\mathcal{S}$, $\mathcal{S}_a$ and $\mathcal{E}$, suppose $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\mathsf{eventualDelivery}(\mathcal{E})$, $\mathsf{causalDelivery}(\mathcal{E})$ and $\psi(\mathcal{S}) = \mathcal{S}_a$.
Since $\mathsf{eventualDelivery}(\mathcal{E})$, we know

$$\forall \mathsf{t}.\ \mathsf{visible}(\mathcal{E}, \mathsf{t}) = \mathsf{orig}(\mathcal{E}).$$

By $\mathsf{CRDT\text{-}CW}_{\psi}(\Pi, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$, we know there exist $+$, $-$, $\rightarrowtail$ and $\mathcal{V}$ such that

$$\mathsf{sameRVal}_{\varphi}(\Pi, \Gamma),\ \mathsf{step\text{-}CW}_{\varphi}(\Pi, (\Gamma, \bowtie), \mathcal{V}, +, -, \rightarrowtail),\ \mathsf{uniqView}_{\psi,\Pi}(\mathcal{V}),\ \mathsf{wfC}_{\Pi}(\rightarrowtail, \mathcal{V}, (\Gamma, \bowtie)),\ \mathsf{wfWL}_{\Pi}(+, -, \mathcal{V}, (\Gamma, \bowtie, \blacktriangleleft, \rhd)).$$

Below we prove $\mathsf{XACT}_{\varphi}(\mathcal{E}, \mathcal{S}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$. For any t, we first define $\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \rightarrowtail, +, -, (\Gamma, \bowtie, \rhd))$ in Figure 45. By Lemma 91, we know

$$\mathsf{partialOrder}(\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \rightarrowtail, +, -, (\Gamma, \bowtie, \rhd))).$$

So there exists ar_{t} such that $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar_{\mathsf{t}})$ and $\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \rightarrowtail, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar_{\mathsf{t}}$. Thus

$$\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \subseteq ar_{\mathsf{t}} \text{ and } \mathsf{PresvCancel}(ar_{\mathsf{t}}, \mathsf{t}, \mathcal{E}, (\Gamma, \rhd)).$$

- Below we prove $\mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathsf{t}}))$.
  We first prove $\mathsf{StRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathcal{S}_a, ar_{\mathsf{t}}))$ by applying Lemma 92.
  Then, by Lemma 74, we know $\mathsf{RValRelated}(\mathsf{t}, \mathcal{E}, (\Gamma, \mathcal{S}_a, ar_{\mathsf{t}}))$.
  Thus $\mathsf{ExecRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, ar_{\mathsf{t}}))$.
- We prove $\forall \mathsf{t}' \neq \mathsf{t}.\ \mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((ar_{\mathsf{t}}, ar_{\mathsf{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$ by Lemma 98.

Thus we are done. □

Lemma 91 (vpa is partial order)**.** If

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\mathcal{S} \in dom(\psi)$, $\mathsf{eventualDelivery}(\mathcal{E})$, $\mathsf{causalDelivery}(\mathcal{E})$,
2. $\mathsf{loserWinnerDisj}(+, -)$, $\mathsf{wlConflict}(+, -, (\Pi, \Gamma, \bowtie, \rhd))$, $\mathsf{conflictWL}(+, -, (\Pi, \Gamma, \bowtie))$,
3. $\rhd \subseteq \bowtie$,

then $\mathsf{partialOrder}(\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \rightarrowtail, +, -, (\Gamma, \bowtie, \rhd)))$.

Proof. Let $rel = (\xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \cup (\xmapsto{\mathsf{vis}}_{\mathcal{E}} \cap \rhd_{\Gamma}) \cup \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\rhd})$. We only need to prove $\neg\mathsf{cyclic}(rel)$.

By contradiction. Suppose there exist $n, e_1, \ldots, e_n$ such that $\forall i \in [1..n-1].\ (e_i, e_{i+1}) \in rel$ and $(e_n, e_1) \in rel$. Without loss of generality, we can suppose n is the length of the smallest cycle. We analyze the following two cases:

- $n = 1$. We know it is impossible from $\mathsf{loserWinnerDisj}(+, -)$ and the definition of rel.
- $n > 1$.
  Since $\mathsf{eventualDelivery}(\mathcal{E})$, we know
  $$\{e_1, \ldots, e_n\} \subseteq \mathsf{visible}(\mathcal{E}, \mathsf{t}).$$
  Without loss of generality, we can suppose e_n is the last event among $e_1, \ldots, e_n$ that t applies, that is, $\forall i \in [1..n-1].\ e_i \prec^{\mathsf{t}}_{\mathcal{E}} e_n$.
  By the definition of rel, we know
  $$(e_n, e_1) \in \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\rhd}.$$
  Thus
  $$\Gamma \models e_n \bowtie e_1,\ e_n \in -,\ e_1 \in +,$$
  $$\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e_n, e_1),\ \neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e_1, e_n).$$
  Since $\mathsf{loserWinnerDisj}(+, -)$, we know
  $$(e_1, e_2) \notin \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\rhd} \text{ and } (e_{n-1}, e_n) \notin \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\rhd}.$$
  Since $(e_1, e_2) \in ar$, we know
  $$(e_1, e_2) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} \cup (\xmapsto{\mathsf{vis}}_{\mathcal{E}} \cap \rhd_{\Gamma}).$$

  1. $(e_1, e_2) \in (\xmapsto{\mathsf{vis}}_{\mathcal{E}} \cap \rhd_{\Gamma})$. Thus we know
     $$\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e_1, e_n).$$
     So we get a contradiction.
  2. $(e_1, e_2) \in \xmapsto[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$ and $\neg(\Gamma \models e_1 \rhd e_2)$.
     We have two cases:

a. $\neg(\Gamma \models e_1(\bowtie)^+ e_2)$.
Thus $\neg(\Gamma \models e_2(\bowtie)^+ e_n)$. Since $\forall i \in [1..n-1].\ (e_i, e_{i+1}) \in \mathit{rel}$, we know there exists i such that $2 \le i < n$ and
$$\neg(\Gamma \models e_i \bowtie e_{i+1}).$$
Since $\rhd \subseteq \bowtie$, we know
$$(e_i, e_{i+1}) \in \xrightarrow[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}\ .$$

- If $e_1 \prec^{\mathsf{t}}_{\mathcal{E}} e_{i+1}$, then $e_1 \xrightarrow[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} e_{i+1}$. So we can construct a smaller cycle $e_1, e_{i+1}, \ldots, e_n, e_1$. Thus we get a contradiction.
- If $e_{i+1} \prec^{\mathsf{t}}_{\mathcal{E}} e_1$, then $e_i \prec^{\mathsf{t}}_{\mathcal{E}} e_2$. Thus $e_i \xrightarrow[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}} e_2$. So we can construct a smaller cycle $e_2, \ldots, e_i, e_2$. Thus we get a contradiction.

