A Framework for Transactional Consistency Models with Atomic Visibility

Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman

IMDEA Software Institute, Madrid, Spain

Abstract

Modern distributed systems often rely on databases that achieve scalability by providing only weak guarantees about the consistency of distributed transaction processing. The semantics of programs interacting with such a database depends on its consistency model, defining these guarantees. Unfortunately, consistency models are usually stated informally or using disparate formalisms, often tied to the database internals. To deal with this problem, we propose a framework for specifying a variety of consistency models for transactions uniformly and declaratively. Our specifications are given in the style of weak memory models, using structures of events and relations on them. The specifications are particularly concise because they exploit the property of atomic visibility guaranteed by many consistency models: either all or none of the updates by a transaction can be visible to another one. This allows the specifications to abstract from individual events inside transactions. We illustrate the use of our framework by specifying several existing consistency models. To validate our specifications, we prove that they are equivalent to alternative operational ones, given as algorithms closer to actual implementations. Our work provides a rigorous foundation for developing the metatheory of the novel form of concurrency arising in weakly consistent large-scale databases.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Replication, Consistency models, Transactions

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

To achieve availability and scalability, modern distributed systems often rely on *replicated databases*, which maintain multiple *replicas* of shared data. The database clients can execute transactions on the data at any of the replicas, which communicate changes to each other using message passing. For example, large-scale Internet services use data replicas in geographically distinct locations, and applications for mobile devices keep replicas locally as well as in the cloud to support offline use. Ideally, we want the concurrent and distributed processing in a replicated database to be transparent, as formalised by the classical notion of serialisability [20]: the database behaves as if it executed transactions serially on a non-replicated copy of the data. However, achieving this ideal requires extensive coordination between replicas, which slows down the database and even makes it unavailable if network connections between replicas fail [1]. For this reason, nowadays replicated databases often provide weaker consistency guarantees, which allow non-serialisable behaviours, called *anomalies*. For example, consider the following program issuing transactions concurrently:

$$\begin{array}{c}\texttt{txn}\,\{\,x.\texttt{write}(\mathit{post});\ y.\texttt{write}(\mathit{empty})\,\}\ \parallel\ \texttt{txn}\,\{\,u = x.\texttt{read}();\ y.\texttt{write}(\mathit{comment})\,\}\\ \parallel\ \texttt{txn}\,\{\,v = x.\texttt{read}();\ w = y.\texttt{read}()\,\}\end{array} \tag{1}$$

where x, y are database objects and u, v, w local variables. In some databases the above program can execute so that the last transaction observes the *comment*, but not the *post*:

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–42

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$u = post$, $v = empty$, $w = comment$. This result cannot be obtained by executing the three transactions in any sequence and, hence, is not serialisable. In an implementation it may arise if the first two transactions are executed at a replica r, and the third one at another replica r', and the messages carrying the updates by the first two transactions arrive to r' out of order.

The semantics of programs interacting with a replicated database thus depends on its *consistency model*, restricting the anomalies it can exhibit and, as a consequence, the possible performance optimisations in its implementation. Recent years have seen a plethora of proposals of consistency models for replicated databases [4, 6, 12, 13, 19, 21, 24] that make different trade-offs between consistency and performance. Unfortunately, these subtle models are usually specified informally or using disparate formalisms, often tied to database internals. Whereas some progress in formalising the consistency models has been recently made for replicated databases without transactions [11, 12], the situation is worse for databases providing these. The lack of a uniform specification formalism represents a major hurdle in developing the metatheory of the novel form of concurrency arising in weakly consistent replicated databases and, in particular, methods for formal reasoning about application programs using them.

To deal with this problem, we propose a framework to uniformly specify a variety of modern transactional consistency models. We apply the framework to specify six existing consistency models for replicated databases; the results are summarised in Figure 1, page 6. Specifications in our framework are declarative, i.e., they do not refer to the database internals and thus allow reasoning about the database behaviour at a higher abstraction level. To achieve this, we take an *axiomatic* approach similar to the one used to define the semantics of weak memory models of multiprocessors and shared-memory programming languages [3]: our specifications model database computations by *abstract executions*, which are structures of events and relations on them, reminiscent of event structures [26]. For example, Figure 3(a), page 6 gives an execution that could arise from the program (1). The boxes named T_1, T_2 and T_3 depict transactions, which are sequences of events ordered by the *program order* po, reflecting the program syntax. The *visibility* edges $T_1 \xrightarrow{\mathsf{VIS}} T_2$ and $T_2 \xrightarrow{\mathsf{VIS}} T_3$ mean that the transaction T_2 (T_3) is aware of the updates made by T_1 (T_2). Consistency models are specified by *consistency axioms*, constraining abstract executions; e.g., a consistency axiom may require the visibility relation to be transitive and thereby disallow the execution in Figure 3(a).

The key observation we exploit in our framework is that modern consistency models for replicated databases usually guarantee *atomic visibility*: either all or none of the events in a transaction can be visible to another transaction; it is the flexibility in *when* a transaction becomes visible that leads to anomalies. Thanks to atomic visibility, in abstract executions we can use relations on whole transactions (such as VIS in Figure 3(a)), rather than on separate events inside them, thereby achieving particularly concise specifications. We further illustrate the benefits of this form of the specifications by exploiting it to obtain sufficient and necessary conditions for *observational refinement* [16] between transactions. This allows replacing a transaction in an abstract execution by another one without invalidating the consistency axioms of a given model. One can think of our conditions as characterising the optimisations that the database can soundly perform inside a transaction due to its atomic visibility.

To ensure that our declarative axiomatic specifications indeed faithfully describe the database behaviour, we prove that they are equivalent to alternative *operational* ones, given as algorithms closer to actual implementations (Theorem 6, §4). This correspondence also

highlights implementation features that motivate the form of the consistency axioms.

Our work systematises the knowledge about consistency models of replicated databases and provides insights into relationships between them (§3). The proposed specification framework also gives a basis to develop methods for reasoning about application programs using weakly consistent databases. Finally, our framework is an effective tool for exploring the space of consistency models, because their concise axiomatic specifications allow easily experimenting with alternative designs. In particular, our formalisation naturally suggests a new consistency model (§3).

2 Abstract Executions

We consider a database storing *objects* $\mathsf{Obj} = \{x, y, \ldots\}$, which for simplicity we assume to be integer-valued. Clients interact with the database by issuing `read` and `write` operations on the objects, grouped into *transactions*. We let $\mathsf{Op} = \{\mathtt{read}(x, n), \mathtt{write}(x, n) \mid x \in \mathsf{Obj}, n \in \mathbb{Z}\}$ describe the possible operation invocations: reading a value n from an object x or writing n to x.

To specify a consistency model, we need to define the set of all client-database interactions that it allows. We start by introducing structures for recording such interactions in a single database computation, called *histories*. In these, we denote operation invocations using *history events* of the form (ι, o), where ι is an identifier from a countably infinite set $\mathsf{EventId}$ and $o \in \mathsf{Op}$. We use e, f, g to range over history events. We let $\mathsf{WEvent}_x = \{(\iota, \mathtt{write}(x, n)) \mid \iota \in \mathsf{EventId}, n \in \mathbb{Z}\}$, define the set REvent_x of read events similarly, and let $\mathsf{HEvent}_x = \mathsf{REvent}_x \cup \mathsf{WEvent}_x$. A relation is a *total order* if it is transitive, irreflexive, and relates every two distinct elements one way or another.

▶ **Definition 1.** A *transaction* $T, S, \ldots$ is a pair (E, po), where $E \subseteq \mathsf{HEvent}$ is a finite, non-empty set of events with distinct identifiers, and the *program order* po is a total order over E. A *history* $\mathcal{H}$ is a (finite or infinite) set of transactions with disjoint sets of event identifiers.

All transactions in a history are assumed to be committed: to simplify presentation, our specifications do not constrain values read inside aborted or ongoing transactions.

To define the set of histories allowed by a given consistency model, we introduce *abstract executions*, which enrich histories with certain relations on transactions, declaratively describing how the database processes them. Consistency models are then defined by constraining these relations. We call a relation *prefix-finite*, if every element has finitely many predecessors in the transitive closure of the relation.

▶ **Definition 2.** An *abstract execution* is a triple $\mathcal{A} = (\mathcal{H}, \mathsf{VIS}, \mathsf{AR})$ where:

- *visibility* $\mathsf{VIS} \subseteq \mathcal{H} \times \mathcal{H}$ is a prefix-finite, acyclic relation; and
- *arbitration* $\mathsf{AR} \subseteq \mathcal{H} \times \mathcal{H}$ is a prefix-finite, total order such that $\mathsf{AR} \supseteq \mathsf{VIS}$.

We often write $T \xrightarrow{\mathsf{VIS}} S$ in lieu of $(T, S) \in \mathsf{VIS}$, and similarly for AR. Figure 3(a) gives an execution corresponding to the anomaly explained in §1. Informally, $T \xrightarrow{\mathsf{VIS}} S$ means that S is aware of T, and thus T's effects can influence the results of operations in S. In implementation terms, this may be the case if the updates performed by T have been delivered to the replica performing S; the prefix-finiteness requirement ensures that there may only be finitely many such transactions T. We call transactions unrelated by visibility *concurrent*. The relationship $T \xrightarrow{\mathsf{AR}} S$ means that the versions of objects written by S supersede those written by T; e.g., *comment* supersedes *empty* in Figure 3(a). The

constraint $\mathsf{AR} \supseteq \mathsf{VIS}$ ensures that writes by a transaction T supersede those that T is aware of; thus AR essentially orders writes only by concurrent transactions. In an implementation, arbitration can be established by assigning timestamps to transactions.

A consistency model specification is a set of *consistency axioms* Φ constraining executions. The model allows those histories for which there exists an execution that satisfies the axioms:

$$\mathsf{Hist}_\Phi = \{\mathcal{H} \mid \exists \mathsf{VIS}, \mathsf{AR}.\, (\mathcal{H}, \mathsf{VIS}, \mathsf{AR}) \models \Phi\}. \tag{2}$$

Our consistency axioms do not restrict the operations done by the database clients. We can obtain the set of histories produced by a particular program interacting with the database, such as (1), by restricting the above set, as is standard in weak memory model definitions [7].

3 Specifying Transactional Consistency Models

We now apply the concepts introduced to define several existing consistency models; see Figures 1-3. For a total order R and a set A, we let $\max_{\mathsf{R}}(A)$ be the element $u \in A$ such that $\forall v \in A.\, v = u \vee (v, u) \in \mathsf{R}$; if $A = \emptyset$, then $\max_{\mathsf{R}}(A)$ is undefined. In the following, the use of $\max_{\mathsf{R}}(A)$ in an expression implicitly assumes that it is defined. For a relation $\mathsf{R} \subseteq A \times A$ and an element $u \in A$, we let $\mathsf{R}^{-1}(u) = \{v \mid (v, u) \in \mathsf{R}\}$. We denote the sequential composition of relations R_1 and R_2 by $\mathsf{R}_1; \mathsf{R}_2$. We write _ for a value that is irrelevant and implicitly existentially quantified.

Baseline consistency model: Read Atomic. The weakest consistency model we consider, Read Atomic (Figure 1), is defined by the axioms Int and Ext (Figure 2), which determine the outcomes of reads in terms of the visibility and arbitration relations. Consistency models stronger than Read Atomic are defined by adding axioms that constrain these relations. The *internal consistency axiom* Int ensures that, within a transaction, the database provides sequential semantics: a read from an object returns the same value as the last write to or read from this object in the transaction. In particular, Int guarantees that, if a transaction writes to an object and then reads the object, then it will observe its last write. The axiom also disallows so-called *unrepeatable reads*: if a transaction reads an object twice without writing to it in-between, it will read the same value in both cases.

If a read is not preceded in the program order by an operation on the same object, then its value is determined in terms of writes by other transactions using the *external consistency axiom* Ext. The formulation of Ext relies on the following notation, defining certain attributes of a transaction $T = (E, \mathsf{po})$. We let $T \vdash \mathsf{Write}\ x : n$ if T writes to x and the last value written is n: $\max_{\mathsf{po}}(E \cap \mathsf{WEvent}_x) = (\_, \mathtt{write}(x, n))$. We let $T \vdash \mathsf{Read}\ x : n$ if T makes an *external* read from x, i.e., one before writing to x, and n is the value returned by the first such read: $\min_{\mathsf{po}}(E \cap \mathsf{HEvent}_x) = (\_, \mathtt{read}(x, n))$. In this case, Int ensures that n will be the result of all external reads from x in T. According to Ext, the value returned by an external read in T is determined by the transactions VIS-preceding T that write to x: if there are no such transactions, then T reads the initial value 0; otherwise it reads the final value written by the last such transaction in AR. (In examples we sometimes use initial values other than 0.) For example, the execution in Figure 3(a) satisfies Ext; if it included the edge $T_1 \xrightarrow{\mathsf{VIS}} T_3$, then Ext would force the read from x in T_3 to return *post*. The axiom Ext implies the absence of so-called *dirty reads*: a committed transaction cannot read a value written by an aborted or an ongoing transaction (which are not present in abstract executions), and a transaction cannot read a value that was overwritten by the

transaction that wrote it (ensured by the definition of $T \vdash \mathsf{Write}\ x : n$). Finally, Ext guarantees *atomic visibility* of a transaction: either all or none of its writes can be visible to another transaction. For example, Ext disallows the execution in Figure 3(b) and, in fact, any execution with the same history. This illustrates a *fractured reads* anomaly: T_1 makes *Alice* and *Bob* friends, but T_2 observes only one direction of the friendship relationship. Thus, the consistency guarantees provided by Read Atomic are useful because they allow maintaining integrity invariants, such as the symmetry of the friendship relation.

Stronger consistency models. Even though Read Atomic ensures that all writes by a transaction become visible together, it does not constrain *when* this happens. This leads to a number of anomalies, including the causality violation shown in Figure 3(a). We now consider stronger consistency models that provide additional guarantees about the visibility of transactions. We specify the first model of *causal consistency* by requiring VIS to be transitive (TransVis). This implies that transactions ordered by VIS (such as T_1 and T_2 in Figure 3(a)), are observed by others (such as T_3) in this order. Hence, the axiom TransVis disallows the anomaly in Figure 3(a).

Both Read Atomic and causal consistency can be implemented without requiring any coordination among replicas [6, 19]: a replica can decide to commit a transaction without consulting other replicas. This allows the database to stay available even during network failures. However, the above consistency models allow the *lost update* anomaly illustrated by the execution in Figure 3(c), which satisfies the axioms of causal consistency. This execution could arise from the code, also shown in the figure, that uses transactions T_1 and T_2 to make deposits into an account. The two transactions read the initial balance of the account and concurrently modify it, resulting in one deposit getting lost. The next consistency model we consider, parallel snapshot isolation, prohibits such anomalies in exchange for requiring replica coordination in its implementations [24]. We specify it by strengthening causal consistency with the axiom NoConflict, which does not allow transactions writing to the same object to be concurrent. This rules out any execution with the history in Figure 3(c): it forces T_1 and T_2 to be ordered by VIS, so that they cannot both read 0 from `acct`.

The axiom TransVis in causal consistency and parallel snapshot isolation guarantees that VIS-ordered transactions are observed by others in this order (cf. Figure 3(a)). However, the axiom allows two *concurrent* transactions to be observed in different orders, as illustrated by the *long fork* anomaly in Figure 3(d), allowed by both models. Concurrent transactions T_1 and T_2 write to x and y, respectively. A transaction T_3 observes the write to x, but not y, and a transaction T_4 observes the write to y, but not x. Thus, from the perspectives of T_3 and T_4, the writes of T_1 and T_2 happen in different orders.

The next pair of consistency models that we consider disallow this anomaly. We specify prefix consistency and snapshot isolation by strengthening causal consistency, respectively, parallel snapshot isolation, with the requirement that all transactions become visible throughout the system in the same order given by AR. This is formalised by the axiom Prefix: if T observes S, then it also observes all AR-predecessors of S. Since $\mathsf{AR} \supseteq \mathsf{VIS}$, Prefix implies TransVis. The axiom Prefix disallows any execution with the history in Figure 3(d): T_1 and T_2 have to be related by AR one way or another; but then by Prefix, either T_4 has to observe *post1* or T_3 has to observe *post2*.

Even though consistent prefix and snapshot isolation ensure that transactions become visible to others in the same order, they allow this to happen with a delay, caused by asynchronous propagation of updates in implementations. This leads to the *write skew* anomaly shown in Figure 3(e). Here each of T_1 and T_2 checks that the combined balance

Φ	Consistency model	Axioms (Figure 2)	Fractured reads	Causality violation	Lost update	Long fork	Write skew
RA	Read Atomic [6]	INT, EXT	✘	✓	✓	✓	✓
CC	Causal consistency [12, 19]	INT, EXT, TRANSVIS	✘	✘	✓	✓	✓
PSI	Parallel snapshot isolation [21, 24]	INT, EXT, TRANSVIS, NOCONFLICT	✘	✘	✘	✓	✓
PC	Prefix consistency [13]	INT, EXT, PREFIX	✘	✘	✓	✘	✓
SI	Snapshot isolation [8]	INT, EXT, PREFIX, NOCONFLICT	✘	✘	✘	✘	✓
SER	Serialisability [20]	INT, EXT, TOTALVIS	✘	✘	✘	✘	✘

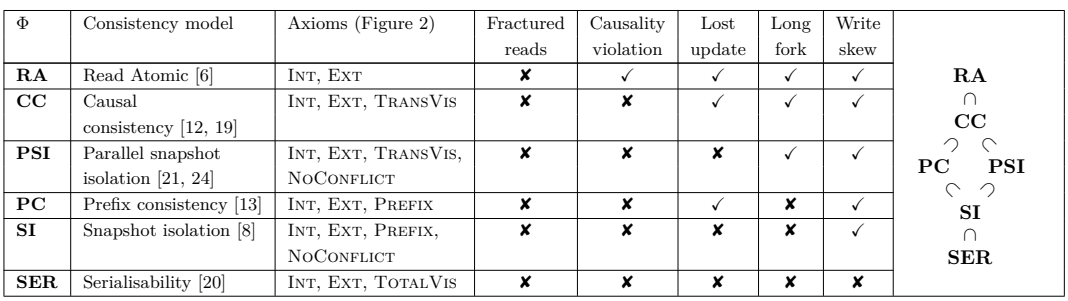

Figure 1 Consistency model definitions, anomalies and relationships.

$$\forall (E, \mathsf{po}) \in \mathcal{H}.\, \forall e \in E.\, \forall x, n.\, (e = (\_, \mathsf{read}(x, n)) \wedge (\mathsf{po}^{-1}(e) \cap \mathsf{HEvent}_x \neq \emptyset)) \implies \max_{\mathsf{po}}(\mathsf{po}^{-1}(e) \cap \mathsf{HEvent}_x) = (\_, \_(x, n)) \quad \text{(INT)}$$

$$\forall T \in \mathcal{H}.\, \forall x, n.\, T \vdash \mathsf{Read}\ x : n \implies ((\mathsf{VIS}^{-1}(T) \cap \{S \mid S \vdash \mathsf{Write}\ x : \_\} = \emptyset \wedge n = 0) \vee \max_{\mathsf{AR}}(\mathsf{VIS}^{-1}(T) \cap \{S \mid S \vdash \mathsf{Write}\ x : \_\}) \vdash \mathsf{Write}\ x : n) \quad \text{(EXT)}$$

VIS is transitive (TRANSVIS)	AR; VIS $\subseteq$ VIS (PREFIX)	VIS is total (TOTALVIS)

$$\forall T, S \in \mathcal{H}.\, (T \neq S \wedge T \vdash \mathsf{Write}\ x : \_ \wedge S \vdash \mathsf{Write}\ x : \_) \implies (T \xrightarrow{\mathsf{VIS}} S \vee S \xrightarrow{\mathsf{VIS}} T) \quad \text{(NOCONFLICT)}$$

Figure 2 Consistency axioms, constraining an execution $(\mathcal{H}, \mathsf{VIS}, \mathsf{AR})$.

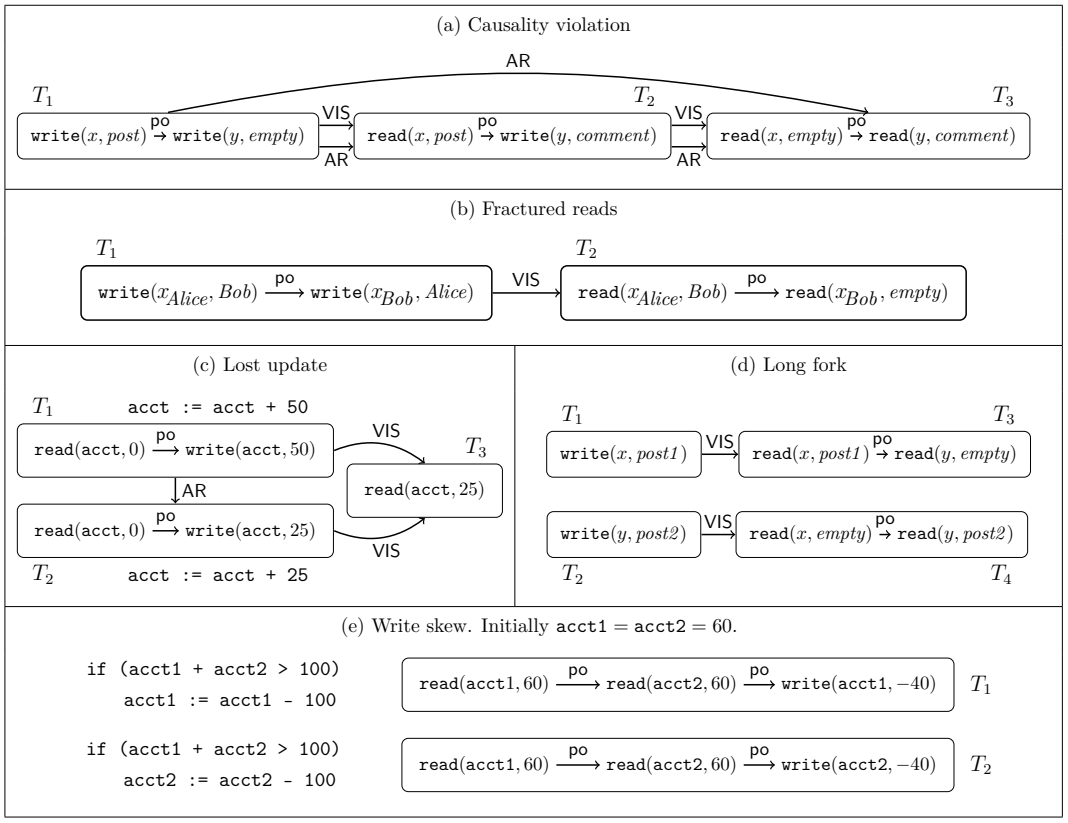

Figure 3 Executions illustrating anomalies allowed by different consistency models. The boxes group events into transactions. We sometimes omit irrelevant AR edges.

of two accounts exceeds 100 and, if so, withdraws 100 from one of them. Both transactions pass the checks and make the withdrawals from different accounts, resulting in the combined balance going negative. NoConflict allows the transactions to be concurrent, because they write to different objects.

Write skew and all the other anomalies mentioned above are disallowed by the classical consistency model of serialisability. Informally, a history is serialisable if the results of operations in it could be obtained by executing its (committed) transactions in some total order according to the usual sequential semantics. We formalise this in our framework by the axiom TotalVis, which requires the visibility relation VIS to be total. Since we always have $\mathsf{AR} \supseteq \mathsf{VIS}$, it is easy to see that there is no execution with the history in Figure 3(e) and a total VIS that would satisfy Ext.

Ramifications. The above specifications demonstrate the benefits of using our framework. First, the specifications are *declarative*, since they state constraints on database processing in terms of VIS and AR relations, rather than the database internals. The specifications thus allow checking whether a consistency model admits a given history solely in terms of these relations, as per (2).

The declarative nature of our specifications also provides a *better understanding* of consistency models. In particular, it makes apparent the relationships between different models and highlights the main mechanism of strengthening consistency—mandating that more edges be included into visibility.

▶ Proposition 3. The strict inclusions between the consistency models in Figure 1 hold.

The strictness of the inclusions in Figure 1 follows from the examples of histories in Figure 3.

Axiomatic specifications also provide an effective tool for *designing new consistency models*. For example, the existing consistency models do not include a counterpart of Read Atomic obtained by adding the NoConflict axiom. Such an "Update Atomic" consistency model would prevent lost update anomalies without having to enforce causal consistency (as in parallel snapshot isolation), which incurs performance overheads [5]. Update Atomic could be particularly useful when *mixed* with Read Atomic, so that the NoConflict axiom apply only to some transactions specified by the programmer. This provides a lightweight way of strengthening consistency where necessary.

Atomic visibility and observational refinement. Our specifications are particularly concise because they are tailored to consistency models providing atomic visibility. With axioms Int and Ext establishing this property, additional guarantees can be specified while abstracting from the internal events in transactions: solely in terms of VIS and AR relations on whole transactions and transaction attributes given by the $\vdash$-judgements. To further illustrate the benefits of this way of specification, we now exploit it to establish sufficient and necessary conditions for when one transaction *observationally refines* another, i.e., we can replace it in an execution without invalidating the consistency axioms. This notion is inspired by that of testing preorders in process algebras [16]. We can think of it as characterising the optimisations that the database can soundly perform inside the transaction due to its atomic visibility. As it happens, the conditions we establish differ subtly depending on the consistency model.

To formulate observational refinement, we introduce ***contexts*** $\mathcal{X}$—abstract executions with a hole $[\,]$ that represents a transaction with an unspecified behaviour: $\mathcal{X} = (\mathcal{H} \cup \{[\,]\}, \mathsf{VIS}, \mathsf{AR})$, where $\mathsf{VIS}, \mathsf{AR} \subseteq (\mathcal{H} \cup \{[\,]\}) \times (\mathcal{H} \cup \{[\,]\})$ satisfy the conditions in Definition 2. We can fill in the hole in the above context $\mathcal{X}$ by a transaction T, provided that the sets

of event identifiers appearing in T and $\mathcal{H}$ are disjoint. This yields the abstract execution $\mathcal{X}[T] = (\mathcal{H} \cup \{T\}, \mathsf{VIS}[[\,] \mapsto T], \mathsf{AR}[[\,] \mapsto T])$, where $\mathsf{VIS}[[\,] \mapsto T]$ treats T in the same way as VIS treats $[\,]$ and similarly for $\mathsf{AR}[[\,] \mapsto T]$ (we omit the formal definition to conserve space). We say that a transaction T_1 ***observationally refines*** a transaction T_2 on the consistency model Φ, written $T_1 \sqsubseteq_\Phi T_2$, if $\forall \mathcal{X}.\, \mathcal{X}[T_1] \models \Phi \implies \mathcal{X}[T_2] \models \Phi$.

▶ **Theorem 4.** *Let T_1, T_2 be such that $(\{T_1, T_2\}, \emptyset, \emptyset) \models$ Int. We have $T_1 \sqsubseteq_{\mathbf{RA}} T_2$ if and only if for all x, n:*

$$\big(\neg(T_1 \vdash \mathsf{Read}\ x : n) \implies \neg(T_2 \vdash \mathsf{Read}\ x : n)\big) \wedge \big(T_1 \vdash \mathsf{Write}\ x : n \iff T_2 \vdash \mathsf{Write}\ x : n\big).$$

For $\Phi \in \{\mathbf{CC}, \mathbf{PC}, \mathbf{SER}\}$ we have $T_1 \sqsubseteq_\Phi T_2$ if and only if for all x, n, m, l:

$$\big(\neg(T_1 \vdash \mathsf{Read}\ x : n) \implies (\neg(T_2 \vdash \mathsf{Read}\ x : n) \wedge (T_1 \vdash \mathsf{Write}\ x : n \iff T_2 \vdash \mathsf{Write}\ x : n))\big) \wedge$$
$$\big((T_1 \vdash \mathsf{Read}\ x : n \wedge (T_1 \vdash \mathsf{Write}\ x : m \implies m = n)) \implies (T_2 \vdash \mathsf{Write}\ x : l \implies l = n)\big).$$

For $\Phi \in \{\mathbf{SI}, \mathbf{PSI}\}$ we have $T_1 \sqsubseteq_\Phi T_2$ if and only if $T_1 \sqsubseteq_{\mathbf{CC}} T_2$ and for all x, n:

$$\neg(T_1 \vdash \mathsf{Write}\ x : n) \implies \neg(T_2 \vdash \mathsf{Write}\ x : n).$$

We prove the theorem in §A. In the case of $\Phi = \mathbf{RA}$, we prohibit T_2 from reading more objects than T_1 or changing the values read by T_1; however, it is safe for T_2 to read less than T_1. We also require T_1 and T_2 to have the same sets of final writes. The case of $\Phi \in \{\mathbf{CC}, \mathbf{PC}\}$ introduces two exception to the latter requirement. One exception is when T_1 reads an object and writes the same value to it. Then T_2 may not change the value written, but may omit the write. Another exception is when T_1 reads an object, but does not write to it. Then T_2 can write the value read without invalidating the reads in the context. This is disallowed when $\Phi \in \{\mathbf{SI}, \mathbf{PSI}\}$.

4 Operational Specifications

To justify that our axiomatic specifications of weak consistency models indeed faithfully describe the intended database behaviour, we now prove that they are equivalent to alternative *operational* ones. These are given as algorithms that are close to actual implementations [6, 13, 19, 24], yet abstract from some of the more low-level features that such implementations have. We start by giving an operational specification of the weakest consistency model we consider, Read Atomic. We then specify other models weaker than serialisability by assuming additional guarantees about the communication between replicas in this algorithm.

4.1 Operational Specification of Read Atomic

Informally, the idealised algorithm for Read Atomic operates as follows. The database consists of a set of *replicas*, identified by $\mathsf{RId} = \{r_0, r_1, \ldots\}$, each maintaining a copy of all objects. The set RId is infinite, to model dynamic replica creation. We assume that the system is fully connected: each replica can broadcast messages to all others. All client operations within a given transaction are initially executed at a single replica (though operations in *different* transactions can be executed at different replicas). For simplicity, we assume that every transaction eventually terminates. When this happens, the replica decides whether to commit or abort it. In the former case, the replica sends a message to all other replicas containing the *transaction log*, which describes the updates done by the transaction. The

replicas incorporate the updates into their state upon receiving the message. A transaction log has the form $t : \rho$, where $\rho \in \{\mathtt{write}(x, n) \mid x \in \mathsf{Obj}, n \in \mathbb{N}\}^* \triangleq \mathsf{UpdateList}$. This gives the sequence of values written to objects and the unique *timestamp* $t \in \mathbb{N}$ of the transaction, which is used to determine the precedence of different object versions (and thus implements the AR relation in abstract executions). We denote the set of all sets of logs with distinct timestamps by LogSet.

