Understanding Eventual Consistency

March 25, 2013

Technical Report
MSR-TR-2013-39

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Important

This document is *work in progress*. Feel free to cite, but note that we will update the contents without warning (the first page contains a timestamp), and that we are likely going to publish the content in some future venue, at which point we will update this paragraph.

Understanding Eventual Consistency

Sebastian Burckhardt
Microsoft Research

Alexey Gotsman
IMDEA Software Institute

Hongseok Yang
University of Oxford

Abstract. Modern geo-replicated databases underlying large-scale Internet services guarantee immediate availability and tolerate network partitions at the expense of providing only weak forms of consistency, commonly dubbed eventual consistency. At the moment there is a lot of confusion about the semantics of eventual consistency, as different systems implement it with different sets of features and in subtly different forms, stated either informally or using disparate and low-level formalisms.

We address this problem by proposing a framework for formal and declarative specification of the semantics of eventually consistent systems using axioms. Our framework is fully customisable: by varying the set of axioms, we can rigorously define the semantics of systems that combine any subset of typical guarantees or features, including conflict resolution policies, session guarantees, causality guarantees, multiple consistency levels and transactions. We prove that our specifications are validated by an example abstract implementation, based on algorithms used in real-world systems. These results demonstrate that our framework provides system architects with a tool for exploring the design space, and lays the foundation for formal reasoning about eventually consistent systems.

1. Introduction

Modern large-scale Internet services rely on distributed database systems that maintain multiple replicas of data. Often such systems are *geo-replicated*, meaning that the replicas are located in geographically distinct locations. Geo-replication requires the systems to tolerate network partitions, yet end-user applications also require them to provide immediate availability. Ideally, we would like to achieve these two requirements while also providing *strong consistency*, which roughly guarantees that the outcome of a set of concurrent requests to the database is the same as what one can obtain by executing these requests atomically in some sequence. Unfortunately, the famous CAP theorem [18] shows that this is impossible. For this reason, modern geo-replicated systems provide weaker forms of consistency, commonly dubbed *eventual consistency* [32]. Here the word 'eventual' refers to the guarantee that

> if update requests stop arriving to the database, then it will eventually reach a consistent state. (1)

Geo-replication is a hot research area, and new architectures for eventually consistent systems appear every year [5, 14, 15, 17, 21, 24, 29, 30]. Unfortunately, whereas consistency models of classical relational databases have been well-studied [9, 26], those of geo-replicated systems are poorly understood. The very term eventual consistency is a catch-all buzzword, and different systems claiming to be eventually consistent actually provide subtly different guarantees and features. Commonly used ways of their specification are inadequate for several reasons:

- **Disparate and low-level formalisms.** Specifications of consistency models proposed for various systems are stated informally or using disparate formalisms, often tied to system implementations. This makes it hard to compare guarantees provided by different systems or apply ideas from one of them in another.
- **Weak guarantees.** More declarative attempts to formalise eventual consistency [29] have identified it with property (1), which actually corresponds to a form of *quiescent consistency* from distributed computing [19]. However, such reading of eventual consistency does not allow making conclusions about the behaviour of the database in realistic scenarios, when updates *never* stop arriving.
- **Conflict resolution policies.** To satisfy the requirement of availability, geo-replicated systems have to allow making updates to the same object on different, potentially disconnected replicas. The systems then have to resolve conflicts, arising when replicas exchange the updates, according to certain policies, often encapsulated in *replicated data types* [27, 29]. The use of such policies complicates the semantics provided by eventually consistent systems and makes its formal specification challenging.
- **Combinations of different consistency levels.** Even in applications where basic eventual consistency is sufficient most of the time, stronger consistency may be needed occasionally. This has given rise to a wide variety of features for strengthening consistency on demand. Thus, some systems now provide a mixture of eventual and strong consistency [1, 13, 21], and researchers have argued for doing the same with different forms of eventual consistency [5]. Other systems have allowed strengthening consistency by implementing transactions, usually not provided by geo-replicated systems [14, 24, 30]. Understanding the semantics of such features and their combinations is very difficult.

The absence of a uniform and widely applicable specification formalism complicates the development and use of eventually consistent systems. Currently, there is no easy way for developers of such systems to answer basic questions when designing their programming interfaces: Are the requirements of my application okay with a given form of eventual consistency? Can I use a replicated data type implemented in a system X in a different system Y? What is the semantics of combining two given forms of eventual consistency?

We address this problem by proposing a formal and declarative framework for specifying the semantics of eventually consistent systems. In our proposal, the specification of a consistency model consists of the following components:

- **Replicated data type specifications** define the basic data types supported by the database and determine the conflict resolution policies at the level of individual objects (§2). We show how to specify abstractly a variety of data types, including registers, counters, multi-value registers [17] and observed-remove sets [28].
- **Consistency specification** extends the semantics of individual objects to that of the entire database (§3). It is defined by a set of *axioms*, constraining requests submitted to the database by its clients and relations on these requests that abstract the way the database processes them. We show how to specify consistency guarantees implemented by a range of existing eventually consistent systems, including weak forms of eventual consistency [2, 17], session guarantees [31] and different kinds of causal consistency [14, 24, 29, 30].
- **Interfaces for strengthening consistency** can include *consistency annotations*, which allow users to specify the consistency level of a single operation (§4.1), *fences*, which affect the consistency level of multiple operations (§4.2), and *transactions*, which ensure that a group of operations executes atomically (§5). We provide the analysis of the trade-offs between some of these features (§4.3).

The main technical challenge we have to deal with is that the above aspects of the system behaviour interact in subtle ways, making it difficult to specify each of them separately. We resolve this problem by representing the information about the database execution that all of the specification components rely on abstractly, in a way not tied to the database implementation (visibility and arbitration relations in §2). This interface between different specification components yields a fully customisable framework, which can define the semantics of any combination of typical guarantees or features of eventually consistent systems. Our technique is a generalisation of approaches used in the context of *weak shared-memory consistency models* [3, 8], which have been extensively studied by the programming languages and verification communities[1]. Overall, we make the following contributions:

- We systematise the knowledge about the existing forms of eventual consistency and provide a single specification framework that can express many guarantees and features of existing systems [2, 13, 14, 17, 21, 24, 29–31]. While some of the concepts that we specify have been formally defined before individually, to our best knowledge, our framework is the first to allow handling all of them uniformly and drawing conclusions about their interactions. This provides the developers of eventually consistent systems with a tool for exploring the design space.
- We justify the correspondence of our specifications to real-world systems by proving that the specifications are validated by an example abstract implementation, based on algorithms used in such systems.
- We prove that our specifications are more powerful than quiescent consistency, allowing us to make conclusions about the database behaviour even in the presence of a continuous stream of updates arriving from its clients(§3.4).
- We prove several results demonstrating how our framework supports rigorous comparisons between various features and guarantees (Theorem 6, Theorem 8, Proposition 9). In particular, we establish that fences are sufficient to recover sequential consistency and serializable transactions in an eventually consistent system. An analogous result for modern shared-memory models was proved only recently [7].

Due to the volume of our technical development, we have relegated all proofs to §A. The definition of the abstract implementation and the proof of its correspondence with our specifications are deferred to §B; however, we present the algorithms used in the implementation informally throughout the paper.

[1] Although our axioms may appear different on the surface, we discover that, when specialised to the integer register data type, common forms of eventual consistency correspond almost exactly to fragments of the memory model adopted in the 2011 C and C++ standards [8] (§3.4). This suggests that existing techniques for testing and verifying programs running on weak memory models [6, 12, 23] may prove beneficial for eventually consistent systems.

2. Replicated Data Types

In this paper we consider a replicated database storing ***objects*** $\mathsf{Obj} = \{x, y, \dots\}$ that have ***values*** from a set Val. For simplicity, we assume that the set of objects in the database is fixed. Every object $x \in \mathsf{Obj}$ has a ***type*** $\mathsf{type}(x) \in \mathsf{Type}$ that determines the set $\mathsf{Op}_{\mathsf{type}(x)}$ of ***operations*** that clients of the database can perform on it. For example, the data types can include a counter ctr and an integer register intreg with operations for reading, incrementing or writing an integer k: $\mathsf{Op}_{\mathsf{ctr}} = \{\mathsf{rd}, \mathsf{inc}\}$ and $\mathsf{Op}_{\mathsf{intreg}} = \{\mathsf{rd}\} \cup \{\mathsf{wr}(k) \mid k \in \mathbb{Z}\}$.

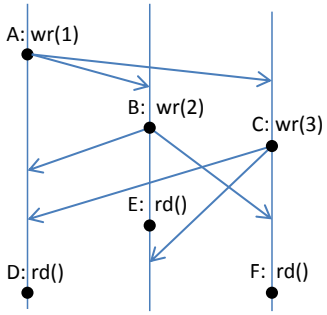
Even though the database may store the same object on multiple replicas, its interface is agnostic to this: a client just requests an operation on an object, without specifying a replica it is stored on. We now present a way of defining the semantics of common replicated data types that matches this level of abstraction by hiding implementation aspects, such as the replication strategy, network topology or transport layer. This allows us to understand how the same data type can be implemented on different system architectures. Our discussion focuses on the behaviour of individual operations on a single object.