b. $(\Gamma \models e_1(\bowtie)^+ e_2)$.
Since $\neg(\Gamma \models e_1 \rhd e_2)$ and $e_1 \in +$, from $\mathsf{wlConflict}(+, -, (\Pi, \Gamma, \bowtie, \rhd))$, we know
$$e_2 \notin -.$$
Since $\Gamma \models e_n \bowtie e_1$, we know $\Gamma \models e_n(\bowtie)^+ e_2$. Since $\mathsf{conflictWL}(+, -, (\Pi, \Gamma, \bowtie))$, we know $e_2 \in + \vee e_2 \in -$. Thus
$$e_2 \in +.$$
Since $\mathsf{wlConflict}(+, -, (\Pi, \Gamma, \bowtie, \rhd))$, we know
$$\Gamma \models e_n \rhd e_2.$$
Since $\rhd \subseteq \bowtie$, we know
$$\Gamma \models e_n \bowtie e_2.$$
Since $e_2 \prec^{\mathsf{t}}_{\mathcal{E}} e_n$ and $\mathsf{causalDelivery}(\mathcal{E})$, we know
$$\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e_n, e_2).$$

- Below we prove $\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e_2, e_n)$.
By contradiction. Suppose $\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e_2, e_n)$. That is, there exists e'' such that $(\Gamma \models e_2 \rhd e'') \wedge (e_2 \xrightarrow{\mathsf{vis}}_{\mathcal{E}} e'') \wedge (e'' \prec^{\mathsf{t}}_{\mathcal{E}} e_n \vee e'' = e_n)$. Since $(e_1, e_2) \in \xrightarrow[\mathsf{t}]{\mathsf{vis}}_{\mathcal{E}}$ and $e_2 \xrightarrow{\mathsf{vis}}_{\mathcal{E}} e''$, from $\mathsf{causalDelivery}(\mathcal{E})$, we know
$$e_1 \xrightarrow{\mathsf{vis}}_{\mathcal{E}} e''.$$
Since $\Gamma \models e_2 \rhd e''$, from $\rhd \subseteq \bowtie$, we know
$$\Gamma \models e_2 \bowtie e''.$$
From $\mathsf{conflictWL}(+, -, (\Pi, \Gamma, \bowtie))$, we know
$$e'' \in -.$$
Since $\Gamma \models e_1(\bowtie)^+ e_2$ and $\Gamma \models e_2 \bowtie e''$, we know
$$\Gamma \models e_1(\bowtie)^+ e''.$$
Since $e_1 \in +$ and $e'' \in -$, from $\mathsf{wlConflict}(+, -, (\Pi, \Gamma, \bowtie, \rhd))$, we know
$$\Gamma \models e_1 \rhd e''.$$
Thus we have
$$\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e_1, e_n).$$
So we get a contradiction. Thus $\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e_2, e_n)$.

As a result, we know
$$(e_n, e_2) \in \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\rhd}.$$
So we can construct a smaller cycle $e_2, \ldots, e_n, e_2$. Thus we get a contradiction.

Thus we are done. □

Lemma 92 (tStRelated)**.** If

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\mathsf{eventualDelivery}(\mathcal{E})$, $\mathsf{causalDelivery}(\mathcal{E})$, $\psi(\mathcal{S}) = \mathcal{S}_a$, $\psi \Rightarrow \varphi$,
2. $\mathsf{step\text{-}CW}_{\varphi}(\Pi, (\Gamma, \bowtie), \mathcal{V}, +, -, \rightarrowtail)$,
3. $\mathsf{genNotLose}(+, -, \mathcal{V}, \rightarrowtail, (\Pi, \Gamma, \bowtie))$, $\mathsf{notVCancelee}(\rightarrowtail, \mathcal{V})$, $\mathsf{wfCGen}_{\Pi}(\rightarrowtail, \mathcal{V})$, $\mathsf{wfV}_{\psi}(\mathcal{V})$, $\mathsf{uniqView}_{\Pi}(\mathcal{V})$, $\mathsf{winnerSee}(+, \mathcal{V})$, $\mathsf{cancelAbsCancel}(\rightarrowtail, (\Pi, \Gamma, \rhd))$, $\mathsf{conflictWL}(+, -, (\Pi, \Gamma, \bowtie))$, $\mathsf{canceleeNotV}(\rightarrowtail, \mathcal{V})$,
4. $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar)$, $\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \rightarrowtail, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar$, $\mathsf{partialOrder}(\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \rightarrowtail, +, -, (\Gamma, \bowtie, \rhd)))$,
5. $\mathsf{nonComm}(\Gamma, \bowtie)$, $(\bowtie = (\rhd \cup \rhd^{-1}))$, $\mathsf{cancel}(\rhd)$, $\mathsf{cancel}(\rhd^{-1})$,

then $\mathsf{StRelated}_{\varphi}(\mathsf{t}, (\mathcal{E}, \mathcal{S}), (\Gamma, \mathcal{S}_a, ar))$.

Proof. For any $\mathcal{E}'$ if $\mathcal{E}' \leqslant \mathcal{E}$, we want to prove $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar)$. Suppose $|\mathcal{E}'| = n$. By induction over n.

1. $n = 0$. Trivial.