Every replica processes transactions locally without interleaving. This idealisation does not limit generality, since all anomalies that would result from concurrent execution of transactions at a single replica arise anyway because of the asynchronous propagation of updates between replicas. The above assumption allows us to maintain the state of a replica r in the algorithm by a pair $(D, l) \in \mathsf{RState} \triangleq \mathsf{LogSet} \times (\mathsf{UpdateList} \uplus \{\mathsf{idle}\})$, where:

- l is either the sequence of updates done so far by the (single) transaction currently executing at r, or idle, signifying that no transaction is currently executing; and
- D is the database copy of r, represented by the set of logs of transactions that have committed at r or have been received from other replicas.

Then a *configuration* of the whole system $(R, M) \in \mathsf{Config} \triangleq (\mathsf{RId} \to \mathsf{RState}) \times \mathsf{LogSet}$ is described by the state $R(r)$ of every replica r and the pool of messages M in transit among replicas.

Formally, our algorithm is defined using the transition relation $\twoheadrightarrow$: $\mathsf{Config} \times \mathsf{LEvent} \times \mathsf{Config}$ in Figure 4, which describes how system configurations change in response to *low-level events* from a set LEvent, describing actions by clients and message receipts by replicas. The set LEvent consists of triples of the form $(\iota, r, \mathbf{o})$, where $\iota \in \mathsf{EventId}$ is the event identifier, $r \in \mathsf{RId}$ is the replica the event occurs at, and $\mathbf{o}$ is a *low-level operation* from the set

$$\mathsf{COp} = \{\mathtt{start}, \mathtt{read}(x, n), \mathtt{write}(x, n), \mathtt{commit}(t), \mathtt{abort}, \mathtt{receive}(t : \rho) \mid x \in \mathsf{Obj}, n \in \mathbb{Z}, t \in \mathbb{N}, \rho \in \mathsf{UpdateList}\}.$$

We use $\mathbf{e}, \mathbf{f}, \mathbf{g}$ to range over low-level events.

According to $\twoheadrightarrow$, when a client starts a transaction at a replica r (Start), the database initialises the current sequence of updates to signify that a transaction is in progress. Since a replica processes transactions serially, a transaction can start only if r is not already executing a transaction. When a client writes n to an object x at a replica r (Write), the corresponding record $\mathtt{write}(x, n)$ is appended to the current sequence of updates. This rule can be applied only when r is not executing a transaction. A read of an object x at r (Read) returns the value determined by a $\mathsf{lastval}$ function based on the transactions in r's database copy and the current transaction. For $D' \in \mathsf{LogSet}$ we define $\mathsf{lastval}(x, D')$ as the last value written to x by the transaction with the highest timestamp among those in D', or 0 if x is not mentioned in D'. Since the timestamps of transactions in D' are distinct, this defines $\mathsf{lastval}(x, D')$ uniquely. For brevity, we omit its formal definition. Note that (Read) implies that a transaction always reads from its own writes and a snapshot of the database the replica had at its start; the transaction is not affected by writes concurrently executing at other replicas, thus ensuring the absence of unrepeatable reads (§3).

If a transaction aborts at a replica r (Abort), the current sequence of updates of r is cleared. If the transaction commits (Commit), it gets assigned a timestamp t, and its log is added to the message pool, as well as to r's database copy. The timestamp t is chosen to be greater than the timestamps of all the transactions in r's database copy, which validates the condition $\mathsf{AR} \supseteq \mathsf{VIS}$ in Definition 2. The timestamp t also has to be distinct from any timestamp assigned previously in the execution. The fact that (Commit) sends all updates by a transaction in a single message ensures atomic visibility. Note that, in Read Atomic, a

$$\text{(Start)}\quad \frac{\mathbf{e} = (\_, r, \texttt{start})}{(R[r \mapsto (D, \mathsf{idle})], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \varepsilon)], M)}$$

$$\text{(Write)}\quad \frac{\mathbf{e} = (\_, r, \texttt{write}(x, n))}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \rho \cdot \texttt{write}(x, n))], M)}$$

$$\text{(Read)}\quad \frac{\mathbf{e} = (\_, r, \texttt{read}(x, n)) \qquad n = \mathsf{lastval}(x, D \cup \{\infty : \rho\})}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \rho)], M)}$$

$$\text{(Abort)}\quad \frac{\mathbf{e} = (\_, r, \texttt{abort})}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \mathsf{idle})], M)}$$

$$\text{(Commit)}\quad \frac{\begin{array}{l}\mathbf{e} = (\_, r, \texttt{commit}(t)) \\ (\forall r', D'.\, R(r') = (D', \_) \implies (t : \_) \notin D') \qquad (\forall t'.\, (t' : \_) \in D \implies t > t')\end{array}}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D \cup \{t : \rho\}, \mathsf{idle})], M \cup \{t : \rho\})}$$

$$\text{(Receive)}\quad \frac{\mathbf{e} = (\_, r, \texttt{receive}(t : \rho))}{(R[r \mapsto (D, \mathsf{idle})], M \cup \{(t : \rho)\}) \xrightarrow{\mathbf{e}} (R[r \mapsto (D \cup \{(t : \rho)\}, \mathsf{idle})], M \cup \{t : \rho\})}$$

Figure 4 Transition relation $\rightarrow : \mathsf{Config} \times \mathsf{LEvent} \times \mathsf{Config}$ for defining the operational specification. We let $R[r \mapsto u]$ be the function that has the same value as R everywhere except r, where it has the value u; $\cdot$ denotes sequence concatenation, and ε the empty sequence.

$$\big(\mathbf{e}_1 \in \{(\_, r, \texttt{receive}(t_1 : \_)), (\_, r, \texttt{commit}(t_1))\} \wedge \mathbf{e}_2 = (\_, r, \texttt{commit}(t_2)) \wedge \mathbf{e}_1 \prec \mathbf{e}_2 \wedge r \neq r' \wedge \mathbf{f}_2 = (\_, r', \texttt{receive}(t_2 : \_))\big) \implies \big(\exists \mathbf{f}_1 \in \{(\_, r', \texttt{receive}(t_1 : \_)), (\_, r', \texttt{commit}(t_1))\}.\, \mathbf{f}_1 \prec \mathbf{f}_2\big) \tag{CausalDeliv}$$

$$\big(\mathbf{e}_1 = (\_, \_, \texttt{commit}(t_1)) \wedge \mathbf{e}_2 = (\_, \_, \texttt{commit}(t_2)) \wedge \mathbf{e}_1 \prec \mathbf{e}_2\big) \implies t_1 < t_2 \tag{MonTS}$$

$$\big(\mathbf{g} = (\_, r, \texttt{start}) \wedge \mathbf{e}_2 \in \{(\_, r, \texttt{commit}(t_2)), (\_, r, \texttt{receive}(t_2 : \_))\} \wedge \mathbf{f} = (\_, \_, \texttt{commit}(t_1)) \wedge t_1 < t_2 \wedge \mathbf{e}_2 \prec \mathbf{g}\big) \implies \big(\exists \mathbf{e}_1 \in \{(\_, r, \texttt{commit}(t_1)), (\_, r, \texttt{receive}(t_1 : \_))\}.\, \mathbf{e}_1 \prec \mathbf{g}\big) \tag{TotalDeliv}$$

$$\begin{array}{l}\big(\mathbf{e}_1 = (\_, r, \texttt{write}(x, \_)) \wedge \mathbf{f}_1 = (\_, r, \texttt{commit}(t_1)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_1) = t_1 \wedge \\ \mathbf{e}_2 = (\_, r', \texttt{write}(x, \_)) \wedge \mathbf{f}_2 = (\_, r', \texttt{commit}(t_2)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_2) = t_2 \wedge \mathbf{f}_2 \prec \mathbf{f}_1 \wedge r \neq r'\big) \\ \implies \big(\exists \mathbf{g} \in \mathbf{E}.\, \mathbf{g} = (\_, r, \texttt{receive}(t_2 : \_)) \wedge \mathbf{g} \prec \mathbf{f}_1\big),\end{array} \tag{ConflictCheck}$$

where for $\mathbf{e} \in \mathbf{E}$ we let

$$\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = \begin{cases} t, & \text{if } \exists r.\, \mathbf{e} \in \{(\_, r, \texttt{read}(\_, \_)), (\_, r, \texttt{write}(\_, \_))\} \wedge \\ & \quad \exists \mathbf{g} \in \mathbf{E}.\, \mathbf{g} = (\_, r, \texttt{commit}(t)) \wedge \\ & \quad \neg(\exists \mathbf{f} \in \{(\_, r, \texttt{commit}(\_)), (\_, r, \texttt{abort})\}.\, (\mathbf{e} \prec \mathbf{f} \prec \mathbf{g})) \\ \text{undefined}, & \text{otherwise} \end{cases}$$

Φ	Constraints	Φ	Constraints	Φ	Constraints
RA	None	**PSI**	(CausalDeliv), (ConflictCheck)	**SI**	(MonTS), (TotalDeliv), (ConflictCheck)
CC	(CausalDeliv)	**PC**	(MonTS), (TotalDeliv)		

Figure 5 Constraints on concrete executions $\mathcal{C} = (\mathbf{E}, \prec)$ required by various consistency models. Free variables are universally quantified and range over the following domains: $\mathbf{e}_i, \mathbf{f}, \mathbf{f}_i, \mathbf{g} \in \mathbf{E}$ for $i = 1, 2$; $t_1, t_2 \in \mathbb{N}$; $r, r' \in \mathsf{RId}$.

transaction can always commit; as we explain in the following, this is not the case for some of the other consistency models. Finally, a replica r that is not executing a transaction can receive a transaction log from the message pool (Receive), adding it to the database copy.

We define the semantics of Read Atomic by considering all sequences of transitions generated by $\twoheadrightarrow$ from an initial configuration where the log sets of all replicas and the message pool are empty. We thereby consider all possible operations that clients could issue to the database.

▶ **Definition 5.** Let $(R_0, M_0) = (\lambda r.\, (\emptyset, \mathsf{idle}), \emptyset)$. A *concrete execution* is a pair $\mathcal{C} = (\mathbf{E}, \prec)$, where: $\mathbf{E} \subseteq \mathsf{LEvent}$; $\prec$ is a prefix-finite, total order on $\mathbf{E}$; and if $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \ldots$ is the enumeration of the events in $\mathbf{E}$ defined by $\prec$, then for some configurations $(R_1, M_1), (R_2, M_2), \ldots \in \mathsf{Config}$ we have $(R_0, M_0) \xrightarrow{\mathbf{e}_1} (R_1, M_1) \xrightarrow{\mathbf{e}_2} (R_2, M_2) \xrightarrow{\mathbf{e}_3} \ldots$

4.2 Correspondence to Axiomatic Specifications and Other Models

We next show that the above operational specification indeed defines the semantics of Read Atomic, and that stronger models can be defined by assuming additional guarantees about communication between replicas. These guarantees are formalised by the constraints on concrete executions in Figure 5; in implementations they would be ensured by distributed protocols that our specifications abstract from.

We first map each concrete execution into a history, which includes only reads and writes in its committed transactions. The history of $\mathcal{C} = (\mathbf{E}, \prec)$ is defined as follows:

$$\begin{aligned}
&\mathsf{history}(\mathcal{C}) = \big\{T_t \mid \{\mathbf{e} \in \mathbf{E} \mid \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t\} \neq \emptyset\big\}, \text{ where } T_t = (E_t, \mathsf{po}_t) \text{ for}\\
&E_t = \{(\iota, \mathbf{o}) \mid \exists \mathbf{e} \in \mathbf{E}.\, \mathbf{e} = (\iota, _, \mathbf{o}) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t\};\\
&\mathsf{po}_t = \{(\iota_1, \mathbf{o}_1), (\iota_2, \mathbf{o}_2) \mid (\iota_1, \mathbf{o}_1), (\iota_2, \mathbf{o}_2) \in E_t \wedge (\iota_1, _, \mathbf{o}_1) \prec (\iota_2, _, \mathbf{o}_2)\},
\end{aligned}$$

where $\mathsf{TS}_{\mathcal{C}}$ is defined in Figure 5. We lift the function $\mathsf{history}$ to sets of concrete executions as expected.

▶ **Theorem 6.** *For a consistency model Φ let $\mathsf{ConcExec}_\Phi$ be the set of concrete executions satisfying the model-specific constraints in Figure 5. Then $\mathsf{history}(\mathsf{ConcExec}_\Phi) = \mathsf{Hist}_\Phi$.*

Proof outline. We defer the full proof to §B. Here we sketch the argument for one set inclusion ($\subseteq$) and, on the way, explain the constraints in Figure 5. Fix a Φ and let $\mathcal{C} = (\mathbf{E}, \prec) \in \mathsf{ConcExec}_\Phi$. To show $\mathsf{history}(\mathcal{C}) \in \mathsf{Hist}_\Phi$ we let $\mathcal{A} = (\mathsf{history}(\mathcal{C}), \mathsf{VIS}, \mathsf{AR})$, where $\mathsf{AR} = \{(T_{t_1}, T_{t_2}) \mid t_1 < t_2\}$ and

$$\mathsf{VIS} = \big\{(T_{t_1}, T_{t_2}) \mid \exists \mathbf{e}_1, \mathbf{e}_2 \in \mathbf{E}.\, \exists r.\, \mathbf{e}_1 \in \{(\_, r, \mathtt{commit}(t_1)), (\_, r, \mathtt{receive}(t_1 : \_))\} \wedge \mathbf{e}_2 = (\_, r, \mathtt{commit}(t_2)) \wedge \mathbf{e}_1 \prec \mathbf{e}_2\big\}.$$

While AR merely lifts the order on timestamps to transactions, VIS reflects message delivery: $T_{t_1} \xrightarrow{\mathsf{VIS}} T_{t_2}$ if the effects of T_{t_1} have been incorporated into the state of the replica where T_{t_2} is executed. We can show that any abstraction execution $\mathcal{A}$ constructed from a concrete execution $\mathcal{C}$ as above satisfies Int and Ext, and hence, its history belongs to $\mathsf{Hist}_{\mathbf{RA}}$. The constraints on a concrete execution $\mathcal{C}$ in Figure 5 ensure that the abstract execution $\mathcal{A}$ constructed from it satisfies other axioms in Figure 2.

Constraint (CausalDeliv) implies the axiom TransVis, because it ensures that the message delivery is *causal* [9]: if a replica r sends the log of a transaction t_2 (event $\mathbf{e}_2$) after it sends or receives the log of t_1 (event $\mathbf{e}_1$), then every other replica r' will receive the log of t_2 (event $\mathbf{f}_2$) only after it receives or sends the log of t_1 (event $\mathbf{f}_1$).

The axiom Prefix follows from constraints (MonTS) and (TotalDeliv). The constraint (MonTS) requires that timestamps agree with the order in which transactions commit. The constraint (TotalDeliv) requires that each transaction access a database snapshot that is closed under adding transactions with timestamps (t_1) smaller than the ones already present in the snapshot (t_2). In an implementation, the above constraints can be satisfied if replicas communicate via a central server, which assigns timestamps to transactions when they commit, and propagates their logs to replicas in the order of their timestamps [13].

The axiom NoConflict follows from the constraint (ConflictCheck), similar to that in the original definitions of **SI** [8] and **PSI** [24]. The constraint allows a transaction t_1 to commit at a replica r (event $\mathbf{f}_1$) only if it passes a *conflict detection check*: if t_1 updates an object x (event $\mathbf{e}_1$) that is also updated by a transaction t_2 (event $\mathbf{e}_2$) committed at another replica r' (event $\mathbf{f}_2$), then the replica r must have received the log of t_2 (event $\mathbf{g}$). If this check fails, the only option left for the database is to abort t using the rule (Abort). Implementing the check in a realistic system would require the replica r to coordinate with others on commit [24]. □

The above operational specifications are closer to the intuition of practitioners [6, 13, 19, 24] and thus serve to validate our axiomatic specifications. However, they are more verbose and reasoning about database behaviour using them may get unwieldy. It requires us to keep track of low-level information about the system state, such as the logs at all replicas and the set of messages in transit. We then need to reason about how the system state is affected by a large number of possible interleavings of operations at different replicas. In contrast, our axiomatic specifications (§3) are more declarative and, in particular, do not refer to implementation-level details, such as message exchanges between replicas. These specifications thereby facilitate reasoning about the database behaviour.

5 Related Work

Our specification framework builds on the axiomatic approach to specifying consistency models, previously applied to weak shared-memory models [3] and eventual consistency [11, 12]. In particular, the visibility and arbitration relations were first introduced for specifying eventual consistency and causally consistent transactions [12]. In comparison to prior work, we handle more sophisticated transactional consistency models. Furthermore, our framework is specifically tailored to transactional models with atomic visibility, by defining visibility and arbitration relations on whole transactions as opposed to events. This avoids the need to enforce atomic visibility explicitly in all axioms [12], thus simplifying specifications.

Adya [2] has previously proposed specifications for weak consistency models of transactions in classical databases. His framework also broadly follows the axiomatic specification approach, but uses relations different from visibility and arbitration. Adya's work did not address the variety of consistency models for large-scale databases proposed recently, while our framework is particularly appropriate for these. On the other hand, Adya handled transactional consistency models that do not guarantee atomic visibility, such as Read Committed, which we do not address. Adya also specified snapshot isolation (**SI**), which is a weak consistency model older than the others we consider. However, his specification is low-level, since it introduces additional events to denote the times at which a transaction takes a snapshot of the database state. Saeida Ardekani et al. [22] have since proposed a higher-level specification for snapshot isolation; this specification still uses relations on individual events and thus does not exploit atomic visibility.

Partial orders have been used to define semantics of concurrent and distributed programs,

e.g., by event structures [26]. Our results extend this research line by considering new kinds of relations among events, appropriate to describe computations of weakly consistent databases, and by relating the resulting abstract specifications to lower-level algorithms.

Prior work has investigated calculi with transactions communicating via message passing: cJoin [10], TCCS^m [17] and RCCS [15]. Even though replicated database implementations and our operational specifications are also based on message passing, the database interface that we consider allows client programs only to read and write objects. Thereby, it provides the programs with an (imperfect) illusion of *shared memory*, and our goal was to provide specifications for this interface that abstract from its message passing-based implementation.

6 Conclusion

We have proposed a framework for uniformly specifying transactional consistency models of modern replicated databases. The axiomatic nature of our framework makes specifications declarative and concise, with further simplicity brought by exploiting atomic visibility. We have illustrated the use of the framework by specifying several existing consistency models and thereby systematising the knowledge about them. We have also validated our axiomatic specifications by proving their equivalence to operational specifications that are closer to implementations.

We hope that our work will promote an exchange of ideas between the research communities of large-scale databases and concurrency theory. In particular, our framework provides a basis to develop techniques for reasoning about the correctness of application programs using modern databases; this is the subject of our ongoing work.

Finally, axiomatic specifications are well-suited for systematically exploring the design space of consistency models. In particular, insights provided by the specifications may suggest new models, as we illustrated by the Update Atomic model in §3. This is likely to help in the design of the sophisticated programming interfaces that replicated databases are starting to provide to compensate for the weakness of their consistency models. For example, so-called replicated data types [23] avoid lost updates by eventually merging concurrent updates without coordination between replicas, and sessions [25] provide additional consistency guarantees for transactions issued by the same client. Finally, there are also interfaces that allow the programmer to request different consistency models for different transactions [18], analogous to fences in weak memory models [3]. In the future we plan to generalise our techniques to handle the above features. We expect to handle replicated data types by integrating our framework with their specifications proposed in [11], and to handle sessions and mixed consistency models by studying additional constraints on the visibility and arbitration relations. We believe that the complexity of database consistency models and the above programming interfaces makes it indispensable to specify them formally and declaratively. Our work provides the necessary foundation for achieving this.

Acknowledgements. We thank Artem Khyzha, Vasileios Koutavas, Hongseok Yang and the anonymous reviewers for comments that helped improve the paper. This work was supported by the EU FET project ADVENT.

References

1 D. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, 45(2), 2012.

2 A. Adya. Weak consistency: A generalized theory and optimistic implementations for distributed transactions. PhD thesis, MIT, 1999.

3 J. Alglave. A formal hierarchy of weak memory models. *FMSD*, 41(2), 2012.

4 P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Highly Available Transactions: virtues and limitations. In *VLDB*, 2014.

5 P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. The potential dangers of causal consistency and an explicit solution. In *SOCC*, 2012.

6 P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Scalable atomic visibility with RAMP transactions. In *SIGMOD*, 2014.

7 M. Batty, M. Dodds, and A. Gotsman. Library abstraction for C/C++ concurrency. In *POPL*, 2013.

8 H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD*, 1995.

9 K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1), 1987.

10 R. Bruni, H. C. Melgratti, and U. Montanari. cJoin: Join with communicating transactions. *Mathematical Structures in Computer Science*, 25(3), 2015.

11 S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski. Replicated data types: specification, verification, optimality. In *POPL*, 2014.

12 S. Burckhardt, D. Leijen, M. Fähndrich, and M. Sagiv. Eventually consistent transactions. In *ESOP*, 2012.

13 S. Burckhardt, D. Leijen, J. Protzenko, and M. Fähndrich. Global sequence protocol: A robust abstraction for replicated shared state. In *ECOOP*, 2015.

14 A. Cerone, G. Bernardi, and A. Gotsman. A framework for transactional consistency models with atomic visibility (extended version). Available from http://software.imdea.org/∼gotsman/.

15 V. Danos and J. Krivine. Transactions in RCCS. In *CONCUR*, 2005.

16 R. De Nicola and M. Hennessy. Testing equivalence for processes. In *ICALP*, 1983.

17 V. Koutavas, C. Spaccasassi, and M. Hennessy. Bisimulations for communicating transactions - (extended abstract). In *FOSSACS*, 2014.

18 C. Li, D. Porto, A. Clement, R. Rodrigues, N. Preguiça, and J. Gehrke. Making geo-replicated systems fast if possible, consistent when necessary. In *OSDI*, 2012.

19 W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In *SOSP*, 2011.

20 C. H. Papadimitriou. The serializability of concurrent database updates. *J. ACM*, 26(4), 1979.

21 M. Saeida Ardekani, P. Sutra, and M. Shapiro. Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems. In *SRDS*, 2013.

22 M. Saeida Ardekani, P. Sutra, M. Shapiro, and N. Preguiça. On the scalability of snapshot isolation. In *Euro-Par*, 2013.

23 M. Shapiro, N. M. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In *SSS*, 2011.

24 Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *SOSP*, 2011.

25 D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. W. Welch. Session guarantees for weakly consistent replicated data. In *PDIS*, 1994.

26 G. Winskel. Event structure semantics for CCS and related languages. In *ICALP*, 1982.

A Proof of Theorem 4

In the following, we adopt the notation discussed below. Given two sets of events E, F, we let $E \Cap F = \{e \in E \mid \exists \iota.e = (\iota, \_) \wedge \exists f \in F.f = (\iota, \_)\} \cup \{f \in F \mid \exists \iota.f = (\iota, \_) \wedge \exists e \in E.e = (\iota, \_)\}$. This is the sets of events belonging to either E of F, whose identifier identifies at least one event in both sets.

Given a set of transactions $\mathcal{H}$, we let $\mathcal{H}_{[]} = \mathcal{H} \cup \{[]\}$. Given a transaction T such that $T \Cap T' = \emptyset$ for any $T' \in \mathcal{H}$, we define $\mathcal{H}[T] = \mathcal{H} \cup \{T\}$. Let $\mathcal{R} \subseteq \mathcal{H}_{[]} \times \mathcal{H}[]$, and a transaction T such that for any $T' \in \mathcal{H}$, $T \Cap T' = \emptyset$. We define $\mathcal{R}[[] \mapsto T] = \mathcal{R}|_{\mathcal{H}} \cup \{(T, T') \mid T' \in \mathcal{R}([])\} \cup \{(T', T) \mid T' \in \mathcal{R}^{-1}([])\}$, where we recall that $\mathcal{R}|_{\mathcal{H}}$ is the restriction of $\mathcal{R}$ to the set $\mathcal{H}$. To maintain an easy notation, we will often write $\mathcal{R}[T]$ in lieu of $\mathcal{R}[[] \mapsto T]$.

▶ **Lemma 7.** *Let $\mathcal{H}$ be a set of transactions. If $(\mathcal{H}, \mathsf{AR}, \mathsf{VIS}) \models$ Int, then for any AR', VIS' and $\mathcal{H}' \subseteq \mathcal{H}$ we have that $(\mathcal{H}, \mathsf{AR}', \mathsf{VIS}') \models$ Int. Also, if $\mathcal{H}_1, \mathcal{H}_2$ are such that $(\mathcal{H}_1, \_, \_) \models$ Int, $(\mathcal{H}_2, \_, \_) \models$ Int, then $(\mathcal{H}_1 \cup \mathcal{H}_2, \_, \_) \models$ Int.*

Proof. This is because neither AR, nor VIS, appear in the Definition of Int in Figure 2. Also, Int is a property that quantifies over all transactions in a history $\mathcal{H}$. ◀

▶ **Lemma 8.** *Let T_1, T_2 be two transactions such that $(\{T_2\}, \_, \_) \models$ Int. Then, for any context $\mathcal{X}[]$ such that both $\mathcal{X}[T_1], \mathcal{X}[T_2]$ are defined, $\mathcal{X}[T_1] \models$ Int implies that $\mathcal{X}[T_2] \models$ Int.*

Proof. Let $\mathcal{X} = (\mathcal{H}_{[]}, \mathsf{AR}, \mathsf{VIS})$. Recall that $\mathcal{H}[T_1] = \mathcal{H} \cup \{T_1\}$. Since $\mathcal{X}[T_1] \models$ Int by hypothesis, by Lemma 7 it follows that $(\mathcal{H}, \_, \_) \models$ Int. By hypothesis, $(\{T_2\}, \_, \_) \models$ Int, hence by Lemma 7 and the definition $\mathcal{H}[T_2] = \mathcal{H} \cup \{T_2\}$ again we get that $(\mathcal{H}[T_2], \mathsf{AR}[T_2], \mathsf{VIS}[T_2]) \models$ Int This last abstract execution is exactly $\mathcal{X}[T_2]$. ◀

▶ **Definition 9.** For any two transactions T_1, T_2, we write $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{RA}} T_2$ if, for any $x \in \mathsf{Obj}$ and $n \in \mathbb{N}$,

- if $\neg(T_1 \vdash \mathsf{Read}\ x : n)$ then $\neg(T_2 \vdash \mathsf{Read}\ x : n)$, and
- $T_1 \vdash \mathsf{Write}\ x : n$ if and only if $T_2 \vdash \mathsf{Write}\ x : n$.

▶ Proposition 10. Let $\mathcal{X}$ be a context, and let T_1, T_2 be two transactions such that $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{RA}} T_2$, and both $\mathcal{X}[T_1], \mathcal{X}[T_2]$ are defined. If $\mathcal{X}[T_1] \models$ Ext, then $\mathcal{X}[T_2] \models$ Ext.

Proof. We let $\mathcal{X} = (\mathcal{H}_{[]}, \mathsf{VIS}, \mathsf{AR})$. We show that, for any $T \in \mathcal{H}[T_2]$ such that $T \vdash \mathsf{Read}\ x : n$, either $\max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x) = \emptyset$ and $n = 0$, or $\max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x) \vdash \mathsf{Write}\ x : n$.

Let then $T \in \mathcal{H}[T_2]$ be such that $T \vdash \mathsf{Read}\ x : n$; we distinguish two cases:

- $T = T_2$; note that **(a)** $\mathsf{VIS}[T_1]^{-1}(T_1) = \mathsf{VIS}[T_2]^{-1}(T_2)$, and **(b)** $\mathsf{AR}[T_1]^{-1}(T_1) = \mathsf{AR}[T_2]^{-1}(T_2)$. Since $T_2 \vdash \mathsf{Read}\ x : n$, and because $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{RA}} T_2$, it has to be the case that $T_1 \vdash \mathsf{Read}\ x : n$. Also, $\mathcal{X}[T_1] \models$ Ext by hypothesis, hence there are two possible cases:
  - $(\mathsf{VIS}[T_1]^{-1}(T_1) \cap \mathsf{Write}_x) = \emptyset$ and $n = 0$; In this case it is tirival to see that **(a)** above implies $(\mathsf{VIS}[T_2]^{-1}(T_2) \cap \mathsf{Write}_x) = \emptyset$, and there is nothing to prove,
  - otherwise, let $W = \max_{\mathsf{AR}[T_1]}(\mathsf{VIS}[T_1]^{-1}(T_1) \cap \mathsf{WEvent}_x)$; since $T_1 \vdash \mathsf{Read}\ x : n$ and $\mathcal{X}[T_1] \models$ Ext, then $W \vdash \mathsf{Write}\ x : n$. Now it remains to note that **(a)** and **(b)** above imply that $\max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T_2) \cap \mathsf{WEvent}_x) = W$, hence there is nothing left to prove;
- $T \neq T_2$. Let us distinguish two subcases:

- if $(\mathsf{VIS}[T_1]^{-1}(T) \cap \mathsf{WEvent}_x) = \emptyset$ then $n = 0$, because $\mathcal{X}[T_1] \models \textsc{Ext}$; note that $(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x) \subseteq \{T_2\}$. In practice, we prove that it always has to be the case that $(\mathsf{VIS}[T_2]^{-1}(T_2) \cap \mathsf{WEvent}_x) = \emptyset$ by showing that $T_2 \notin (\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x)$. Since $(\mathsf{VIS}[T_1]^{-1}(T) \cap \mathsf{WEvent}_x) = \emptyset$, either $\neg(T_1 \vdash \mathsf{Write}\ x : \_)$, or $\neg(T_1 \xrightarrow{\mathsf{VIS}[T_1]} T$. If $\neg(T_1 \vdash \mathsf{Write}\ x : \_)$, then $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$ because $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{RA}} T_2$; if$\neg(T_1 \xrightarrow{\mathsf{VIS}[T_1]} T)$, then $\neg(T_2 \xrightarrow{\mathsf{VIS}[T_2]} T)$. In both cases, we obtain that $T_2 \notin (\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x)$,
- otherwise, let $W = \max_{\mathsf{AR}[T_1]}(\mathsf{VIS}[T_1]^{-1}(T) \cap \mathsf{WEvent}_x)$; then, since $T \vdash \mathsf{Read}\ x : n$ and $\mathcal{X}[T_1] \models \textsc{Ext}$, it needs to be $W \vdash \mathsf{Write}\ x : n$. We are left with two sub-cases:
  - $W = T_1$; since $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{RA}} T_2$, it follows that $T_2 \vdash \mathsf{Write}\ x : n$. Also, in this case we have that $T_2 = \max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x)$, and there is nothing left to prove,
  - $W \neq T_1$; in this case we have that $W = \max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x)$, as we wanted to prove.