In strongly consistent systems, the outcome of a set of concurrent operations on an object can be obtained by executing them atomically in some sequence. In this setting, the semantics of a data type τ can be completely specified by a function $\mathcal{S}_\tau : \mathsf{Op}_\tau^+ \to \mathsf{Val}$, which, given a nonempty sequence of operations performed on an object, specifies the return value of the last operation (we assume that $\bot \in \mathsf{Val}$ is used for operations that return no value). For a counter, read operations return the number of preceding increment operations. For an integer register, read operations return the value of the last preceding write, or zero if there was no prior write. We can thus define $\mathcal{S}_{\mathsf{ctr}}$ and $\mathcal{S}_{\mathsf{intreg}}$ as follows: for any sequence of operations σ,

$$\begin{aligned}
\mathcal{S}_{\mathsf{ctr}}(\sigma\,\mathsf{rd}) &= (\text{the number of } \mathsf{inc} \text{ operations in } \sigma);\\
\mathcal{S}_{\mathsf{intreg}}(\sigma\,\mathsf{rd}) &= k, \text{ if } \mathsf{wr}(0)\,\sigma = \sigma_1\,\mathsf{wr}(k)\,\sigma_2 \text{ and}\\
&\qquad \sigma_2 \text{ does not contain } \mathsf{wr} \text{ operations};\\
\mathcal{S}_{\mathsf{ctr}}(\sigma\,\mathsf{wr}(k)) &= \mathcal{S}_{\mathsf{intreg}}(\sigma\,\mathsf{inc}) = \bot.
\end{aligned}$$

In an eventually consistent system, data type semantics is more complicated. In such a system, operations on the same

object can be issued concurrently at multiple replicas. To achieve availability and partition tolerance, the system performs operations at each replica immediately, and communicates this to other replicas only after the fact. The following diagram illustrates an execution of a system with integer registers by arranging labelled operations on several vertical timelines corresponding to replicas, with the delivery of updates to other replicas shown as diagonal arrows:



Previous work has described a variety of implementations of *replicated data types* that operate in this manner (see [28] for a survey). Since replicas in these implementations cannot be immediately aware of all operations, they may at times be inconsistent. The key challenge is to ensure that, once all updates are delivered to all replicas, they resolve conflicts between them uniformly and converge to the same state. Exactly how this is achieved varies greatly; we identify the following strategies:

1. **Make concurrent operations commutative**. We require that all operations be commutative, so that we can apply them in any order using the standard sequential semantics. This strategy works for counters, but not integer registers.
2. **Order concurrent operations**. The system totally orders all concurrent operations in some disciplined way, e.g., using timestamps. This allows applying the sequential semantics.
3. **Flag conflicts**. The system detects the presence of a conflict and lets the user deal with it. For example, the multi-value register used in Amazon's Dynamo key-value store [17] defines the return value of the read D in the above example as the set $\{2, 3\}$ of conflicting values.
4. **Resolve conflicts semantically**. The system detects a conflict and takes some data-type-dependent action to resolve it [27, 29]. For example, a replicated data type of *observed-remove set* [10] resolves conflicting operations trying to add and remove the same element so that an add always wins.

We now present a single formalization that is sufficiently general to represent all the four strategies and give an example of a replicated data type for each of them. We specify a data type τ by a function $\mathcal{F}_\tau$, generalizing $\mathcal{S}_\tau$. Just like with $\mathcal{S}_\tau$, we want $\mathcal{F}_\tau$ to determine the return value of an operation based on prior operations performed on the object. However, there are some important differences:

1. In the sequential setting, an operation takes into account the effect of all operations preceding it in the sequence. In the concurrent setting, the result depends on what subset of operations is visible in a given context. For example, in the diagram above, C is visible to D, but not to E.
2. In the concurrent setting, the result may also depend on additional information used to order events. For example, the relative order of the concurrent writes B and C in the diagram above may be determined using a timestamp, thus guaranteeing that reads D and F return the same result, even though they receive B and C in different orders.

The main insight of our formalisation is that we can specify the information about such relationships between events declaratively, without referring to implementation-level concepts, such as replicas or messages. Namely, $\mathcal{F}_\tau$ takes as a parameter not a sequence, but an *operation context*, which encapsulates "all we need to know" about a system execution to determine the return value of a given operation.

DEFINITION 1. *An **operation context** for a data type τ is a tuple $C = (f, V, \mathsf{ar}, \mathsf{vis})$, where*

- *$f \in \mathsf{Op}_\tau$ is the operation about to be performed.*
- *V is a set of **operation events** of the form (e, g), where e is a unique event identifier and $g \in \mathsf{Op}_\tau$. The set V includes all operations that are visible to f and can thus affect its result.*
- *$\mathsf{vis} \subseteq V \times V$ is a **visibility relation**, recording the relationships between operations in V motivated by point 1 above.*
- *$\mathsf{ar} \subseteq V \times V$ is a total and irreflexive **arbitration relation**, recording the relationships motivated by point 2 above.*

For a relation r we write $(u, v) \in \mathsf{r}$ and $u \xrightarrow{\mathsf{r}} v$ interchangeably. The vis relation describes the relative visibility of events in V: $u \xrightarrow{\mathsf{vis}} v$ means that the effect of u is visible to v. As we show below, it is needed to define certain data types. In implementation terms, an event u might be considered visible to an event v if the update performed by u has been delivered to the replica performing v before v is issued. The ar relation represents the ordering information provided by the system; $u \xrightarrow{\mathsf{ar}} v$ means that u is ordered before v. In an implementation it can be constructed using timestamps or tie-breaking mechanisms.

DEFINITION 2. *A **replicated data type specification** for a type τ is a function $\mathcal{F}_\tau$ that, given an operation context C for τ, specifies a return value $\mathcal{F}_\tau(C) \in \mathsf{Val}$, and that does not depend on the operation identifiers in C.*

Note that $\mathcal{F}_\tau(f, \emptyset, \emptyset, \emptyset)$ returns the value resulting from performing f on the default state for the data type (e.g., zero for an integer register). Also, $\mathcal{F}_\tau$ is deterministic: all the nondeterminism present in the distributed system is resolved by the operation context. We emphasise that our specifications do not prescribe a particular way in which the visibility and arbitration relations are actually represented in an implementation: the above references to implementations are only illustrative. Thus, the specifications can be viewed as generalising the concept of an *abstract data type* [22] to the replicated case. We now give several examples of data type specifications, corresponding to the four implementation strategies mentioned earlier.

Example 1: Counter (ctr) is defined by

$$\mathcal{F}_{\mathsf{ctr}}(\mathsf{inc}, V, \mathsf{vis}, \mathsf{ar}) = \bot;$$
$$\mathcal{F}_{\mathsf{ctr}}(\mathsf{rd}, V, \mathsf{vis}, \mathsf{ar}) = (\text{the number of } \mathsf{inc} \text{ operations in } V).$$

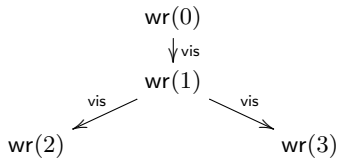
Note that the result does not depend on ar or vis, but only on V. Hence, this example is representative of strategy 1.

Example 2: Integer register (intreg) is defined by $\mathcal{F}_{\mathsf{intreg}}(f, V, \mathsf{vis}, \mathsf{ar}) = \mathcal{S}_{\mathsf{intreg}}(V^{\mathsf{ar}} f)$, where V^{ar} denotes the sequence obtained by ordering the operations in the set V of events visible to f according to the arbitration relation ar. This represents strategy 2: the return value is determined by establishing a total order of the visible operations and applying the regular sequential semantics. Note that the relative visibility between events in V, given by vis, is not used. We can similarly obtain a concurrent semantics $\mathcal{F}_\tau$ of any data type τ based on its sequential semantics $\mathcal{S}_\tau$. For example, $\mathcal{F}_{\mathsf{ctr}}$ as defined in Example 1 is equivalent to what we obtain using this generic construction. The next two examples go beyond this.

Example 3: Multi-value register (mvr)**.** A multi-value register has the same operations as the integer register, but its reads return a set of values, rather than a single one:

$$\mathcal{F}_{\mathsf{mvr}}(\mathsf{rd}, V, \mathsf{vis}, \mathsf{ar}) = \{k \mid \exists e.\, (e, \mathsf{wr}(k)) \text{ is maximal in } V\},$$

where an element $v \in V$ is maximal in V if there exists no $v' \in V$ such that $v \xrightarrow{\mathsf{vis}} v'$. Here a read returns all versions of the object that are not superseded by later writes, as determined by vis; the ar relation is not used. A multi-value register thus detects conflicting writes, as in strategy 3. For example, a rd operation would return $\{2, 3\}$ in the following context:



Example 4: Observed-remove set (orset)**.** Assume we want to implement a set of integers. Consider the operation context

$$(\mathsf{rd}, \{(\mathsf{A}, \mathsf{add}(42)), (\mathsf{B}, \mathsf{remove}(42))\}, \mathsf{vis}, \mathsf{ar}).$$

If we use the generic construction from Example 2, the result of rd will depend on the arbitration relation: $\emptyset$ if $\mathsf{A} \xrightarrow{\mathsf{ar}} \mathsf{B}$, and $\{42\}$ otherwise. In some cases, the application semantics may require a different outcome, e.g., that an add operation always win against concurrent remove operations. Bieniusa et al. [10] propose the ***observed-remove set*** orset that achieves this semantics by mandating that remove operations cancel out only the add operations that are visible to them:

$$\mathcal{F}_{\mathsf{orset}}(\mathsf{rd}, V, \mathsf{vis}, \mathsf{ar}) = \{k \mid \exists \text{ live } v = (e, \mathsf{add}(k)) \in V\},$$

where $v = (e, \mathsf{add}(k)) \in V$ is live if there does not exist $v' = (e', \mathsf{remove}(k)) \in V$ such that $v \xrightarrow{\mathsf{vis}} v'$. In the above operation context rd will return $\emptyset$ if $\mathsf{A} \xrightarrow{\mathsf{vis}} \mathsf{B}$, and $\{42\}$ otherwise.