2. $n = m + 1$. Suppose $\mathcal{E}' = \mathcal{E}''{+}{+}[e]$. By the induction hypothesis, we know
$$\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}''|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}'', \mathsf{t}) \downharpoonleft ar).$$
Let $\mathcal{S}'' = \mathsf{exec\_st}(\mathcal{S}, \mathcal{E}''|_{\mathsf{t}})$ and $\mathcal{S}_a'' = \varphi(\mathcal{S}'')$. We do case analysis over e.
   a. $e = (mid, \mathsf{t}, (f, n, n', \delta))$. From the semantics we know there exists $\mathcal{S}'$ such that
$$\mathsf{genAt}_\Pi(\mathcal{S}'', \delta) \text{ and } \delta(\mathcal{S}'') = \mathcal{S}'.$$
   From $\mathsf{genNotLose}(+, -, \mathcal{V}, \succ\!\!\circ, (\Pi, \Gamma, \bowtie))$, we know
$$\neg\mathsf{loseAt}_\Pi(\delta, \mathcal{S}'', \mathcal{V}, +, -, \succ\!\!\circ, (\Gamma, \bowtie)).$$
   From $\mathsf{step\text{-}CW}_\varphi(\Pi, (\Gamma, \bowtie), \mathcal{V}, +, -, \succ\!\!\circ)$, we know there exists $\mathcal{S}_a'$ such that
$$\varphi(\mathcal{S}') = \mathcal{S}_a' \text{ and } \Gamma(f, n)(\mathcal{S}_a'') = (\_, \mathcal{S}_a').$$
   Also, since $\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \succ\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar$, we know
$$\forall e' \in \mathsf{visible}(\mathcal{E}'', \mathsf{t}).\ (e', e) \in ar.$$
   Thus we know
$$\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar).$$
   b. $e = (mid, \mathsf{t}, (f, n), \delta)$.
      i. $\neg\mathsf{loseAt}_\Pi(\delta, \mathcal{S}'', \mathcal{V}, +, -, \succ\!\!\circ, (\Gamma, \bowtie))$.
      Let $rel = \{(e', e) \mid e' \in \mathsf{visible}(\mathcal{E}'', \mathsf{t})\}$ and $rel' = (\mathsf{vpa}(\mathsf{t}, \mathcal{E}', \succ\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \cup rel)^+$. By Lemma 97, we know
$$\mathsf{partialOrder}(rel').$$
      Thus there exists ar' such that $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar')$ and $rel' \subseteq ar'$.
      Also, since $\mathcal{E}' \leqslant \mathcal{E}$ and $\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \succ\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar$, we know $\mathsf{vpa}(\mathsf{t}, \mathcal{E}', \succ\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar$. From Lemma 94, we know
$$\mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar').$$
      From $\mathsf{step\text{-}CW}_\varphi(\Pi, (\Gamma, \bowtie), \mathcal{V}, +, -, \succ\!\!\circ)$, we know there exists $\mathcal{S}_a'$ such that
$$\varphi(\mathcal{S}') = \mathcal{S}_a' \text{ and } \Gamma(f, n)(\mathcal{S}_a'') = (\_, \mathcal{S}_a').$$
      Also, since $rel \subseteq ar'$, we know
$$\forall e' \in \mathsf{visible}(\mathcal{E}'', \mathsf{t}).\ (e', e) \in ar'.$$
      Thus we know
$$\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar').$$
      Thus
$$\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar).$$
      ii. $\mathsf{loseAt}_\Pi(\delta, \mathcal{S}'', \mathcal{V}, +, -, \succ\!\!\circ, (\Gamma, \bowtie))$.
      Thus
$$\delta \in - \wedge \exists \delta'.\ \delta' \in \mathcal{V}(\mathcal{S}'') \wedge \delta' \in + \wedge (\delta' \bowtie_{\Pi,\Gamma} \delta) \wedge \neg(\delta' \succ\!\!\circ\, \delta).$$
      From $\mathsf{wfV}_\psi(\mathcal{V})$, we know there exists e' such that $\mathsf{eff}(e') = \delta'$ and $e' \in \mathsf{visible}(\mathcal{E}'', \mathsf{t})$. Since $\mathsf{causalDelivery}(\mathcal{E})$, we know
$$\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e, e').$$
      Also, by Lemma 93, we know
$$\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e', e).$$
      From $\delta' \bowtie_{\Pi,\Gamma} \delta$, we know $\Gamma \models e \bowtie e'$. Thus
$$\mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\rhd}(e, e').$$
      Since $\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \succ\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar$, we know
$$(e, e') \in ar.$$
      Since $\Gamma \models e \bowtie e'$, from $(\bowtie = (\rhd \cup \rhd^{-1}))$, $\mathsf{cancel}(\rhd)$ and $\mathsf{cancel}(\rhd^{-1})$, we know
$$\mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}'', \mathsf{t}) \downharpoonleft ar) = \mathcal{S}_a''.$$
      From $\mathsf{step\text{-}CW}_\varphi(\Pi, (\Gamma, \bowtie), \mathcal{V}, +, -, \succ\!\!\circ)$, we know
$$\varphi(\mathcal{S}') = \mathcal{S}_a''.$$
      Thus
$$\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar).$$
   c. $\mathsf{tid}(e) \neq \mathsf{t}$. Thus $\mathcal{E}'|_{\mathsf{t}} = \mathcal{E}''|_{\mathsf{t}}$ and $\mathsf{visible}(\mathcal{E}', \mathsf{t}) = \mathsf{visible}(\mathcal{E}'', \mathsf{t})$. Thus $\varphi(\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})) = \mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar)$.

Thus we are done. □

Lemma 93. If

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\mathcal{S} \in dom(\psi)$, $\mathsf{causalDelivery}(\mathcal{E})$,

2. $(\mathcal{E}'{+}{+}[e]) \leqslant \mathcal{E}$, $\mathcal{S} = \mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})$, $e' \in \mathsf{visible}(\mathcal{E}', \mathsf{t})$,
3. $\mathsf{eff}(e) = \delta$, $\mathsf{eff}(e') = \delta'$, $\delta' \in \mathbf{+}$, $\delta' \in \mathcal{V}(\mathcal{S})$, $\neg(\delta' \succ\!\!\!\!\!\!\lhd \delta)$,
4. $\mathsf{conflictWL}(\mathbf{+}, \mathbf{-}, (\Pi, \Gamma, \bowtie))$, $\mathsf{genNotLose}(\mathbf{+}, \mathbf{-}, \mathcal{V}, \succ\!\!\!\!\!\!\lhd, (\Pi, \Gamma, \bowtie))$, $\mathsf{notVCancelee}(\succ\!\!\!\!\!\!\lhd, \mathcal{V})$, $\mathsf{canceleeNotV}(\succ\!\!\!\!\!\!\lhd, \mathcal{V})$, $\mathsf{winnerSee}(\mathbf{+}, \mathcal{V})$, $\mathsf{wfCGen}_\Pi(\succ\!\!\!\!\!\!\lhd, \mathcal{V})$, $\mathsf{wfV}_\psi(\mathcal{V})$, $\mathsf{uniqView}_\Pi(\mathcal{V})$,
5. $\rhd \subseteq \bowtie$,

then $\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e', e)$.

Proof. By contradiction. Suppose there exists e'' such that $(\Gamma \models e' \rhd e'')$, $(e' \xmapsto{\mathsf{vis}}_{\mathcal{E}} e'')$ and $(e'' \prec^{\mathsf{t}}_{\mathcal{E}} e \vee e'' = e)$.

Let $\mathsf{t}' = \mathsf{tid}(e'')$ and $\delta'' = \mathsf{eff}(e'')$. Thus $e' \xmapsto[\mathsf{t}']{\mathsf{vis}}_{\mathcal{E}} e''$ and there exists $\mathcal{S}''$ such that $\mathsf{genAt}_\Pi(\mathcal{S}'', \delta'')$. Since $\rhd \subseteq \bowtie$, we know $\delta' \bowtie_{\Pi,\Gamma} \delta''$. From $\mathsf{conflictWL}(\mathbf{+}, \mathbf{-}, (\Pi, \Gamma, \bowtie))$, since $\delta' \in \mathbf{+}$, we know

$$\delta'' \in \mathbf{-}.$$

From $\mathsf{genNotLose}(\mathbf{+}, \mathbf{-}, \mathcal{V}, \succ\!\!\!\!\!\!\lhd, (\Pi, \Gamma, \bowtie))$, we know

$$\neg\mathsf{loseAt}_\Pi(\delta'', \mathcal{S}'', \mathcal{V}, \mathbf{+}, \mathbf{-}, \succ\!\!\!\!\!\!\lhd, (\Gamma, \bowtie)).$$

Thus

$$\delta' \notin \mathcal{V}(\mathcal{S}'') \text{ or } \delta' \succ\!\!\!\!\!\!\lhd \delta''.$$