◀

▶ **Definition 11.** Let T_1, T_2, be two transactions. For $\Phi \in \{\mathbf{CC}, \mathbf{PC}, \mathbf{SER}\}$ we write $T_1 \mathrel{\widehat{\sqsubseteq}}_{\Phi} T_2$ if, for any $x \in \mathsf{Obj}$ and $n \in \mathbb{N}$

- if $\neg(T_1 \vdash \mathsf{Read}\ x : n)$ then $\neg(T_2 \vdash \mathsf{Read}\ x : n)$, and $(T_1 \vdash \mathsf{Write}\ x : n)$ if and only if $(T_2 \vdash \mathsf{Write}\ x : n)$,
- otherwise, if there exists no $m \neq n$ such that $T_1 \vdash \mathsf{Write}\ x : m$, then there exists no $l \neq n$ such that $T_2 \vdash \mathsf{Write}\ x : l$.

▶ Proposition 12. Let $\mathcal{X}$ be a context, and let T_1, T_2 be two transactions such that $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{SER}} T_2$, and both $\mathcal{X}[T_1], \mathcal{X}[T_2]$ are defined. If $\mathcal{X}[T_1] \models \textsc{Ext}$, and $\mathcal{X}[T_1] \models \textsc{TransVis}$, then $\mathcal{X}[T_2] \models \textsc{Ext}$.

Proof. Assume that $\mathcal{X} = (\mathcal{H}_{[]}, \mathsf{AR}, \mathsf{VIS})$. We need to show that, for any transaction $T \in \mathcal{X}[T_2]$, either $\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x = \emptyset$ and $n = 0$, or $\max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x) \vdash \mathsf{Write}\ x : n$. Let $T \in \mathcal{X}[T_2]$ be a transaction such that $T \vdash \mathsf{Read}\ x : n$. There are two possible cases:

- $T \neq T_2$; then $T \in \mathcal{H}$, hence $T \in \mathcal{H}[T_1]$. We distinguish between two sub-cases:
  - Suppose that there are no transactions $T' \in \mathcal{H}[T_1]$ such that $T' \xrightarrow{\mathsf{VIS}[T_1]} T$ and $T' \vdash \mathsf{Write}\ x : \_$. Since $\mathcal{X}[T_1] \models \textsc{Ext}$, then it is $n = 0$.

    If there are no $T'' \in \mathcal{H}[T_2]$ such that $T'' \xrightarrow{\mathsf{VIS}[T_2]} T$ and $T'' \vdash \mathsf{Write}\ x : \_$, then there is nothing left to prove. Otherwise, we have that $\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x = \{T_2\}$. In fact, whenever $T'' \in \mathcal{H}[T_2], T'' \xrightarrow{\mathsf{VIS}[T_2]} T$ and $T'' \vdash \mathsf{Write}\ x : \_$, then it cannot be $T'' \in \mathcal{H}$, since otherwise it would also be $T'' \in \mathcal{H}[T_1]$ and $T'' \xrightarrow{\mathsf{VIS}[T_1]} T$. In particular, $T_2 = \max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x)$, and $T_2 \vdash \mathsf{Write}\ x : m$ for some m. Also, since $T_2 \xrightarrow{\mathsf{VIS}[T_2]} T$, then it is the case that $T_1 \xrightarrow{\mathsf{VIS}[T_1]} T$. There are other two sub-cases to consider:
    - $\neg(T_1 \vdash \mathsf{Read}\ x : m)$. Since we are assuming that $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{SER}} T_2$, and $T_2 \vdash \mathsf{Write}\ x : m$, then it follows that $T_1 \vdash \mathsf{Write}\ x : m$. However, since $T_2 \xrightarrow{\mathsf{VIS}[T_2]} T$, then it also has to be $T_1 \xrightarrow{\mathsf{VIS}[T_1]} T$, contradicting the statement that there exists no $T' \in \mathcal{H}[T_1]$ such that $T' \xrightarrow{\mathsf{VIS}[T_1]} T$ and $T' \vdash \mathsf{Write}\ x : \_$.
    - Therefore it has to be that $T_1 \vdash \mathsf{Read}\ x : m$. Since there are no transactions $T' \in \mathcal{H}[T_1]$ such that $T' \vdash \mathsf{Write}\ x : \_$ and $T' \xrightarrow{\mathsf{VIS}[T_1]} T$, then it is also the

case that there exists no transaction $T' \in \mathcal{H}[T_1]$ such that $T' \vdash \mathsf{Write}\ x : \_$ and $T' \xrightarrow{\mathsf{VIS}[T_1]} T_1$. If it were the case, the transitivity of $\mathsf{VIS}[T_1]$ and the fact that $T_1 \xrightarrow{\mathsf{VIS}[T_1]} T$ would lead to $T' \vdash \mathsf{Write}\ x : \_$ and $T' \xrightarrow{\mathsf{VIS}} T$, causing a contradiction. Since $\mathcal{X}[T_1] \models \textsc{Ext}$, it follows that $m = 0$, as we wanted to prove.

- Otherwise, let $T' \in \mathcal{H}[T_1]$ be such that $T' = \max_{\mathsf{AR}[T_1]}(\mathsf{VIS}[T_1]^{-1}(T) \cap \mathsf{WEvent}_x)$. Since $\mathcal{X}[T_1] \models \textsc{Ext}$, then $T' \vdash \mathsf{Write}\ x : n$. There are two sub-cases to consider:
  - $\neg\big(T' \xrightarrow{\mathsf{VIS}[T_1]} T_1 \xrightarrow{\mathsf{VIS}[T_1]} T\big)$. In this case it follows immediately that $\neg\big(T' \xrightarrow{\mathsf{VIS}[T_2]} T_2 \xrightarrow{\mathsf{VIS}[T_1]} T\big)$, and more in general, whenever T'' is a transaction such that $T \xrightarrow{\mathsf{VIS}[T_1]} T'' \xrightarrow{\mathsf{VIS}[T_1]} T$, then $T \xrightarrow{\mathsf{VIS}[T_2]} T'' \xrightarrow{\mathsf{VIS}[T_2]} T$. As a consequence, we obtain that $T' = \max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x)$, and there is nothing left to prove.
  - $T' \xrightarrow{\mathsf{VIS}[T_1]} T_1 \xrightarrow{\mathsf{VIS}[T_1]} T$. Because of $T' = \max_{\mathsf{AR}[T_1]}(\mathsf{VIS}[T_1]^{-1}(T) \cap \mathsf{WEvent}_x)$, it has to be the case that $\neg(T_1 \vdash \mathsf{Write}\ x : \_)$. If $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$ then there is nothing to prove, since in this case we have that $T' = \max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x)$. Otherwise, $T_2 \vdash \mathsf{Write}\ x : m$ for some m; further, in this case we have that $T_2 = \max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x)$. We can proceed as in the previous case to show that, since $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{SER}} T_2$, it must be $T_2 \vdash \mathsf{Read}\ x : m$; the hypothesis $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{SER}} T_2$ leads then to $T_1 \vdash \mathsf{Read}\ x : m$, and the transitivity of $\mathsf{VIS}[T_1]$ gives that $\max_{\mathsf{AR}[T_1]}(\mathsf{VIS}[T_1]^{-1}(T_1) \cap \mathsf{WEvent}_x) = T'$. Since $\mathcal{X}[T_1] \models \textsc{Ext}$, we obtain $m = n$, as we wanted to prove.
- $T = T_2$; since $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{SER}} T_2$, it has to be the case that $T_1 \vdash \mathsf{Read}\ x : n$. Now note that, if there exists no transaction $T' \in \mathcal{H}[T_1]$ such that $T' \vdash \mathsf{Write}\ x : \_$ and $T' \xrightarrow{\mathsf{VIS}[T_1]} T_1$, then $n = 0$ because $\mathcal{X}[T_1] \models \textsc{Ext}$; also, there exists no $T'' \in \mathcal{H}[T_2]$ such that $T'' \vdash \mathsf{Write}\ x : \_$ and $T'' \xrightarrow{\mathsf{VIS}[T_2]} T_2$, as we wanted to prove.
  Otherwise, let $T' = \max_{\mathsf{AR}[T_1]}(\mathsf{VIS}[T_1]^{-1}(T_1) \cap \mathsf{WEvent}_x)$. Since $\mathcal{X}[T_1] \models \textsc{Ext}$, it follows that $T' \vdash \mathsf{Write}\ x : n$. Now it is not difficult to see that it also has to be $T' = \max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(T_2) \cap \mathsf{WEvent}_x)$, and there is nothing left to prove.

◀

▶ **Definition 13.** Let T_1, T_2, be two transactions. For $\Phi \in \{\mathbf{SI}, \mathbf{PSI}\}$ we write $T_1 \mathrel{\widehat{\sqsubseteq}}_{\Phi} T_2$ if

- $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{CC}} T_2$, and
- for any $x \in \mathsf{Obj}$, if $\neg(T_1 \vdash \mathsf{Write}\ x : \_)$ then $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$.

▶ Proposition 14. Let T_1, T_2 be two transitions such that, whenever $T_2 \vdash \mathsf{Write}\ x : \_$ then $T_1 \vdash \mathsf{Write}\ x : \_$. Let $\mathcal{X}$ be a context such that both $\mathcal{X}[T_1]$ and $\mathcal{X}[T_2]$ are defined. If $\mathcal{X}[T_1] \models \textsc{NoConflict}$, then $\mathcal{X}[T_2] \models \textsc{NoConflict}$.

Proof. Let $\mathcal{X} = (\mathcal{H}_{[]}, \mathsf{AR}, \mathsf{VIS})$. Consider two transactions $S, T \in \mathcal{H}[T_2]$ such that $S \vdash \mathsf{Write}\ x : \_$, $T \vdash \mathsf{Write}\ x : \_$, and $S \neq T$. We show that either $S \xrightarrow{\mathsf{VIS}[T_2]} T$ or $T \xrightarrow{\mathsf{VIS}[T_2]} S$. There are three different cases to consider:

- $S \neq T_2, T \neq T_2$. Then both $S, T \in \mathcal{H}$, and particularly $S, T \in \mathcal{H}[T_1]$. Since $\mathcal{X}[T_1] \models \textsc{NoConflict}$ by hypothesis, it has to be the case that either $S \xrightarrow{\mathsf{VIS}[T_1]} T$ or $T \xrightarrow{\mathsf{VIS}[T_1]} S$. In this case it is immediate to note that either $S \xrightarrow{\mathsf{VIS}[T_2]} T$ or $T \xrightarrow{\mathsf{VIS}[T_2]} S$.
- $S = T_2$; then $T_2 \vdash \mathsf{Write}\ x : \_$, and by hypothesis we have that $T_1 \vdash \mathsf{Write}\ x : \_$. Since $\mathcal{X}[T_1] \models \textsc{NoConflict}$, then either $T_1 \xrightarrow{\mathsf{VIS}[T_1]} T$ or $T \xrightarrow{\mathsf{VIS}[T_1]} T_1$. Without loss of generality, suppose that $T_1 \xrightarrow{\mathsf{VIS}[T_1]} T$; a trivial consequence of this fact is that $T_2 \xrightarrow{\mathsf{VIS}[T_2]} T$, and since $S = T_2$ there is nothing left to prove.

- $T = T_2$; this case is symmetric to the previous one, and it is therefore omitted.

◀

▶ **Theorem 15.** *Let Φ be a consistency model. Let T_1, T_2 be two transactions, and assume that $(\{T_2\}, \_, \_) \models \textsc{Int}$. If $T_1 \mathrel{\widehat{\sqsubseteq}}_\Phi T_2$, then $T_1 \sqsubseteq_\Phi T_2$.*

Proof. Let $\mathcal{X}$ be a context such that both $\mathcal{X}[T_1]$ and $\mathcal{X}[T_2]$ are defined. In the following we let $\mathcal{X} = (\mathcal{H}_{[]}, \mathsf{AR}, \mathsf{VIS}$. We show that the relevant properties of $\mathcal{X}[T_1]$ are preserved in $\mathcal{X}[T_2]$, by performing a case analysis over Φ:

1. $\Phi = \mathbf{RA}$; suppose that $\mathcal{X}[T_1] \models \textsc{Int}, \textsc{Ext}$. By Lemma 8 it follows that $\mathcal{X}[T_2] \models \textsc{Int}$, while Proposition 10 it follows that $\mathcal{X}[T_2] \models \textsc{Ext}$,
2. $\Phi = \mathbf{CC}$; suppose that $\mathcal{X}[T_1] \models \textsc{Int}, \textsc{Ext}, \textsc{TransVis}$. Since $T_1 \mathrel{\widehat{\sqsubseteq}}_{\mathbf{CC}} T_2$, Lemma 8 and Proposition 12 imply that $\mathcal{X}[T_2] \models \textsc{Int}$, $\mathcal{X}[T_2] \models \textsc{Ext}$, respectively. The fact that $\mathcal{X}[T_2] \models \textsc{TransVis}$ follows directly from $\mathcal{X}[T_1] \models \textsc{TransVis}$,
3. $\Phi = \mathbf{PC}$; suppose that $\mathcal{X}[T_1] \models \textsc{Int}, \textsc{Ext}, \textsc{Prefix}$. Since $\mathcal{X}[T_1] \models \textsc{Prefix}$ and $\mathsf{VIS}[T_1] \subseteq \mathsf{AR}[T_1]$, it is immediate to note that $\mathsf{VIS}[T_1]$ is transitive, hence $\mathcal{X}[T_1] \models \textsc{TransVis}$. Following the case above, we can prove that $\mathcal{X}[T_2] \models \textsc{Int}, \textsc{Ext}$; further, $\mathcal{X}[T_2] \models \textsc{Prefix}$ is a trivial consequence of $\mathcal{X}[T_1] \models \textsc{Prefix}$,
4. $\Phi = \mathbf{PSI}$; suppose that $\mathcal{X}[T_1] \models \textsc{Int}, \textsc{Ext}, \textsc{TransVis}, \textsc{NoConflict}$. We have already shown that in this case $\mathcal{X}[T_2] \models \textsc{Int}, \textsc{Ext}, \textsc{TransVis}$. Also, by Proposition 14 it follows that $\mathcal{X}[T_2] \models \textsc{NoConflict}$,
5. $\Phi = \mathbf{SI}$; suppose that $\mathcal{X}[T_1] \models \textsc{Int}, \textsc{Ext}, \textsc{Prefix}, \textsc{NoConflict}$. We have already proved that in this case $\mathcal{X}[T_2] \models \textsc{Int}, \textsc{Ext}, \textsc{Prefix}, \textsc{NoConflict}$.
6. $\Phi = \mathbf{SER}$; suppose that $\mathcal{X}[T_1] \models \textsc{Int}, \textsc{Ext}, \textsc{TotalVis}$. Since $\mathsf{VIS}[T_1]$ is a total order, it is also reflexive, hence $\mathcal{X}[T_1] \models \textsc{TransVis}$. In this case we know that $\mathcal{X}[T_2] \models \textsc{Int}, \textsc{Ext}$. Further, the totality of $\mathsf{VIS}[T_1]$ implies the totality of $\mathsf{VIS}[T_2]$, or equivalently $\mathcal{X}[T_2] \models \textsc{TotalVis}$.

◀

▶ Proposition 16. Let T_1, T_2 be two transactions such that $(\{T_2\}, \_, \_) \models \textsc{Int}$, and $T_1 \sqsubseteq_\Psi T_2$ for some $\Psi \in \{\mathbf{RA}, \mathbf{CC}, \mathbf{PC}, \mathbf{PSI}, \mathbf{SI}, \mathbf{SER}\}$. If $\neg(T_1 \vdash \mathsf{Read}\ x : n)$, for some object x and value n, then $\neg(T_2 \vdash \mathsf{Read}\ x : n)$.

Proof. If $\neg(T_1 \vdash \mathsf{Read}\ x : n)$, there are two possible cases:

- $T_1 \vdash \mathsf{Read}\ x : m$, for some $m \neq n$. Take the context $\mathcal{X} = (\{S\}_{[]}, \{(S, [])\}, \{(S, [])\}$, where $S \vdash \mathsf{Write}\ x : m$. Then $\mathcal{X}[T_1] \models \Psi$, and $\mathcal{X}[T_2] \models \Psi$. In particular, $\mathcal{X}[T_2] \models \textsc{Ext}$; therefore, either $\neg(T_2 \vdash \mathsf{Read}\ x : \_)$ or $T_2 \vdash \mathsf{Read}\ x : m$. In any case, we have that $\neg(T_2 \vdash \mathsf{Read}\ x : n)$.
- $\neg(T_1 \vdash \mathsf{Read}\ x : \_)$; in this case, choose one arbitrary value $m \neq n$ and take again the context $\mathcal{X} = (\{S\}_{[]}, \{(S, [])\}, \{(S, [])\}$. We have that $\mathcal{X}[T_1] \models \Psi$, hence by hypothesis $\mathcal{X}[T_2] \models \Psi$. In particular, $\mathcal{X}[T_2] \models \textsc{Ext}$, which is possible only if either $T_2 \vdash \mathsf{Read}\ x : m$, or $\neg(T_2 \vdash \mathsf{Read}\ x : \_)$. In both cases, we obtain that $\neg(T_2 \vdash \mathsf{Read}\ x : \_)$.

◀

▶ Proposition 17. Let T_1, T_2 be two transactions such that $T_1 \sqsubseteq_{\mathbf{RA}} T_2$; then $T_1 \vdash \mathsf{Write}\ x : n$ if and only if $T_2 \vdash \mathsf{Write}\ x : n$.

Proof. Suppose that $T_1 \vdash \mathsf{Write}\ x : n$. We consider two sub-cases as follows:

- $\neg(T_1 \vdash \mathsf{Read}\, x : \_)$; in this case, it suffices to consider the context $\mathcal{X} = (\{W, [], R), \mathsf{VIS}, \mathsf{AR})$, where $W \vdash \mathsf{Write}\ x : m$ for some $m \neq n$, $R \vdash \mathsf{Read}\, x : n$, $\mathsf{VIS} = \{(W, R), ([], R)\}$ and $\mathsf{AR} = \{(W, []), ([], R)\}^+$. It is immediate to show that $\mathcal{X}[T_1] \models \mathbf{RA}$, hence $\mathcal{X}[T_2] \models \mathbf{RA}$ by hypothesis. In particular, $\mathcal{X}[T_2] \models \textsc{Ext}$, from which it follows that $M := \max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(R) \cap \mathsf{WEvent}_x) \vdash \mathsf{Write}\ x : n$. Note that $\mathsf{VIS}[T_2]^{-1}(R) = \{W, T_2\}$, hence it is either $M = W$, or $M = T_2$. However, since $W \vdash \mathsf{Write}\ x : m$, and $m \neq n$, it has to be the case that $M = T_2$, and there is nothing left to prove,
- otherwise, suppose that $(T_1 \vdash \mathsf{Read}\, x : m)$ for some m (not necessarily distinct from n); let $\mathcal{X} = (\{W_1, W_2, [], R\}), \mathsf{VIS}, \mathsf{AR})$, where $W_1 \vdash \mathsf{Write}\ x : m$, $W_2 \vdash \mathsf{Write}\ x : l$ for some $l \neq n$, $R \vdash \mathsf{Read}\, x : n$, $\mathsf{VIS} = \{(W_1, []), ([], R), (W_2, R)\}$, and $\mathsf{AR} = \{(W_1, W_2), (W_2, []), ([], R)\}^+$. Note that VIS is not transitive: in particular, $(W_1, []), ([], R)) \in \mathsf{VIS}$, but $(W_1, R) \notin \mathsf{VIS}$. Because $T_1 = \max_{\mathsf{AR}[T_1]}(\mathsf{VIS}[T_1]^{-1}(R) \cap \mathsf{WEvent}_x)$, $T_1 \vdash \mathsf{Write}\ x : n$, $R \vdash \mathsf{Read}\, x : n$, then $\mathcal{X}[T_1] \models \mathbf{RA}$. It follows that $\mathcal{X}[T_2] \models \mathbf{RA}$, and in particular $\mathcal{X}[T_2] \models \textsc{Ext}$. Let $M := \max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(R) \cap \mathsf{WEvent}_x)$; it has to be $M \vdash \mathsf{Write}\ x : n$. Since $\mathsf{VIS}[T_2]^{-1}(R) = \{W_2, T_2\}$ (recall that $\neg(W_1 \xrightarrow{\mathsf{VIS}[T_2]} T_2)$), then either $M = W_2$ or $M = T_2$. But since $W_2 \vdash \mathsf{Write}\ x : l$, and $l \neq m$, it cannot be $M = W_2$. We are left with $M = T_2$, and there is nothing left to prove.

Next, suppose that $\neg(T_1 \vdash \mathsf{Write}\ x : n)$. There are two possibilities:

- $\neg(T_1 \vdash \mathsf{Write}\ x : \_)$; in this case it suffices to consider the context $\mathcal{X} = (\{W, [], R\}, \mathsf{AR}, \mathsf{VIS})$, where $\mathsf{AR} = \mathsf{VIS} = \{(W, []), ([], R)\}^+$, $W \vdash \mathsf{Write}\ x : m$ and $R \vdash \mathsf{Read}\ x : m$ for some $m \neq n$. Since $\neg(T_1 \vdash \mathsf{Write}\ X : \_)$, we have that $\mathcal{X}[T_1] \models \mathbf{RA}$, hence $\mathcal{X}[T_2] \models \mathbf{RA}$, hence $\mathcal{X}[T_2] \models \textsc{Ext}$. In particular, $\max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(R) \cap \mathsf{WEvent}_x) \vdash \mathsf{Write}\ x : m$; now note that $\mathsf{VIS}[T_2]^{-1}(R) = \{W, T_2\}$; if $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$, then there is nothing to prove. Otherwise, we have that $T_2 = \max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1}(R) \cap \mathsf{WEvent}_x)$, hence $T_2 \vdash \mathsf{Write}\ x : m$. Because $m \neq n$, it follows that $\neg(T_2 \vdash \mathsf{Write}\ x : n)$.
- $T_1 \vdash \mathsf{Write}\ x : m$ for some $m \neq n$; we can proceed as in the first part of the proof to show that $T_2 \vdash \mathsf{Write}\ x : m$, hence $\neg(T_2 \vdash \mathsf{Write}\ x : n)$.

◀

▶ Proposition 18. Let T_1, T_2 be two transactions such that $(\{T_2\}, \_, \_) \models \textsc{Int}$, and $T_1 \sqsubseteq_\Psi T_2$. If $\neg(T_1 \vdash \mathsf{Read}\ x : n)$ and $(T_1 \vdash \mathsf{Write}\ x : n)$ for some object x and value n, then $(T_2 \vdash \mathsf{Write}\ x : n)$.

Proof. We have to consider two possible cases; either $(T_1 \vdash \mathsf{Read}\ x : m)$ for some $m \neq n$, or $\neg(T_1 \vdash \mathsf{Read}\ x : \_)$. We consider only the first case, as the second one can be handled similarly.

Suppose then that $(T_1 \vdash \mathsf{Read}\ x : m)$ for some $m \neq n$. Consider the context $\mathcal{X} = (\{S, T\}_{[]}, \mathsf{AR}, \mathsf{VIS})$, where $\mathsf{AR} = \mathsf{VIS} = \{(S, []), ([], T)\}^+$, where $S \vdash \mathsf{Write}\ x : m$ and $T \vdash \mathsf{Read}\ x : n$. It is immediate to note that $\mathcal{X}[T_1] \models \Psi$; in fact $\mathcal{X}[T_1] \models \mathbf{SER}$, hence $\mathcal{X}[T_1] \models \Psi$ for any consistency level Ψ. It follows that $\mathcal{X}[T_2] \models \Psi$. Consider the transaction $W = \max_{AR[T_2]}(\mathsf{VIS}[T_2]^{-1}(T) \cap \mathsf{WEvent}_x)$; this transaction is well defined, and either $W = S$ or $W = T_2$. Further, since $\mathcal{X}[T_2] \models \textsc{Ext}$, then $W \vdash \mathsf{Write}\ x : n$. But we already know that $S \vdash \mathsf{Write}\ x : m$ for some $m \neq n$, so that it has to be $W = T_2$. We have now proved that $T_2 \vdash \mathsf{Write}\ x : n$. ◀

▶ Proposition 19. Let T_1, T_2 be two transactions such that $(\{T_2\}, \_, \_) \models \textsc{Int}$, and $T_1 \sqsubseteq_\Psi T_2$. If $\neg(T_1 \vdash \mathsf{Read}\ x : n)$ and $\neg(T_1 \vdash \mathsf{Write}\ x : n)$ for some object x and value n, then $\neg(T_2 \vdash \mathsf{Write}\ x : n)$.

Proof. First, assume that $\neg(T_1 \vdash \mathsf{Read}\ x : n)$ because $\neg(T_1 \vdash \mathsf{Read}\ x : \_)$. If $\neg(T_1 \vdash \mathsf{Write}\ x : n)$, then there are two possible cases:

- $(T_1 \vdash \mathsf{Write}\ x : m)$ for some value $m \neq n$. In this case, we have that $\neg(T_1 \vdash \mathsf{Read}\ x : m)$ by assumption, hence $(T_2 \vdash \mathsf{Write}\ x : m)$ because of Proposition 18. Since $(T_2 \vdash \mathsf{Write}\ x : m)$ and $m \neq n$, then $\neg(T_1 \vdash \mathsf{Write}\ x : n)$,
- otherwise $\neg(T_1 \vdash \mathsf{Write}\ x : \_)$. In this case, we consider two contexts $\mathcal{X}_1, \mathcal{X}_2$, where for $i = 1, 2$, $\mathcal{X}_i = (\{W_i, R_i\}_{[]}, \mathsf{AR}_i, \mathsf{VIS}_i)$, $\mathsf{AR}_i = \mathsf{VIS}_i = \{(W_i, []), (R_i, [])\}^+$, $W_i \vdash \mathsf{Write}\ x : i$ and $R_i \vdash \mathsf{Read}\ x : i$. Note that both $\mathcal{X}_1[T_1] \models \Psi$ and $\mathcal{X}_2[T_1] \models \Psi$. By hypothesis, it follows that both $\mathcal{X}_1[T_2] \models \Psi$ and $\mathcal{X}_2[T_2] \models \Psi$. In particular, $\mathcal{X}_1[T_2] \models \textsc{Ext}$ implies that either $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$ or $T_2 \vdash \mathsf{Write}\ x : 1$. But $T_2 \vdash \mathsf{Write}\ x : 1$ contradicts the fact that $\mathcal{X}[T_2] \models \textsc{Ext}$, since $T_2 = \max_{AR_2[T_2]}(\mathsf{VIS}_2[T_2]^{-1}(R_2))$ but $\neg(T_2 \vdash \mathsf{Write}\ x : 2)$. Therefore it has to be the case that $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$, and in particular $\neg(T_2 \vdash \mathsf{Write}\ x : n)$.

Now suppose that $T_1 \vdash \mathsf{Read}\ x : m$ for some $m \neq n$. There are three different subcases:

- $T_1 \vdash \mathsf{Write}\ x : m'$ for some $m' \neq m$; note that in this case it also has to be $m' \neq n$, since $\neg(T_1 \vdash \mathsf{Write}\ x : n)$ by hypothesis. Here we have that $\neg(T_1 \vdash \mathsf{Read}\ x : m')$, $T_1 \vdash \mathsf{Write}\ x : m'$, so that it follows from Proposition 18 that $T_2 \vdash \mathsf{Write}\ x : m'$, and in particular $\neg(T_2 \vdash \mathsf{Write}\ x : n)$,
- $T_1 \vdash \mathsf{Write}\ x : m$. Choose the context $(\{W, R\}_{[]}, \mathsf{AR}, \mathsf{VIS})$ where $\mathsf{AR} = \mathsf{VIS} = \{(W, []), ([], R)\}^+$, $W \vdash \mathsf{Write}\ x : m$ and $R \vdash \mathsf{Read}\ x : m$. Note that $\mathcal{X}[T_1] \models \Psi$, so that $\mathcal{X}[T_2] \models \Psi$ by hypothesis. In particular, $\mathcal{X}[T_2] \models \textsc{Ext}$ which is possible if either $T_2 \vdash \mathsf{Write}\ x : m)$ or $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$. In both cases, we obtain that $\neg(T_2 \vdash \mathsf{Write}\ x : n)$,
- $\neg(T_1 \vdash \mathsf{Write}\ x : \_)$; this case is analogous to the previous one.

◀

▶ Proposition 20. Let T_1, T_2 be two transactions such that $(\{T_2\}, \_, \_) \models \textsc{Int}$, and $T_1 \sqsubseteq_\Psi T_2$, for any consistency level Ψ. If $(T_1 \vdash \mathsf{Read}\ x : n)$ and $\neg(T_1 \vdash \mathsf{Write}\ x : \_)$ for some object x and value n, then either $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$ or $(T_2 \vdash \mathsf{Write}\ x : n)$.