Although replicated data types can simplify the programming of eventually consistent systems, they are not by themselves sufficient for general systems that contain more than one object. For example, consider a client program that updates a set object `friends` and then a list object `wall`:

```
friends.remove(boss); wall.append(photo);
```

In this case, we may want the database to ensure that the order of the updates is preserved. In terms of our framework, such guarantees depend on properties of vis and ar relations that are not defined by data type specifications. In the following sections we provide means of their specification.

3. Axiomatic Specification Framework

We extend our framework for specifying replicated data types (§2) to defining the semantics of the entire database. Our specifications can capture a range of forms of eventual consistency:

- **Basic eventual consistency**, which only guarantees that the effect of every operation will eventually become visible to all participants (§3.2).
- **Ordering guarantees**, which ensure that the system preserves the order in which operations are performed (§3.3) .
- **Interfaces for on-demand consistency strengthening**, which include consistency annotations and fences (§4), as well as transactions (§5).

We start by formalising client-database interactions (§3.1). We describe a run of the database by a *history*, which records all (possibly infinitely many) such interactions. We then specify its consistency model by a number of *axioms* (see Figure 1), which define the set of all histories it can produce.

3.1 Client Interaction Model

Clients often wish to perform multiple operations within some context, and to capture this, we introduce the notion of a ***session***, identified by session identifiers $\mathsf{Sid} = \{1, 2, 3, \dots\}$. Sessions are different from transactions (introduced in §5): rather than providing advanced guarantees, such as atomicity or isolation, they are a means of tracking client identity across multiple requests. The purpose of sessions is to satisfy basic ordering expectations [31], such as whether a read following a write is guaranteed to see the effects of that write. In some systems, this is ensured by permanently binding a session to a single replica.

Each client operation is described in a history by an *action*, which enriches operation events (Definition 1) with session identifiers and return values.

DEFINITION 3. *An **action** is a tuple $(e, s, [x.f : k])$, where e is a unique action identifier, $s \in \mathsf{Sid}$ is the identifier of the session the action takes place in, $f \in \mathsf{Op}_{\mathsf{type}(x)}$ is an operation performed on an object $x \in \mathsf{Obj}$, and $k \in \mathsf{Val}$ is its return value. We denote the set of actions by* Act.

We omit return values equal to $\bot$. For $a = (e, s, [x.f : k])$, we let $\mathsf{ses}(a) = s$, $\mathsf{obj}(a) = x$, $\mathsf{op}(a) = f$, $\mathsf{rval}(a) = k$, $\mathsf{type}(a) = \mathsf{type}(x)$ and $\mathsf{event}(a) = (e, f)$. We also use the event projection on sets of actions and relations over them.

A history consists of a set of actions, together with a ***session order***, relating actions within a session according to the order in which they were issued by the client.

DEFINITION 4. *A **history** is a pair (A, so), where $A \subseteq \mathsf{Act}$, A has no duplicate action identifiers,* $\mathsf{so} \subseteq A \times A$*, and* so *satisfies the axiom* SOWF *in Figure 1.*

Histories abstract from internal database execution completely. However, in order to define concepts arising in differ-

ent forms of eventual consistency, we need to know more about how operations relate to each other. To this end, we enrich histories with additional information about the internal database execution. As in §2, we record this information in an abstract form using the visibility and arbitration relations.

DEFINITION 5. *An **execution** is a tuple* $X = (A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$*, where* (A, so) *is a history, and* $\mathsf{vis}, \mathsf{ar} \subseteq A \times A$ *are such that the axioms* VISWF *and* ARWF *from Figure 1 hold.*

The vis and ar relations have the same intuitive meaning as those in §2. The difference is that, previously, we only considered these relations on a set of events needed to determine the outcome of a particular operation. In contrast, here the relations are defined on the set of all operations on all objects accessed in a run of a database. Given $X = (A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$ and $a \in A$, we can easily extract the operation context of a by selecting all actions visible to it according to vis:

$$\mathsf{ctxt}(a) = (\mathsf{op}(a), \mathsf{event}(B), \mathsf{event}(\mathsf{vis}|_B), \mathsf{event}(\mathsf{ar}|_B)),$$

where $B = \mathsf{vis}^{-1}(a)$. ARWF implies that the arbitration relation is sufficient to fully order all operations in $\mathsf{vis}^{-1}(a)$, and, thus, $\mathsf{ctxt}(a)$ is indeed an operation context.

Figure 2(a) shows an example of an execution, corresponding to the scenario of posting a photo from §2, but expressed using integer registers. In diagrams throughout this paper, we omit action and session identifiers. In the execution, a session first writes to a register x, setting the access permission to `all`. Some time later, it changes the permission to `noboss` and then writes to a register y to post a photo. The arbitration relation ar states that any session that sees both writes to x should assume that `noboss` overwrites `all`. However, as shown by the vis edges, in this example another session sees the photo, but not the updated access permission. In an implementation this anomaly can happen when the replica that session is connected to receives the updates corresponding to the writes of `noboss` and `photo` out of the order they were issued in [2, 17].

Our notion of an execution is similar to structures used for defining memory consistency models of hardware [3] and programming languages [8]. In particular, the visibility relation is similar to the "reads-from" relation used in such models. Unlike "reads-from", our visibility relation captures *all* delivered updates, so as to handle replicated data types.

3.2 Axiomatic System Specifications

The notion of a history corresponds straightforwardly to runs of a real-world database. Thus, we consider a ***system specification*** to be simply a set of histories. We can check if a particular database correctly implements the specification by determining whether, for all its runs, the corresponding history is in the set. In our specification framework, we obtain a system specification by choosing a set of ***axioms*** A from Figure 1 that constrain executions. The corresponding specification includes all histories that can be extended to an execution satisfying A:

$$(A, \mathsf{so}) \in [\![\mathsf{A}]\!] \iff \exists \mathsf{vis}, \mathsf{ar}.\ (\mathsf{A}, \mathsf{so}, \mathsf{vis}, \mathsf{ar}) \text{ satisfies } \mathsf{A}.$$

The fewer axioms are chosen, the weaker the consistency model is. The weakest sensible specification in our framework is $[\![\text{SOWF}, \text{VISWF}, \text{ARWF}, \text{RVAL}, \text{EVENTUAL}, \text{THINAIR}]\!]$. We call it ***basic eventual consistency***, and executions satisfying its axioms, ***eventually consistent***. The specification includes no ordering guarantees, but satisfies basic expectations, and is implemented, e.g., by Dynamo [17]. Its axioms have the following purpose:

- The well-formedness axioms SOWF, VISWF, ARWF state basic properties of the relations that we are working with.
- RVAL ensures that data types behave according to their specifications, i.e., the return value of every action a is computed on its operation context using the specification $\mathcal{F}_{\mathsf{type}(a)}$ for the type of the object accessed in a (§2). For example, in the execution in Figure 2(a) the operation context of the read from x includes only the write of `all`, so by RVAL and $\mathcal{F}_{\mathsf{intreg}}$, the read returns `all`. If the execution contained a vis edge from the write of `noboss` to the read from x, then the context would include both writes to x, with `noboss` taking precedence over `all` according to ar. By RVAL and $\mathcal{F}_{\mathsf{intreg}}$, the read would then have to return `noboss`.
- EVENTUAL ensures that an action cannot be invisible to infinitely many other actions on the same object and thus formalises (1) from §1.
- THINAIR rules out counterintuitive behaviours that are not produced by most real-world eventually consistent systems. For example, the execution in Figure 2(b) is allowed by RVAL and EVENTUAL, but not by THINAIR. Here the value 42 appears "out-of-thin-air", due to a cycle in $\mathsf{so} \cup \mathsf{vis}$ (recall that registers are initialised to 0). Such cycles usually arise from effects of speculative computations, which are done by some older eventually consistent systems [32], but not by modern ones. In this paper, we restrict ourselves to forms of eventual consistency implemented by the latter [29].

Our framework allows checking easily whether a given outcome is allowed by the consistency model without considering the system implementation: one just needs to ensure that all the axioms are satisfied. For example, the execution in Figure 2(a) is allowed by the axioms of basic eventual consistency.

Like in §2, our formalisation does not prescribe a particular way of representing visibility and arbitration in an implementation. Hence, these relations form an abstract interface between replicated data types used for conflict resolution and the underlying consistency model of the database. This makes our specification framework fully customisable and allows exploring the design space of consistency models easily: each of these two aspects can be varied separately, and the framework will define the semantics of any possible combination. We achieve this level of modularity even though, in real-world systems, implementations of the two aspects are often tightly interlinked [28].

Abstract implementation. We justify the soundness of our axioms by proving that they are validated by an abstract implementation based on algorithms used in existing systems [2, 14, 17, 24, 30, 31]: the set of histories it produces is included into the corresponding system specification (Theorem 11, §B). Our implementation of basic eventual consistency corresponds straightforwardly to the operational explanations we have given so far. The database consists of an arbi-

Figure 1. Axioms of eventual consistency. Here $\mathsf{r}|_B$ denotes the projection of a relation r to B: $\mathsf{r}|_B = (\mathsf{r} \cap (B \times B))$.