For the case $\delta' \notin \mathcal{V}(\mathcal{S}'')$, from $\mathsf{winnerSee}(\mathbf{+}, \mathcal{V})$ and $\mathsf{notVCancelee}(\succ\!\!\!\!\!\!\lhd, \mathcal{V})$, we know there exists e''' such that

$$e' \succ\!\!\!\!\!\!\lhd e''' \text{ and } e''' \prec^{\mathsf{t}'}_{\mathcal{E}} e''.$$

Thus, for both cases, we know there exists e_0 such that

$$e' \succ\!\!\!\!\!\!\lhd e_0 \quad \text{and} \quad e_0 \xmapsto[\mathsf{t}']{\mathsf{vis}}_{\mathcal{E}} e'' \vee e_0 = e''.$$

Since $\mathsf{wfCGen}_\Pi(\succ\!\!\!\!\!\!\lhd, \mathcal{V})$, $\mathsf{wfV}_\psi(\mathcal{V})$ and $\mathsf{uniqView}_\Pi(\mathcal{V})$, from Lemma 99, we know

$$e' \xmapsto{\mathsf{vis}}_{\mathcal{E}} e_0.$$

Since $\mathsf{causalDelivery}(\mathcal{E})$, we know

$$e' \prec^{\mathsf{t}}_{\mathcal{E}} e_0 \quad \text{and} \quad e_0 \prec^{\mathsf{t}}_{\mathcal{E}} e'' \vee e_0 = e''.$$

Since $(e'' \prec^{\mathsf{t}}_{\mathcal{E}} e \vee e'' = e)$, we know

$$e_0 \prec^{\mathsf{t}}_{\mathcal{E}} e \vee e_0 = e.$$

Since $\neg(\delta' \succ\!\!\!\!\!\!\lhd \delta)$, we know the case $e_0 = e$ is impossible. Thus

$$e_0 \prec^{\mathsf{t}}_{\mathcal{E}} e.$$

Since $e' \succ\!\!\!\!\!\!\lhd e_0$, from $\mathsf{winnerSee}(\mathbf{+}, \mathcal{V})$, $\mathsf{canceleeNotV}(\succ\!\!\!\!\!\!\lhd, \mathcal{V})$ and $\mathsf{uniqView}_\Pi(\mathcal{V})$, we know

$$\delta' \notin \mathcal{V}(\mathcal{S}).$$

This contradicts with $\delta' \in \mathcal{V}(\mathcal{S})$. So we are done. □

Lemma 94. If

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\mathcal{E}' \leqslant \mathcal{E}$, $\mathcal{E}_1 = (\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar)$, $\mathcal{E}_2 = (\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonleft ar')$,
2. $\mathsf{nonComm}(\Gamma, \bowtie)$, $\mathsf{cancel}(\rhd)$, $\mathsf{conflictWL}(\mathbf{+}, \mathbf{-}, (\Pi, \Gamma, \bowtie))$,
3. $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar)$, $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar')$, $\mathsf{vpa}(\mathsf{t}, \mathcal{E}', \succ\!\!\!\!\!\!\lhd, \mathbf{+}, \mathbf{-}, (\Gamma, \bowtie, \rhd)) \subseteq ar \cap ar'$,
4. $\mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathcal{E}_1) = \mathcal{S}'_a$,

then $\mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathcal{E}_2) = \mathcal{S}'_a$.

Proof. Below we first prove $\mathsf{ccCoh}(\mathcal{E}_1, \mathcal{E}_2, (\Gamma, \bowtie, \rhd))$ which is defined in Fig. 45. For any e_0 and e_1 such that $e_0 \prec_{\mathcal{E}_1} e_1$, $e_1 \prec_{\mathcal{E}_2} e_0$ and $\Gamma \models e_0 \bowtie e_1$, we want to prove $\exists i \in \{0, 1\}.\ \exists e.\ (\Gamma \models e_i \rhd e) \wedge (e_i \prec_{\mathcal{E}_1} e) \wedge (e_i \prec_{\mathcal{E}_2} e)$. Since $e_0 \prec_{\mathcal{E}_1} e_1$ and $e_1 \prec_{\mathcal{E}_2} e_0$, we know

$$\{e_0, e_1\} \subseteq \mathsf{visible}(\mathcal{E}', \mathsf{t}),\ e_0\ ar\, e_1 \text{ and } e_1\ ar'\ e_0.$$

Since $\mathsf{vpa}(\mathsf{t}, \mathcal{E}', \succ\!\!\!\!\circ, +, -, (\Gamma, \bowtie, \triangleright)) \subseteq ar \cap ar'$, we know

$$\mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E}',\Gamma,\bowtie,\triangleright} \subseteq ar \cap ar'.$$

Thus

$$(e_0, e_1) \notin \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E}',\Gamma,\bowtie,\triangleright} \text{ and } (e_1, e_0) \notin \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E}',\Gamma,\bowtie,\triangleright}.$$

Since $\Gamma \models e_0 \bowtie e_1$ and $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, by $\mathsf{conflictWL}(+, -, (\Pi, \Gamma, \bowtie))$, we know

$$e_0 \in + \wedge e_1 \in - \text{ or } e_0 \in - \wedge e_1 \in +$$

Thus we know there exists $i \in \{0, 1\}$ such that

$$\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E}',\Gamma,\bowtie}(e_i, e_{1-i}).$$

Thus there exists e such that

$$(\Gamma \models e_i \triangleright e) \wedge (e_i \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}'} e) \wedge (e \prec^{\mathsf{t}}_{\mathcal{E}'} e_{1-i} \vee e = e_{1-i})$$

Since $\mathsf{vpa}(\mathsf{t}, \mathcal{E}', \succ\!\!\!\!\circ, +, -, (\Gamma, \bowtie, \triangleright)) \subseteq ar \cap ar'$, we know

$$(\overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}'} \cap \triangleright_\Gamma) \subseteq ar \cap ar'.$$

Thus

$$(e_i, e) \in ar \cap ar'.$$

Since $(e \prec^{\mathsf{t}}_{\mathcal{E}'} e_{1-i} \vee e = e_{1-i})$ and $e_{1-i} \in \mathsf{visible}(\mathcal{E}', \mathsf{t})$, we know

$$e \in \mathsf{visible}(\mathcal{E}', \mathsf{t}).$$

Since $\mathcal{E}_1 = (\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar)$ and $\mathcal{E}_2 = (\mathsf{visible}(\mathcal{E}', \mathsf{t}) \downharpoonright ar')$, we know

$$e_i \prec_{\mathcal{E}_1} e \text{ and } e_i \prec_{\mathcal{E}_2} e.$$

Thus we know $\mathsf{ccCoh}(\mathcal{E}_1, \mathcal{E}_2, (\Gamma, \bowtie, \triangleright))$.

Finally by Lemma 95, we know $\mathsf{aexecST}(\Gamma, \mathcal{S}_a, \mathcal{E}_2) = \mathcal{S}'$. Thus we are done. □

Lemma 95. If

1. $\lfloor \mathcal{E}_1 \rfloor = \lfloor \mathcal{E}_2 \rfloor$, $\mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_1) = \mathcal{S}'$,
2. $\mathsf{nonComm}(\Gamma, \bowtie)$, $\mathsf{cancel}(\triangleright)$,
3. $\mathsf{ccCoh}(\mathcal{E}_1, \mathcal{E}_2, (\Gamma, \bowtie, \triangleright))$,

then $\mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_2) = \mathcal{S}'$.