Proof. Consider the context $\mathcal{X} = (\{W, R\}_{[]}, \mathsf{AR}, \mathsf{VIS})$, where $\mathsf{AR} = \mathsf{VIS} = \{(W, []), ([], R)\}^+$, $W \vdash \mathsf{Write}\ x : n$ and $R \vdash \mathsf{Read}\ x : n$. Note that $\mathcal{X}[T_1] \models \Psi$, and by hypothesis $\mathcal{X}[T_2] \models \Psi$. Therefore, $\max_{AR[T_2]}(\mathsf{VIS}[T_2]^{-1} \cap \mathsf{WEvent}_x) \vdash \mathsf{Read}\ x : n$. We have $\mathsf{VIS}[T_2]^{-1}(R) = \{W, T_2\}$, and $W \vdash \mathsf{Write}\ x; n$; there are two possibilities:

1. $\max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1} \cap \mathsf{WEvent}_x) = W$; this is possible only if $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$, so that there is nothing left to prove, or
2. $\max_{\mathsf{AR}[T_2]}(\mathsf{VIS}[T_2]^{-1} \cap \mathsf{WEvent}_x) = T_2$, hence $T_2 \vdash \mathsf{Write}\ x : n$.

◀

▶ Proposition 21. Let T_1, T_2 be two transactions such that $(\{T_2\}, \_, \_) \models \textsc{Int}$, and $T_1 \sqsubseteq_\Psi T_2$. If $(T_1 \vdash \mathsf{Read}\ x : n)$ and $T_1 \vdash \mathsf{Write}\ x : n$ for some object x and value n, then either $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$ or $(T_2 \vdash \mathsf{Write}\ x : n)$.

Proof. Identical to the one of Proposition 20 ◀

▶ Proposition 22. Let T_1, T_2 be two transactions such that $(\{T_2\}, \_, \_) \models \textsc{Int}$, and $T_1 \sqsubseteq_\Psi T_2$, where $\Psi \in \{\mathbf{SI}, \mathbf{PSI}\}$. If $\neg(T_1 \vdash \mathsf{Write}\ x : \_)$ then $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$.

Proof. Take the context $\mathcal{X} = (\{W\}_{[]}, \emptyset, \emptyset\}$, where $W \vdash \mathsf{Write}\ x : \_$. Since $\neg(T_1 \vdash \mathsf{Write}\ x : \_)$ it follows that $\mathcal{X}[T_1] \models \textsc{NoConflict}$, and more specifically $\chi[T_1] \models \Psi$. By hypothesis, $\mathcal{X}[T_2] \models \Psi$, hence $\mathcal{X}[T_2] \models \textsc{NoConflict}$. Since there is no visibility edge between W and T_2 in $\mathcal{X}[T_2]$, and $W \vdash \mathsf{Write}\ x : \_$, it has to be the case that $\neg(T_2 \vdash \mathsf{Write}\ x : \_)$. ◀

▶ **Theorem 23.** *Consider a consistency level Ψ. and let T_1, T_2 be transactions such that $(\{T_2\}, \_, \_) \models \textsc{Int}$. If $T_1 \sqsubseteq_\Psi T_2$, then $T_1 \mathrel{\widehat{\sqsubseteq}}_\Psi T_2$.*

Proof. A straightforward consequence of propositions 16, 17, 18, 19, 20, 21 and 22. ◀

B Proof of Theorem 6

B.1 Technical Results for Concrete Executions

In the following, given a partial order $\prec$, we use the symbol $\preceq$ to denote its reflexive closure.

▶ **Lemma 24.** *Let $\{D_r\}_{r\in\mathsf{RId}}$, $\{D'_r\}_{r\in\mathsf{RId}}$ be two RId-indexed collections of sets in LogSet, and let $\mathbf{e}$ be an event such that $((\lambda r.D_r, \_), \_) \xrightarrow{\mathbf{e}} ((\lambda r.D'_r, \_), \_)$; then $D_r \subseteq D'_r$ for any $r \in \mathsf{RId}$.*

Proof. A straightforward case analysis on the proof of the transition $((\lambda r.D_r, \_), \_) \xrightarrow{\mathbf{e}} ((\lambda r.D'_r, \_), \_)$. ◀

▶ **Lemma 25.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution, and suppose that $\mathbf{e}, \mathbf{f} \in \mathbf{E}$ with $\mathbf{e} = (\_, \_, \texttt{commit}(t))$, $\mathbf{f} = (\_, \_, \texttt{commit}(t))$ for some $t \in \mathbb{N}$. Then $\mathbf{e} = \mathbf{f}$.*

Proof. Since $\mathbf{e}, \mathbf{f} \in \mathbf{E}$, either $\mathbf{e} \prec \mathbf{f}$, $\mathbf{f} \prec \mathbf{e}$ or $\mathbf{e} = \mathbf{f}$. We show that the first two cases lead to a contradiction, hence it has to be $\mathbf{e} = \mathbf{f}$.

Suppose that $\mathbf{e} \prec \mathbf{f}$; the case $\mathbf{f} \prec \mathbf{e}$ is analogous, and it will be omitted. There exists a sequence of transitions

$$(R_0, \_) \xrightarrow{\mathbf{e}_1} (R_1, \_) \xrightarrow{\mathbf{e}_2} \cdots \xrightarrow{\mathbf{e}_n} (R_n, \_)$$

such that $\mathbf{e} = \mathbf{e}_i, \mathbf{f} = \mathbf{e}_j$ for some $i, j : 1 \leq i < j \leq n$. Since $\mathbf{e}_i = \mathbf{e} = (\_, \_, \texttt{commit}(t))$, the transition $(R_{i-1}, \_) \xrightarrow{\mathbf{e}_i} (R_i, \_)$ can be derived only via an application of Rule (Commit) from Figure 4; therefore there exists $r \in \mathsf{RId}$ such that $R_i(r) = (D \cup (t : \_), \_)$. By Lemma 27 then **(a)** $(t : \_) \in R_{j-1}(r)$ (recall that $i < j$, hence $i \leq j - 1$). Since $\mathbf{e}_j = \mathbf{f} = (\_, \_, \texttt{commit}(t))$, the transition $(R_{j-1}, \_) \xrightarrow{\mathbf{e}j} (R_j, \_)$ can also be derived via an application of Rule (Commit). This requires that $(t : \_) \notin R_{j-1}(r')$ for any $r' \in \mathsf{RId}$, contradicting **(a)** above. ◀

▶ **Lemma 26.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. If $\mathbf{e}, \mathbf{f}$ are two events in $\mathbf{E}$ such that $\mathsf{TS}_\mathcal{C}(\mathbf{e}), \mathsf{TS}_\mathcal{C}(\mathbf{f})$ are defined and coincide. Then $\mathbf{e} = (\_, r, \_), \mathbf{f} = (\_, r, \_)$ for some $r \in \mathsf{RId}$.*

Proof. By definition, if $\mathsf{TS}_\mathcal{C}(\mathbf{e}) = t$ andt $\mathbf{e} = (\_, r, \_) \in \mathbf{E}$ for some $r \in \mathsf{RId}$ and $t \in \mathbb{N}$, then there exists $\mathbf{h} \in \mathbf{E}$ such that $\mathbf{h} = (\_, r, \texttt{commit}(t))$ and $\mathbf{e} \prec \mathbf{f}$. Simlarly, if $\mathbf{f} = (\_, r', \_)$ and $\mathsf{TS}_\mathcal{C}(\mathbf{f}) = t$, then there exists $\mathbf{h}' \in \mathbf{E}$ such that $\mathbf{h}' = (\_, r', \texttt{commit}(t))$ and $\mathbf{f} \prec \mathbf{h}'$. By Lemma 25, it must be the case that $\mathbf{h} = \mathbf{h}'$, and in particular $r = r'$. ◀

▶ **Lemma 27.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. Let $\mathbf{e} = (\_, r, \_)$ be an event in $\mathbf{E}$, where $r \in \mathsf{RId}$ is an arbitrary replica identifier. Assume that $(R, M) \xrightarrow{\mathbf{e}} (R', M')$, for some $(R, M), (R', M')$. Then $R(r') = R'(r')$ for any $r' \neq r$, .*

Proof. A simple case analysis on the proof of the transition $(R, M) \xrightarrow{\mathbf{e}} (R', M')$ reveals that if $\mathbf{e} = (\_, r, \_)$ then there exists a R'' such that $R = R''[r \mapsto \_]$ and $R' = R''[r \mapsto \_]$. Consequently, for any $r' \neq r$, $R(r') = R''(r') = R'(r')$. ◀

▶ **Lemma 28.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution, with an event $\mathbf{e} = (\_, \_, \texttt{read}(\_, \_))$. If $(R, M) \xrightarrow{\mathbf{e}} (R', M')$ for some $(R, M), (R', M')$, then $R = R'$.*

Proof. This is true because the transition $(R, M) \xrightarrow{\mathbf{e}} (R', M')$ can be obtained only via an application of Rule (Read), which ensures that $R = R'$ ◀

▶ **Lemma 29.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution, and suppose that $\mathbf{e} = (\_, r, \mathbf{o})$ is an event in $\mathbf{E}$ such that $\mathbf{o} \neq \texttt{receive}(\_)$. Then $\mathbf{E}$ contains an event $\mathbf{f} = (\_, r, \texttt{start})$ such that $\mathbf{f} \preceq \mathbf{e}$.*

Proof. Note that if $\mathbf{e} = (\_, r, \texttt{start})$ then the proof of the statement is trivial; therefore, suppose that $\mathbf{e} \neq (\_, \texttt{start}, \_)$.

Let $\mathbf{e}_0, \mathbf{e}_1, \cdots, \mathbf{e}_n$ be the maximal sequence of events in $\mathbf{e}$ such that $\mathbf{e}_n = \mathbf{e}$, and for any $i, j : 0 \leq i, j \leq n, \mathbf{e}_i \prec \mathbf{e}_j$. The maximality of the sequence ensures that we can find $R_0, R_1, \cdots, R_{n+1}$ such that

$$(R_0, \_) \xrightarrow{\mathbf{e}_0} (R_1, \_) \xrightarrow{\mathbf{e}_1} \cdots \xrightarrow{\mathbf{e}_{n-1}} (R_n, \_) \xrightarrow{\mathbf{e}_n} (R_{n+1}, \_)$$

where we recall that $R_0 = \lambda r \in \mathsf{RId}.(\emptyset, \mathsf{idle})$, and in particular $R_0(r) = (\_, \mathsf{idle})$. Since $\mathbf{e}_n = (\_, r, \mathbf{o})$ and $\mathbf{o} \notin \{\texttt{start}, \texttt{receive}(t : \rho) \mid t \in \mathbb{N}, \rho \in \mathsf{UpdateList}\}$, a simple case analysis on the rules of Figure 4 reveals that $R_n(r) = (\_, \rho)$ for some $\rho \neq \mathsf{idle}$. Therefore, there exists an index $i : 0 \leq i < n$ such that $R_i(r) = (\_, \mathsf{idle})$ and $R_{i+1}(r) = (\_, \rho')$ for some $\rho' \neq \mathsf{idle}$. Another case analysis on the rules of Figure 4 reveal that the transition $(R_i, \_) \xrightarrow{\mathbf{e}_i} (R_{i+1}, \_)$ can be derived only via Rule (Start), hence $\mathbf{e}_i = (\_, r, \texttt{start})$. ◀

▶ **Definition 30.** Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. For any event $\mathbf{e} \in \mathbf{E}$ such that $\mathbf{e} = (\_, r, \mathbf{o})$ and $\mathbf{o} \neq \texttt{receive}(\_)$, the event $\texttt{start}_{\mathcal{C}}(\mathbf{e})$ is defined as $\max_{\preceq}(\mathbf{f} \in \mathbf{E} \mid \mathbf{f} \preceq \mathbf{e} \wedge \mathbf{f} = (\_, r, \texttt{start})$.

Also, if there exists an event $\mathbf{e} = (\_, r, \texttt{commit}(t)) \in \mathbf{E}$. then we let $\texttt{start}_{\mathcal{C}}(t) = \texttt{start}_{\mathcal{C}}(\mathbf{e})$.

▶ **Lemma 31.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. Let $\mathbf{e} = (\_, r, \mathbf{o})$ be an event in $\mathbf{E}$ such that $\mathbf{o} \neq \texttt{receive}(\_)$ and $r \in \mathsf{RId}$. Then the event $\texttt{start}_{\mathcal{C}}(\mathbf{e})$ is well-defined, and whenever $\texttt{start}_{\mathcal{C}}(\mathbf{e}) \prec \mathbf{f} \prec \mathbf{e}$ for some $\mathbf{f} = (\_, r, \mathbf{o}')$, then $\mathbf{o}' \notin \{\texttt{start}, \texttt{abort}, \texttt{commit}(t), \texttt{receive}(t : \rho) \mid t \in \mathbb{N}, \rho \in \mathsf{UpdateList}\}$.*

Proof. The fact that $\texttt{start}_{\mathcal{C}}(\mathbf{e})$ is well-defined is an immediate consequence of Lemma 29. Next, suppose that $\mathbf{f} = (\_, r, \mathbf{o}')$ is an event in $\mathbf{f} \in \mathbf{E}$ such that $\texttt{start}_{\mathcal{C}}(\mathbf{e}) \prec \mathbf{f} \prec \mathbf{e}$ and $\mathbf{f} = (\_, r, \mathbf{o}')$. Clearly it cannot be $\mathbf{o}' = \texttt{start}$, since this would contradict the definition of $\texttt{start}_{\mathcal{C}}(\mathbf{e})$.

We have the sequence of transitions

$$(R_1, \_) \xrightarrow{\mathbf{e}_1} (R_2, \_) \xrightarrow{\cdots \mathbf{e}_n} (R_{n+1}, \_)$$

such that $\mathbf{e}_1 = \texttt{start}_{\mathcal{C}}(\mathbf{e})$, $\mathbf{e}_i = \mathbf{f}$ for some $1 < i < n$, and $\mathbf{e}_n = \mathbf{e}$. Now note that if it were $\mathbf{o}' = \texttt{commit}(\_)$, then $R_{i+1}(r) = (\_, \mathsf{idle})$; by using the same argument of the proof of Lemma 29, we may conclude that there exists an index $j : i < j < n$ such that $\mathbf{e}_j = (\_, r, \texttt{start})$, contradicting the definition of $\texttt{start}_{\mathcal{C}}(\mathbf{e})$. We have proved that whenever $\mathbf{f} = (\_, r, \mathbf{o}')$ and $\texttt{start}_{\mathcal{C}}(\mathbf{e}) \prec \mathbf{f} \prec \mathbf{e}$ then $\mathbf{o}' \notin \{\texttt{commit}(t), \mid t \in \mathbb{N}\}$. A similar argument can be used to prove that $\mathbf{o}' \neq \texttt{abort}, \mathbf{o}' \neq \texttt{receive}(\_)$. ◀

▶ **Lemma 32.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. Let $\mathbf{e} = (\_, r, \mathbf{o})$ be an event in $\mathbf{E}$ such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t$ Then the event $\mathtt{start}_{\mathcal{C}}(t)$ is well-defined.*

Proof. It suffices to note that, since $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t$, there exists an event $\mathbf{f} = (\_, r, \mathtt{commit}(t))$ such that $\mathbf{e} \prec \mathbf{f}$, and whenever $\mathbf{g}$ is an event such that $\mathbf{e} \prec \mathbf{g} \prec \mathbf{f}$, then $\mathbf{g} \neq (\_, r, \mathtt{commit}(\_))$. By Lemma 25, whenever $\mathbf{f}' = (\_, r, \mathtt{commit}(t))$ for some $\mathbf{f}' \in \mathbf{E}$, then $\mathbf{f} = \mathbf{g}$. Therefore, $\mathtt{start}_{\mathcal{C}}(t)$ is well-defined and we have that $\mathtt{start}_{\mathcal{C}}(t) = \mathtt{start}_{\mathcal{C}}(\mathbf{f})$. ◀

▶ **Lemma 33.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. For any two events $\mathbf{e}, \mathbf{f}$ such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = \mathsf{TS}_{\mathcal{C}}(\mathbf{f}) = t$, then $\mathtt{start}_{\mathcal{C}}(\mathbf{e}) = \mathtt{start}_{\mathcal{C}}(\mathbf{f}) = \mathtt{start}_{\mathcal{C}}(t)$.*

Proof. Let $\mathbf{e} = (\_, r, \_)$, $\mathbf{f} = (\_, r, \_)$. We can assume that $\mathbf{e}, \mathbf{f}$ occur at the same replica because of the hypothesis $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = \mathsf{TS}_{\mathcal{C}}(\mathbf{f})$ and because of Lemma 26.

First note that if $\mathsf{TS}_{\mathcal{C}}(\mathbf{e})$ is defined and equal to t, then there exists an event $\mathbf{g} = (\_, \_, \mathtt{commit}(t))$ which occurs in $\mathcal{C}$, such that $\mathbf{e} \prec \mathbf{g}$, and whenever $\mathbf{e} \prec \mathbf{e}' \prec \mathbf{g}$ then $\mathbf{e}' \neq (\_, r, \mathbf{o})$ where $\mathbf{o} \in \{\mathtt{abort}, \mathtt{commit}(t') \mid t' \in \mathbb{N}\}$. Therefore the event $\mathtt{start}_{\mathcal{C}}(t) = \max_{\prec}\{\mathbf{h} \prec \mathbf{g} \mid \mathbf{h} = (\_, r, \mathtt{start})\}$ is well-defined by lemmas 29 and 25.

Next, we show that there exists no event $\mathbf{h}' \in \mathcal{C}$ such that $\mathbf{e} \prec \mathbf{h}' \prec \mathit{ccg}$ and $\mathbf{h}' = (\_, r, \mathtt{start})$. The proof of this statement is carried out by contradiction: let then $\mathbf{h}'$ be an event in $\mathbf{E}$ as described above. Its existence ensures that we can derive a sequence of transitions

$$(R, \_) \xrightarrow{\mathbf{e}} (R_1, \_) \xrightarrow{\mathbf{e}_1} \cdots \xrightarrow{\mathbf{e}_n} (R_2, \_) \xrightarrow{\mathbf{h}'} (R_3, \_)$$

Since $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t$, it means that $\mathbf{e} = (\_, r, \mathbf{o})$ for some $\mathbf{o} \in \{\mathtt{read}(x, n), \mathtt{write}(x, n) \mid x \in \mathsf{Obj}, n \in \mathbb{N}\}$. A case analysis on the derivation of the transition $(R, \_) \xrightarrow{\mathbf{e}} (R_1, \_)$ reveals that it has to be the case that $R_1(r) \neq (\_, \mathsf{idle})$.

However, since $\mathbf{h} = (\_, r, \mathtt{start})$, then the transition $(R_2, \_) \xrightarrow{\mathbf{h}'} (R_3, \_)$ can be derived only if $R_2(r) = (\_, \mathsf{idle})$. Thus, the sequence of transitions

$$(R_1, \_) \xrightarrow{\mathbf{e}_1} \cdots \xrightarrow{\mathbf{e}_n} (R_2, \_)$$

includes a transition $(R', \_) \xrightarrow{\mathbf{e}_k} (R'', \_)$, where $1 \leq k \leq n$, such that $R'(r) \neq \mathsf{idle}$ and $R''(r) = \mathsf{idle}$. This is possible only if $\mathbf{e}_k = (\_, r, \mathbf{o}')$ with $\mathbf{o}' \in \{\mathtt{abort}, \mathtt{commit}(t') \mid t' \in T\}$. Define $\mathbf{g}' := \mathbf{e}_k$.

If $\mathbf{g}' = (\_, r, \mathtt{abort})$ then we obtain a contradiction, since in this case we have $\mathbf{e} \prec \mathbf{g}' \prec \mathbf{g}$, contradicting the definition of $\mathsf{TS}_{\mathcal{C}}(\mathbf{g})$ according to which there are no event of the form $(\_, r, \mathtt{abort})$ between $\mathbf{e}$ and $\mathbf{g}$.

If $\mathbf{g}' = (\_, r, \mathtt{commit}(t')$ then we also obtain a contradiction; in fact, in this case we have that $\mathbf{e} \prec \mathbf{g}' \prec \mathbf{h}' \prec \mathbf{g}$, hence $\mathbf{g} \neq \mathbf{g}'$. Since $\mathsf{TS}_{\mathcal{C}} = t$, it has to be the case that there exists an event $\mathbf{g}''$ such that $\mathbf{e} \prec \mathbf{g}'' \preceq \mathbf{g}'$ and $\mathbf{g}'' = (\_, r, \mathtt{commit}(t))$. By Lemma 25, this contradicts the hypothesis that $\mathbf{g} = (\_, r, \mathtt{commit}(t))$.

This can be done in a way analogous to the proof of Lemma 29. It can be shown that the existence of such a $\mathbf{h}'$ implies the existence of an event $\mathbf{g}' \in \{\mathtt{abort}, \mathtt{commit}(t') \mid t' \in \mathbb{N}\}$, such that $\mathbf{e} \prec \mathbf{g}' \prec \mathbf{h}'$. This gives in turn $\mathbf{e} \prec \mathbf{g}' \prec\prec \mathbf{g}$, contradicting the hypothesis that there exists no commit or abort event at replica r, between $\mathbf{e}$ and $\mathbf{g}$ (this assumption is given by the fact that $\mathsf{TS}_{\mathbf{e}} = \mathbf{g}$ and that $\mathbf{g}$ is the unique event in $\mathbf{E}$ such that $\mathbf{g} = (\_, \_, \mathtt{commit}(t))$, cf Lemma 25).

Since there exists no event $\mathbf{h}'$ such that $\mathbf{e} \prec \mathbf{h}' \prec \mathbf{g}$, and $\mathbf{h} = (\_, r, \mathtt{start})$, we obtain that $\mathtt{start}_{\mathcal{C}}(\mathbf{e}) = \max_{\prec}(\mathbf{h} \prec \mathbf{e} \mid \mathbf{h} = (\_, r, \mathtt{start})\} = \max_{\prec}(\mathbf{h} \prec \mathbf{g} \mid \mathbf{h} = (\_, r, \mathtt{start})\} =$

$\mathsf{TS}_{\mathcal{C}}(\mathbf{g}) = \mathsf{TS}_{\mathcal{C}}(t)$. We can repeat this kind of reasoning for the event $\mathbf{f}$, showing that $\mathtt{start}_{\mathcal{C}}(\mathbf{f}) = \mathsf{TS}_{\mathcal{C}}(t)$. At this point it is immediate to note that $\mathtt{start}_{\mathcal{C}}(\mathbf{e}) = \mathtt{start}_{\mathcal{C}}(\mathbf{f})$. ◀

▶ **Lemma 34.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. Let $t \in \mathbb{N}$, $\mathbf{e} \in \mathbf{E}$ be such that $\mathbf{e} = (\_, r, \mathbf{o})$ with $\mathbf{o} \in \{\mathtt{write}(x, n), \mathtt{read}(x, n) \mid x \in \mathsf{Obj}, n \in \mathbb{N}\}$, and $\mathtt{start}_{\mathcal{C}}(\mathbf{e}) = \mathtt{start}_{\mathcal{C}}(t)$. Then $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t$.*

Proof. By definition there exists an event $\mathbf{f} = (\_, r, \mathtt{commit})$ for some r, such that $\mathtt{start}_{\mathcal{C}}(t) = \max_{\prec}\{\mathbf{g} \prec \mathbf{e} \mid \mathbf{g} = (\_, r, \mathtt{start})\}$. Since $\mathtt{start}_{\mathcal{C}}(t) = \mathtt{start}_{\mathcal{C}}(e)$, we have that $\mathtt{start}_{\mathcal{C}}(t) \preceq \mathbf{e}$.

There are three possible cases: $\mathbf{f} \prec \mathbf{e}, \mathbf{e} \prec \mathbf{f}$ and $\mathbf{e} = \mathbf{f}$. The case $\mathbf{e} = \mathbf{f}$ is not possible, since $\mathbf{f} = (\_, r, \mathtt{commit})$ and $\mathbf{e} = (\_, r, \mathbf{o})$, where $\mathbf{o} \in \{\mathtt{read}(x, n), \mathtt{write}(x, n) \mid x \in \mathsf{Obj}, n \in \mathbb{N}\}$.

Suppose that $\mathbf{f} \prec \mathbf{e}$, then we would have a sequence of transitions

$$(R, \_) \xrightarrow{\mathtt{start}_{\mathcal{C}}(t)} (R_1, \_) \xrightarrow{\mathbf{e}_1} \cdots \xrightarrow{\mathbf{e}_n} (R_{n+1}, \_) \xrightarrow{\mathbf{f}} (R', \_) \xrightarrow{\mathbf{f}_1} \cdots \xrightarrow{\mathbf{f}_k} (R'_k, \_) \xrightarrow{\mathbf{e}} (R'', \_)$$

A simple case analysis over the proof of the transitions $(R_{n+1}, \_) \xrightarrow{\mathbf{f}} (R', \_)$ and $(R'_k, \_) \xrightarrow{\mathbf{e}} (R'', \_)$ proves that $R'(r) = (\_, \mathsf{idle})$ and $R''(r) = (\_, \rho)$ for some $\rho \neq \mathsf{idle}$. This is possible only if there exists an index $h < k$ such that $\mathbf{f}'_h = (\_, r, \mathtt{start})$, contradicting the hypothesis that $\mathtt{start}_{\mathcal{C}}(\mathbf{e}) = \mathtt{start}_{\mathcal{C}}(t)$. Therefore it cannot be $\mathbf{f} \prec \mathbf{e}$.

We are left with the only possibility $\mathbf{e} \prec \mathbf{f}$. A similar analysis, reveals that there exist no event $\mathbf{g} = (\_, r, \mathbf{o}')$ such that $\mathbf{o}' \in \{\mathtt{abort}, \mathtt{commit}(t') \mid t' \in \mathbb{N}\}$ and $\mathbf{e} \prec \mathbf{h} \prec \mathbf{f}$. At least intuitively, the existence of such a $\mathbf{h}$ implies that the state of replica r in $\mathcal{C}$, immediately after the event $\mathbf{g}$ has been performed, has the form $(\_, \mathsf{idle})$, while the state of the same replica before executing $\mathbf{f}$ has the form $(\_, \rho)$, where $\rho \neq \mathsf{idle}$. This is possible only if there exists an event $\mathbf{h} = (\_, r, \mathtt{start})$ such that $\mathbf{g} \prec \mathbf{h} \prec \mathbf{f}$; since $\mathbf{e} \prec \mathbf{g}$, this contradicts the assumption that $\mathtt{start}_{\mathcal{C}}(\mathbf{e}) = \mathtt{start}_{\mathcal{C}}(\mathbf{f})$.

It remains to apply the definition of $\mathsf{TS}_{\mathcal{C}}(\cdot)$ to prove that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e})$ is defined and equal to t. ◀

▶ **Lemma 35.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. For any events $\mathbf{e}, \mathbf{f}, \mathbf{g} \in \mathbf{E}$ and replica r such that $\mathbf{e} = (\_, r, \_)$, $\mathbf{f} = (\_, r, \_)$, $\mathbf{g} = (\_, r, \_)$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = \mathsf{TS}_{\mathcal{C}} = t$. Then $\mathsf{TS}_{\mathcal{C}}(\mathbf{g}) = t$.*

Proof. By Lemma 33 we know that $\mathtt{start}_{\mathcal{C}}(\mathbf{e}) = \mathtt{start}_{\mathcal{C}}(\mathbf{f}) = \mathtt{start}_{\mathcal{C}}(t)$. It is easy to note that in this case it has to be $\mathtt{start}_{\mathcal{C}}(\mathbf{g}) = \mathtt{start}_{\mathcal{C}}(t)$, and by Lemma 34 it follows that $\mathsf{TS}_{\mathcal{C}}(\mathbf{g}) = t$. ◀

▶ **Definition 36.** Let $\mathcal{C}$ be a concrete execution. Given a (possibly empty) finite sequence of events $\sigma = \mathbf{e}_1 \cdots \mathbf{e}_n$ and a timestamp t, we define the *transaction log of t* for σ in $\mathcal{C}$ by letting $\mathsf{LogOf}_{\mathcal{C}}(t, \sigma) = \mathsf{idle}$ if the event $\mathtt{start}_{\mathcal{C}}(t)$ does not occur in σ, otherwise it is defined as $t : \mathsf{LogAux}_{\mathcal{C}}(t, \sigma)$, where

$$\mathsf{LogAux}_{\mathcal{C}}(t, \sigma) = \begin{cases} \varepsilon & \text{if } \sigma = \varepsilon \\ \mathsf{LogAux}_{\mathcal{C}}(t, \sigma') \cdot \mathtt{write}(x, n) & \text{if } \sigma = \sigma' \cdot \mathbf{e} \text{ where} \\ & \mathbf{e} = (\_, \_, \mathtt{write}(x, n)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t \\ \mathsf{LogAux}_{\mathcal{C}}(t, \sigma') & \text{if } \sigma = \sigma' \cdot \mathbf{e} \text{ where} \\ & \neg(\mathbf{e} = (\_, \_, \mathtt{write}(x, n)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t) \end{cases}$$

▶ **Lemma 37.** *Let $\mathcal{C}$ be a concrete execution, $t \in \mathbb{N}$ and σ be a sequence of events such that $\texttt{start}_{\mathcal{C}}(t)$ occurs in σ. Suppose that $\texttt{start}_{\mathcal{C}}(t) = (\_, r, \_)$ and let $\mathbf{e} = (\_, r', \_)$ for some $r' \neq r$. Then $\mathsf{LogOf}_{\mathcal{C}}(t, \sigma) = \mathsf{LogOf}_{\mathcal{C}}(t, \sigma \cdot \mathbf{e})$.*

Proof. First note, that if either $\mathsf{TS}_{\mathcal{C}}(\mathbf{e})$ is undefined or different from t, then the statement follows immediately from Definition 36.

Suppose then that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t$. By Lemma 33 we know that $\texttt{start}_{\mathcal{C}}(\mathbf{e}) = \texttt{start}_{\mathcal{C}}(t) = (\_, r, \texttt{start})$. By Definition, we also have that $\texttt{start}_{\mathcal{C}}(\mathbf{e}) = \max_{\prec}(\mathbf{g} \prec \mathbf{e} \mid \mathbf{g} = (\_, r', \texttt{start})\}$, from which it follows that $r' = r$. Contradiction. ◀

▶ **Lemma 38.** *Let*

$$(R_0, M_0) \xrightarrow{\mathbf{e}_1} (R_1, M_1) \xrightarrow{\mathbf{e}_2} \cdots$$

such that $\mathcal{C} = (\{e_i\}_{i \in \mathbb{N}_0}, \{(\mathbf{e}_i, \mathbf{e}_j) \mid i < j\})$ is a concrete execution. For any $i = 1, \cdots$ and replica $r \in \mathsf{RId}$ such that $R_i(r) = (\_, \rho_r)$, where $\rho_r \neq \mathsf{idle}$, and $\texttt{start}_{\mathcal{C}}(t_r) = \max_{j \leq i}\{\mathbf{e}_j = (\_, r, \texttt{start})\}$. Then $t_r : \rho_r = \mathsf{LogOf}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_i)$.