WELL-FORMEDNESS AXIOMS

SOWF: so is the union of transitive, irreflexive and total orders on actions by each session

VISWF: $\forall a, b.\, a \xrightarrow{\mathsf{vis}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$

ARWF: $\forall a, b.\, a \xrightarrow{\mathsf{ar}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$, ar is transitive and irreflexive, and $\mathsf{ar}|_{\mathsf{vis}^{-1}(a)}$ is a total order for all $a \in A$

DATA TYPE AXIOM

RVAL: $\forall a \in A.\, \mathsf{rval}(a) = \mathcal{F}_{\mathsf{type}(a)}(\mathsf{ctxt}(a))$

BASIC EVENTUAL CONSISTENCY AXIOMS

EVENTUAL: $\forall a \in A. \neg(\exists \text{ infinitely many } b \in A.\, \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

THINAIR: $\mathsf{so} \cup \mathsf{vis}$ is acyclic

AUXILIARY RELATIONS

Per-object session order: $\mathsf{soo} = (\mathsf{so} \cap \mathsf{sameobj})$

Per-object causality order: $\mathsf{hbo} = (\mathsf{soo} \cup \mathsf{vis})^+$

Causality order: $\mathsf{hb} = (\mathsf{so} \cup \mathsf{vis})^+$

SESSION GUARANTEES

RYW (Read Your Writes). An operation sees all previous operations by the same session: $\mathsf{soo} \subseteq \mathsf{vis}$

MR (Monotonic Reads). An operation sees all operations previously seen by the same session: $(\mathsf{vis}; \mathsf{soo}) \subseteq \mathsf{vis}$

WFRV (Writes Follow Reads in Visibility). Operations are made visible at other replicas after operations on the same object that were previously seen by the same session: $(\mathsf{vis}; \mathsf{soo}^*; \mathsf{vis}) \subseteq \mathsf{vis}$

WFRA (Writes Follow Reads in Arbitration). Arbitration orders an operation after other operations previously seen by the same session: $(\mathsf{vis}; \mathsf{soo}^*) \subseteq \mathsf{ar}$

MWV (Monotonic Writes in Visibility). Operations are made visible at other replicas after all previous operations on the same object by the same session: $(\mathsf{soo}; \mathsf{vis}) \subseteq \mathsf{vis}$

MWA (Monotonic Writes in Arbitration). Arbitration orders an operation after all previous operations by the same session: $\mathsf{soo} \subseteq \mathsf{ar}$

CAUSALITY AXIOMS

POCV (Per-Object Causal Visibility): $\mathsf{hbo} \subseteq \mathsf{vis}$

POCA (Per-Object Causal Arbitration): $\mathsf{hbo} \subseteq \mathsf{ar}$

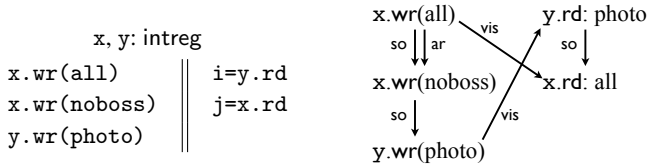
COCV (Cross-Object Causal Visibility): $(\mathsf{hb} \cap \mathsf{sameobj}) \subseteq \mathsf{vis}$

COCA (Cross-Object Causal Arbitration): $\mathsf{hb} \cup \mathsf{ar}$ is acyclic

trary number of replicas, each storing a log of actions. Every client session is connected to some replica, although a session can switch to another one at any time. When a session issues an operation, its return value is computed immediately on the basis of the state of the replica the session is connected to, and the corresponding action is appended to its log. From time to time, each replica broadcasts updates to the others, subject to a fairness constraint that every update will eventually get delivered to every replica (needed to validate EVENTUAL). Then $a \xrightarrow{\mathsf{vis}} b$ if a has been delivered to the replica performing b before b is issued. Thus, so and vis edges go forwards in time, so that THINAIR is validated. The arbitration relation is computed using Lamport timestamps [20].

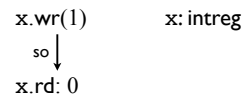
Figure 2. Anomalies allowed or disallowed by different axioms from Figure 1. In some cases, we show the code of client sessions that could produce the execution (where `i` and `j` are client-local variables).

(a) Disallowed by COCV:

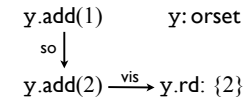


(b) Disallowed by THINAIR:

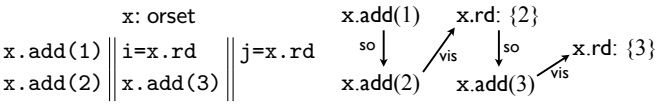
(c) Disallowed by RYW:



(d) Disallowed by MWV:



(e) Disallowed by POCV:



(f) Allowed by all axioms in Figure 1:

3.3 Classification of Ordering Guarantees

Basic eventual consistency provides very few guarantees to clients, allowing the anomaly in Figure 2(a) and, in fact, even more straightforward anomalies shown in Figures 2(c) and 2(d). In Figure 2(c), a session does not see a write it made (as signified by the absence of a vis edge) and thus reads the initial value of the register `x`. In an implementation this can happen when the read connects to another replica, which has not yet received the update corresponding to the write operation. In Figure 2(d), a session inserts 1 and then 2 into a set `y`; another session sees the first insertion, but not the second. In an implementation this can happen when the two insertions are propagated to other replicas out of order.

Many systems achieve availability and partition tolerance while providing stronger guarantees on the ordering of operations. In this paper we formalise the following ones:

Form of eventual consistency	Implementations
Basic eventual consistency	[2, 17]
Session guarantees	[31]
Per-object causal consistency	[29]
(Cross-object) causal consistency	[14, 21, 24, 30]

The differences can be subtle; it is one of our contributions to provide a means for precise comparisons. The cornerstones of our discussion are the axioms shown in Figure 1 and the anomalies shown in Figure 2.

Session guarantees. Let r^* denote the reflexive and transitive closure of a relation r, $\mathsf{r}_1; \mathsf{r}_2$ the composition of binary relations r_1 and r_2, and let $\mathsf{sameobj}(a, b) \Leftrightarrow \mathsf{obj}(a) = \mathsf{obj}(b)$. For

an execution $X = (A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$, we define the ***per-object session order*** as follows: $\mathsf{soo} = (\mathsf{so} \cap \mathsf{sameobj})$.

The axioms RYW–MWA in Figure 1 formalise ***session guarantees***, ensuring that operations within a session observe a view of the database that is consistent with their own actions, even if, in an implementation, they can access various, potentially inconsistent replicas. The guarantees are due to Terry et al. [31], who defined them in a low-level operational framework. Here we recast them into axioms appropriate for arbitrary replicated data types. However, we have preserved the original terminology, and thus refer to reads and writes in the names of the axioms. As an illustration, RYW and MWV disallow the executions in Figures 2(c) and 2(d), respectively. Note also that WFRV implies that vis is transitive, and WFRA implies that $\mathsf{vis} \subseteq \mathsf{ar}$. In our **abstract implementation**, we implement session guarantees as proposed by Terry et al. [31]. For example, to ensure RYW, each session maintains the "write set" of actions it has issued; the session can then only connect to replicas that contain all of its write set.

The axioms for session guarantees highlight the general principle of formalising stronger consistency models: mandating that certain edges be included into vis and ar, so that clients have more up-to-date information. We now use the same approach to formalise other models.

Per-object causal consistency. Let $\mathsf{hbo} = (\mathsf{soo} \cup \mathsf{vis})^+$ be the ***per-object causality order***, which orders every action after those that can affect it via a chain of computation involving the same object; in other contexts, the name *happens-before* is used instead of causality. For example, in Figure 2(e), $\mathsf{add}(1)$ in the first session and rd in the third are related by hbo.

An execution is ***per-object causally consistent***, if it is eventually consistent and satisfies the axioms POCV and POCA in Figure 1. POCV guarantees that an operation sees all operations on the same object that causally affect it, and POCA correspondingly restricts the arbitration relation. POCV disallows the executions in Figures 2(c), 2(d) and 2(e). In fact, using our formalisation we can show that per-object causal consistency is equivalent to all of Terry et al.'s session guarantees.

THEOREM 6. POCV *is equivalent to the conjunction of* RYW*,* MR*,* WFRV *and* MWV. POCA *is equivalent to the conjunction of* WFRA *and* MWA.

(Cross-object) causal consistency. We define a consistency model that preserves a stronger notion of causality than per-object one, and is implemented, e.g., by COPS [24], Walter [30] and Concurrent Revisions [14]. Let the ***causality order*** $\mathsf{hb} = (\mathsf{so} \cup \mathsf{vis})^+$ be the transitive closure of the session and visibility relations. Unlike hbo, this relation considers any chain of computation, possibly involving multiple objects, to be a causal dependency. For example, in Figure 2(a) the write of `noboss` to x in the first session hb-precedes the read from x in the second; these actions are not related by hbo.

An execution is ***(cross-object) causally consistent***, if it is eventually consistent and satisfies the axioms COCV and COCA in Figure 1. These axioms are similar to those for per-object causal consistency, but without restrictions to causal dependencies via the same object (the use of acyclicity in COCA is explained below). For example, the execution in Figure 2(a), is allowed by per-object causal consistency, but not by cross-object causal consistency. Causal consistency is weaker than strong consistency, as it allows reading stale data—it is this feature that allows implementing it while guaranteeing availability and partition tolerance. For example, it allows the execution in Figure 2(f), where both reads fetch the initial value of the register, despite writes to it by the other session. It is easy to check that this outcome cannot be produced by any interleaving of the sessions' actions, and is thus not strongly consistent.