Proof. Suppose the length of $\mathcal{E}_1$ is n. By induction over n.

- $n = 0$. Trivial.
- $n = m + 1$. Suppose $\mathcal{E}_1 = e_1 :: \mathcal{E}'_1$ and $\mathcal{E}_2 = e'_1 :: \mathcal{E}'_2$.
  - $e_1 = e'_1$. Let $\mathcal{S}'' = \mathsf{aexecST}(\Gamma, \mathcal{S}, [e_1])$. Then we know
  $$\lfloor \mathcal{E}'_1 \rfloor = \lfloor \mathcal{E}'_2 \rfloor,\ \mathsf{aexecST}(\Gamma, \mathcal{S}'', \mathcal{E}'_1) = \mathcal{S}' \text{ and } \mathsf{ccCoh}(\mathcal{E}'_1, \mathcal{E}'_2, (\Gamma, \bowtie, \triangleright)).$$
  Then, by the induction hypothesis, we know
  $$\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}'', \mathcal{E}'_2).$$
  Thus $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_2)$.
  - $e_1 \neq e'_1$. Suppose $\mathcal{E}_1 = e_1 :: e_2 :: \ldots :: e_n$ and $\mathcal{E}_2 = e'_1 :: e'_2 :: \ldots :: e'_n$.
  Since $\lfloor \mathcal{E}_1 \rfloor = \lfloor \mathcal{E}_2 \rfloor$, we know there exists $i > 1$ such that $e_1 = e'_i$.
  Let $\mathcal{E}_3 = e'_i :: \mathcal{E}'_3$ and $\mathcal{E}'_3 = e'_1 :: \ldots :: e'_{i-1} :: e'_{i+1} :: \ldots :: e'_n$.
  Below we first prove $\mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_3) = \mathcal{S}'$.
  Since $\lfloor \mathcal{E}_1 \rfloor = \lfloor \mathcal{E}_2 \rfloor$ and $\mathsf{ccCoh}(\mathcal{E}_1, \mathcal{E}_2, (\Gamma, \bowtie, \triangleright))$, we know
  $$\lfloor \mathcal{E}'_1 \rfloor = \lfloor \mathcal{E}'_3 \rfloor \text{ and } \mathsf{ccCoh}(\mathcal{E}'_1, \mathcal{E}'_3, (\Gamma, \bowtie, \triangleright)).$$
  Let $\mathcal{S}'' = \mathsf{aexecST}(\Gamma, \mathcal{S}, [e_1])$. Thus $\mathsf{aexecST}(\Gamma, \mathcal{S}'', \mathcal{E}'_1) = \mathcal{S}'$. Then, by the induction hypothesis, we know

$$\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}'', \mathcal{E}_3').$$

Thus $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_3)$.
Next, we prove $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_2)$.
Since $\mathsf{ccCoh}(\mathcal{E}_1, \mathcal{E}_2, (\Gamma, \bowtie, \rhd))$, we know

$$\mathsf{ccCoh}(\mathcal{E}_3, \mathcal{E}_2, (\Gamma, \bowtie, \rhd)).$$

By Lemma 96, we know $\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_2)$.

Thus we are done. □

Lemma 96. If

1. $\mathcal{E}_1 = [e_1]{+\!+}\mathcal{E}_1'{+\!+}\mathcal{E}_1''$, $\mathcal{E}_2 = \mathcal{E}_1'{+\!+}[e_1]{+\!+}\mathcal{E}_1''$, $\mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_1) = \mathcal{S}'$,
2. $\mathsf{nonComm}(\Gamma, \bowtie)$, $\mathsf{cancel}(\rhd)$,
3. $\mathsf{ccCoh}(\mathcal{E}_1, \mathcal{E}_2, (\Gamma, \bowtie, \rhd))$,

then $\mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_2) = \mathcal{S}'$.

Proof. Suppose the length of $\mathcal{E}_1'$ is n. By induction over n.

- $n = 0$. Trivial.
- $n = m + 1$. Suppose $\mathcal{E}_1' = \mathcal{E}_2'{+\!+}[e_2]$. Thus

$$\mathcal{E}_1 = [e_1]{+\!+}\mathcal{E}_2'{+\!+}[e_2]{+\!+}\mathcal{E}_1'' \text{ and } \mathcal{E}_2 = \mathcal{E}_2'{+\!+}[e_2]{+\!+}[e_1]{+\!+}\mathcal{E}_1''.$$

  Let $\mathcal{E}_3 = \mathcal{E}_2'{+\!+}[e_1]{+\!+}[e_2]{+\!+}\mathcal{E}_1''$.
  Below we first prove $\mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_3) = \mathcal{S}'$.
  Since $\mathsf{ccCoh}(\mathcal{E}_1, \mathcal{E}_2, (\Gamma, \bowtie, \rhd))$, we know

$$\mathsf{ccCoh}(\mathcal{E}_1, \mathcal{E}_3, (\Gamma, \bowtie, \rhd)).$$

  Then, by the induction hypothesis, we know

$$\mathcal{S}' = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_3).$$

  Next, we prove $\mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_3) = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_2)$.
  Let $\mathcal{S}_2 = \mathsf{aexecST}(\Gamma, \mathcal{S}, \mathcal{E}_2')$. So we only need to prove $\mathsf{aexecST}(\Gamma, \mathcal{S}_2, [e_1]{+\!+}[e_2]{+\!+}\mathcal{E}_1'') = \mathsf{aexecST}(\Gamma, \mathcal{S}_2, [e_2]{+\!+}[e_1]{+\!+}\mathcal{E}_1'')$.
  Since $e_1 <_{\mathcal{E}_1} e_2$ and $e_2 <_{\mathcal{E}_2} e_1$, by $\mathsf{ccCoh}(\mathcal{E}_1, \mathcal{E}_2, (\Gamma, \bowtie, \rhd))$, we know

$$\begin{aligned}&\neg(\Gamma \models e_1 \bowtie e_2)\\ &\vee \exists i \in \{1, 2\}.\ \exists e.\ (\Gamma \models e_1 \rhd e) \wedge (e_i <_{\mathcal{E}_1} e) \wedge (e_i <_{\mathcal{E}_2} e)\end{aligned}$$

  - $\neg(\Gamma \models e_1 \bowtie e_2)$.
    Since $\mathsf{nonComm}(\Gamma, \bowtie)$, we know

$$\mathsf{aexecST}(\Gamma, \mathcal{S}_2, [e_2]{+\!+}[e_1]) = \mathsf{aexecST}(\Gamma, \mathcal{S}_2, [e_1]{+\!+}[e_2]).$$