Proof. By induction on i. If $i = 0$, then for any replica r, $R_0(r) = (\_, \mathsf{idle})$ (cf. Definition 5), hence this case is vacuous.

Let then $i > 0$, and suppose that the statement holds for $i - 1$. Note that if there exists no index $j \leq i$ such that $\texttt{start}_{\mathcal{C}}(t_r) \neq \mathbf{e}_j$, then the statement is vacuous. Assume then that $\texttt{start}_{\mathcal{C}}(t_r) = \mathbf{e}_j$ for some $j \leq i$.

We perform a case analysis on the event $\mathbf{e}_i$.

- Assume first that $\mathbf{e}_i = (\_, r', \_)$ for some $r' \neq r$; in this case it cannot be $\texttt{start}_{\mathcal{C}}(t_r) = \mathbf{e}_i$, since $\texttt{start}_{\mathcal{C}}(t_r) = (\_, r, \texttt{start})$ and $r' \neq r$. Thus, $\texttt{start}_{\mathcal{C}}(t_r) = \mathbf{e}_j$ for some $j < i$. By Lemma 27 we know that $R_{i-1}(r) = R_i(r)$; In particular, since $R_i(r) = (\_, \rho_r)$, then $R_{i-1}(r) = (\_, \rho_r)$; by inductive hypothesis $\mathsf{LogOf}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_{i-1}) = t_r : \rho_r$. Finally, since $r \neq r'$, and $\texttt{start}_{\mathcal{C}}(t_r) = (\_, r, \texttt{start})$, we can apply Lemma 37, from which it follows that $\mathsf{LogOf}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_i) = \mathsf{LogOf}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_{i-1}) = t_r : \rho_r$.
  Since $\mathbf{e}_i = (\_, r', \_)$, it cannot be that $\mathsf{TS}(\mathbf{e}_i) = t_r$. In fact, if it were $\mathsf{TS}_{\mathbf{e}_i} = t_r$, then it would be the case that $\texttt{start}_{\mathcal{C}}(t_r) = \texttt{start}_{cexec}(\mathbf{e}_i) = (\_, r', \texttt{start})$, contradicting the hypothesis that $\texttt{start}_{\mathcal{C}}(t_r) = (\_, r, \texttt{start})$.
- Suppose that $\mathbf{e}_i = (, r, \texttt{start})$. By hypothesis, it follows that $\mathbf{e}_i = \texttt{start}_{\mathcal{C}}(t_r)$. A simple case analysis on the proof of the transition $(R_{i-1}, \_) \xrightarrow{\mathbf{e}_i} (R_i, \_)$ reveals that $R_i(r) = (\_, \varepsilon)$. Hence $\rho_r = \varepsilon$. In thix case there exists no event $\mathbf{e}_j$ such that $j < i$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) = t_r$. In fact, if such an event $\mathbf{e}_j$ existed, then by Lemma 29 we would have that $\texttt{start}_{\mathcal{C}}(t_r) = \texttt{start}_{\mathcal{C}}(\mathbf{e}_j) \prec \texttt{start}_{\mathcal{C}}(\mathbf{e}_i) = \texttt{start}_{\mathcal{C}}(t_r)$, leading to a contradiction.
  Since the event $\texttt{start}_{\mathcal{C}}(t_r)$ occurs in $\mathbf{e}_1 \cdots \mathbf{e}_i$, by Definition we have that $\mathsf{LogOf}_{\mathcal{C}}(t_r, \mathbf{e}_1, \cdots, \mathbf{e}_i) = t_r : \mathsf{LogAux}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_i)$. Using the fact that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) \neq t_r$ for any $j < i$, it is immediate to note that $\mathsf{LogAux}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_i) = \varepsilon$.
- Assume that $\mathbf{e}_i = (\_, r, \texttt{write}(x, n))$. Since $\mathbf{e}_i \neq (\_, r, \texttt{start})$, it has to be the case that $\texttt{start}_{\mathcal{C}}(t) = \mathbf{e}_j$ for some $j < i$. Thus we can apply the inductive hypothesis, and infer that $R_{i-1}(r) = \mathsf{LogAux}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_{i-1})$. By Performing a case analysis over the proof of the transition $(R_{i-1}, \_) \xrightarrow{\mathbf{e}_i} (R_i, \_)$, we get that $R_i(r) = (\_, \mathsf{LogAux}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_{i-1}) \cdots \texttt{write}(x, n))$. Therefore, in this case $\rho_r = \mathsf{LogAux}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_{i-1}) \cdots \texttt{write}(x, n)$.
  By Lemma 34 it has to be the case that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i) = t_r$. By Definition, $\mathsf{LogOf}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_i) = t_r : \mathsf{LogAux}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_i) = t_r : (\mathsf{LogAux}_{\mathcal{C}}(t_r, \mathbf{e}_1 \cdots \mathbf{e}_{i-1}) \cdot (\texttt{write}(x, n)) = t_r : \rho_r$.

- The case $\mathbf{e}_i \triangleright t_r : \texttt{read}(x, n)$ @ r is analogous to the previous one, where Lemma 28 is used to prove that $R_{i-1}(r) = R_i(r)$.
- The remaining cases, $\mathbf{e}_i \triangleright t_r : \{\texttt{abort}, \texttt{commit}, \texttt{receive}(\_)\}$ @ r, are vacuous, since an analysis on the proof of the transition $(R_{i-1}, M_{i-1}) \xrightarrow{\mathbf{e}_i} (R_i, M_i)$ reveals that $R_i(r) = (\_, \mathsf{idle})$.

◀

▶ **Lemma 39.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. For any $\mathbf{e} = (\_, \_\texttt{receive}(t : \_))$ there exists $\mathbf{f} \prec \mathbf{e}$ such that $\mathbf{f} = (\_, \_, \texttt{commit}(t))$.*

Proof. If $\mathbf{e} = (\_, \_, \texttt{receive}(t : \_))$ appears in $\mathbf{E}$, then there exists a sequence of transitions

$$(R_0, M_0) \xrightarrow{\mathbf{e}_0} (R_1, M_1) \xrightarrow{\mathbf{e}_1} \cdots \xrightarrow{\mathbf{e}_n} (R_{n+1}, M_{n+1})$$

where $R_0(r) = \lambda r.(\emptyset, \mathsf{idle})$, $M_0 = \emptyset$ and $\mathbf{e}_n = \mathbf{e}$. An inspection of the derivation $(R_n, M_n) \xrightarrow{\mathbf{e}} (R_{n+1}, M_{n+1})$ reveals that $t : \rho \in M_n$ for some ρ. Since $M_0 = \emptyset$, there exists an index $i < n$ such that $t : \rho \notin M_i, t : \rho \in M_{i+1}$. This is possible only if $\mathbf{e}_i = (\_, \_, \texttt{commit}(t))$. ◀

▶ **Lemma 40.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. For any $\mathbf{e} = (\_, r, \texttt{receive}(t : \_)) \in \mathbf{E}$ and $\mathbf{f} = (\_, r, \texttt{start})$ such that $\mathbf{f} \prec \mathbf{e}$, there exists $\mathbf{g} \in \mathbf{E}$ such that $\mathbf{f} \prec \mathbf{g} \prec \mathbf{e}$ and $\mathbf{g} = (\_, r, \mathbf{o})$ for some $\mathbf{o} \in \{\texttt{abort}, \texttt{commit}(t') \mid t' \in \mathbb{N}\}$.*

Outline. The proof is similar in style to the previous ones. It suffices to note that, since $\mathbf{e} = (\_, r, \texttt{receive}(t : \_))$, then the state of replica r in $\mathcal{C}$ before performing $\mathbf{e}$ has the form $(\_, \mathsf{idle})$. On the other hand, the state of the same replica after performing the event $\mathbf{f}$ has the form $(\_, \rho)$, where $\rho \in \mathsf{UpdateList}$. This is possible only if there is an event $\mathbf{g}$ occurring between $\mathbf{e}$ and $\mathbf{f}$, which causes the state of replica r to evolve from a value in $\mathsf{UpdateList}$ to idle. But then this means that $\mathbf{g}$ corresponds to either an $\texttt{abort}$ or a $\texttt{commit}(\_)$ operation. ◀

▶ **Lemma 41.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution; let also $\mathbf{e}, \mathbf{f} \in \mathbf{E}$ be such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = \mathsf{TS}_{\mathcal{C}}(\mathbf{f}) = t$. If $\mathbf{g} \in \mathbf{E}$ is such that $\mathbf{g} = (\_, r, \mathbf{o})$ and $\mathbf{e} \preceq \mathbf{g} \prec f$, then $\mathbf{o}_{\mathbf{g}} \in \{\texttt{read}(\_, \_), \texttt{write}(\_, \_)\}$.*

Proof. An immediate consequence of Lemma 35 and the Definition of $\mathsf{TS}_{\mathcal{C}}(\cdot)$. ◀

▶ **Definition 42.** For any list of records ρ we let

$$\mathsf{lastval}(x, \rho) = \begin{cases} \textbf{undefined} & \text{if } \rho = \varepsilon \\ n & \text{if } \rho = \rho' \cdot \texttt{write}(x, n) \\ \mathsf{lastval}(x, \rho') & \text{if } \rho = \rho' \cdot \texttt{write}(y, \_) \text{ and } y \neq x \end{cases}$$

We lift this notation to transaction logs $t : \rho$ by letting $\mathsf{lastval}(x, t : \rho) = \mathsf{lastval}(x, \rho)$.

For a set of transaction logs D such that whenever $t : \rho, t' : \rho' \in D$ and $t = t'$ then $\rho = \rho'$, we define $\mathsf{lastval}(x, D) = n$ if there exists $t : \rho \in D$ such that $\mathsf{lastval}(x, t : \rho) = n$, and $t = \max_<\{t' : \rho' \in D \mid \mathsf{lastval}(x, t : \rho') \neq \textbf{undefined}\}$; otherwise, we let $\mathsf{lastval}(x, D) = 0$.

▶ **Lemma 43.** *Consider a (possibly infinite) computation*

$$(R_0, M_0) \xrightarrow{\mathbf{e}_1} (R_1, M_1) \xrightarrow{\mathbf{e}_2} \cdots$$

such that $\mathcal{C} = (\{e_i\}_{i \in \mathbb{N}_0}, \{(\mathbf{e}_i, \mathbf{e}_j) \mid i < j\})$ is a concrete execution. Let $i \geq 0$ and $r \in \mathsf{RId}$ such that $R_i(r) \neq (\_, \mathsf{idle})$, and choose D, ρ such that $R_i(r) = (D, \rho)$. Then

$$\mathsf{lastval}(x, D \cup \{\infty : \rho\}) = \begin{cases} \mathsf{lastval}(x,\rho) & \textit{if } \mathsf{lastval}(x,\rho) \neq \textbf{undefined} \\ \mathsf{lastval}(x, D) & \textit{otherwise} \end{cases}$$

Proof. Immediate from the definition of $\mathsf{lastval}(x, D)$. ◀

▶ **Lemma 44.** *Consider a (possibly infinite) computation*

$$(R_0, M_0) \xrightarrow{\mathbf{e}_1} (R_1, M_1) \xrightarrow{\mathbf{e}_2} \cdots$$

such that the execution $\mathcal{C} = (\mathbf{E}, \prec)$, where $\mathbf{E} = (\{e_i\}_{i\in\mathbb{N}_0}, \prec = \{(\mathbf{e}_i, \mathbf{e}_j) \mid i < j\})$ is a concrete execution. Let $\mathbf{e}_i$ be such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i) = t$ for some $t \in \mathbb{N}$, and $\mathbf{e}_i = (\_, r, \mathtt{read}(x, n))$ for some $r \in \mathsf{RId}$. Assume that $R_i(r) = (\_, \rho)$. If $\mathsf{lastval}(x, t : \rho) = n$, the event $\mathbf{f} := \max_{\prec}\{\mathbf{e}' \mid \mathbf{e}' \prec \mathbf{e}_i \wedge \mathbf{e}' = (\_, r, \mathtt{write}(x, \_) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}') = t\}$ is defined, and $\mathbf{f} = (\_, r, \mathtt{write}(x, n))$.

Proof. First note that, if $\mathsf{lastval}(x, \rho) = n$, then $\rho = rho' \cdot \mathtt{write}(x, n) \cdot \rho''$, for some ρ', ρ'' such that no record of the form $\mathtt{write}(x, \_)$ appears in ρ''.

By Lemma 38 we know that $t : \rho = \mathsf{LogOf}_{\mathcal{C}}(t, \mathbf{e}_1, \cdots, \mathbf{e}_i) = \rho' \cdot \mathtt{write}(x, n) \cdot \rho''$. By the definition of $\mathsf{LogOf}_{\mathcal{C}}(\cdot)$ there exists an index j such that $\mathbf{e}_j = (\_, \_, \mathtt{write}(x, n))$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) = t$, and whenever $\mathbf{e}_k = (\_, r, \mathtt{write}(x, \_)$ for some $j < k$, then we also have that $k < i$. In fact, it cannot be $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_k) = t$, since otherwise we would have that $\mathtt{write}(x, \_)$ occurs in ρ''. Then, by Lemma 35 and the fact that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) = \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i) = t$, it follows that $\neg(j < k < i)$. Since $j < k$, either it has to be $\mathbf{e}_j = \mathbf{e}_k$ or $\mathbf{e}_i \prec \mathbf{e}_k$. Since $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) \neq \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_k)$, it has to be $\mathbf{e}_i \prec \mathbf{e}_k$.

We have proved that there exists an event $\mathbf{e}_j \prec \mathbf{e}_i$ such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) = t$, $\mathbf{e}_j = (\_, \_, \mathtt{write}(x, n))$ and whenever $\mathbf{f}$ is such that $\mathsf{TS}(\mathbf{f}) = t, \mathbf{f} = (\_, \_, \mathtt{write}(x, \_))$, then $\mathbf{e}_i \prec \mathbf{e}_f$. By definition, $\mathbf{e}_j = \max_{\prec}(\mathbf{e}' \prec \mathbf{e}_i \mid \mathsf{TS}_{\mathcal{C}}(\mathbf{e}') = t \wedge \mathbf{e}' = (\_, \_, \mathtt{write}(x, \_))\}$. ◀

▶ **Lemma 45.** *Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. Let $\mathbf{e}, \mathbf{f}$ be such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = \mathsf{TS}_{\mathcal{C}}(\mathbf{f}) = t$, $\mathbf{e} = (\_, r, \mathtt{read}(x, n_{\mathbf{e}}))$, $\mathbf{f} = (\_, r, \mathtt{read}(x, n_{\mathbf{f}})$ and whenever $\mathbf{e} \prec \mathbf{g} \prec \mathbf{f}$ for some $\mathbf{g}$ such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{g}) = t$, then $\neg(\mathbf{g} = (\_, \_, \mathtt{write}(x, \_))$. Then $n_{\mathbf{e}} = n_{\mathbf{f}}$.*

Proof. As in the previous proof, we let $\mathbf{E} = \{\mathbf{e}_i\}_{i=1}^{n}$ (with possibly $n = \infty$) and such that for any $i < j$, $\mathbf{e}_i \prec \mathbf{e}_j$. Also, for each $i = 1, \cdots, n$, we let $\mathbf{e}_i = (\iota_i, r_i, \mathbf{o}_i)$ and we let $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i) = t_i$ whenever it is defined.

Since $\mathbf{e} \prec \mathbf{f}$, there exist two indices i, j such that $\mathbf{e} = \mathbf{e}_i, \mathbf{f} = \mathbf{e}_j$ and $i < j$. Also, by Lemma 41 we know that for any index $k = 0, \cdots, j - i$, if $r_{i+k} = r, t_{i+k}$ is defined and equal to t, then either $\mathbf{o}_{i+k} = \mathtt{read}(y_{i+k}, n_{i+k})$ or $\mathbf{o}_{i+k} = \mathtt{write}(y_{i+k}, n_{i+k})$, for some y_{i+k}, n_{i+k}. Furthermore, in this second case the hypothesis of the Lemma implies that $y_{i+k} \neq x$.

By Definition 5 there exists a sequence of states $(R_{i-1}, M_{i-1}), \cdots, (R_j, M_j)$ such that

$$(R_{i-1}, M_{i-1}) \xrightarrow{\mathbf{e}_i} (R_i, M_i) \xrightarrow{\mathbf{e}_{i+1}} \cdots \xrightarrow{\mathbf{e}_{j-1}} (R_{j-1}, M_{j-1}) \xrightarrow{\mathbf{e}_j} (R_j, M_j) \tag{3}$$

For any $k = 0, (j - i) + 1$, we let $R_{(i-1)+k}(r) = (D_{(i-1)+k}, t : \rho_{(i-1)+k})$ (note that the cases $R_{(i-1)+k} = (\_, \mathsf{idle})$ or $R_{(i-1)+k} = (\_, t' : \rho)$ for some $t' \neq t$ are not possible, since replica r is already executing a transaction, and transactions are processed sequentially by replicas.).

By hypothesis both $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i) = \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) = t$, $\mathbf{e}_i = (\_, \_, \mathtt{read}(x, n_{\mathbf{e}}))$, $\mathbf{e}_j = (\_, \_, \mathtt{read}(x, n_{\mathbf{f}}))$. The judgements $(R_{i-1}, M_{i-1}) \xrightarrow{\mathbf{e}_i} (R_i, M_i)$ and $(R_{j-1}, M_{j-1}) \xrightarrow{\mathbf{e}_j} (R_j, M_j)$ could have been derived only via an application of Rule (Read), in the operational semantics of Figure 4. It follows that $n_{\mathbf{e}} = \mathsf{lastval}(x, D_{(i-1)} \cup \{t : \rho_{i-1}\}$, $n_{\mathbf{f}} = \mathsf{lastval}(x, D_{(j-1)} \cup \{t : \rho_{j-1}\}$.

Next we show that $D_{(i-1)} = D_{(j-1)}$; To do this, we show that for any $k = 0, \cdots, (j-i)+1$ then $D_{(i-1)+k} = D_{i+k}$; we do this by performing a case analysis on $\mathbf{e}_{i+k}$.

- If $\mathbf{e}_{i+k} = (\_, r', cco_{i+k})$. for some $r' \neq r$, then, the claim follows by a direct application of Lemma 27;
- if $\mathbf{e}_{i+k} = (\_, r, \texttt{read}(y_{i+k}, n_{i+k}))$, then $(R_{i+k-1}, \_) \overset{\mathbf{e}_{i+k}}{\to} (R_{(i+k)}, \_)$ has been derived by applying Rule (Read). This ensures that $R_{i+k-1} = R_{i+k}$, and in particular $D_{i+k-1} = D_{i+k}$.
- If $\mathbf{e}_{i+k} = (\_, r, \texttt{write}(y_{i+k}, n_{i+k}))$, then the judgement $(R_{i+k-1}, \_) \overset{\mathbf{e}_{i+k}}{\to} (R_{(i+k)}, \_)$ has been derived by applying Rule (Write). Again, we have that $(D_{i+k-1}, \_) = R_{i+k-1}(r) = R_{i+k}(r) = (D_{i+k}, \_)$.

Another point to be noted is that $\mathsf{lastval}(x, t : \rho_{i-1}) = \mathsf{lastval}(x, t : \rho_{j-1}$. This can be shown by proving that, for any $k = 0, \cdots, (j-i)+1$, $\mathsf{lastval}(x, t : \rho_{i-1}) = \mathsf{lastval}(x, t : \rho_{(i-1)+k}$. The proof is performed by induction on k; the case $k = 0$ is trivial. Suppose then that $k > 0$, and the claim holds for $k-1$. We perform a case analysis on the operation performed by $\mathbf{e}_{i+k}$; we give the details for the only non-trivial case, which is $\mathbf{e}_{i+k} = (\_, r, \texttt{write}(y_{i+k}, n_{i+k}))$ (in all other cases, it is easy to show that $t : \rho_{i+k-1} = t : \rho_{i+k}$). If $\mathbf{e}_{i+k} = (\_, r, \texttt{write}(y_{i+k}, n_{i+k}))$ then the facts that $\mathbf{e}_i \prec \mathbf{e}_{i+k} \prec \mathbf{e}_j$, $\mathbf{e}_i = (\_, r, \_)$, $\mathbf{e}_j = (\_, r, \_)$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i) = \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) = t$, give that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_{i+k}) = t$. This is a direct consequence of Lemma 35. Since $\rho_{i+k} = \texttt{write}(y_{i+k}, n_{i+k}) \cdot \rho_{i+k-1}$, and since $y_{i+k} \neq x$ it follows by Definition 42 that $\mathsf{lastval}(x, \texttt{write}(y_{i+k}, n_{i+k}) \cdot \rho_{i+k-1}) = \mathsf{lastval}(x, \rho_{i+k-1})$.

The last step to be performed is that of proving that $n_{\mathbf{e}} = \mathsf{lastval}(x, D_{i-1} \cup \{t : \rho_{i-1}\} = \mathsf{lastval}(x, D_{j-1} \cup \{t : \rho_{i-1}\} = n_{\mathbf{f}}$. This is an immediate consequence of Lemma 43. ◀

▶ **Lemma 46.** *Let $(\mathbf{E}, \prec)$ be a concrete execution. Let also $(\mathbf{e} = (\_, r, \texttt{commit}(t))$ be an event in $\mathbf{E}$. For any $\mathbf{f} \in \mathbf{E}$ such that $\mathbf{e} \prec \mathbf{f}$, and $\mathsf{TS}_{\mathcal{C}}(\mathbf{f}) = t'$, then $t' \neq t$.*

Proof. If $\mathbf{f}$ be such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{f}) = t$, then there exists an event $\mathbf{g} = (\_, r, \texttt{commit}(t))$ such that $\mathbf{f} \prec \mathbf{g}$. But we also have that $\mathbf{e} \prec \mathbf{f}$. By Lemma 25 we obtain that $\mathbf{e} = \mathbf{f}$. Contradiction. ◀

▶ **Lemma 47.** *Consider a (possibly infinite) computation*

$$(R_0, M_0) \overset{\mathbf{e}_1}{\to} (R_1, M_1) \overset{\mathbf{e}_2}{\to} \cdots$$

and assume that the pair $(\mathbf{E}, \prec)$, where $\mathbf{E} = (\{e_i\}_{i \in \mathbb{N}_0}, \prec = \{(\mathbf{e}_i, \mathbf{e}_j) \mid i < j\})$ is a concrete execution.

For any $i \geq 0$, we let D_j^r, l_j^r be such that $R_i(r) = (D_i^r, l_i^r)$.

Then, for any $i \geq 0$ and $r \in \mathsf{RId}$, $D_i^r \subseteq M_i$.

Proof. First note that, for any $i \geq 0$, we have that $M_i \subseteq M_{i+1}$. This can be proved by a simple inspection of the rules of the operational semantics illustrated in Figure 4.

We prove the statement by performing an induction on i. If $i = 0$, then $D_i^r = \emptyset$ and there is nothing to prove.

If $i > 0$, suppose that the claim is valid for $i-1$. We perform a case analysis over $\mathbf{e}_i$. If $\mathbf{e}_i = (\_, r', \_)$ for some $r' \neq r$, then $D_i^r = D_{i-1}^r$ by Lemma 27. By inductive hypothesis $D_{i-1}^r \subseteq M_{i-1}$, and finally $M_{i-1} \subseteq M_i$.

If $\mathbf{e}_i = (\_, r, \_)$ where $\mathbf{o} \in \{\texttt{start}, \texttt{read}(\_, \_), \texttt{write}(\_, \_)\}$ then we also have $D_i^r = D_{i-1}^r$, and we can proceed as in the last case.

If $\mathbf{e}_i = (\_, r, \texttt{commit}(t))$ then $D_i^r = D_{i-1}^r \cup \{t : \rho\}$, $M_i = M_{i-1} \cup \{t : \rho\}$ for some $\rho \in \mathsf{UpdateList}$, and $D_{i-1}^r \subseteq M_{i-1}$ by inductive hypothesis. Hence $D_i^r \subseteq M_i$.

Finally, if $\mathbf{e}_i = (\_, r, \texttt{receive}(t : \rho))$ then $D_i^r = D_{i-1}^r \cup \{t : \rho\}$, $M_{i-1} = M_i$ and $t : \rho \in M_{i-1}$. Since $D_{i-1}^r \subseteq D_i^r$ by inductive hypothesis, we have that $D_i^r = D_{i-1}^r \cup \{t : \rho\} \subseteq M_{i-1} \cup \{t : \rho\} = M_{i-1} = M_i$.

◀

▶ **Lemma 48.** *Consider a (possibly infinite) computation*

$$(R_0, M_0) \overset{\mathbf{e}_1}{\rightarrowtriangle} (R_1, M_1) \overset{\mathbf{e}_2}{\rightarrowtriangle} \cdots$$

and assume that the pair $(\mathbf{E}, \prec)$*, where* $\mathbf{E} = (\{e_i\}_{i \in \mathbb{N}_0}, \prec = \{(\mathbf{e}_i, \mathbf{e}_j) \mid i < j\})$ *is a concrete execution.*

If $t : \rho$ *is such that* $t : \rho \in M_i$ *for some* $i \geq 0$*, then there exists an event* $\mathbf{e}_j$*,* $j \leq i$*, such that* $\mathbf{e}_j = (\_, r, \texttt{commit}(t))$*.*

Proof. By induction on i. If $i = 0$, then $M_i = \emptyset$, and there is nothing to prove.

Suppose that $i > 0$, and assume that the statement holds for $i - 1$. If $\mathbf{e} = (\_, \_, \texttt{commit}(t))$, then there is nothing to prove.

If $\mathbf{e} = (\_, \_, \texttt{commit}(t'))$ for some $t' \neq t$, then $M_i = M_{i-1} \cup \{t' : \rho\}$ for some ρ. The hypothesis that $t : \rho \in M_i$, and the fact that $t \neq t'$, imply that $t : \rho \in M_{i-1}$, and by inductive hypothesis there exists $j \leq i - 1 \leq i$ such that $\mathbf{e}_j = (\_, \_, \texttt{commit}(t))$.

Finally, if $\mathbf{e}_j \neq (\_, \_, \texttt{commit}(t'))$ for any $t \in \mathbb{N}$, then $M_{i-1} = M_i$, and the statement follows by inductive hypothesis as before. ◀

▶ **Lemma 49.** *Consider a (possibly infinite) computation*

$$(R_0, M_0) \overset{\mathbf{e}_1}{\rightarrowtriangle} (R_1, M_1) \overset{\mathbf{e}_2}{\rightarrowtriangle} \cdots$$

and assume that the pair $(\mathbf{E}, \prec)$*, where* $\mathbf{E} = (\{e_i\}_{i \in \mathbb{N}_0}, \prec = \{(\mathbf{e}_i, \mathbf{e}_j) \mid i < j\})$ *is a concrete execution.*

For any $i \geq 0$*, we let* D_j^r, l_j^r *be such that* $R_i(r) = (D_i^r, l_i^r)$*.*

Let $r \in \mathsf{RId}$*,* $t \in \mathbb{N}$ *and* $\rho, \rho' \in \mathsf{UpdateList}$*.*

1. *if* $\{t : \rho, t : \rho'\} \subseteq D_i^r$*, then* $\rho = \rho'$*,*
2. $\{t : \rho, t : \rho'\} \subseteq M_i$*, then* $\rho = \rho'$*.*

Proof. It suffices to prove the second statement. In fact, in this case we would have that if $\{t : \rho, t : \rho'\} \subseteq D_i^r$, then by Lemma 47 we would have that $\{t : \rho, t' : \rho'\} \subseteq M^i$, hence $\rho = \rho'$.

We prove by induction on i that, if $t : \rho, t : \rho' \in M_i$ for some $i \geq 0$, then $\rho = \rho'$. The case $i = 0$ is vacuous, since $M_i = \emptyset$.

Let then $i > 0$, and suppose that the statement is true for $i - 1$. Suppose that $\mathbf{e}_i = (\_, \_, \mathbf{o})$ for some $\mathbf{o}$

If $\mathbf{o} \neq \texttt{commit}(\_)$, then $M_{i-1} = M_i$. Thus, $t : \rho, t : \rho' \in M_{i-1}$, and by inductive hypothesis $\rho = \rho'$.

If $\mathbf{o} = \texttt{commit}(t')$ for some $t' \neq t$, then $M_i = M_{i-1} \cup \{t' : \rho''\}$ for some ρ'', and again it has to be the case that $t : \rho, t : \rho' \in M_{i-1}$. Hence $\rho = \rho'$ by inductive hypothesis.

Finally, if $\mathbf{o} = \texttt{commit}(t)$, then $M_i = M_{i-1} \cup \{t : \rho''\}$ for some ρ''. Note that if $t : \rho \in M_{i-1}$, then by Lemma 48 there exists an event $\mathbf{e}_j \preceq \mathbf{e}_{i-1} \prec \mathbf{e}_i$ such that $\mathbf{e}_j = (\_, \_, \texttt{commit}(t))$. But Lemma 25 gives that $\mathbf{e}_j = \mathbf{e}_i$, causing a contradiction. Therefore, $t : \rho \notin M_{i-1}$; the same argument can be used to prove that $t : \rho' \notin M_{i-1}$. Therefore, it has to be the case that $t : \rho, t : \rho' \in \{t : \rho''\}$, from which the equality $\rho = \rho'' = \rho'$ follows.