Abstract implementation of causal consistency. Our abstract implementation of per-object and cross-object causal consistency is based on the algorithm used in the COPS system [24]. When propagating actions between replicas, the algorithm tags every one of them with a list of other actions it causally depends on: those preceding it in hbo for per-object causal consistency, and in hb for cross-object one. A replica receiving an update performed by an action must wait until it receives all of the action's dependencies before making it available to clients. For the case of cross-object causal consistency, in Figure 2(a) the write of the photo by the first session will depend on the write of `noboss`. Hence, when the replica the second session is connected to receives the photo, it will have to wait until it receives the write of `noboss` before making the photo available to the client. The arbitration relation is computed using a system-wide Lamport clock [20]. This guarantees the acyclicity of $\mathsf{hb} \cup \mathsf{ar}$ and is the reason for not formulating COCA in the same way as POCA, i.e., $\mathsf{hb} \cap \mathsf{sameobj} \subseteq \mathsf{ar}$.

3.4 Comparison to Other Definitions

Quiescent consistency. We now show that our specifications of eventual consistency describe the semantics of the system more precisely than quiescent consistency, as stated by (1). To formalise the latter, assume that all operations are divided into queries (Query) and updates (Update), and that return values computed by $\mathcal{F}_\tau$ are insensitive to queries: $\mathcal{F}_\tau(f, V, \mathsf{vis}, \mathsf{ar}) = \mathcal{F}_\tau(f, V_u, \mathsf{vis}_u, \mathsf{ar}_u)$ where V_u, vis_u and ar_u are the restrictions of V, vis and ar to update operations. The following straightforward proposition shows that even the basic notion of eventual consistency in Figure 1 implies quiescent consistency.

PROPOSITION 7. *Consider an eventually consistent execution* $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$ *with finitely many update actions. Then there are only finitely many query actions* $a \in A$ *that do not see all updates on the object they are accessing:*

$$\neg(\forall b \in A.\, \mathsf{op}(b) \in \mathsf{Update} \wedge \mathsf{sameobj}(a, b) \implies b \xrightarrow{\mathsf{vis}} a).$$

Furthermore, all other query actions return the same value.

The converse is not true. Consider a program with a counter object x (initially zero), where the first session adds 1 to x, and the second continuously adds 2 to x until it is odd:

```
x.add(1)  ||  do { x.add(2) } while(x.read() % 2 == 0)
```

Under basic eventual consistency as defined above, termination is guaranteed: the axiom EVENTUAL guarantees that all but finitely many reads of x must see the $x.\mathsf{add}(1)$ operation, and thus the loop cannot repeat forever. However, under quiescent consistency termination is not guaranteed: since x is updated continuously, the system is not quiescent, and all bets are off.

Shared-memory consistency models. Interestingly, the specialisations of the consistency levels defined by the axioms in Figure 1 to the type intreg from §2 correspond almost exactly to those adopted by the memory model in the 2011 C and C++ standards [8]. Thus, POCA and POCV define the semantics of so-called *relaxed* operations in C/C++, which provide the weakest consistency. COCV and COCA are close to the semantics of *release-acquire* operations, providing consistency in between relaxed and strong. However, C/C++ does not validate THINAIR, since it makes the effects of processor or compiler speculation visible to the programmer. We formalise the correspondence to the C/C++ memory model in §C.

The similarity to C/C++ might come as a surprise. It stems from the fact that modern multiprocessors have a complicated microarchitecture, including a hierarchy of caches with coherence maintained via message passing—under the hood, a processor is really a distributed system. Optimisations that processors perform produce the same effects as delays and reorderings of messages occurring in a distributed system.

4. Combining Different Consistency Levels

Even though a given flavour of eventual consistency may be sufficient for many applications, stronger consistency levels are needed from time to time. For example, consider a shopping cart in an e-commerce application: while a user is shopping, it is acceptable for the information about the items in the shopping cart to be temporarily inconsistent; however, during a check-out, we need to be sure the user is paying only for what has actually been ordered. Therefore, Amazon's Dynamo [1] allows requesting strong consistency for some operations.

In a similar vein, causal consistency is desirable for many applications, but algorithms used to implement it (§3.3) increase the latency of propagating operations through the system. Bailis et al. [5] have argued that, given the huge size of causality graphs in real-world applications, this makes it problematic to provide causal consistency throughout the system and suggested letting the programmer request it on demand.

Due to the complexity of eventually consistent models, formulating their combinations precisely and choosing the programming interfaces for requesting stronger consistency are delicate. These issues have not been addressed by existing proposals for combining different models of eventual consistency [5]. Our contribution in this section is to show how, using our specification framework, we can define the semantics of such combinations and assess the trade-offs between different design choices. In particular, we present two mechanisms for requesting stronger consistency, widely used in shared-memory models [3, 8], and discuss their trade-offs in the context of eventually consistent systems. We first illustrate the techniques for combining consistency levels using the example of requesting cross-object causal consistency on demand in a system providing per-object causal consistency (§4.1). We then add on-demand strong consistency (§4.2). Combinations with weaker consistency levels could be handled similarly.

4.1 Consistency Annotations

Consider a system providing per-object causal consistency, i.e., satisfying the axioms SOWF–POCA in Figure 1. We wish to give the programmer the ability to request cross-object causal consistency on demand, as defined by COCV and COCA. One way to achieve this is to annotate every operation accepted by the database by a ***consistency annotation***, which classifies the operation as ordinary or causal. Accordingly, we change the form of actions from §3 to $a = (e, s, [x.f_\mu : k])$, where $\mu \in \{\mathsf{ORD}, \mathsf{CSL}\}$. We let $\mathsf{level}(a) = \mu$, and let the event selector ignore μ. The intention is that, when an action a annotated by CSL is visible to another action b, this establishes a causal relationship between a and b. Formally, consider an execution $X = (A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$. We define a ***causal visibility relation*** as follows:

$$\forall a, b.\, a \xrightarrow{\mathsf{cvis}} b \iff a \xrightarrow{\mathsf{vis}} b \wedge \mathsf{level}(a) = \mathsf{CSL}.$$

We then redefine the causality order as the transitive closure of the session and causal visibility relations: $\mathsf{hb} = (\mathsf{so} \cup \mathsf{cvis})^+$. Consistency annotations thus allow the programmer to specify which visibility relationships represent causal dependencies that the system has to preserve (we could similarly let the programmer treat only a subset of so as causal dependencies).

The combined model is given by all of the axioms in Figure 1, but with hb defined as above. Note that, since hbo is still defined as in Figure 1, per-object causal consistency is always guaranteed. Also, if every access is annotated as causal, then $\mathsf{cvis} = \mathsf{vis}$ and hb becomes defined as in Figure 1, so we come back to the causal consistency model from §3.3.

To illustrate the combined model, consider the execution in Figure 2(a), previously discussed in §3. To rule out this execution and make sure that a session that sees the photo will also see the updated access permission, we do not need to require all accesses to be causally consistent: only posting the photo has to be a causal operation. Indeed, in this case, the vis edge from the write to y in the first session to the read from it in the second will be included into cvis. Since the $\mathsf{so} \subseteq \mathsf{hb}$, the write of `noboss` to x in the first session will causally precede the read from x in the second. Then, by COCV, the read from x in the second session will see the write of `noboss` and fetch the correct permission. This is ensured even though the write to x is annotated as ordinary. Note that, if we performed additional ordinary operations in between the writes to x and y in the first session and then read them in the second, we would get the same guarantees.

Abstract implementation. To provide the combined consistency model in our abstract implementation, we modify the COPS algorithm (§3.3) to tag actions with their $(\mathsf{hb} \cup \mathsf{hbo})$-predecessors. Because of the tigher definition of hb, this decreases the number of dependencies in comparison to providing causal consistency throughout the system.

4.2 Fences

We now extend the the consistency model from §4.1 to allow the programmer to request strong consistency as well. Here we illustrate a different approach: instead of consistency annotations, we use ***fences***, which affect multiple actions instead of a single one. We extend the set of actions with $a = (e, s, \mathsf{fence})$, for which we let $\mathsf{op}(a) = \mathsf{fence}$.

In our **abstract implementation**, we treat a fence as a two-phase commit across all replicas, whereby the replica that ex-

ecutes it propagates all the updates it knows about to the other replicas, and does not proceed further until they acknowledge the receipt. Note that, as expected, this violates the availability requirement: if some replica becomes disconnected, the execution of a fence will have to wait until it reconnects.

We show the axioms for the consistency model with fences in Figure 3, where we also integrated the new axioms from §4.1. To define the semantics of fences, we adjust the notion of an execution to contain an additional relation sc satisfying the SCWF axiom: $X = (A, \mathsf{so}, \mathsf{vis}, \mathsf{ar}, \mathsf{sc})$. In implementation terms, sc can be viewed as the total order in which the two-phase commits initiated by fences take place. We modify VISWF and ARWF so that vis and ar relations did not relate fence actions, as they do not make sense for fences; we also assume that the sameobj relation from Figure 1 does not relate fence actions. We adjust THINAIR to take sc into account.