    Thus $\mathsf{aexecST}(\Gamma, \mathcal{S}_2, [e_1]{+\!+}[e_2]{+\!+}\mathcal{E}_1'') = \mathsf{aexecST}(\Gamma, \mathcal{S}_2, [e_2]{+\!+}[e_1]{+\!+}\mathcal{E}_1'')$.
  - $\exists i \in \{1, 2\}.\ \exists e.\ (\Gamma \models e_1 \rhd e) \wedge (e_i <_{\mathcal{E}_1} e) \wedge (e_i <_{\mathcal{E}_2} e)$.
    Thus we know $e \in \mathcal{E}_1''$.
    Since $\mathsf{nonComm}(\Gamma, \bowtie)$ and $\mathsf{cancel}(\rhd)$, we know

$$\begin{aligned}&\mathsf{aexecST}(\Gamma, \mathcal{S}_2, [e_1]{+\!+}[e_2]{+\!+}\mathcal{E}_1'')\\ &= \mathsf{aexecST}(\Gamma, \mathcal{S}_2, [e_2]{+\!+}\mathcal{E}_1'')\\ &= \mathsf{aexecST}(\Gamma, \mathcal{S}_2, [e_2]{+\!+}[e_1]{+\!+}\mathcal{E}_1'')\end{aligned}$$

Thus we are done. □

Lemma 97. If

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, $\mathcal{S} \in dom(\psi)$, $\mathsf{eventualDelivery}(\mathcal{E})$,
2. $\mathcal{E} = \mathcal{E}'{+\!+}[e]$, $\mathcal{S} = \mathsf{exec\_st}(\mathcal{S}, \mathcal{E}'|_{\mathsf{t}})$, $e = (mid, \mathsf{t}, (f, n), \delta)$,
3. $\neg\mathsf{loseAt}_{\Pi}(\delta, \mathcal{S}, \mathcal{V}, +, -, \gg\!\!\circ, (\Gamma, \bowtie))$,
4. $\mathsf{partialOrder}(\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \gg\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)))$, $\mathsf{wfCGen}_{\Pi}(\gg\!\!\circ, \mathcal{V})$, $\mathsf{wfV}_{\psi}(\mathcal{V})$, $\mathsf{uniqView}_{\Pi}(\mathcal{V})$, $\mathsf{winnerSee}(+, \mathcal{V})$, $\mathsf{notVCancelee}(\gg\!\!\circ, \mathcal{V})$, $\mathsf{cancelAbsCancel}(\gg\!\!\circ, (\Pi, \Gamma, \rhd))$,
5. $rel = \{(e', e) \mid e' \in \mathsf{visible}(\mathcal{E}', \mathsf{t})\}$, $rel' = (\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \gg\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \cup rel)^+$,

then $\mathsf{partialOrder}(rel')$.

Proof. Let $rel'' = (\underset{\mathsf{t}}{\xmapsto{\mathsf{vis}}}_{\mathcal{E}} \cup (\underset{\mathsf{t}}{\xmapsto{\mathsf{vis}}}_{\mathcal{E}} \cap \rhd_{\Gamma}) \cup \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\rhd} \cup rel)$. We only need to prove $\neg\mathsf{cyclic}(rel'')$.

By contradiction. Suppose there exist $n, e_1, \ldots, e_n$ such that $\forall i \in [1..n-1]$. $(e_i, e_{i+1}) \in rel''$ and $(e_n, e_1) \in rel''$. Without loss of generality, we can suppose n is the length of the smallest cycle. We analyze the following two cases:

- $n = 1$. We know it is impossible from $\mathsf{partialOrder}(\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \rtimes, +, -, (\Gamma, \bowtie, \rhd)))$ and the definition of rel''.
- $n > 1$.
  Since $\mathsf{eventualDelivery}(\mathcal{E})$, we know
  $$\{e_1, \ldots, e_n\} \subseteq \mathsf{visible}(\mathcal{E}, \mathsf{t}).$$
  Without loss of generality, we can suppose e_n is the last event among $e_1, \ldots, e_n$ that t applies, that is, $\forall i \in [1..n-1].\ e_i \prec^{\mathsf{t}}_{\mathcal{E}} e_n$.
  By the definition of rel'', we know
  $$(e_n, e_1) \in \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\rhd}.$$
  Since $\mathsf{partialOrder}(\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \rtimes, +, -, (\Gamma, \bowtie, \rhd)))$, we know
  $$\neg\mathsf{cyclic}(\underset{\mathsf{t}}{\overset{\mathsf{vis}}{\longmapsto}}_{\mathcal{E}} \cup (\overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} \cap \rhd_{\Gamma}) \cup \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\rhd}).$$
  Thus $\exists i.\ (e_i, e_{i+1}) \in rel$. Since $\mathcal{E} = \mathcal{E}'{+}{+}[e]$ and $\forall i \in [1..n-1].\ e_i \prec^{\mathsf{t}}_{\mathcal{E}} e_n$, we know
  $$(e_{n-1}, e_n) \in rel.$$
  Thus
  $$e_n = e \text{ and } (e_1, e) \in rel.$$
  From $(e, e_1) \in \mathsf{Win}^{+,-}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie,\rhd}$, we know
  $$\Gamma \models e \bowtie e_1,\ e \in -,\ e_1 \in +,$$
  $$\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e, e_1),\ \neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\rhd}(e_1, e).$$
  Since $\neg\mathsf{loseAt}_{\Pi}(\delta, \mathcal{S}, \mathcal{V}, +, -, \rtimes, (\Gamma, \bowtie))$, we know
  $$\forall \delta'.\ \delta' \in + \wedge (\delta' \bowtie_{\Pi,\Gamma} \delta) \implies (\delta' \rtimes \delta) \vee (\delta' \notin \mathcal{V}(\mathcal{S})).$$
  Let $\mathsf{eff}(e_1) = \delta_1$. Thus $(\delta_1 \rtimes \delta) \vee (\delta_1 \notin \mathcal{V}(\mathcal{S}))$.
  - $\delta_1 \rtimes \delta$. Since $\mathsf{wfCGen}_{\Pi}(\rtimes, \mathcal{V})$, $\mathsf{wfV}_{\psi}(\mathcal{V})$ and $\mathsf{uniqView}_{\Pi}(\mathcal{V})$, from Lemma 99, we know
    $$e_1 \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e.$$
    Since $\mathsf{cancelAbsCancel}(\rtimes, (\Pi, \Gamma, \rhd))$ and $e_1 \rtimes e$, we know
    $$\Gamma \models e_1 \rhd e.$$
    This contradicts with $\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie}(e_1, e)$.
  - $\delta_1 \notin \mathcal{V}(\mathcal{S})$. From $\mathsf{winnerSee}(+, \mathcal{V})$ and $\mathsf{notVCancelee}(\rtimes, \mathcal{V})$, we know there exists e'' such that
    $$e_1 \rtimes e'' \text{ and } e'' \prec^{\mathsf{t}}_{\mathcal{E}} e.$$
    Since $\mathsf{wfCGen}_{\Pi}(\rtimes, \mathcal{V})$, $\mathsf{wfV}_{\psi}(\mathcal{V})$ and $\mathsf{uniqView}_{\Pi}(\mathcal{V})$, from Lemma 99, we know
    $$e_1 \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e''.$$
    Since $\mathsf{cancelAbsCancel}(\rtimes, (\Pi, \Gamma, \rhd))$ and $e_1 \rtimes e''$, we know
    $$\Gamma \models e_1 \rhd e''.$$
    This contradicts with $\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathsf{t},\mathcal{E},\Gamma,\bowtie}(e_1, e)$.