The two statements are proved simultaneously by induction over i. If $i = 0$, there is nothing to prove. Suppose then that $i > 0$, and the claim holds for $i - 1$. We perform a case analysis on the proof of the derivation $(R_{i-1}, M_{i-1}) \overset{\mathbf{e}_i}{\rightarrowtriangle} (R_i, M_i)$. ◀

▶ **Lemma 50.** *Consider a (possibly infinite) computation*

$$(R_0, M_0) \xrightarrow{\mathbf{e}_1} (R_1, M_1) \xrightarrow{\mathbf{e}_2} \cdots$$

and assume that the pair $(\mathbf{E}, \prec)$*, where* $\mathbf{E} = (\{e_i\}_{i\in\mathbb{N}_0}, \prec = \{(\mathbf{e}_i, \mathbf{e}_j) \mid i < j\})$ *is a concrete execution. let* $\{D_i^r\}_{i\geq 0, r\in\mathsf{RId}}$ *and* $\{l_i^r\}_{i\geq 0, r\in\mathsf{RId}}$ *be two sets such that let* $R_i(r) = (D_i^r, I_i^r)$*. Whenever* $t : \rho$ *is a transaction log such that* $t : \rho \in D_i^r$*, for some* $i \geq 0 <$ *there exists an index* $j \leq i$ *such that* $\mathbf{e}_j = (\_, r, \mathbf{o})$ *where* $\mathbf{o} \in \{receive(t : \rho), \mathtt{commit}(t)\}$*. and* $\mathsf{LogOf}_{\mathcal{C}}(t, \mathbf{e}_1 \cdots \mathbf{e}_i) = t : \rho$*.*

Proof. We perform an induction over i. If $i = 0$ then $D_i^r = \mathsf{idle}$ for any $r \in \mathsf{RId}$, by Definition 5. This case is vacuous. Let then $i > 0$, and suppose that the claim holds for $i - 1$. We perform a case analysis over $\mathbf{e}_i$:

- if $\neg(\mathbf{e}_i = (\_, \_, \mathbf{o}_i))$, where$\mathbf{o}_i \in \{\mathtt{receive}(t' : \rho), \mathtt{commit}(t')\}$, then it can be proved $D_i^r = D_{i-1}^r$ for any $r \in \mathsf{RId}$; the proof is performed via a case analysis on the proof of the transition $(R_{i-1}, M_{i-1}) \xrightarrow{\mathbf{e}_i} (R_i, M_i)$.
  Let $r \in \mathsf{RId}$ be such that $t : \rho \in D_i^r$; then $t : \rho \in D_{i-1}^r$ and by inductive hypothesis there exists an index $j < i-1 < i$ such that $cce_j = (\_, \_, \mathbf{o}_j)$ with $\mathbf{o}_j \in \{\mathtt{commit}(t), \mathtt{receive}(t : \rho) \mid \rho \in \mathsf{UpdateList}\}$, and $\mathsf{LogOf}_{\mathcal{C}}(t, \mathbf{e}_1 \cdots \mathbf{e}_{i-1}) = t : \rho$.
  Finally, note that there exists an index $h \leq j < i$ such that $\mathbf{e}_h = (\_, \_, \mathtt{commit}(t))$. In fact, if $\mathbf{o}_j = \mathtt{commit}(t))$, simply take $h := j$; otherwise, $\mathbf{o}_j = \mathtt{receive}(t : \rho)$ and the existence of such an event $\mathbf{e}_h$ follows from an application of Lemma 39. We have that $\mathbf{e}_h \preceq \mathbf{e}_j \prec \mathbf{e}_i$, hence by Lemma 46 it has to be $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i) \neq t$. By definition, $\mathsf{LogOf}_{\mathcal{C}}(t, \mathbf{e}_1 \cdots \mathbf{e}_i) = \mathsf{LogOf}_{\mathcal{C}}(t, \mathbf{e}_1 \cdots \mathbf{e}_{i-1}) = (t : \rho)$,
- if $\mathbf{e}_i = (\_, r, \mathbf{o}_i)$ where $\mathbf{o}_i \in \{\mathtt{receive}(t : \rho), \mathtt{commit}(t) \mid \rho \in \mathsf{UpdateList}\}$. If $t' \neq t$, then whenever $t : \rho \in D_i^{r'}$ for some replica r' (possibly $r' = r$) it has to be the case that $t : \rho \in D_i^{r'}$, and the claim follows from the inductive hypothesis and the definition of $\mathsf{LogOf}_{\mathcal{C}}(t, \mathbf{e}_1 \cdots \mathbf{e}_i)$. Otherwise $t = t'$; note that, for any replca $r' \neq r$, $D_i^{r'} = D_{i-1}^{r'}$, hence if $t : \rho \in D_i^{r'}$ then $t : \rho \in D_{i-1}^{r-1}$, and by inductive hypothesis $t : \rho = \mathsf{LogOf}_{\mathcal{C}}(\mathbf{e}_1 \cdots \mathbf{e}_{i-1})$. Since $\mathbf{o}_i$ is not an operation of the form $\mathtt{write}(\_, \_), \mathtt{read}(\_, \_)$, $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i)$ is not defined, and by definition $\mathsf{LogOf}_{\mathcal{C}}(\mathbf{e}_1 \cdots \mathbf{e}_i = \mathsf{LogOf}_{\mathcal{C}}(\mathbf{e}_1 \cdots \mathbf{e}_{i-1}) = t : \rho$.
  Finally, suppose that $t : \rho \in D_{i-1}^{r'}$. We distinguishing between two sub-cases:
  - $\mathbf{e}_i = (\_, r, \mathtt{commit}(t))$; since $(R_{i-1}, M_{i-1}) \xrightarrow{\mathbf{e}_i} (R_i, M_i)$ can be derived only by applying Rule (Commit), it has to be the case that $D_i^r = D_{i-1}^r \cup \{t : \rho\}$, and $l_{i-1}^r = t : \rho$. By Lemma 38 it follows that $t : \rho = l_{i-1}^r = \mathsf{LogOf}_{\mathcal{C}}(t, \mathbf{e}_1 \cdots \ cce_{i-1})$. Since $\mathbf{e}_i = (\_, r, \mathtt{commit}(t))$, then $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i)$ is undefined, and by definition of $\mathsf{LogOf}_{\mathcal{C}}$ we obtain that $\mathsf{LogOf}(t, \mathbf{e}_1 \cdots \mathbf{e}_i) = \mathsf{LogOf}(t, \mathbf{e}_1 \cdots \mathbf{e}_{i-1}) = t : \rho$.
  - Finally, suppose that $\mathbf{e}_i = (\_, r, \mathtt{receive}(t : \rho))$. This is possible only if $t : \rho \in M_{i-1}$. By Lemma 48, there exists an event $\mathbf{e}_j \preceq \mathbf{e}_i$ such that $\mathbf{e}_j = (\_, r', \mathtt{commit}(t))$; note that it cannot be $\mathbf{e}_j = \mathbf{e}_i$, hence it has to be $\mathbf{e}_j \prec \mathbf{e}_i$. Since $\mathbf{e}_j = (\_, r', \mathtt{commit}(t))$, it has to be the case that $t : \rho' \in D_j^r$ for some ρ'. By Lemma 38 we know that $\rho' = \mathsf{LogAux}_{\mathcal{C}}(t, \mathbf{e}_1, \cdots, \mathbf{e}_j)$. Also, since $\mathbf{e}_j = \mathtt{commit}(t)$, by Lemma 46, whenever $\mathbf{e}_h$ is such that $\mathbf{e}_j \prec \mathbf{e}_h$, then $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_h) \neq t$. By definition, $\rho' = \mathsf{LogAux}_{\mathcal{C}}(t, \mathbf{e}_1 \cdots \mathbf{e}_j) = \mathsf{LogAux}_{\mathcal{C}}(t, \mathbf{e}_1 \cdots \mathbf{e}_i)$. Finally, since $\rho' \in D_j^{r'}$, by Lemma we obtain that $t : \rho' \in M_j$, hence $t : \rho' \in M_{i-1}$ because $\mathbf{e}_j \preceq \mathbf{e}_{i-1}$. We have proved that $t : \rho', t : \rho \in M_{i-1}$; By Lemma 49 we obtain that $\rho = \rho'$.
    In this case $D_i^r = D_{i-1}^r \cup \{t : \rho'\}$.
    First note that $\rho' = \rho$; in fact, if it were $\rho' \neq \rho$, then $t : \rho \in D_i$ implies that $t : \rho \in D_{i-1}$.

◀

B.2 Encoding Concrete Executions into Abstract Ones

We write again the definition of $\mathsf{history}(\mathcal{C})$ for the sake of clarity. We also show how the two relaitons $\mathsf{AR}(\mathcal{C}) \subseteq \mathsf{history}(\mathcal{C}) \times \mathsf{history}(\mathcal{C})$ and $\mathsf{VIS}(\mathcal{C}) \subseteq \mathsf{history}(\mathcal{C}) \times \mathsf{history}(\mathcal{C})$ can be derived from a given concrete execution $\mathcal{C}$.

▶ **Definition 51.** Let $\mathcal{C} = (\mathbf{E}, \prec)$ be a concrete execution. We define the *underlying abstract execution* of $\mathcal{C}$ as $\mathsf{AbsExec}(\mathcal{C}) = (\mathsf{history}(\mathcal{C}), \mathsf{VIS}(\mathcal{C}), \mathsf{AR}(\mathcal{C}))$, where

- For any $t \in \mathbb{N}$ let $T_t = (E_t, \mathsf{po}_t)$, where $E_t = \{(\iota, \mathbf{o}) \mid (\iota, \_, \mathbf{o}) \in \mathbf{E} \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t\}$ and $\mathsf{po}_t = \{(\iota_1, \mathbf{o}_1), (\iota_2, \mathbf{o}_2) \mid (\iota_1, \_, \mathbf{o}_1) \prec (\iota_2, \_, \mathbf{o}_2)\}$ (note that $E_t = \emptyset, \mathsf{po}_t = \emptyset$ if there exists no $\mathbf{e} \in \mathbf{E}$ such that $\mathbf{e} \triangleright t : \_ \ @ \ \_$); then $\mathsf{history}(\mathcal{C}) = \{T_t \mid T_t \neq \emptyset\}$;,
- $\mathsf{AR}(\mathcal{C}) = \{(T_t, T_s) \in \mathsf{history}(\mathcal{C}) \mid t < s\}$,
- $\mathsf{VIS}(\mathcal{C}) = \{(T_t, T_s) \mid \exists r, \mathbf{o}, \iota.\, (\_, r, \mathbf{o}) \prec (\iota, r, \mathtt{start}) \wedge \mathbf{o} \in \{\mathtt{commit}(t), \mathtt{receive}(t : \_)\} \wedge (\iota, r, \mathtt{start}) = \mathtt{start}_{\mathcal{C}}(t)\}$.

▶ Proposition 52. For any concrete execution $\mathcal{C}$, $\mathsf{AR}(\mathcal{C})$ is a total order.

Proof. A simple consequence of the fact that $\mathsf{AR}(\mathcal{C})$ orders transactions according to the total order $<$ over timestamps assigned in $\mathcal{C}$. ◀

▶ Proposition 53. For any concrete execution $\mathcal{C}$, $\mathsf{VIS}(\mathcal{C}) \subseteq \mathsf{AR}(\mathcal{C})$.

Proof. Consider a concrete execution $\mathcal{C} = (\mathbf{E}, \prec)$; for convenience, we let $\mathbf{E} = \{\mathbf{e}_i\}_{i=1}^{n}$, with possibly $n = \infty$, and such that for any $i < j$, $\mathbf{e}_i \prec \mathbf{e}_j$. Let now $(E, \mathsf{po}) \in \mathsf{history}(\mathcal{C})$, and consider an event $e \in E$ such that $e = (\iota, \mathtt{read}(x, n))$. By construction of $\mathsf{history}(\mathcal{C})$ it follows that there exists a timestamp t such that $E = E_t, \mathsf{po} = \mathsf{po}_t$, as per Definition 51. Let also (R_i, M_i), $i = 1, \cdots$ be such that

$$(R_0, M_0) \xrightarrow{\mathbf{e}_1} (R_1, M_1) \xrightarrow{e_2} \cdots$$

and for any $i = 1, \cdots$ let $R_i(r) = (D_i^r, l_i^r)$.

Let $S, T \in \mathsf{history}(\mathcal{C})$ be such that $S \xrightarrow{\mathsf{VIS}(\mathcal{C})} T$. By definition 51, there exist t, t' such that $S = T'_t, T = T_t$. Also, there exists two indexes i, j such that $j < i$, $\mathbf{e}_j = (\_, r, \mathbf{o})$ where $\mathbf{o} \in \{\mathtt{receive}(t' : \_), \mathtt{commit}(t')\}$ and $\mathbf{e}_s = (\_, r, \mathtt{start}) = \mathtt{start}_{\mathcal{C}}(t)$. It follows that $(t' : \_) \in D_j^r$, and by Lemma 24, $(t' : \_) \in D_{s-1}^r$. By looking at the structure of the proof of $(R_{s-1}, M_{s-1}) \xrightarrow{\mathbf{e}_s} (R_s, M_s)$, recalling that $\mathbf{e} = (\_, r, \mathtt{start})$, it has to be the case that $t' < t$, hence $T_{t'} \xrightarrow{\mathsf{AR}(\mathcal{C})} T_t$ by Definition 51. ◀

▶ Proposition 54. For any concrete execution $\mathcal{C}$, both $\mathsf{AR}(\mathcal{C})$ and $\mathsf{VIS}(\mathcal{C})$ are prefix-finite.

Proof. If suffices to show that $\mathsf{AR}(\mathcal{C})$ is prefix-finite; the result for $HB(\mathcal{C})$ follows then from Proposition 53. Let $(E, \mathsf{po}) \in \mathsf{history}(\mathcal{C})$. Then $(E, \mathsf{po}) = (E_t, \mathsf{po}_t)$ for some timestamp t, where E_t, po_t are defined as per Definition 51. Also, note that whenever $S \xrightarrow{\mathsf{AR}(\mathcal{C})} T$, then $S = (E_{t'}, \mathsf{po}_{t'})$ for some timestamp $t' < t$. It follows that $|\{S \mid S \xrightarrow{\mathsf{AR}(\mathcal{C})} T\}| \leq |\{t' \mid t' < t\}| \leq t < +\infty$. ◀

▶ Proposition 55. Let $\mathcal{C}$ be a concrete execution. Then, for any $(E, \mathsf{po}) \in \mathsf{history}\mathcal{C}$, $|E| < +\infty$.

Proof. Consider a concrete execution $\mathcal{C} = (\mathbf{E}, \prec)$; for convenience, we let $\mathbf{E} = \{\mathbf{e}_i\}_{i=1}^{n}$, with possibly $n = \infty$, and such that for any $i < j$, $\mathbf{e}_i \prec \mathbf{e}_j$. Let $(E, \mathsf{po}) \in \mathsf{history}(\mathcal{C})$. Then there exists a timestamp $t \in \mathbb{N}$ such that $E = E_t$, as per Definition 51. Recall that $E_t = \{(\iota, \mathbf{o}) \mid \exists \mathbf{e} \in \mathbf{E}.\mathbf{e} = (\iota, \_, \mathbf{o}) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t$. Also, it is easy to note that

$\{\mathbf{e} \in \mathbf{E} \mid \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t\} \subseteq \{\mathbf{e} \in \mathbf{E} \mid \mathbf{e} \prec (\_, \_, \texttt{commit}(t))\}$. Recall that by Lemma 25 there exists a unique event of the form $(\_, \_, \texttt{commit}(t))$, and assume that such an event is $\mathbf{e}_i$, for some $i \geq 0$.

Now we have that

$$|E_t| = |\{\mathbf{e} \in \mathbf{E} \mid \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t\}| \leq |\{\mathbf{e} \in \mathbf{E} \mid \mathbf{e} \prec \mathbf{e}_i\}| \leq i < +\infty.$$

◀

▶ Proposition 56. Let $\mathcal{C}$ be a concrete execution. For any $(E, \mathsf{po}) \in \mathsf{history}(\mathcal{C})$, po is a total order.

Proof. Assume $\mathcal{C} = (\mathbf{E}, \prec)$. IF $(E, \mathsf{po}) \in \mathsf{history}(\mathcal{C})$, there exists a timestamp t such that $E = E_t, \mathsf{po} = \mathsf{po}_t$, as per Definition 51. For any $e = (\iota_1, \_), f = (\iota_2, \_) \in E_t$, with $\iota_1 \neq \iota_2$. Definition 51 ensures that there exist two events $\mathbf{e} = (\iota_1, \_, \_)\mathbf{f} = (\iota_2, \_, \_)$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = \mathsf{TS}_{\mathcal{C}}(\mathbf{f}) = t$. Since $\iota_1 \neq \iota_2$, then $\mathbf{e} \neq \mathbf{f}$. Since $\prec$ is a total order, either $\mathbf{e} \prec \mathbf{f}$ or $\mathbf{f} \prec \mathbf{e}$. Without loss of generality, let $\mathbf{e} \prec \mathbf{f}$. By Definition 51, it follows that $e \xrightarrow{\mathsf{po}_t} f$.

In a similar way, we can prove that po_t is both irreflexive and transitive. ◀

▶ Proposition 57. Let $\mathcal{C}$ be a concrete execution. If $(E, \mathsf{po}_1), (F, \mathsf{po}_2) \in \mathsf{history}(\mathcal{C})$, and $E \cap F \neq \emptyset$, then $E = F$.

Proof. By Definition 51 there exist two timestamps t, t' such that $(E, \mathsf{po}_1) = (E_t, \mathsf{po}_t)$, $(F, \mathsf{po}_2) = (E_{t'}, \mathsf{po}_{t'})$, where the right hand sides of the equations are defined according to Definition 51. Let $e \in E_t \cap E_{t'}$, and assume that $e = (\iota, \_)$. By Definition 51, there exists an event $\mathbf{e}$ such that both $\mathbf{e} = (\iota, \_, \_)$, and $t' = \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t$. Therefore, $t = t'$, hence $(E_t, \mathsf{po}_t) = (E_{t'}, \mathsf{po}_{t'})$, as we wanted to prove. ◀

▶ Proposition 58. For any $\mathcal{C} \in \mathsf{ConcExec}_{\mathbf{RA}}$, $\mathsf{AbsExec}(\mathcal{C}) \models \textsc{Int}$.

Proof. Consider a concrete execution $\mathcal{C} = (\mathbf{E}, \prec)$; for convenience, we let $\mathbf{E} = \{\mathbf{e}_i\}_{i=1}^{n}$, with possibly $n = \infty$, and such that for any $i < j$, $\mathbf{e}_i \prec \mathbf{e}_j$. Let now $(E, \mathsf{po}) \in \mathsf{history}(\mathcal{C})$, and consider an event $e \in E$ such that $e = (\iota, \texttt{read}(x, n))$. By construction of $\mathsf{history}(\mathcal{C})$ it follows that there exists a timestamp t such that $E = E_t, \mathsf{po} = \mathsf{po}_t$, as per Definition 51. This means that there exists a (possibly infinite) computation

$$(R_0, M_0) \xrightarrow{\mathbf{e}_1} (R_1, M_1) \xrightarrow{\mathbf{e}_2} \cdots$$

and an index i such that $\mathbf{e}_i = (\iota, \_, \texttt{read}(x, n))$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i) = t$. In particular, this means that $R_{i-1} = (D_i, \rho_i)$, and $n = \mathsf{lastval}(x, D \cup \{\infty : \rho_i\})$. By Lemma 43 then either $n = \mathsf{lastval}(x, \rho_i)$, or $\mathsf{lastval}(x, \rho_i)$ is undefined and $n = \mathsf{lastval}(x, D)$. We perform a case analysis on whether $n = \mathsf{lastval}(x, \rho_i)$ or $\mathsf{lastval}(x, \rho_i)$ is undefined $n = \mathsf{lastval}(x, D)$, to prove that if the event $\mathbf{e}_h := \max_{\prec}\{\mathbf{e} \mid \mathbf{e}' \prec \mathbf{e}_i \wedge \exists \mathbf{o}'.\mathbf{e}' = (\_, \_, \mathbf{o}') \wedge \mathbf{o}' \in \{\texttt{read}(x, \_), \texttt{write}(x, \_)\} \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{o}') = t\}$ is defined, then $\mathbf{e}_h = (\_, \_, \_(x, n))$.

- If $n = \mathsf{lastval}(x, \rho_i)$, then by Lemma 44 the event $\mathbf{e}_j := \max_{\prec}\{\mathbf{e}' \mid \mathbf{e}' \prec \mathbf{e}_i \wedge \mathbf{e}' = (\_, \_, \texttt{write}(x, \_)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}') = t\}$ is well defined with $j < i$, and $\mathbf{e}_j = (\_, \_, \texttt{write}(x, n))$. Since $\mathbf{e}_j$ is defined, then also $\mathbf{e}_h$ is defined (recall that $\prec$ is assumed to be a total order, hence $\max_{\prec}(X)$ is not defined for some set X, if and only if $X = \emptyset$). Also, it has to be the case that $j \leq h < i$. If $j = h$ then we already know that $\mathbf{e}_j = (\_, r, \_(x, n))$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) = t$; if $j < h$, instead, it has to be the case that $\mathbf{e}_h = (\_, \_, \texttt{read}(x, m))$ for some m, and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_h) = t$. Also, for no index k such that such $h < k < i$ we would have $\mathbf{e}_k = (\_, \_, \texttt{write}(x, \_))$ with $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_k) = t$, since it would contradict the assumption

that $\mathbf{e}_j := \max_{\prec}\{\mathbf{e}' \mid \mathbf{e}' \prec \mathbf{e}_i \wedge \mathbf{e}' = (\_, \_, \texttt{write}(x, \_)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}') = t\}$. It follows from Lemma 45 that $m = n$, and in particular $\mathbf{e}_h = (\_, \_, \texttt{read}(x, n))$, as we wanted to prove.

- Otherwise, suppose that $n = \mathsf{lastval}(x, D)$, which means that $\mathsf{lastval}(x, t : rho) =$ **undefined**; equivalently, there exists no index of the form $\texttt{write}(x, \_)$ in ρ. By Lemma 38, it follows that there exists no event $\mathbf{e}_j = (\_, \_, \texttt{write}(x, n))$ such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) = t$ and $j < i$. Thus, if the event $\mathbf{e}_h$ is defined, it has to be the case that $\mathbf{e}_h = (\_, \_, \texttt{read}(x, m))$, $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_h) = t$, and there exist no index k such such $\mathbf{e}_k = (\_, \_, \texttt{write}(x, \_))$, $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_k) = t$ and $h < k < i$. Again, it follows from Lemma 45 that $m = n$, and in particular $\mathbf{e}_h = (\_, \_, \texttt{read}(x, n))$ with $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_h) = t$.

Next, it remains to note that an event $e = (\iota, \texttt{read}(x, n))$ is included in E_t if and only if the event $\mathbf{e} = (\iota, \_, \texttt{read}(x, n))$ is such that $\mathbf{e} \in \mathbf{E}$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t$. Similarly, for any $e, f \in E_t$, with $e = (\iota_1, \_)$, $f = (\iota_2, \_)$, we have that $e\mathsf{po}_t f$ if and only if $\mathbf{e} \prec \mathbf{f}$, where $\mathbf{e} = (\iota_1, \_)$ and $\mathbf{f} = (\iota_2, \_)$. It follows that the event $\mathbf{e}_h = \max_{\prec}\{\mathbf{e} \mid \mathbf{e}' \prec \mathbf{e}_i \wedge \exists \mathbf{o}'.\mathbf{e}' = (\_, \_, \mathbf{o}') \wedge \mathbf{o}' \in \{\texttt{read}(x, \_), \texttt{write}(x, \_)\} \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{o}') = t\}$ is defined if and only if the event $f := \max_{\mathsf{po}_t}(\mathsf{po}_t^{-1}(e) \cap \mathsf{HEvent}_x)$ is defined (recall that e is the event that corresponds to $\mathbf{e}$). Also, $\mathbf{e}_h = \texttt{read}(x, n)$ if and only if $f = (\_, \texttt{read}(x, n))$, and similarly for writes. This concludes the proof.

◀

▶ Proposition 59. For any $\mathcal{C} \in \mathsf{ConcExec}_{\mathbf{RA}}, \mathsf{AbsExec}(\mathcal{C}) \models \textsc{Ext}$.

Proof. Consider a concrete execution $\mathcal{C} = (\mathbf{E}, \prec)$; for convenience, we let $\mathbf{E} = \{\mathbf{e}_i\}_{i=1}^{n}$, with possibly $n = \infty$, and such that for any $i < j$, $\mathbf{e}_i \prec \mathbf{e}_j$. Also, let $\{(R_i, M_i)\}_{i=0}^{n+1}$ be the sequence of states for which $(R_{i-1}, M_{i-1}) \xrightarrow{\mathbf{e}_i} (R_i, M_i)$, for any $i = 1, \cdots n$; finally, for any $i = 1, \cdots, n$, let $R_i(r) = (D_i^r, l_i^r)$, where either $l_i^r = \mathsf{idle}$ or $l_i^r = \rho_i^r$ for some $\rho_i^r \in \mathsf{UpdateList}$.

Let now $(E, \mathsf{po}) \in \mathsf{history}(\mathcal{C})$ be such that $(E, \mathsf{po}) \vdash \mathsf{Read}\ x : n$. By definition, there exists an event $e \in E$ such that $e = (\iota, \texttt{read}(x, n))$, and for any event $f \in \mathsf{po}_t^{-1}(e)$, $f \neq (\_, \texttt{write}(x, n))$.

Since $(E, \mathsf{po}) \in \mathsf{history}(\mathcal{C})$, then there exists a timestamp $t \in \mathbb{N}$ such that $E = E_t, \mathsf{po} = \mathsf{po}_t$, as per Definition 51. It follows that there exists an event $\mathbf{e} \in \mathbf{E}$ such that $\mathbf{e}_i = (\_, r, \texttt{read}(x, n))$ with $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i) = t$, and for any $\mathbf{e}_j$ with $j < i$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_j) = t$, then $\mathbf{e}_j \neq (\_, \_, \texttt{write}(x, n))$.

By looking at the derivation of the transition $(R_i - 1, M_i - 1) \xrightarrow{\mathbf{e}_i} (R_i, M_i)$, it has to be the case that $l_{i-1}^r = \rho_{i-1}^r$ for some $\rho_{i-1}^r \in \mathsf{UpdateList}$; by Lemma 38 it follows that $\mathsf{LogOf}_{\mathcal{C}}(t, \cdots \mathbf{e}_1, \cdots, cce_i) = (t_{i-1} : \rho_{i-1})$, and since there exists no index $j < i$ such that $\mathsf{TS}_{\mathbf{e}_j} = t$ and $\mathbf{e}_j = (\_, \texttt{write}(x, \_))$, then ρ_{i-1} does not contain a record of the form $\texttt{write}(x, \_)$. It follows that $\mathsf{lastval}(x, \rho_{i-1}) =$ **undefined**. By Lemma 43 then $n = \mathsf{lastval}(x, D_{i-1})$.

Let us now look at the set of transaction logs D_{i-1}^r.

If there exists no log of the form $t' : \rho$ such that $t' : \rho \in D_{i-1}^r$ and $\mathsf{lastval}(x, t' : \rho) \neq$ **undefined**, then by Definition 42 $n = 0$. In this case, it suffices to show that there exists no transaction $(E_{t'}, \mathsf{po}_{t'}) \in \mathsf{history}(\mathcal{C})$ such that $(E_{t'}, \mathsf{po}_{t'}) \vdash \mathsf{Write}\ x : \_$ and $(E_{t'}, \mathsf{po}_{t'} \xrightarrow{\mathsf{VIS}(\mathcal{C})} (E_t, \mathsf{po}_t)$.

Suppose then that $\mathsf{lastval}(x, t' : \rho) =$ **undefined** for any $t' : \rho \in D^{r-1}$. Fix a transaction log t' such that $t' : \rho \in D^{r-1}$ and let let $(E_{t'}, \mathsf{po}_{t'}) \in \mathsf{history}(\mathcal{C})$ be a transaction such that $(E_{t'}, \mathsf{po}_{t'}) \xrightarrow{\mathsf{VIS}(\mathcal{C}} (E_t, \mathsf{po}_t)$.

By Definition 51, we have that there exists an index j such that $\mathbf{e}_j = (\_, r, \mathbf{o})$, where $\mathbf{o} \in \{\texttt{commit}(t'), \texttt{receive}(t' : \rho) \mid \rho \in \mathsf{UpdateList}\}$ and $\mathbf{e}_j \prec \mathtt{start}_{\mathcal{C}}(t) \prec \mathbf{e}_i$ (recall that

$\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i) = t$, hence $\mathtt{start}_{\mathcal{C}}(\mathbf{e}_i) = \mathtt{start}_{\mathcal{C}}(t)$. By looking at the proof of the transition $(R_{j-1}, M_{j-1}) \xrightarrow{\mathbf{e}_j} (R_j, M_j)$, we can infer that there exists a transaction log $\rho' \in \mathsf{UpdateList}$ such that $(t' : \rho' \in D^r_j)$, and since $j < i$ it follows from Lemma 24 that $(t' : \rho' \in D_{i-1})$. Therefore, we have that $(t' : \rho), (t' : \rho') \in D^r_{i-1}$, and by Lemma 49 it follows that $\rho = \rho'$. Hence $\mathsf{lastval}(x, t' : \rho') = \textbf{undefined}$.

By Lemma 50 we have that $\mathsf{LogOf}_{\mathcal{C}}(t', \mathbf{e}_1, \cdots \mathbf{e}_i) = t' : \rho'$, and since $\mathsf{lastval}(x, t' : \rho') = \textbf{undefined}$ it follows that whenever h is an index such that $h < i$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_h) = t'$, then $\mathbf{e}_h \neq (\_, \_, \mathtt{write}(x, n))$ for any $h < i$. Equivalently, there exists no event $e \in E_{t'}$ such that $e = (\_, \mathtt{write}(x, \_))$, from which it follows that $(E_{t'}, \mathsf{po}_{t'} \not\vdash \mathsf{Write}\ x : \_$, as we wanted to prove.

Assume instead that there exists at least one transaction log $t' : \rho \in D^r_{i-1}$ such that $\mathsf{lastval}(x, t' : \rho)$ is defined. Let $\mathbf{e}_k = (\_, \_, \mathtt{commit}(t))$ be the unique event in $\mathbf{E}$ such that $\mathbf{e}_i \prec \mathbf{e}_k$; this event exist because $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_i)$ is defined; by Lemma 24, $t' : \rho \in D^r_{k-1}$, and by analysing the proof of the derivation $(R_{k-1}, M_{k-1}) \xrightarrow{\mathbf{e}_k} (R_k, M_k)$, this is possible only if $t' < t$.