The role of fences is to provide additional guarantees about the visibility of certain actions, which we capture in our axioms two ways. First, we redefine the hb relation to include sc. It is then used by COCV and COCA, which allows taking sc into account when determining visibility and arbitration. To illustrate this, consider the execution in Figure 2(f), which is causally consistent, but not strongly consistent. If each session executes a fence after its write, then the outcome shown will be disallowed. This is because, by SCWF, the total order sc has to order the two fences one way or another; then by COCV, the write of the session whose fence comes first in sc is guaranteed to be seen by the read in the other session. Fences can thus be used to avoid anomalies where sessions read stale values. In implementation terms, we can justify including sc into hb as follows: if fences are implemented by two-phase commits, then a replica R_1 that executed a fence after a replica R_2 did is guaranteed to see all the updates that were visible to R_2 at the time it issued the fence.

Another way in which fences strengthen consistency is captured by a new clause into the definition of cvis: if an operation b hbo-follows an operation c in another session following a fence, then it causally depends on the fence. In implementation terms, this is justified as follows: $a \xrightarrow{\mathsf{so}} c \xrightarrow{\mathsf{hbo}} b$ guarantees that b executes after the two-phase commit triggered by the fence a has been completed. This two-phase commit propagates all updates known to the replica on which a was issued to all others, including the replica of b, which lets us include (a, b) into hb.

To illustrate this guarantee provided by fences, consider the execution in Figure 2(a) and assume that all writes are annotated as ordinary, but the first session issues a fence in between writing `noboss` and `photo`. In this case, by the definition of cvis and COCV, the second session will be guaranteed to see the correct access permissions. Thus, a fence subsumes the effect of annotating the write of the photo as causal. In fact, a fence has a much stronger effect. For example, if the first session posts several photos using ordinary write operations, then the fence would ensure that a session reading any of these photos will see the updated permissions. To achieve the same effect using consistency annotations, we would have to annotate all the writes of the photos as causal. This is the fundamental difference between fences and consistency annotations: the latter affect only a single operation, whereas the former affect many.

Figure 3. Axioms for a combination of per-object causal, cross-object causal and strong consistency. The definition of hbo and the axioms SOWF, RVAL, EVENTUAL, POCV, POCA, COCV, COCA are the same as in Figure 1, but with hb defined here.

VISWF: $\forall a, b.\, a \xrightarrow{\mathsf{vis}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$ and vis does not have edges involving fence actions

ARWF: $\forall a, b.\, a \xrightarrow{\mathsf{ar}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$, ar is transitive and irreflexive, does not have edges involving fence actions, and $\mathsf{ar}|_{\mathsf{vis}^{-1}(a)}$ is a total order for all $a \in A$

SCWF: sc is a total, transitive and irreflexive relation on fence actions

THINAIR: $\mathsf{so} \cup \mathsf{vis} \cup \mathsf{sc}$ is acyclic

Causal visibility relation:

$$\forall a, b.\, a \xrightarrow{\mathsf{cvis}} b \iff (a \xrightarrow{\mathsf{vis}} b \wedge \mathsf{level}(a) = \mathsf{CSL}) \vee (\exists c.\, \mathsf{op}(a) = \mathsf{fence} \wedge a \xrightarrow{\mathsf{so}} c \xrightarrow{\mathsf{hbo}} b)$$

Causality order: $\mathsf{hb} = (\mathsf{so} \cup \mathsf{cvis} \cup \mathsf{sc})^+$

As we show in §4.3, choosing one over the other to request stronger consistency impacts the implementation.

We now justify that fences enforce strong consistency: in a system with integer registers, putting a fence in between every pair of operations in a session guarantees strong consistency.

THEOREM 8. *Assume an execution* $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar}, \mathsf{sc})$ *with only* intreg *objects such that it satisfies the axioms in Figure 3 and* $\forall a, b \in A.\, \mathsf{op}(a) \neq \mathsf{fence} \wedge \mathsf{op}(b) \neq \mathsf{fence} \wedge a \xrightarrow{\mathsf{so}} b \implies \exists c.\, \mathsf{op}(c) = \mathsf{fence} \wedge a \xrightarrow{\mathsf{so}} c \xrightarrow{\mathsf{so}} b$. *Then there exists a total, transitive and irreflexive order* r *on all actions in* A *such that every read* r *from a register* x *fetches the value written by the last write to* x *preceding* r *in* r, *or* 0 *if there is no such write.*

4.3 Consistency Annotations vs Fences

In §4.1 and §4.2, we illustrated different mechanisms for requesting a stronger level of consistency: consistency annotations for causal consistency and fences for strong consistency. We now argue that the choice of the mechanisms impacts the implementation significantly.

Consider a per-object causally consistent system. Instead of using consistency annotations to request cross-object causal consistency, we could introduce a special kind of a fence, $\mathsf{fence}_{\mathsf{CSL}}$, and define cvis similarly to how it was done in §4.2:

$$\forall a, b.\, a \xrightarrow{\mathsf{cvis}} b \iff \exists c.\, \mathsf{op}(a) = \mathsf{fence}_{\mathsf{CSL}} \wedge a \xrightarrow{\mathsf{so}} c \xrightarrow{\mathsf{vis}} b.$$

Then a session that sees *any* operation c following a fence a is guaranteed to see all operations preceding the fence. To achieve this effect using consistency annotations, we would have to annotate every such operation c as causal. Hence, requesting causal consistency using fences makes it easier for the programmer to mark a series of operations as enforcing causal relationships. However, this can potentially affect the efficiency of the implementation. For the COPS algorithm (§3.3) to validate the above axiom, we would have to tag *every* action following a fence in a session with the list of actions visible to it at the time the fence was issued. A replica receiving such an action would then have to wait until all its dependencies are satisfied before making it available to clients. Unlike a consistency annotation, a fence thus increases the number of dependencies for all actions following it, with a potential impact on latency.

Such considerations need to be taken into account when designing programming interfaces for combined consistency models. A system may also provide both consistency annotations and fences (as done, e.g., in C/C++ [8]), to give the programmer maximal control at the expense of complicating the programming model. In all cases, our framework helps in exploring this design space by allowing a system developer to quickly establish the semantic consequences of various decisions and their impact on programmability.

5. Transactions

The semantics of transactions has been extensively studied in the context of databases, and various consistency models have been proposed for them, including ANSI SQL isolation levels, snapshot isolation [9] and serializability [26]. However, classical consistency models, such as the latter two, cannot be implemented while satisfying the requirements of availability and partition tolerance. For this reason, eventually consistent systems implement transactions with weaker guarantees [24, 30] that do not contradict these requirements. In this section, we show how they can be specified in our framework.

We illustrate our approach by extending the combined consistency model from §4 to accommodate transactions. The resulting semantics is similar to snapshot isolation [9], but without write-write conflict detection. Since we focus on the use of replicated data types in this paper, we do not need to disallow write-write conflicts: when an appropriate data type is used, its semantics automatically resolves conflicts, merging the two conflicting versions if needed, and the "lost update" anomaly does not occur. Hence, we do not have to consider the consequences of transactions aborting due to conflicts with others.

When a fence is used inside every transaction, our semantics coincides with serializability (Proposition 9 below). When specialised to causal consistency, it coincides with a variant of *parallel snapshot isolation*, implemented in Walter [30] and, for read-only transactions, in COPS [24]. We have not yet incorporated transactions into the single abstract implementation we use for the other features (§2–4). Hence, we justify the correspondence of our definitions to existing systems by a separate proof that their specialisation to causal consistency is equivalent to the abstract implementation of parallel snapshot isolation given by Sovran et al. [30]. The main idea of this implementation is to append writes performed by a transaction to the state of the replica it executes on atomically, and to propagate these writes to other replicas together. This ensures that transactions take effect atomically, but possibly with a delay. We formalise and prove the correspondence in §D.

To define the semantics of transactions, we assume that every action in an execution belongs to a transaction. Let us again extend the set of actions with one that separates different transactions: $a = (e, s, \mathsf{commit})$, for which we let $\mathsf{op}(a) = \mathsf{commit}$. We do not consider explicit abort operations. The changes to the axioms from Figure 3 needed to accommodate transactions are shown in Figure 4. We assume that vis, ar and $\mathsf{sameobj}$ relations do not relate commit actions.

Intuitively, if several actions are executed within a transaction, then the system should treat them as a single atomic action. Our key insight is that this can be captured by factoring the relations in an execution over an equivalence relation $\sim$, grouping actions in the same transaction. For a relation r, its factoring $\mathsf{r}/{\sim}$ includes the edges from r and those obtained from such edges by relating any other actions coming from the same transactions as their endpoints. The latter excludes the case when the endpoints themselves are from the same transaction. For vis, ar and sc we require that they be preserved under factoring (TRANSACT). We also include factoring explicitly into the definition of hb and COCA. Finally, ISOLATION ensures that uncommitted transactions are invisible to other sessions. In the following, we explain the semantics of transactions and justify the particular ways in which we use factoring over $\sim$ to define it by examples, summarised in Figure 5.

We start by showing that our notion of transactions provides basic isolation properties, ensured by factoring vis over $\sim$. Consider the execution in Figure 5(a), similar to the one in Figure 2(a), but where the user posts a photo and changes the access permission in a different order and within the same transaction. This execution is disallowed by the clause for vis in TRANSACT: even though both writes are ordinary, a session that sees the photo is guaranteed to see the updated permission. Thus, every session sees a transaction as happening either completely, or not at all. Furthermore, if the first transaction in the execution in Figure 5(a) also read the permission it wrote, it would be guaranteed to see its own write, since we assume per-object causality by default. If the second transaction read the photo twice, it would be guaranteed to see the same result due to the clause for vis in TRANSACT. Hence, transactions cannot read uncommitted data or observe a non-repeatable read.