Thus we are done. □

Lemma 98 (Coherence). For any pa, pa', ar, ar', t, t' and $\mathcal{E}$, if

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$,
2. $\mathsf{conflictWL}(+, -, (\Pi, \Gamma, \bowtie))$, $\mathsf{abswonbyWL}(+, -, (\Pi, \Gamma, \blacktriangleleft))$,
3. $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar_{\mathsf{t}})$, $\mathsf{totalOrder}_{\mathsf{orig}(\mathcal{E})}(ar_{\mathsf{t}'})$,
4. $\mathsf{vpa}(\mathsf{t}, \mathcal{E}, \rtimes, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar_{\mathsf{t}}$, $\mathsf{vpa}(\mathsf{t}', \mathcal{E}, \rtimes, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar_{\mathsf{t}'}$,

then $\mathsf{RCoh}_{(\mathsf{t},\mathsf{t}')}((ar_{\mathsf{t}}, ar_{\mathsf{t}'}), \mathcal{E}, (\Gamma, \bowtie, \blacktriangleleft, \rhd))$.

Proof. For any $\mathcal{E}'$, $\mathcal{E}''$, e_0 and e_1, suppose $\mathcal{E}' \leqslant \mathcal{E}$, $\mathcal{E}'' \leqslant \mathcal{E}$, $e_0 \bowtie_{\Gamma} e_1$ and $\{e_0, e_1\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}', \mathsf{t}, (\Gamma, \rhd)) \cap \mathsf{nc\text{-}vis}(\mathcal{E}'', \mathsf{t}', (\Gamma, \rhd))$.
We want to prove:

(1) $(e_0, e_1) \in ar_{\mathsf{t}} \cap ar_{\mathsf{t}'} \vee (e_1, e_0) \in ar_{\mathsf{t}} \cap ar_{\mathsf{t}'}$;
(2) $\mathsf{Concurrent}_{\mathcal{E}}(e_0, e_1) \wedge (e_0 \blacktriangleleft_{\Gamma} e_1) \implies (e_0, e_1) \in ar_{\mathsf{t}}$.

We consider three cases:

1. $e_0 \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e_1$. We know
   $$(e_0, e_1) \in \mathsf{vpa}(\mathsf{t}, \mathcal{E}, \rtimes, +, -, (\Gamma, \bowtie)) \text{ and } (e_0, e_1) \in \mathsf{vpa}(\mathsf{t}', \mathcal{E}, \rtimes, +, -, (\Gamma, \bowtie))$$

Since $\mathsf{vpa}(\mathrm{t}, \mathcal{E}, \bowtie\!\!\!\circ, +, -, (\Gamma, \bowtie)) \subseteq ar_{\mathrm{t}}$ and $\mathsf{vpa}(\mathrm{t}', \mathcal{E}, \bowtie\!\!\!\circ, +, -, (\Gamma, \bowtie)) \subseteq ar_{\mathrm{t}'}$, we know

$$(e_0, e_1) \in ar_{\mathrm{t}} \cap ar_{\mathrm{t}'}.$$

So (1) holds.

Since $e_0 \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e_1$, we know $\neg\mathsf{Concurrent}_{\mathcal{E}}(e_0, e_1)$. So (2) holds.

2. $e_1 \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e_0$. Similar to the first case.
3. $\neg(e_0 \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e_1)$ and $\neg(e_1 \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e_0)$. Since $\mathsf{conflictWL}(+, -, (\Pi, \Gamma, \bowtie))$, we know

$$(e_0 \in + \wedge e_1 \in -) \vee (e_1 \in + \wedge e_0 \in -)$$

Since $\{e_0, e_1\} \subseteq \mathsf{nc\text{-}vis}(\mathcal{E}', \mathrm{t}, (\Gamma, \rhd)) \cap \mathsf{nc\text{-}vis}(\mathcal{E}'', \mathrm{t}', (\Gamma, \rhd))$, we know

$$\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathrm{t},\mathcal{E},\Gamma,\rhd}(e_0, e_1),\ \neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathrm{t},\mathcal{E},\Gamma,\rhd}(e_1, e_0),$$
$$\neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathrm{t}',\mathcal{E},\Gamma,\rhd}(e_0, e_1),\ \neg\mathsf{canceled\text{-}bef\text{-}or\text{-}by}_{\mathrm{t}',\mathcal{E},\Gamma,\rhd}(e_1, e_0).$$

a. If $(e_0 \in - \wedge e_1 \in +)$, then

$$(e_0, e_1) \in \mathsf{Win}^{+,-}_{\mathrm{t},\mathcal{E},\Gamma,\bowtie,\rhd} \text{ and } (e_0, e_1) \in \mathsf{Win}^{+,-}_{\mathrm{t}',\mathcal{E},\Gamma,\bowtie,\rhd}.$$

We know

$$(e_0, e_1) \in \mathsf{vpa}(\mathrm{t}, \mathcal{E}, \bowtie\!\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \text{ and } (e_0, e_1) \in \mathsf{vpa}(\mathrm{t}', \mathcal{E}, \bowtie\!\!\!\circ, +, -, (\Gamma, \bowtie, \rhd))$$

Since $\mathsf{vpa}(\mathrm{t}, \mathcal{E}, \bowtie\!\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar_{\mathrm{t}}$ and $\mathsf{vpa}(\mathrm{t}', \mathcal{E}, \bowtie\!\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar_{\mathrm{t}'}$, we know

$$(e_0, e_1) \in ar_{\mathrm{t}} \cap ar_{\mathrm{t}'}.$$

b. If $(e_1 \in - \wedge e_0 \in +)$, then

$$(e_1, e_0) \in \mathsf{Win}^{+,-}_{\mathrm{t},\mathcal{E},\Gamma,\bowtie,\rhd} \text{ and } (e_1, e_0) \in \mathsf{Win}^{+,-}_{\mathrm{t}',\mathcal{E},\Gamma,\bowtie,\rhd}.$$

We know

$$(e_1, e_0) \in \mathsf{vpa}(\mathrm{t}, \mathcal{E}, \bowtie\!\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \text{ and } (e_1, e_0) \in \mathsf{vpa}(\mathrm{t}', \mathcal{E}, \bowtie\!\!\!\circ, +, -, (\Gamma, \bowtie, \rhd))$$

Since $\mathsf{vpa}(\mathrm{t}, \mathcal{E}, \bowtie\!\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar_{\mathrm{t}}$ and $\mathsf{vpa}(\mathrm{t}', \mathcal{E}, \bowtie\!\!\!\circ, +, -, (\Gamma, \bowtie, \rhd)) \subseteq ar_{\mathrm{t}'}$, we know

$$(e_1, e_0) \in ar_{\mathrm{t}} \cap ar_{\mathrm{t}'}.$$

So (1) holds.