Without loss of generality, assume that $t' = \max_<\{t'' \mid t'' : \rho'' \in D^r_{i-1} \wedge \mathsf{lastval}(x, t'' : \rho'') \neq \textbf{undefined}\}$; in this case we have that $\mathsf{lastval}(x, t' : \rho') = n$. We show that $(E_{t'}, \mathsf{po}'_t) \vdash \mathsf{Write}\ x : n$, and $(E_{t'}, \mathsf{po}_{t'}) = \max_{\mathsf{AR}(\mathcal{C})}\{(E_{t''}, \mathsf{po}_{t''}) \mid (E_{t''}, \mathsf{po}_{t''}) \xrightarrow{\mathsf{VIS}(\mathcal{C})} (E_t, \mathsf{po}_t) \wedge (E_{t''}, \mathsf{po}_{t''}) \vdash \mathsf{Write}\ x : \_\}$.

First, note that since $t' : \rho \in D^r_{i-1}$, then $(E'_t, \mathsf{po}'_t) \xrightarrow{\mathsf{VIS}(\mathcal{C})} (E_t, \mathsf{po}_t)$. In fact, note that by Lemma 50 there exists an index j such that $j < i$, and $\mathbf{e}_j = (\_, r, \mathbf{o})$ with $\mathbf{o} \in \{\mathtt{commit}(t'), \mathtt{receive}(t' : \rho') \mid \rho' \in \mathsf{UpdateList}\}$. Because of Lemma 35, and since $t' \neq t$, it has to be the case that $\mathbf{e}_j \prec \mathtt{start}_{\mathcal{C}}(t) \prec \mathbf{e}_i$.

By Definition 51 this gives $(E_{t'}, \mathsf{po}_{t'}) \xrightarrow{\mathsf{VIS}(\mathcal{C})} (E_t, \rho_t)$. Lemma 50 also ensures that $t' : \rho = \mathsf{LogOf}_{\mathcal{C}}(t', \mathbf{e}_1, \cdots, \mathbf{e}_i)$. Since $\mathsf{lastval}(x, t : \rho') = n$, this means that there exists an event $\mathbf{e}_h$, $h < i$, such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_h) = t'$, $\mathbf{e}_j = (\_, \_, \mathtt{write}(x, n))$ and whenever $k : i < h < k$, if $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_k) = t'$ then $\mathbf{e}_k \neq (\_, \_, \mathtt{write}(x, \_))$. By Definition 51, there exists an event $f \in E_{t'}$ such that $\mathsf{op}(f) = \mathtt{write}(x, n)$, and for any $g : f \xrightarrow{\mathsf{po}_{t'}} g$, then $\mathsf{op}(g) \neq \mathtt{write}(x, \_)$. By definition, $f = \max_{\mathsf{po}_{t'}}\{g \mid \mathsf{op}(g) = \mathtt{write}(x, \_)\}$, hence $E_{t'} \vdash \mathsf{Write}\ x : n$.

Let now $(E_{t''}, \mathsf{po}_{t''}) \in \mathsf{history}(\mathcal{C})$ be a transition such that $(E_{t''}, \mathsf{po}_{t''}) \xrightarrow{\mathsf{VIS}(\mathcal{C})} (E_t, \mathsf{po}_t)$, and $(E_{t''}, \mathsf{po}_{t''}) \vdash \mathsf{Write}\ x : \_$. We show that either $(E_{t''}, \mathsf{po}_{t''}) = (E_{t'}, \mathsf{po}_{t'})$, or $(E_{t''}, \mathsf{po}_{t''}) \xrightarrow{\mathsf{AR}(\mathcal{C})} (E_{t'}, \mathsf{po}_{t'})$. Since we have already proved that $t' < t$, then it follows from $(E_{t'}, \mathsf{po}_{t'}) \xrightarrow{\mathsf{AR}(\mathcal{C})} (E_t, \mathsf{po}_t)$ and by Proposition 52 that $(E_{t''}, \mathsf{po}_{t''} \xrightarrow{\mathsf{AR}(\mathcal{C})} (E_t, \mathsf{po}_t)$; as a simple consequence, we get that $(E_{t'}, \mathsf{po}_{t'}) = \max_{\mathsf{AR}(\mathcal{C})}\{(E_{t''}, \mathsf{po}_{t''}) \mid (E_{t''}, \mathsf{po}_{t''}) \xrightarrow{\mathsf{VIS}(\mathcal{C})} (E_t, \mathsf{po}_t) \wedge (E_{t''}, \mathsf{po}_{t''}) \vdash \mathsf{Write}\ x : \_\}$.

Since $(E_{t''}, \mathsf{po}_{t''}) \in \mathsf{history}(\mathcal{C})$, by definition 51 there exists an event $\mathbf{e}_j$, such that $\mathbf{e}_j \prec \mathtt{start}_{\mathcal{C}}(t)$, $\mathbf{e}_j = (\_, r, \mathbf{o}_j)$ where $\mathbf{o}_j \in \{\mathtt{receive}(t'' : \rho''), \mathtt{commit}\}$ @ r. Using 24, we can show that $t'' : \rho'' \in D^r$. By hypothesis, $t' = \max_<(t'' \mid t'' : \rho'' \in D_{i-1} \wedge \mathsf{lastval}(x, t'' : \rho'') \neq \textbf{undefined}$; therefore either $t'' \leq t'$ or there is no record of the form $\mathtt{write}(x, n)$ occurs in ρ''.

Since we are assuming that $(E_{t''}, \mathsf{po}_{t''}) \vdash \mathsf{Write}\ x : \_$, there exists an event $f \in E_{t''}$ such that $f = (\_, \mathtt{write}(x, \_))$. By Definition 51, it follows that there exists an index $h < i$ such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_h) = t''$ and $\mathbf{e}_h = (\_, \_, \mathtt{write}(x, \_))$. We also have that $t'' : \rho'' = \mathsf{LogOf}(t'', \mathbf{e}_1 \cdots \mathbf{e}_i)$, By (Lemma 38), hence it has to be the case that $\mathtt{write}(x, \_)$ occurs in ρ''. Therefore, the only possibility is that $t'' \leq t'$. That is, either $t'' = t'$, from which we get that $(E_{t''}, \mathsf{po}_{t''}) = (E_{t'}, \mathsf{po}_{t'}$, or $t'' < t'$, and by Definition 51 it follows that

$(E_{t''}, \mathsf{po}_{t''}) \xrightarrow{\mathsf{AR}(\mathcal{C})} (E_{t'}, \mathsf{po}_{t'})$, as we wanted to prove. ◀

▶ Proposition 60. Let $(\mathbf{E}, \prec)$ be a concrete execution satisfying (**??**). Then $\mathsf{AbsExec}(\mathcal{C}) \models$ TransVis.

Proof. Let S, T, V be three transactions in $\mathsf{history}(\mathbf{E}, \prec)$, such that $S \xrightarrow{\mathsf{VIS}(\mathcal{C})} T \xrightarrow{\mathsf{VIS}(\mathcal{C})} V$. Then there exist three timestamps t, t', t'' such that $S = T_t, T = T_{t'}$ and $V = T_{t''}$ as per Definition 51. Also, since $T_t \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t'}$, there exist $\mathbf{e}_r, \mathbf{e}_{s'} \in \mathbf{E}$ such that $\mathbf{e}_r = (\_, r, \mathbf{o})$ with $\mathbf{o} \in \{\mathtt{commit}(t), \mathtt{receive}(t : \rho) \mid \rho \in \mathsf{UpdateList}\}$, and $\mathbf{e}_r \prec (\_, r, \mathtt{commit}(t'))$. Note that by Lemma 25, the event $(\_, r, \mathtt{commit}(t))$ is uniquely defined in $\mathbf{E}$.

Also, since $T_{t'} \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t''}$. there exist $\mathbf{f}_{r'} \in \mathbf{E}$ such that $\mathbf{f}_{r'} = (\_, r', \mathbf{o}')$ with $\mathbf{o}' \in \{\mathtt{commit}(t'), \mathtt{receive}(t' : \rho') \mid \rho' \in \mathsf{UpdateList}\}$ and $\mathbf{e}_{r'} \prec (\_, r', \mathtt{commit}(t''))$.

By Lemma 39 we get that $\mathbf{e}_r \prec (\_, r, \mathtt{commit}(t')) \preceq \mathbf{e}_{r'} \prec (\_, r', \mathtt{commit}(t''))$. If $r = r'$, By Definition 51 and the fact that $\mathbf{e}_r \prec (\_, r', \mathtt{commit}(t'')) = (\_, r, \mathtt{commit}(t''))$, we obtain that $T_t \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t''}$, as we wanted to prove.

Suppose then that $r' \neq r$; since we are assuming that $\mathcal{C}$ satisfies (CausalDeliv), and because $\mathbf{e}_r \prec (\_, r', \mathtt{commit}(t'))$, then there exists an event $\mathbf{e}'_r = (\_, r', \mathbf{o}'')$ with $\mathbf{o}'' \in \{\mathtt{commit}(t''), \mathtt{receive}(t'' : \_)\}$ such that $\mathbf{e}_{r'} \prec \mathbf{f}_{r'} \prec (\_, r', \mathtt{commit}(t''))$. By Definition 51, it follows that $T_t \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t''}$. ◀

▶ Proposition 61. Let $(\mathbf{E}, \prec)$ be a concrete execution satisfying (TotalDeliv). Then $\mathsf{AbsExec}((\mathbf{E}, \prec)) \models$ Prefix.

Proof. Let $T_t, T_{t'}, T_{t''} \in \mathsf{history}(\mathbf{E}, \prec)$, where $T_t, T_{t'}, T_{t''}$ are defined from t, t', t'' according to Definition 51, respectively. Suppose that $T_t \xrightarrow{\mathsf{AR}(\mathcal{C})} T_{t'} \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t''}$. We need to show that $T_t \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t''}$.

Since $T_t \xrightarrow{\mathsf{AR}(\mathcal{C})} T_{t'}$, by Definition 51 it follows that $t < t'$. Also, since $T_{t'} \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t''}$, we have that there exist two events $\mathbf{e}_{c'}, \in \mathbf{E}$ such that $\mathbf{e}_{c'} = (\_, r', \mathbf{o}')$ with $\mathbf{o}' \in \{\mathtt{commit}(t'), \mathtt{receive}(t' : \_)\}$, and $\mathbf{e}_{c'} \prec (\_, r', \mathtt{commit}(t''))$. Here it is also important to note that, because of Lemma 35, it has to be the case that $\mathbf{e}_{c'} \prec \mathtt{start}_{\mathcal{C}}(t')$, where $\mathtt{start}_{\mathcal{C}}(t') = (\_, r', \mathtt{start})$.

Since $T_t \in E$, then there exists at least one event in $\mathbf{e} \in \mathbf{E}$ such that $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t$. This is possible only if an event of the form $(\_, \_, \mathtt{commit}(t))$ appears in $\mathbf{E}$. Thus we have that **(a)** there exists an event of the form $(\_, \_, \mathtt{commit}(t))$ in $\mathbf{E}$, **(b)**, $\mathbf{e}_{c'} \prec \mathtt{start}_{\mathcal{C}}(t'')$, **(c)** $\mathtt{start}_{\mathcal{C}}(t'') = (\_, r', \mathtt{start})$ and **(d)** $t < t'$. Since $\mathcal{C}$ satisfies (TotalDeliv), it has to be the case that there exists an event $\mathbf{e}_c$ such that $\mathbf{e}_c = (\_, r', \mathbf{o})$ with $\mathbf{o} \in \{\mathtt{commit}(t), \mathtt{receive}(t : \_)\}$ and $\mathbf{e}_c \prec \mathtt{start}_{\mathcal{C}}(t'')$. It is immediate to observe now that it has to be the case that $\mathbf{e}_c \prec (\_, r', \mathtt{commit}(t''))$, and by Definition 51, $T_t \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t''}$. ◀

▶ Proposition 62. Let $(\mathbf{E}, \prec)$ be a concrete execution that satisfies ConflictCheck. Then $\mathsf{AbsExec}((\mathbf{E}, \prec)) \models$ NoConflict.

Proof. Let $T_t, T'_t \in \mathsf{history}((\mathbf{E}, \prec))$, where t, t' are timestamps such that $t \neq t'$ and T_t, T'_t are defined according to Definition 51. Suppose that $T_t \vdash \mathsf{Write}\ x : \_$ and $T_{t'} \vdash \mathsf{Write}\ x : \_$. We nedd to show that either $T_t \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t'}$ or $T_{t'} \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_t$.

Since both $T_t, T'_t \vdash \mathsf{Write}\ x : \_$, then $T_t = (E_t, \_), T_{t'} = (E_{t'}, \_)$ for some $E_t, E_{t'}$ such that there exist $e_t \in E_t, e_{t'} \in E_{t'}$ with $e_t = (\_, \mathtt{write}(x, \_)), e_{t'} = (\_, \mathtt{write}(x, \_))$.

By Definition 51, we have that there exist two events $\mathbf{e}_w, \mathbf{e}_{w'}$ such that $cce_w = (\_, r, \texttt{write}(x, \_))$, $\mathbf{e}_w = (, r', \texttt{write}(x, \_))$ and $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_w) = t$, $\mathsf{TS}_{\mathcal{C}}(\mathbf{e}_{w'}) = t'$. By Definition of $\mathsf{TS}_{\mathcal{C}}(\cdot)$, then there exists two unique events of the form $(\_, r, \texttt{commit}(t))$ and $(\_, r', \texttt{commit}(t'))$ appearing in $\mathbf{E}$.

since $\prec$ is total, either $(\_, r, \texttt{commit}(t)) \prec (\_, r, \texttt{commit}(t'))$ or $(\_, r', \texttt{commit}(t')) \prec (\_, r, \texttt{commit}(t))$ (recall that $t \neq t'$ by hypothesis). Without loss of generality, assume that $(\_, r, \texttt{commit}(t)) \prec (\_, r', \texttt{commit}(t'))$. We have two possible cases:

- $r' = r$; in this case it is immediate, by Definition 51, to obtain that $T_t \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t'}$,
- $r' \neq r$; in this case (ConflictCheck) ensures that there exists an event $\mathbf{e}_{rcv} \in \mathbf{E}$ such that $\mathbf{e}_{rcv} = (\_, r', \texttt{receive}(t : \_))$, It follows from Definition 51 that $T_t \xrightarrow{\mathsf{VIS}(\mathcal{C})} T_{t'}$.

◀

B.3 Recovering Concrete Executions from Abstract Ones

In the following, we use γ to range over computation fragments, i.e. $\gamma = (R^0, M^0) \overset{\mathbf{e}_1}{\to} (R^1, M^1) \overset{\mathbf{e}_2}{\to} \cdots \overset{\mathbf{e}_i}{\to} (R^i, M^i)$ for some (R^j, M^j), with $j = 0, \cdots, i$ and $\mathbf{e}_j$,with $j = 1, \cdots, i$. The concatenation of two execution fragments γ_1, γ_2, denoted as $\gamma_1 \cdot \gamma_2$, is defined whether $\gamma_1 = (R^0, M^0) \overset{\mathbf{e}_1}{\to} (R^1, M^1) \overset{\mathbf{e}_2}{\to} \cdots \overset{\mathbf{e}_i}{\to} (R^i, M^i)$, $\gamma_2 = (R^i, M^i) \overset{\mathbf{e}_{i+1}}{\to} (R^{i+1}, M^{i+1}) \overset{\mathbf{e}_{i+2}}{\to} \cdots \overset{\mathbf{e}_{i+j}}{\to} (R^{i+j}, M^{i+j})$ for some $(R_1, M_1), \cdots (R_{i+j}, M_{i+J}) \in \mathsf{RState}$; in this case we let $\gamma_1 \cdot \gamma_2 = (R^0, M^0) \overset{\mathbf{e}_1}{\to} (R^1, M^1) \overset{\mathbf{e}_2}{\to} \cdots \overset{\mathbf{e}_{i+j}}{\to} (R^{i+j}, M^{i+j})$.

We define a concrete execution, given an abstract one $(\mathcal{H}, \mathsf{VIS}, \mathsf{AR})$, by applying the following criteria: any two different transactions in $\mathcal{H}$ to be executed at different replicas; the timestamp of transactions are determined to be consistent with the arbitration relation AR; at any given time, at most one replica is executing a transaction (i.e. there is a global sequential order according to which transactions are executed by some replicas); before starting executing a transaction T at a given replica r, the latter will have received the effects of exactly those transactions which are visible to T (i.e. message delivery is defined to be consistent with the visibility relation). In the following we fix an abstract execution $\mathcal{A} = (\mathcal{H}, \mathsf{VIS}, \mathsf{AR})$. This is done to keep the notation easy; however, since we are not making any assumption about the structure of $\mathcal{A}$, all the results in the Section hold for an arbitrary abstract execution. Also, the notation introduced in the next Definition is maintained throughout this section.

▶ **Definition 63.** If $\mathcal{H} = \emptyset$, then we let $\mathsf{Recover}(\mathcal{A}) = (R_0, M_0)$, where we recall that $R_0 = \lambda r.(\emptyset, \mathsf{idle})$, $M_0 = \emptyset$. Here the pair (R_0, M_0) is meant as a computation fragment with no transitions. Otherwise, we define $\mathsf{Recover}(\mathcal{A})$ as follows:

- For each $T \in \mathcal{H}$, we let t_T, r_T to be a timestamp and replica identifier, respectively, such that $T \neq S$ implies $r_T \neq r_S$, and if $T \xrightarrow{\mathsf{AR}} S$ then $t_T < t_S$. We assume that $\mathsf{RId} = \{r_T \mid T \in \mathcal{H}\}$ (this is possible since $\mathcal{H}$ is either finite or countably infinite). For example, we can assume that $\mathsf{RId} = \mathbb{N} = \mathbb{N}$, and $t_T = r_T := |\{S \mid S \xrightarrow{\mathsf{AR}} T\}|$.
- Given $(E, \mathsf{po}) \in \mathcal{H}$, define $\rho_{(E,\mathsf{po})}$ is defined inductively as follows

$$\rho_{(E,\mathsf{po})} = \begin{cases} \varepsilon & \text{if } E = \emptyset \\ \rho_{(E\setminus\{e\}),\mathsf{po}|_{(E\setminus\{e\})}} & \text{where } e := \min_{\mathsf{po}}(E), \text{ is defined and } \mathsf{op}(e) = \texttt{read}(\_,\_) \\ \texttt{write}(x,n) \cdot \rho_{(E\setminus\{e\}),\mathsf{po}|_{(E\setminus\{e\})}} & \text{where } e := \min_{\mathsf{po}}(E), \text{ is defined and } \mathsf{op}(e) = \texttt{write}(x,n) \end{cases}$$

For any $T \in \mathcal{H}$, we define $l_T := t_T : \rho_T$.

- For any $T \in \mathcal{H}$, we define R_T so that, for any $r_S \in \mathsf{RId}$, $R_T(r_S) = (D_{T,S}, \mathsf{idle})$ where $D_{T,S} = \{l_{T'} \mid T' \xrightarrow{\mathsf{VIS}} S\}$ if $t_S < t_T$, $D_{T,S} = \emptyset$ otherwise.
- We let $M_T := \{l_{T'} \mid T' \xrightarrow{\mathsf{AR}} T\}$.
- For each $T = (E_T, \mathsf{po}_T) \in \mathcal{H}$, we define two execution fragments$\gamma_{T,\mathsf{rcv}}$ and $\gamma_{T,\mathsf{exec}}$ as follows:
  - we assume that the set of event identifiers is $\mathsf{Hist} \times \{\mathsf{rcv}, \mathsf{exec}\} \times \mathbb{N}$,
  - let $S_1, \cdots, S_n$ be the enumeration of $\mathsf{VIS}^{-1}(T)$ determined by AR. Note that VIS is prefix finite by hypothesis, hence $|\mathsf{VIS}^{-1}(T)| < +\infty$. Then, for any $i = 1, \cdots, n$ we let $(\mathbf{e}^i_{T,\mathsf{rcv}} = ((T, \mathsf{rcv}, i), r_T, \mathtt{receive}(t_{S_i} : \rho_{S_i}))$, $R^i_{T,\mathsf{rcv}}(r_S) := R_T(r_S)$, if $S \neq T$, $R^i_{T,\mathsf{rcv}}(r_T) := (\{l_{S_1}, \cdots, l_{S_i}\}, \mathsf{idle})$, and $M^i_{T,\mathsf{rcv}} = M_T$. We also let $R^0_{T,\mathsf{rcv}} := R_T, M^0_{T,\mathsf{rcv}} := M_T$. Then

$$\gamma_{T,\mathsf{rcv}} = (R^0_{T,\mathsf{rcv}}, M^0_{T,\mathsf{rcv}}) \overset{\mathbf{e}^1_{T,\mathsf{rcv}}}{\to} (R^1_{T,\mathsf{rcv}}, M^1_{T,\mathsf{rcv}}) \overset{\mathbf{e}^2_{T,\mathsf{rcv}}}{\to} \cdots \overset{\mathbf{e}^n_{T,\mathsf{rcv}}}{\to} (R^n_{T,\mathsf{rcv}}, M^n_{T,\mathsf{rcv}})$$

  - Let $R^0_{T,\mathsf{exec}} := R_T[r_T \mapsto (\{l_S \mid S \xrightarrow{\mathsf{VIS}} T\}, \mathsf{idle})]$. Take $M^0_{T,\mathsf{exec}} := M_T$. Let $(\mathbf{e}^1_{T,\mathsf{exec}} = ((T, \mathsf{exec}, 1), r_T, \mathtt{start})$. Define $R^1_{T,\mathsf{exec}} = R_T[r_T \mapsto (\{l_S \mid S \xrightarrow{\mathsf{VIS}} T\}, \varepsilon)]$, and $M^1_{T,\mathsf{exec}} = M_T$.
    Now, let $e^2_T, \cdots, e^n_T$ be the enumeration of E_T given by po_T. Without loss of generality, assume that for any $i = 2, \cdots, n$, $e^i_T = ((T, \mathsf{exec}, i), \_)$; note that if this is not the case we can still rename the identifiers of E_T in $\mathcal{A}$ so that $e^i_T = ((T, \mathsf{exec}, i), \_)$.
    If $e^i_T = (\_, o(x, n))$, where $o \in \{\mathtt{read}, \mathtt{write}\}$, we let $\mathbf{e}^i_{T,\mathsf{exec}} := (\iota, r_T, o(x, n))$. We also let $R^i_{T,\mathsf{exec}} := R_T[r_T \mapsto (\{l_S \mid S \xrightarrow{\mathsf{VIS}} T\}, \rho_{\{e^2_T, \cdots, e^i_T\}, \mathsf{po}|_{\{e^2_T, \cdots, e^i_T\}}})]$, $M^i_{T,\mathsf{exec}} := M_T$.
    Finally, let $\mathbf{e}^{n+1}_{T,\mathsf{exec}} := ((T, \mathsf{exec}, n+1), r_T, \mathtt{commit}(t_T))$, and define $R^{n+1}_{T,\mathsf{exec}} = R_{T,\mathsf{exec}}[r_T \mapsto ((\{l_T\} \cup \{l_S \mid S \xrightarrow{\mathsf{VIS}} T\}), \mathsf{idle})]$, $M^{i+1}_{T,\mathsf{exec}} := M_T \cup \{l_T\}$. Then

$$\gamma_{T,\mathsf{exec}} = (R^0_{T,\mathsf{exec}}, M^0_{T,\mathsf{exec}}) \overset{\mathbf{e}^1_{T,\mathsf{rcv}}}{\to} (R^1_{T,\mathsf{exec}}, M^1_{T,\mathsf{exec}}) \overset{\mathbf{e}^2_{T,\mathsf{rcv}}}{\to} \cdots \overset{\mathbf{e}^{n+1}_{T,\mathsf{rcv}}}{\to} (R^{n+1}_{T,\mathsf{exec}}, M^{n+1}_{T,\mathsf{exec}})$$

Suppose that $T_1, \cdots, T_n$ (with possibly $n = \infty$ is the enumeration of $\mathcal{H}$ given by AR. For $i = 1, \cdots, n$ we let $\gamma_{T_i} := \gamma_{T_i,\mathsf{rcv}} \cdot \gamma_{T_i,\mathsf{exec}}$, and we define $\mathsf{Recover}(\mathcal{A}) = \gamma_{T_1} \cdots \gamma_{T_n}$.

▶ Proposition 64. The execution $\mathsf{Recover}(\mathcal{A})$ is well defined.

Proof. Let $T_1, \cdots, T_n$ be the enumeration of $\mathcal{H}$ given by AR; note that possibly $n = \infty$. We first show that,

- for each $i = 1, \cdots, n$, the computation fragment $\gamma_{T_i,\mathsf{rcv}} \cdot \gamma_{T_i,\mathsf{exec}}$ is well defined,
- for any $i = 1, \cdots, n-1$, the computation fragment $\gamma_{T_i,\mathsf{exec}} \cdot \gamma_{T_{i+1},\mathsf{rcv}}$ is well defined.

We only give the details for the first statement, as the second one can be proved similarly.

Let then $i = 1, \cdots, n$. Let also $k := |\{S \mid S \xrightarrow{\mathsf{VIS}} T_i\}$. By definition, we have that

$$\gamma_{T_i,\mathsf{rcv}} = (R^0_{T_i,\mathsf{rcv}}, M^0_{T_i,\mathsf{rcv}}) \overset{\mathbf{e}^1_{T_i,\mathsf{rcv}}}{\to} \cdots \overset{\mathbf{e}^{|\mathsf{VIS}^{-1}(T_i)|}_{T_i,\mathsf{rcv}}}{\to} (R^{|\mathsf{VIS}^{-1}(T_i)|}_{T_i,\mathsf{rcv}}, M^{|\mathsf{VIS}^{-1}(T_i)|}_{T_i,\mathsf{rcv}}),$$

while

$$\gamma_{T_i,\mathsf{exec}} = (R^0_{T_i,\mathsf{exec}}, M^0_{T_i,\mathsf{exec}}) \overset{\mathbf{e}^1_{T_i,\mathsf{rcv}}}{\to} \cdots \overset{\mathbf{e}^{|E_{T_i}|+2}_{T_i,\mathsf{exec}}}{\to} (R^{|E_{T_i}|+2}_{T_i,\mathsf{exec}}, M^{|E_{T_i}|+2}_{T_i,\mathsf{exec}}).$$

In order to prove that $\gamma_{T_i,\mathsf{rcv}} \cdot \gamma_{T_i,\mathsf{exec}}$ is well-defined, it suffices to prove that $(R^{|\mathsf{VIS}^{-1}(T_i)|}_{T_i,\mathsf{rcv}}, M^{|\mathsf{VIS}^{-1}(T_i)|}_{T_i,\mathsf{rcv}}) = (R^0_{T_i,\mathsf{exec}}, M^0_{T_i,\mathsf{exec}})$.

Note that, for any $r \in \mathsf{RId}, r \neq r_{T_i}$, we have that $r = r_S$ for some $S \neq T_i$; also, in this case we have that $R^0_{T_i,\mathsf{exec}}(r_S) = R_{T_i}(r_S) = R^{|\mathsf{VIS}^{-1}(T_i)|}_{T_i,\mathsf{rcv}}(r_S)$. For $r = r_{T_i}$, we have that $R^0_{T_i,\mathsf{exec}}(r_{T_i}) = (\{l_S \mid S \xrightarrow{\mathsf{VIS}} T_i\}, \mathsf{idle})$; on the other hand, $R^{|\mathsf{VIS}^{-1}(T_i)|}_{T_i,\mathsf{rcv}}(r_{T_i}) = (\{l_{S_1}, \cdots, l_{S_{|\mathsf{VIS}^{-1}(T_i)|}}\}, \mathsf{idle})$, where $S_1, \cdots, S_{|\mathsf{VIS}^{-1}(T_i)|}$ correspond to the enumeration of $\mathsf{VIS}^{-1}(T_i)$ induced by AR. Therefore, $\{l_{S_1}, \cdots, l_{S_{|\mathsf{VIS}^{-1}(T_i)|}}\} = \{l_S \mid S \xrightarrow{\mathsf{VIS}} T_i\}$, hence $R^{|\mathsf{VIS}^{-1}(T_i)|}_{T_i,\mathsf{rcv}}(r_{T_i}) = (\{l_S \mid S \xrightarrow{\mathsf{VIS}} T_i\}, \mathsf{idle}) = R^0_{T_i,\mathsf{exec}}(r_{T_i})$.

Finally, by construction we have that $M^0_{T_i,\mathsf{exec}} = M_{T_i} = M^{|\mathsf{VIS}^{-1}(T_i)|}_{T_i,\mathsf{rcv}}$, and there is nothing left to prove.

◀

▶ Proposition 65. Recall that $R_0 = \lambda r.(\emptyset, \mathsf{idle})$, and $M_0 = \emptyset$. Then either $\mathsf{Recover}(\mathcal{A}) = (R_0, M_0)$ or $\mathsf{Recover}(\mathcal{A}) = (R_0, M_0) \xrightarrow{\mathbf{e}_1} \cdots$ for some event $\mathbf{e}_1$.

Proof. If $\mathcal{H} = \emptyset$, then by definition $\mathsf{Recover}(\mathcal{A}) = (R_0, M_0)$. Suppose then that $\mathcal{H} \neq \emptyset$. By definition, $\mathsf{Recover}(\mathcal{A}) = \gamma_{T_1} \cdots$, where $T_1, \cdots$ is the enumeration of transactions in $\mathcal{H}$, according to AR. We show that $\gamma_{T_1} = (R_0, M_0) \xrightarrow{\mathbf{e}_1} (R_1, M_1) \cdot \gamma'$. By Definition 63 we have that $\mathsf{RId} = \{r_S \mid S \in \mathcal{H}\}$, and $\gamma_{T_1} = (R^0_T, M^0_T)$, where $R^0_T(r_S) = (\{l_{T'} \mid T' \xrightarrow{\mathsf{VIS}} S\}, \mathsf{idle})$ if $S \xrightarrow{\mathsf{AR}} T$, $(\emptyset, \mathsf{idle})$ otherwise. Note that $T_1 = \min_{\mathsf{AR}}(\mathcal{H})$, so that $\mathsf{AR}^{-1}(T_1) = \emptyset$, hence for any $S \in \mathcal{H}$, it follows that $R^0_T(r_S) = (\emptyset, \mathsf{idle})$, or equivlently $R^0_T = \lambda r.(\emptyset, \mathsf{idle}) = R_0$. Also $M^T_0 = \emptyset = M_0$. ◀

▶ Proposition 66. Let $\mathbf{e}, \mathbf{f}$ be two events in $\mathsf{Recover}(\mathcal{A})$ such that $\mathbf{e} = (\_, \_, \mathtt{start}), \mathbf{f} = (\_, \_, \mathtt{start})$. Then there exists two transacitons T, S such that $\mathbf{e} = \mathtt{start}_{\mathsf{Recover}(\mathcal{A})}(t_T)$, $\mathbf{f} = \mathtt{start}_{\mathsf{Recover}(\mathcal{A})}(t_S)$.