The execution in Figure 5(b), similar to the one in Figure 2(f), illustrates the analogy with snapshot isolation by showing that our transactions allow the write skew anomaly typical for it. There, each transaction reads the initial value of a register, despite a write to it by the other transaction. This outcome would not be allowed if transactions were serializable. The execution in Figure 5(c) shows that, unlike classical snapshot isolation, our notion allows the lost update anomaly *for integer registers*. To eliminate this anomaly while satisfying the requirements of availability and partition tolerance, we can use an appropriate data type, as in the execution in Figure 5(d).

Finally, we illustrate how our notion of transactions interacts with different consistency levels in the underlying consistency model. If in the examples in Figures 2(a) and 2(f) every write or read were a transaction writing to or reading from multiple registers, then the anomalies shown in both examples would still be allowed: transactions enforce atomicity, but do not strengthen the causality guarantees among different transactions in comparison to that provided by the underlying non-transactional operations. However, ways of strengthening consistency guarantees presented §4 achieve the same effect with transactions. For example, if in the above transactional variant of the execution in Figure 2(a) any write in the photo-writing transaction were annotated as CSL, then the second session would be guaranteed to see the updated permissions.

We can also use fences from §4.2 to achieve serializability for transactions. Figure 5(e) illustrates this by implementing an operation that atomically tests if a register has reach a limit and, and if not, increments it, e.g., to make a reservation.

Figure 4. Changes to Figure 3 needed to accommodate transactions

"Same transaction" relation:
$a \sim b \iff \mathsf{op}(a) \neq \mathsf{commit} \wedge \mathsf{op}(b) \neq \mathsf{commit} \wedge$
$((a \xrightarrow{\mathsf{so}}{}^{*} b \wedge \neg\exists c.\, a \xrightarrow{\mathsf{so}} c \xrightarrow{\mathsf{so}} b \wedge \mathsf{op}(c) = \mathsf{commit}) \vee$
$(b \xrightarrow{\mathsf{so}}{}^{*} a \wedge \neg\exists c.\, b \xrightarrow{\mathsf{so}} c \xrightarrow{\mathsf{so}} a \wedge \mathsf{op}(c) = \mathsf{commit}))$

Factoring:
$\mathsf{r}/\sim \;=\; \mathsf{r} \cup \{(a,b) \mid a \not\sim b \wedge \exists a', b'.\, a \sim a' \wedge b \sim b' \wedge (a',b') \in \mathsf{r}\}$

TRANSACT: $(\mathsf{vis}/\sim) \cap \mathsf{sameobj} = \mathsf{vis}, \quad (\mathsf{ar}/\sim) \cap \mathsf{sameobj} = \mathsf{ar},$
$(\mathsf{sc}/\sim) \cap (\{a \mid \mathsf{op}(a) = \mathsf{fence}\})^2 = \mathsf{sc}$

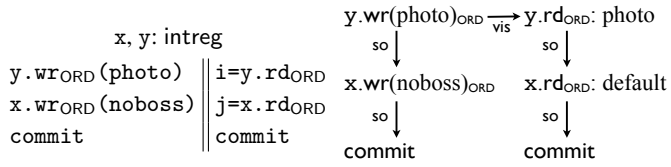
Causality order: $\mathsf{hb} = ((\mathsf{so} \cup \mathsf{cvis} \cup \mathsf{sc})/\sim)^{+}$

COCA. $(\mathsf{hb} \cup \mathsf{ar})/\sim$ is acyclic

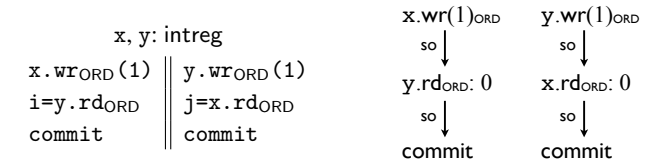
ISOLATION: $\forall a, b \in A.\, (a \xrightarrow{\mathsf{vis}} b \wedge \neg\exists c.\, a \xrightarrow{\mathsf{so}} c \wedge \mathsf{op}(c) = \mathsf{commit})$
$\implies \mathsf{ses}(a) = \mathsf{ses}(b)$

Figure 5. Anomalies allowed or disallowed by our transactions. We have omitted unimportant vis edges.

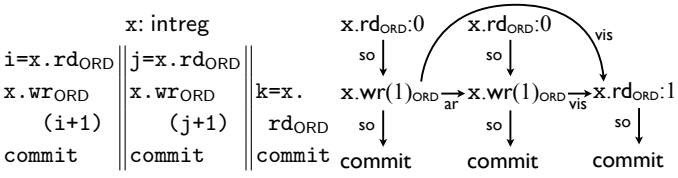
(a) Disallowed by TRANSACT:



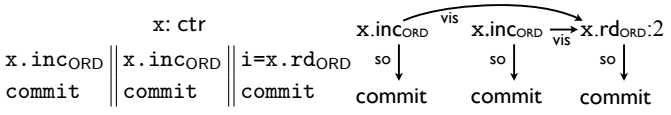
(b) Write skew is allowed:



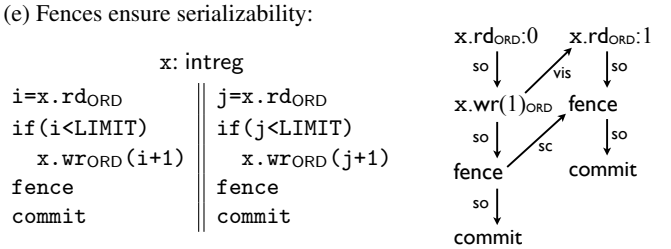
(c) Lost update can happen with integer registers:



(d) Lost update absent with an appropriate choice of data types:



(e) Fences ensure serializability:



In that execution, which is allowed by our axioms, we assume $\texttt{LIMIT} = 1$, and thus only one transaction can succeed in incrementing the register. This is ensured by the use of factoring in the definition of hb: the sc edge between the fences yields an hb edge from the write to x in the first transaction to the read from it in the second; by COCV this means that the second transaction is guaranteed to see the increment by the first one and will thus not perform increment itself. Note that we cannot guarantee this result when using the replicated counter data type: if in Figure 5(d) the first and the second transaction read the counter before incrementing it, they would both read 0. Making sure that only one transaction does an increment requires the use of fences, and correspondingly, giving up availability or partition tolerance. We formalise the guarantee provided by fences for transactions as follows.

PROPOSITION 9. *If an execution satisfies the consistency model in Figure 4, contains only* intreg *objects and has at least one fence inside every transaction, then it is serializable.*

6. Related Work

There exist several formal definitions of eventually consistent models, often proposed together with systems implementing them. In a nutshell, the difference with our work is that such specifications have so far been tied to particular data types or consistency levels, and were often very low-level.

For example, Shapiro et al. [10, 28, 29] described algorithms for a number of replicated data types on per-object causal consistency. As the correctness criterion, they consider quiescent consistency, which is less expressive than our specifications (§3.4). Bosneag and Brockmeyer [11] defined the consistency model of Bayou [32], also concentrating on a form of quiescent consistency. They handle Bayou's speculative operation execution, which we do not cover (§3.2). Fekete et al. [16] specified an eventually consistent model in an operational style, similar to our abstract implementation. Burckhardt et al. [14] defined the consistency model of the Concurrent Revisions system. Like us, they use axioms, but handle only causal consistency and data types with the semantics obtained from the generic construction using arbitration (§2).

Causal consistency was originally defined by Ahamad et al. [4]. However, their definition allows different replicas to have different, though causally consistent, views on the system evolution and thus diverge forever. Recently, Mahajan et al. [25] and Lloyd et al. [24] defined a stronger version of causal consistency that ensures convergence, which we use in this paper. Mahajan et al. define convergence in an operational model, and Lloyd et al. using explicit conflict handling functions. We give a declarative specification, with conflict handling encapsulated in a replicated data type.

Sovran et al. [30] defined a consistency model for transactions called parallel snapshot isolation, which they implemented in the Walter system. Our semantics of transactions for the case of causal consistency is equivalent to the variant of parallel snapshot isolation that Sovran et al. define for a *counting set* replicated data type (§5). In contrast to ours, their definition is given in a low-level operational style.

There exist a number of specifications of shared-memory models weaker than strong consistency [3], including those combining several consistency levels (e.g., [8]). All such models assume read-write memory cells, corresponding to our intreg data type. We handle arbitrary replicated data types and, as a consequence, our notion of executions is somewhat different from the one used to define shared-memory models (§3.1).

There have been some proposals for combining multiple consistency levels within geo-replicated databases. Li et al. [21] proposed *red-blue consistency*, similar to our on-

demand strong consistency (§4.2). In contrast to us, they assume a particular strategy of resolving conflicts using commutativity (§2). Bailis et al. [5] sketched an interface that allows a programmer to specify causality explicitly. Our formalisation of on-demand causal consistency in §4.1 and §4.3 complements their proposal with a formal semantics and a discussion of the trade-offs between different ways of specifying causality.

7. Conclusion

We have presented a flexible specification framework for eventually consistent systems that incorporates and unifies the guarantees and features that have appeared in a wide array of previous work [2, 13, 14, 17, 21, 24, 29–31]. In particular, our framework supports replicated data types and conflict resolution, session guarantees, various causality guarantees, consistency annotations, fences, and transactions.

We have illustrated how our specifications allow programmers to determine if a certain behavior is possible without considering the details of a system's implementation. Moreover, we have shown how our framework enables precise statements and proofs about how various features and guarantees are related, thus providing system architects with a tool for exploring the design space.