If $\mathsf{Concurrent}_{\mathcal{E}}(e_0, e_1) \wedge (e_0 \blacktriangleleft_{\Gamma} e_1)$, from $\mathsf{abswonbyWL}(+, -, (\Pi, \Gamma, \blacktriangleleft))$, we know

$$e_0 \in - \wedge e_1 \in +.$$

Thus $(e_0, e_1) \in ar_{\mathrm{t}} \cap ar_{\mathrm{t}'}$. So (2) holds.

Thus we are done. ☐

Lemma 99 (Canceled-by implies vis-relation). For any $\mathcal{S}$, $\mathcal{E}$, e_1 and e_2, if

1. $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$ and $\mathcal{S} \in dom(\psi)$,
2. $e_1 \bowtie\!\!\!\circ\, e_2$ and $\{e_1, e_2\} \subseteq \mathsf{orig}(\mathcal{E})$,
3. $\mathsf{wfCGen}_{\Pi}(\bowtie\!\!\!\circ, \mathcal{V})$, $\mathsf{wfV}_{\psi}(\mathcal{V})$ and $\mathsf{uniqView}_{\Pi}(\mathcal{V})$,

then $e_1 \overset{\mathsf{vis}}{\longmapsto}_{\mathcal{E}} e_2$.

Proof. Since $\{e_1, e_2\} \subseteq \mathsf{orig}(\mathcal{E})$, we can suppose

$$e_1 = (mid_1, \mathrm{t}_1, (f_1, n_1, n_1', \delta_1)) \text{ and } e_2 = (mid_2, \mathrm{t}_2, (f_2, n_2, n_2', \delta_2)).$$

Since $\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S})$, by the operational semantics, we know there exist $\mathcal{S}_2$ and $\mathcal{E}_2$ such that

$$\mathsf{exec\_st}(\mathcal{S}, \mathcal{E}_2) = \mathcal{S}_2,\ \mathcal{E}_2{+\!+}[e_2] \leqslant (\mathcal{E}|_{\mathrm{t}_2}) \text{ and } \Pi(f_2, n_2)(\mathcal{S}_2) = (n_2', \delta_2).$$

Since $\mathsf{wfCGen}_{\Pi}(\bowtie\!\!\!\circ, \mathcal{V})$ and $\delta_1 \bowtie\!\!\!\circ\, \delta_2$, we know

$$\delta_1 \in \mathcal{V}(\mathcal{S}_2).$$

Since $\mathsf{wfV}_{\psi}(\mathcal{V})$, we know there exists e such that

$$e \in \mathcal{E}_2 \ \wedge\ \mathsf{eff}(e) = \delta_1.$$

From $\mathsf{uniqView}_{\Pi}(\mathcal{V})$, by Lemma 100, we know $\mathsf{uniqSeeT}_{\Pi,\psi}(\mathcal{V})$. Thus we know

$$\mathsf{msgid}(e) = \mathsf{msgid}(e_1).$$

By the operational semantics, we know

$$e_1 \overset{t_2}{\underset{\mathcal{E}}{\Longrightarrow}} e.$$

Thus we know $e_1 \xmapsto{\text{vis}}_{\mathcal{E}} e_2$. So we are done. □

Lemma 100. If $\text{uniqView}_{\Pi}(\mathcal{V})$, then $\text{uniqSeeT}_{\Pi,\psi}(\mathcal{V})$.

Proof. By contradiction. Suppose $\text{uniqSeeT}_{\Pi,\psi}(\mathcal{V})$ does not hold. So there exist $\mathcal{S}_0$, $\mathcal{E}$, e_1, e_2 and δ such that

$$\mathcal{E} \in \mathcal{T}(\Pi, \mathcal{S}_0),\ \ \mathcal{S}_0 \models \psi,\ \ e_1 \in \mathcal{E},\ \ e_2 \in \mathcal{E},\ \ \text{eff}(e_1) = \text{eff}(e_2) = \delta,\ \ \delta \in \mathcal{V},\ \ \text{msgid}(e_1) \neq \text{msgid}(e_2).$$

Then we know there exist $C_1, \ldots, C_n$, W and W' such that

$$((\textbf{let}\ \Pi\ \textbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, \mathcal{S}_0) \xmapsto{\text{load}} W) \wedge (W \xrightarrow{\mathcal{E}}{}^{*} W')$$

Also we know there exist f, n, f', n', $\mathcal{S}$, $\mathcal{S}'$, $\mathcal{E}_1$, $\mathcal{E}_1'$, W_1, W_1', e_1', e_2', t and t′ such that

$$\Pi(f, n)(\mathcal{S}) = (\_, \delta),\ \ \Pi(f', n')(\mathcal{S}') = (\_, \delta),$$
$$\mathcal{E}_1{+\!+}[e_1'] \leqslant \mathcal{E},\ \ e_1' \overset{\text{tid}(e_1)}{\underset{\mathcal{E}}{\Longrightarrow}} e_1,\ \ W \xrightarrow{\mathcal{E}_1}{}^{*} W_1,\ \ W_1.\sigma_o(\mathrm{t}) = (\Pi, \mathcal{S}, \_),$$
$$\mathcal{E}_1'{+\!+}[e_2'] \leqslant \mathcal{E},\ \ e_2' \overset{\text{tid}(e_2)}{\underset{\mathcal{E}}{\Longrightarrow}} e_2,\ \ W \xrightarrow{\mathcal{E}_1'}{}^{*} W_1',\ \ W_1'.\sigma_o(\mathrm{t}') = (\Pi, \mathcal{S}', \_).$$

From $\text{uniqView}_{\Pi}(\mathcal{V})$, since $\Pi(f, n)(\mathcal{S}) = (\_, \delta)$, $\Pi(f', n')(\mathcal{S}') = (\_, \delta)$ and $\delta \in \mathcal{V}$, we know

$$\mathcal{S} = \mathcal{S}'.$$

By the operational semantics, we know

$$\mathrm{t} = \mathrm{t}'.$$

Without loss of generality, we can assume that $\mathcal{E}_1{+\!+}[e_1'] \leqslant \mathcal{E}_1'$. Suppose $\mathcal{E}_1' = \mathcal{E}_1{+\!+}[e_1']{+\!+}\mathcal{E}_1''$. Then we know there exists W_1'' such that

$$W \xrightarrow{\mathcal{E}_1}{}^{*} W_1 \xrightarrow{e_1'} W_1'' \xrightarrow{\mathcal{E}_1''}{}^{*} W_1'.$$

Let $\mathcal{S}_1 = W_1''.\sigma_o(\mathrm{t})$. From $\text{uniqView}_{\Pi}(\mathcal{V})$, we know

$$\mathcal{S} \prec \mathcal{S}_1 \ \text{ and } \ \mathcal{S} \prec \mathcal{S}'.$$

Since $\prec$ is irreflexive, we know $\mathcal{S} \neq \mathcal{S}'$. So we get a contradiction. □