As an immediate consequence, if $S = T$ then $\mathbf{e} = \mathbf{f}$.

Proof. By definition, if $\mathbf{e} = (\_, \_, \mathtt{start})$, then there exists a transaction $T \in \mathcal{H}$ such that $\mathbf{e} = \mathbf{e}^1_{T,\mathsf{exec}}$. This implies that $\mathbf{e} = (\_, r_T, \mathtt{start})$. Now note that there is a unique event $\mathbf{g} = (\_, r_T, \mathtt{commit}(\_))$ appearing in $\mathsf{Recover}(\mathcal{A})$; such an event is exactly $\mathbf{e}^{|E_T|+2}_{T,\mathsf{exec}}$, and in particular we have that $\mathbf{e}^{|,E_T|+2}_{T,\mathsf{exec}} = (\_, r_T, \mathtt{commit}(t_T))$.

Both the events $\mathbf{e} = \mathbf{e}^1_{T,\mathsf{exec}}$ and $\mathbf{g} = \mathbf{e}^{vertE_T|+2}_{T,\mathsf{exec}}$ appear in the computation fragment γ_T, and in particularly $\mathbf{e}^1_{T,\mathsf{exec}}$ precedes $\mathbf{e}^{|E_T|+2}_{T,\mathsf{exec}}$; that is, $\mathbf{e}^1_{T,\mathsf{exec}} \prec \mathbf{e}^{|E_T|+2}_{T,\mathsf{exec}}$. Since the event $\mathbf{e}^{|E_T|+2}_{T,\mathsf{exec}}$ is the unique `commit` event that takes place at replica r_T, and the transaction identifier of such a commit operation is t_T, it follows immediately that $\mathbf{e} = \mathtt{start}_{\mathsf{Recover}(\mathcal{A})}(t_T)$. Analogously, it can be proved that $\mathbf{f} = \mathtt{start}_{\mathsf{Recover}(\mathcal{A})}(t_S)$.

that $\mathbf{e} = \mathbf{f}$. ◀

▶ Proposition 67. Each of the transitions appearing in $\mathsf{Recover}(\mathcal{A})$ can be inferred from the rules given in Figure 4.

Proof. If $\mathcal{H} = \emptyset$, then there is nothing to prove. Suppose then that $\mathcal{H} \neq \emptyset$. We assume that $T_1, \cdots, T_n$ (with possibly $n = \infty$) is the enumeration of $\mathcal{H}$ according to AR. Recall that $\mathsf{Recover}(\mathcal{A}) = \gamma_{T_1} \cdots \gamma_{T_n}$, where for each $i = 1, \cdots, n$, $\gamma_{T_i} = \gamma_{T_i,\mathsf{rcv}} \cdot \gamma_{T_i,\mathsf{exec}}$. Fix an index i in the interval $[1, n]$. We show that any transition $(R, M) \xrightarrow{\mathbf{e}} (R', M')$ which appears in γ_{T_i} can be inferred by the rules given in Figure 4, by performing a case analysis on whether the transition occurs in $\gamma_{T_i,\mathsf{rcv}}$ or in $\gamma_{T_i,\mathsf{exec}}$.

- Suppose that the transition $(R, M) \xrightarrow{\mathbf{e}} (R', M')$ occurs in $\gamma_{T_i,\mathsf{rcv}}$. Let $S_i, \cdots, |\mathsf{VIS}^{-1}(T_i)|$ be the enumeration of $\mathsf{VIS}^{-1}(T_i)$ induced by AR. Then it has to be the case that there exists an index $j = 1, \cdots, |\mathsf{VIS}^{-1}(T_i)|$ such that $(R, M) = (R^{j-1}_{T_i,\mathsf{rcv}}, M^{j-1}_{T_i,\mathsf{rcv}})$, $(R', M') = (R^{j}_{T_i,\mathsf{rcv}}, M^{j}_{T_i,\mathsf{rcv}})$ and $\mathbf{e} = \mathbf{e}^j_{T_i,\mathsf{rcv}} = (\_, r_{T_i} = \mathtt{receive}(t_{S_j} : \rho_{S_j})$.
  By Definition, $M^{j-1}_{T_i,\mathsf{rcv}} = M_{T_i} = \{t_{T'} : \rho_{T'} \mid T' \xrightarrow{\mathsf{AR}} T_i\}$. Since $S_j \xrightarrow{\mathsf{VIS}} T_i$ and $\mathsf{VIS} \subseteq \mathsf{AR}$, then $S_j \xrightarrow{\mathsf{AR}} T_i$, hence $t_{S_j} : \rho_{S_j} \in M^{j-1}_{T_i,\mathsf{rcv}} = M$. Also, we have that $M' = M^j_{T_i,\mathsf{rcv}} = M_{T_i} = M^{j-1}_{T_i,\mathsf{rcv}} = M$.
  Note also that for any $r' \neq r_{T_i}$, we have that $r' = R_{T'}$ for some $T' \neq T$. If $T_i \xrightarrow{\mathsf{AR}} T'$, then $R^{j-1}_{T_i,\mathsf{rcv}}(r_{T'}) = (\emptyset, \mathsf{idle}) = R^j_{T_i,\mathsf{rcv}}(r_T)$; if $T' \xrightarrow{\mathsf{AR}} T_i$, then $R^{j-1}_{T_i,\mathsf{rcv}}(r_{T'}) = (\{l_{T''} \mid T'' \xrightarrow{\mathsf{VIS}} T'\}, \mathsf{idle}) = R^j_{T_i,\mathsf{rcv}}(r_{T'})$. Therefore, whenever $r \in \mathsf{Rld}$ is such that $r \neq r_{T_i}$, then $R(r) = R^{j-1}_{T_i,\mathsf{rcv}}(r) = R^j_{T_i,\mathsf{rcv}}(r) = R$. Finally, we have that $R^j_{T_i,\mathsf{rcv}}(r_{T_i}) = (\{(t_{S_1} : \rho_{S_1}), \cdots, (t_{S_{j-1}} : \rho_{S_{j-1}})\}, \mathsf{idle})$ and $R^j_{T_i,\mathsf{rcv}}(r_{T_i}) = (\{(t_{S_1} : \rho_{S_1}), \cdots, (t_{S_j} : \rho_{S_j})\}, \mathsf{idle})$. If we let $D = \{(t_{S_1} : \rho_{S_1}), \cdots, (t_{S_{j-1}} : \rho_{S_{j-1}})\}$, then it follows that $R(r_{T_i}) = R^{j-1}_{T_i,\mathsf{rcv}}(r_{T_i}) = (D, \mathsf{idle})$. Observe that, since $R(r_{T_i}) = (D, \mathsf{idle})$, we have that $R = R[r_{T_i} \mapsto (D, \mathsf{idle})]$. Since for any $r \neq r_{T_i}$ we also have that $R'(r) = R(r)$, and $R'(r_{T_i}) = (D \cup \{t_{S_j} : \rho_{S_j}\}, \mathsf{idle})$, it follows that $R' = R[r_{T_i} \mapsto (D \cup \{t_{S_j} : \rho_{S_j}\}, \mathsf{idle})$.
  We have proved that $\mathbf{e} = (\_, r_{T_i}, \mathtt{receive}(t_{S_j} : \rho_{S_j})$, $R = R[r_{T_i} \mapsto (D, \mathsf{idle})$ for some set of transaction logs D, $R' = R[r_{T_i} \mapsto (D \cup \{t_{S_j} : \rho_{S_j})$, and $M = M'$. Therefore, we can apply Rule (Receive) from Figure 4 and infer $(R, M) \xrightarrow{\mathbf{e}} (R', M')$, as we wanted to prove.
- Suppose that the transition $(R, M) \xrightarrow{\mathbf{e}} (R', M')$ occurs in $\gamma_{T_i,\mathsf{exec}}$. in this case we have that $\mathbf{e} = (\_, r_{T_i}, \mathbf{o})$ where , where $\mathbf{o} \in \{\mathtt{start}, \mathtt{commit}(t_{T_i}), \mathtt{write}(\_, \_), \mathtt{read}(\_, \_)\}$.
  We perform a case analysis on $\mathbf{o}$:
  - $\mathbf{o} = \mathtt{start}$; this is possible only if $\mathbf{e} = \mathbf{e}^0_{T_i,\mathsf{exec}}$. It follows that $(R, M) = (R^0_{T_i,\mathsf{exec}}, M^0_{T_i,\mathsf{exec}})$, $(R', M') = (R^1_{T_i,\mathsf{exec}}, M^1_{T_i,\mathsf{exec}})$. We can proceed as in the case above to prove that $M = M_{T_i} = M'$, and for any $r \neq r_{T_i}$ we have that $R(r) = R'(r)$. We also have that $R(r_{T_i}) = (\{l_S \mid S \xrightarrow{\mathsf{VIS}} T\}, \mathsf{idle})$ and $R'(r_{T_i}) = (\{l_S \mid S \xrightarrow{\mathsf{VIS}^{-1}} T\}, \varepsilon)$. If we let $D = \{l_S \mid S \xrightarrow{\mathsf{VIS}} T\}$, we can rewrite $R = R[r_{T_i} \mapsto (D, \mathsf{idle})]$, $R' = R[r_{T_i} \mapsto (D, \varepsilon)]$. Now we can infer $(R, M) \xrightarrow{\mathbf{e}} (R', M')$ by applying Rule (Start) from Figure 4.
  - $\mathbf{o} = \mathtt{commit}(t_{T_i})$; by construction, it has to be the case that $\mathbf{e} = \mathbf{e}^{|E_{T_i}|+2}_{T_i,\mathsf{exec}}$, $(R, M) = (R^{|E_{T_i}|+1}_{T_i,\mathsf{exec}}, M^{|E_{T_i}|+1}_{T_i,\mathsf{exec}})$ and $(R', M') = (R^{|E_{T_i}|+2}_{T_i,\mathsf{exec}}, M^{|E_{T_i}|+2}_{T_i,\mathsf{exec}})$.
    Note that whenever $r \in \mathsf{Rld}$, and $R(r) = (D_r, \_)$ for some D_r, then for any $l \in D_r$ we have that $l = t_{T_j} : \rho_{T_j}$ for some $j < i$. This is because, if $l \in D_r$, then $D_r \neq \emptyset$, and by construction this means that $r = r_{T'}$ for some $T' \in \mathcal{H}$ such that $T' \xrightarrow{\mathsf{AR}} T_i$. Furthermore, the fact that $t_{T_j} : \rho_{T_j} \in D_r$ implies that $T_j \xrightarrow{\mathsf{AR}} T'$, and by transitivity of AR we get that $T_j \xrightarrow{\mathsf{AR}} T_i$. Since $T_1 \cdots T_n$ is the sequence of transactions appearing in $\mathcal{H}$, induced by AR, it has to be the case that $j < i$. Also, by construction it has to be the case that $t_{T_j} < t_{T_i}$. We proved that whenever $R(r) = (\{t : \_\} \cup \_, \_)$, then $t < t_{T_i}$, and in particular $t_{T_i} \neq t$ for any t such that $R(r) = (\{t : \_\} \cup \_, \_)$, and $t_{T_i} > t$ for any t such that $R(r_{T_i}) = (\{t : \_\} \cup \_, \_)$.
    Next, observe that we can proceed as in the previous cases, and prove that for any $r \neq r_{T_i}$, it has to be $R(r) = R'(r)$. For $r = r_{T_i}$, by construction we have that $R(r_{T_i}) = (\{t_S : \rho_S \mid S \xrightarrow{\mathsf{VIS}} T_i\}, \rho_{T_i})$, $R'(r_{T_i}) = (\{t_S : \rho_S \mid S \xrightarrow{\mathsf{VIS}} T_i\} \cup \{t_{T_i} : \rho_{T_i}\}, \mathsf{idle})$; if we let $D = \{t_S : \rho_S \mid S \xrightarrow{\mathsf{VIS}} T_i\}$, then we obtain that $R = R[r_{T_i} \mapsto (D, \rho_{T_i})$, $R' = R[r_{T_i} \mapsto \{t_{T_i} : \rho_{T_i}\}, \mathsf{idle})$.

Finally, by construction we have that $M = \{t_{T_1} : \rho_{T_1}, \cdots, t_{T_{i-1}} : \rho_{T_{i-1}}\}$, while $M' = \{t_{T_1} : \rho_{T_1}, \cdots, t_{T_i} : \rho_{T_i}\}$; hence, $M' = M \cup \{t_{T_i} : \rho_{T_i}\}$.
We have proved all the premises necessary to apply Rule (Commit) from Figure 4, from which we infer the transition $(R, M) \xrightarrow{\mathbf{e}} (R', M')$.

- $\mathbf{o} = \texttt{write}(x, n)$ where $x \in \mathsf{Obj}$ and $n \in \mathbb{N}$. Let $T_i = (E_i, \mathsf{po}_i)$ In this case we have that $\mathbf{e} = cce^{h+1}_{T,\mathsf{exec}}$ for some $h = 1, \cdots, |E_i|$. Also, if we let $e_1, \cdots, e^{|E_i|}$ be the enumeration of E_i according to po_i, we also have that $e_h = (\_, \texttt{write}(x, n))$.
  In this case we have that $R = R_{T_i}[r_{T_i} \mapsto (\{l_S \mid S \xrightarrow{\mathsf{VIS}} T_i\}, t_{T_i} : \rho)]$, $\rho = \rho_{\{e_1,\cdots,e_{h-1}\},\mathsf{po}_i|_{\{e_1,\cdots,e_{h-1}\}}}$. It is also the case that $R' = R_{T_i}[r_{T_i} \mapsto \{l_S \mid \xrightarrow{\mathsf{VIS}} T_i\} t_{T_i} : \rho')]$, where $\rho' = \rho_{\{e_1,\cdots,e_h\},\mathsf{po}_i|_{\{e_1,\cdots,e_h\}}}$. Since we already argued that $e_h = (\_, \texttt{write}(x, n))$, it is straightforward to see that in this case we have that $\rho' = \rho \cdot \texttt{write}(x, n)$.
  Finally, we have that $M = M' = M_{T_i}$, where M_{T_i} is given by Definition 63. Putting all this information together, we find that the transition $(R, M) \xrightarrow{\mathbf{e}} (R', M')$ can be inferred via an application of Rule (Write) from Figure 4.
- $\mathbf{o} = \texttt{read}(x, n)$; this is the last, and most interesting case. Let $T_i = (E_i, \mathsf{po}_i)$. We have that $\mathbf{e} = \mathbf{e}^{h+1}_{T,\mathsf{exec}}$ for some $h = 1, \cdots, |E_i|$. Also, let $e_1, \cdots, e_{|E_i|}$ be the enumeration of E_i given by po_i. Then, we have that for any $j = 1, \cdots, |E_i|$, $e_j = (\_, \texttt{read}(y, m))$ if and only if $\mathbf{e}^j_{T_i,\mathsf{exec}} = (\_, r_{T_i}, \texttt{read}(y, m))$; a similar statement follows for the case $e_j = (\_, \texttt{write}(y, m))$.
  In this case we have that $R = R_{T_i}[r_{T_i} \mapsto (\{l_S \mid S \xrightarrow{\mathsf{VIS}} T_i\}, t_{T_i} : \rho)$, where $\rho = \rho_{\{e_1,\cdots,e_{h-1}\},\mathsf{po}_i|_{\{e_1,\cdots,e_{h-1}\}}}$. We also have that $R' = R_{T_i}[r_{T_i} \mapsto (\{l_S \mid S \xrightarrow{\mathsf{VIS}} T_i\}, t_{T_i} : \rho'])$, where $\rho' = \rho_{\{e_1,\cdots,e_{h-1}\},\mathsf{po}_i|_{\{e_1,\cdots,e_{h-1}\}}}$. In this case it is not difficult to see that $\rho = \rho'$.
  Since Definition 63 ensures that $M = M' = M_{T_i}$, where M_{T_i} is defined according to Definition 51, we have that $(R, M) = (R', M')$.
  Next, we show that $\mathsf{lastval}(x, \{l_S \mid S \xrightarrow{\mathsf{VIS}} T_i\} \cup \{\infty : \rho\}) = n$. There are three possible cases:
  - there exists an index $k < h$ such that $\mathbf{e}^k_{T_i,\mathsf{exec}} = (\_, r_{T_i}, \texttt{write}(x, m))$. In this case $\mathsf{lastval}(x, \{l_S \mid S \xrightarrow{\mathsf{VIS}} T_i\} \cup \{\infty : \rho\} = \mathsf{lastval}(x, t_{T_i} : \rho)$. Without loss of generality, assume that there exists no index k' such that $k < k' < h$ and $\mathbf{e}^{k'}_{T_i,\mathsf{exec}} \triangleright t_{T_i} : \texttt{write}(x, \_) \mathbin{@} r_{T_i}$. In this case we have that $\mathsf{lastval}(x, t_{T_i} : \rho) = m$. Also, it is the case that $e_k \in E_{T_i}$, $e_k = (\_, \texttt{write}(x, m))$ and there exists no index k' such that $k < k' < h$ and $\mathsf{op}(k') = \texttt{write}(x, \_)$. Axiom Int now gives $m = n$, as we wanted to prove.
  - there exists no index $k < h$ such that $\mathbf{e}^k_{T_i,\mathsf{exec}} \triangleright t_{T_i} : \texttt{write}(x, m) \mathbin{@} r_{T_i}$. Note that this means that there exists no index $k < h$ such that $e_k = (\_, \texttt{write}(x, \_))$, and by definition we get that $T_i \vdash \mathsf{Read}\ x : n$. However, assume that there exists a transaction S such that $S \xrightarrow{\mathsf{VIS}} T_i$ and $\mathsf{lastval}(x, l_S)$ is defined. Without loss of generality, assume that $t_S = \max_<\{t_{T'} \mid T' \xrightarrow{\mathsf{VIS}} T_i \wedge \mathsf{lastval}(x, l_{T'})$ is defined$\}$. By definition, we have that $\mathsf{lastval}(x, \{l_{T'} \mid T' \xrightarrow{\mathsf{VIS}} T_i\} \cup \{t_{T_i} : rho\}\} = \mathsf{lastval}(x, l_S)$. Also, note that there exists no transaction T' such that $T' \xrightarrow{\mathsf{VIS}} T_i$, and $S \xrightarrow{\mathsf{AR}} T'$, since otherwise we would have $t_S < t_{T'}$; therefore, $S = \max_{AR}\{T' \vdash T' \xrightarrow{\mathsf{VIS}} T_i\}$. Since $T_i \vdash \mathsf{Read}\ x : n$, Axiom Ext ensures that $S \vdash \mathsf{Write}\ x : n$. If we let $S = (E_s, \mathsf{po}_S)$, and we let $f_1, \cdots, f_{|S|}$ be the enumeration of E_S given by po_S, then there exists an index k such that $f_k = (\_, \texttt{write}(x, n))$ and for no k' such

that $k' < k \leq |E_S|$, $f_{k'} = (\_, \texttt{write}(x, \_))$. It remains to note that $l_S = t_S : \rho_S$, from which it is easy to see that $\rho_S = \rho_{\mathsf{pre}} \cdot \texttt{write}(x, n) \cdot \rho_{\mathsf{post}}$, for some $\rho_{\mathsf{pre}}, \rho_{\mathsf{post}}$ such that no record of the form $\texttt{write}(x, \_)$ occurs in ρ_{post}. Now we obtain that $\mathsf{lastval}(t_S : \rho_S) = n$, as we wanted to prove.

* There exists no $k < h$ such that $\mathbf{e}^k_{T_i,\mathsf{exec}} \triangleright t_{T_i} : \texttt{write}(x, m)$ @ r_{T_i}; further, assume that there exists no transaction S such that $S \xrightarrow{\mathsf{VIS}} T_i$ and $\mathsf{lastval}(x, l_S)$ is defined. In this case we have that $\mathsf{lastval}(x, \{l_S \mid S \xrightarrow{\mathsf{VIS}} T_i\} \cup \{t_{T_i} : \rho\}\} = 0$. It is also easy to see that if $\mathsf{lastval}(x, l_S)$ is undefined for some transaction $S = (E_s, \mathsf{po}_S)$, then there exists no event $f \in E_s$ such that $f = (\_, \texttt{write}(x, \_))$. That is, whenever $S \xrightarrow{\mathsf{VIS}} T_i$ we get that $\neg(S \vdash \mathsf{Write}\ x : \_)$, and by Axiom Ext it follows that $n = 0$.

We can now combine all the facts above to show that the transition $(R, M) \xrightarrow{\mathbf{e}} (R', M')$ can be inferred via an application of Rule (Read) from Figure 4. In fact we have ◀

▶ Proposition 68. $\mathsf{Recover}(\mathcal{A})$ satisfies (MonTS).

Proof. Let $\mathbf{e}, \mathbf{f}$ be two event appearing in $\mathsf{Recover}(\mathcal{A})$ such that $\mathbf{e} = (\_, \_, \texttt{commit}(t))$, $\mathbf{f} = (\_, \_, \texttt{commit}(t'))$ and $\mathbf{e} \prec \mathbf{f}$. We show that $t < t'$.

By construction, $\mathbf{e} = (\_, \_, \texttt{commit}(t_{T_i}))$ for some $T_i \in \mathcal{H}$; note that the event $\mathbf{e}$ appears in γ only in the computation fragment γ_{T_i}. Similarly, $\mathbf{f} = (\_, \_, \texttt{commit}(t_{T_j}))$ for some $T_j \in \mathcal{H}$, and the event $\mathbf{f}$ appears in γ only in the computation fragment γ_{T_j}. Since $\mathbf{e} \prec \mathbf{f}$, it has to be the case that $\gamma = \gamma' \cdot \gamma_{T_i} \cdot \gamma'' \cdot \gamma_{T_j} \cdot \gamma'''$ for some $\gamma', \gamma'', \gamma'''$. In particular, we have that $T_i \xrightarrow{\mathsf{AR}} T_j$, and by construction it follows that $t_{T_i} < t_{T_j}$. ◀

▶ Proposition 69. If $\mathcal{A} \models$ TransVis, then $\mathsf{Recover}(\mathcal{A})$ satisfies (CausalDeliv).

Proof. Suppose that $\mathbf{e}_1, \mathbf{e}_2, \mathbf{f}_2 \in \mathbf{E}$ are such that $(\mathbf{e}_1 \in \{(\_, r_1, \texttt{commit}(t_1)), (\_, r, \texttt{receive}(t_1 : \_))\}$,

$(\mathbf{e}_2 = (\_, r_1, \texttt{commit}(t_2))$ and $\mathbf{f}_2 = (\_, r_2, \texttt{receive}(t_2 : \_))$ for some t_1, t_2 and r_1, r_2 such that $r_1 \neq r_2$. Suppose also that $\mathbf{e}_1 \prec \mathbf{e}_2$; this implies that it cannot be $(\mathbf{e}_1 = (\_, r_1, \texttt{commit}(t_1))$, since by Definition 63 only one transaction can be executed at replica r; this is the transaction t_2, as established by the existence of the event $\mathbf{f}_2 = (\_, r, \texttt{commit}(t_2))$; further, since $\mathbf{e}_1 \prec \mathbf{e}_2$, it has to be $t_1 < t_2$. Therefore it has to be $\mathbf{e}_1 = (\_, r, \texttt{receive}(t_1 : \_))$.

Now, note that by Definition 51, it has to be the case that there exists three transactions $S, T, V \in \mathcal{H}$ such that $t_1 = t_S, t_2 = t_T$, and $r_2 = r_V$. Note that $t_1 = t_S < t_T = t_2$, which is possible only if $T \xrightarrow{\mathsf{AR}} S$. Also, we have that $(\mathbf{e}_1 = (\_, r_1, \texttt{receive}(t_S : \_))$, $\mathbf{e}_2 = (\_, r_1, \texttt{commit}(t_T))$; by construction, this is possible only if $\mathbf{e}_1 \prec \mathbf{e}_2$, which in turn is possible only if $S \xrightarrow{\mathsf{VIS}} T$. This also implies that $S \xrightarrow{\mathsf{AR}} T$. Also, we have that $\mathbf{e}_{f_2} = (\_, r_V, \texttt{receive}(t_T : \_))$; by construction, there exists an event $\mathbf{e}_{f_3} = (\_, r_V, \texttt{commit}(t_V))$ such that $\mathbf{e}_{f_2} \prec \mathbf{e}_{f_3}$ (recall that, at least informally, r_V is the replica where transaction V is executed). This is possible only if $T \xrightarrow{\mathsf{VIS}} V$.

We have proved that $S \to \mathsf{VIS}T \xrightarrow{\mathsf{VIS}} V$, and since VIS is transitive by hypothesis, $S \xrightarrow{\mathsf{VIS}} T$. Therefore, there exists an event $\mathbf{f}_1 = (\_, R_V \texttt{receive}(t_S : \_))$ such that $\mathbf{f}_1 \prec \mathbf{f}_3$. Also, since $S \xrightarrow{\mathsf{AR}} T$, by construction it has to be the case that $\mathbf{f}_1 \prec \mathbf{f}_2$. ◀

▶ Proposition 70. If $\mathcal{A} \models$ Prefix, then $\mathsf{Recover}(\mathcal{A})$ satisfies (TotalDeliv).

Proof. Suppose that $\mathbf{e}, \mathbf{f}, \mathbf{g} \in \mathbf{E}$ are such that $(\mathbf{e} \in \{(\_, r, \texttt{commit}(t)), (\_, r, \texttt{receive}(t : \_))\}$, $(\mathbf{f} = (\_, r, \texttt{start})$ and $\mathbf{g} = (\_, r', \texttt{commit}(t'))$ for some t, t', t'' and r, r'. Suppose also that $\mathbf{e} \prec \mathbf{f}$; this implies that it cannot be $\mathbf{e} = (\_, r, \texttt{commit})$, since Definition 63 ensures that only one transaction can be executed at replica r - this is the transaction which starts with the event $\mathbf{f}$, nothing that obviously it cannot be $\mathbf{f} = \texttt{start}_{\mathsf{Recover}(\mathcal{A})}(t)$, since ; $\mathbf{e} \prec \mathbf{f}$. In the following, we let $\mathbf{f} = \texttt{start}_{\mathsf{Recover}(\mathcal{A})}(t'')$ for some $t'' \in \mathbb{N}$. Note that we can assume this since, by construction, any transaction in $\mathsf{Recover}(\mathcal{A})$ eventually commits.

Therefore it has to be $\mathbf{e} = (\_, r, \texttt{receive}(t : \_))$. By Definition 51, it has to be the case that there exist three transactions $S, T, V \in \mathcal{H}$ such that $t = t_S, t' = t_T$ and $t'' = t_V$; further, $r = r_V$. Note that $t_T < t_S$ by hypothesis; according to Definition 63, this is possible only if $T \xrightarrow{\mathsf{AR}} S$. Also, it is possible that $\mathbf{e} = (\_r_V, \texttt{receive}(t_S : \_)) \prec \mathbf{f} = (\_, r_V, \texttt{start})$ only if $S \xrightarrow{\mathsf{VIS}} V$. We have that $T \xrightarrow{\mathsf{AR}} S \xrightarrow{\mathsf{VIS}} V$; by PREFIX, this gives $T \xrightarrow{\mathsf{VIS}} V$. By Definitionwe obtain that $\mathbf{g} = (\_, r_V, \texttt{receive}(t_T : \_)) \prec (\_, r_V, \texttt{start}) = \mathbf{f}$, as we wanted to prove. ◀

▶ Proposition 71. If $\mathcal{A} \models$ NOCONFLICT, then $\mathsf{Recover}(\mathcal{A})$ satisfies (ConflictCheck).

▶ Proposition 72. Let $\mathbf{e}_1, \mathbf{f}_1, \mathbf{e}_2, \mathbf{f}_2 \in \mathbf{E}$ be such that $(\mathbf{e}_1 = (\_, r_1, \texttt{write}(x, \_))$ and $\mathsf{TS}_{\mathsf{Recover}(\mathcal{A})}(\mathbf{e}_1) = t_1$, $\mathbf{f}_1 = (\_, r_1, \texttt{commit}(t_1))$, $(\mathbf{e}_2 = (\_, r_2, \texttt{write}(x, \_))$, $\mathbf{f}_2 = (\_, r_2, \texttt{commit}(t_2))$, and $r_1 \neq r_2$. This also implies that $t_1 \neq t_2$.

Assume that $\mathbf{f}_2 \prec \mathbf{f}_1$. Note that we have that $t_2 = t_S, t_1 = t_T$ for some $S, T \in \mathcal{H}$. Because of NOCONFLICT, it has to be either $S \xrightarrow{\mathsf{VIS}} T$ or $T \xrightarrow{\mathsf{VIS}} S$. Note that it cannot be be $T \xrightarrow{\mathsf{VIS}} S$, because by Definition 63 we would have $(\mathbf{f}_1 = (\_, r_S, \texttt{commit}(t_S)) \prec \mathbf{f}' = (\_, r_T, \texttt{receive}(t_S : \_)) \prec \mathbf{f}'' = (\_r_T, \texttt{start}) \prec (\mathbf{f}_2 = (\_, r_t, \texttt{commit}(t_T))$, contradicting the hypothesis that $\mathbf{f}_2 \prec \mathbf{f}_1$. Then it has to be $S \xrightarrow{\mathsf{VIS}} T$, which by Definition 51 gives that there exists an event $\mathbf{g}$ such that $\mathbf{g} = (\_, r_T, \texttt{receive}(t_S : \_)) \prec (\_, r_T, \texttt{commit}(t_T)) = \mathbf{f}_1$, as we wanted to prove.