Finally, we hope that our use of shared-memory model techniques will help to bridge research communities and promote an exchange of ideas and results. For instance, we plan to apply several techniques developed for shared-memory models to eventually consistent systems, such as the testing and verification of programs [12], automatic inference of consistency annotations and fences [23] and compositional reasoning about components [6].

References

[1] Amazon DynamoDB. http://aws.amazon.com/dynamodb/.

[2] Basho Riak. http://basho.com/products/riak-overview/.

[3] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. *Computer*, 29(12), 1996.

[4] M. Ahamad, G. Neiger, J. Burns, P. Kohli, and P. Hutto. Causal memory: definitions, implementation, and programming. *Distributed Computing*, 9, 1995.

[5] P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. The potential dangers of causal consistency and an explicit solution (vision paper). In *SOCC*, 20012.

[6] M. Batty, M. Dodds, and A. Gotsman. Library abstraction for C/C++ concurrency. In *POPL*, 2013. To appear.

[7] M. Batty, K. Memarian, S. Owens, S. Sarkar, and P. Sewell. Clarifying and compiling C/C++ concurrency: from C++11 to POWER. In *POPL*, 2012.

[8] M. Batty, S. Owens, S. Sarkar, P. Sewell, and T. Weber. Mathematizing C++ concurrency. In *POPL*, 2011.

[9] 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.

[10] A. Bieniusa, M. Zawirski, N. M. Preguiça, M. Shapiro, C. Baquero, V. Balegas, and S. Duarte. Brief announcement: Semantics of eventually consistent replicated sets. In *DISC*, 2012.

[11] A.-M. Bosneag and M. Brockmeyer. A formal model for eventual consistency semantics. In *IASTED PDCS*, 2002.

[12] S. Burckhardt, R. Alur, and M. M. K. Martin. Checkfence: checking consistency of concurrent data types on relaxed memory models. In *PLDI*, 2007.

[13] S. Burckhardt, M. Fähndrich, D. Leijen, and B. P. Wood. Cloud types for eventual consistency. In *ECOOP*, 2012.

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

[15] N. Conway, R. Marczak, P. Alvaro, J. M. Hellerstein, and D. Maier. Logic and lattices for distributed programming. In *SOCC*, 2012.

[16] A. Fekete, D. Gupta, V. Luchangco, N. Lynch, and A. Shvartsman. Eventually-serializable data services. In *PODC*, 1996.

[17] G. DeCandia et al. Dynamo: Amazon's highly available key-value store. In *SOSP*, 2007.

[18] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2), 2002.

[19] M. Herlihy and N. Shavit. *The art of multiprocessor programming*. 2008.

[20] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7), 1978.

[21] 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.

[22] B. Liskov and S. Zilles. Programming with abstract data types. In *ACM Symposium on Very High Level Languages*, 1974.

[23] F. Liu, N. Nedev, N. Prisadnikov, M. T. Vechev, and E. Yahav. Dynamic synthesis for relaxed memory models. In *PLDI*, 2012.

[24] 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.

[25] P. Mahajan, L. Alvisi, and M. Dahlin. Consistency, availability, and convergence. Technical Report TR-11-22, UT Austin, 2011.

[26] C. Papadimitriou. *The theory of database concurrency control*. 1986.

[27] H.-G. Roh, M. Jeon, J.-S. Kim, and J. Lee. Replicated abstract data types: Building blocks for collaborative applications. *J. Parallel Distrib. Comput.*, 71(3), 2011.

[28] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Technical Report 7506, INRIA, 2011.

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

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

[31] 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.

[32] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *SOSP*, 1995.

A. Proofs

A.1 Proof of Theorem 6

POCV is equivalent to the conjunction of RYW, MR, WFRV and MWV. Pick an excution $X = (A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$. Let soo and hbo be, respectively, the per-object session order and the per-object causality order, both induced by X. Consider binary relations r on actions in A that access the same object, i.e.,

$$r \subseteq \{(a,b) \in A \times A \mid \mathsf{sameobj}(a,b)\}. \tag{2}$$

We define two operators H, G on such relations as follows:

$$H(r) = \mathsf{soo} \cup (r; \mathsf{soo}) \cup (r; \mathsf{soo}^*; r) \cup (\mathsf{soo}; r);$$
$$G(r) = (r \cup \mathsf{soo})^+.$$

We will prove that for every r satisfying (2) we have

$$H(r) \subseteq r \iff G(r) \subseteq r.$$

This gives us the required, because POCV is equivalent to $G(\mathsf{vis}) \subseteq \mathsf{vis}$, and the conjunction of RYW, MR, WFRV and MWV to $H(\mathsf{vis}) \subseteq \mathsf{vis}$.

Since $H(r) \subseteq G(r)$ for every r satisfying (2), we have that $G(r) \subseteq r \implies H(r) \subseteq r$. For the other direction, assume that $H(r) \subseteq r$. We will now show that $G(r) \subseteq r$. Pick $a, a' \in A$ such that $(a, a') \in G(r)$. We have to prove that $(a, a') \in r$. By the definition of G and the transitivity of r and soo, we have the following cases of (a, a'), which we handle separately in this proof.

If $(a, a') \in \mathsf{soo}$, then $(a, a') \in H(r)$, because $\mathsf{soo} \subseteq H(r)$. But $H(r) \subseteq r$ by our choice of r. Hence, $(a, a') \in r$, as desired.

If $(a, a') \notin \mathsf{soo}$, then there exist $a_1, b_1, \ldots, a_n, b_n$ with $n \geq 1$ such that

$$\begin{aligned}(a, a_1) \in \mathsf{soo}^* \wedge (b_n, a') \in \mathsf{soo}^* \\ \wedge (\forall i \in \{1, \ldots, n\}.\, (a_i, b_i) \in r) \\ \wedge (\forall i \in \{1, \ldots, n-1\}.\, (b_i, a_{i+1}) \in \mathsf{soo}).\end{aligned}$$

Using our assumption that $H(r) \subseteq r$, we can prove the desired $(a, a') \in r$ as follows:

$$\begin{aligned}&(a, a_1) \in \mathsf{soo}^* \wedge (a_1, b_1) \in r \wedge (b_1, a_2) \in \mathsf{soo} \wedge \ldots \\ &\qquad\qquad (a_n, b_n) \in r \wedge (b_n, a') \in \mathsf{soo}^* \\ &\implies (a, a_1) \in \mathsf{soo}^* \wedge (a_1, b_n) \in r \wedge (b_n, a') \in \mathsf{soo}^* \\ &\implies (a, a_1) \in \mathsf{soo}^* \wedge (a_1, a') \in r \\ &\implies (a, a') \in r.\end{aligned}$$

The first implication comes from the fact that $(r; \mathsf{soo}^*; r) \subseteq H(r)$, the second from $(r; \mathsf{soo}) \subseteq H(r)$, and the third from $(\mathsf{soo}; r) \subseteq H(r)$.

POCA is equivalent to the conjunction of WFRA and MWA. Pick an execution $X = (A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$. Let soo and hbo be, respectively, the per-object session order and the per-object causality order, both induced by X. Then,

$$\mathsf{hbo} = (\mathsf{soo} \cup \mathsf{vis})^+ = (\mathsf{soo} \cup (\mathsf{vis}; \mathsf{soo}^*))^+.$$

Thus, the lower bound on ar set by WFRA and MWA is included in that given by POCA. This means that POCA implies WFRA and MWA. It remains to prove that WFRA and MWA together imply POCA. Consider $(a, b) \in \mathsf{hbo}$. By the definition of hbo, there exist $a_1, \ldots, a_n$ with $n \geq 2$ such that

$$\begin{aligned}a_1 = a \wedge a_n = b \wedge \\ \forall i \in \{1, \ldots, n-1\}.\, (a_i, a_{i+1}) \in (\mathsf{soo} \cup \mathsf{vis}).\end{aligned}$$

Since WFRA and MWA hold, the third conjunct above implies that

$$\forall i \in \{1, \ldots, n-1\}.\, (a_i, a_{i+1}) \in \mathsf{ar}.$$

Furthermore, ar is transitive. Hence, we have $(a, b) \in \mathsf{ar}$, as desired. □

A.2 Proof of Theorem 8

In the following r ranges over read actions, w over write actions and f over fence actions. Consider an execution $X = (A, \mathsf{so}, \mathsf{vis}, \mathsf{ar}, \mathsf{sc})$ satisfying the assumptions of the theorem. For simplicity, we consider only the case when every read in the execution reads a non-default value. Let

$$\begin{aligned}\mathsf{vis}' &= \{(w, r) \in \mathsf{vis} \mid \mathsf{op}(w) = \mathsf{wr} \wedge \mathsf{op}(r) = \mathsf{rd}\}; \\ \mathsf{ar}' &= \{(w_1, w_2) \in \mathsf{ar} \mid \mathsf{op}(w_1) = \mathsf{wr} \wedge \mathsf{op}(w_2) = \mathsf{wr}\}.\end{aligned}$$

Let us show that $\mathsf{ar}' \cup \mathsf{vis}' \cup \mathsf{so} \cup \mathsf{sc}$ is acyclic. Assume there is a cycle in this relation. By the assumptions of the theorem, we can assume that any so edge on the cycle has a fence as one of its endpoints. Then there is at least one fence on the cycle, since ar' and vis' cannot form one due to the types of their endpoints. Consider any segment of the cycle starting an ending with a fence that has no fences in the middle. If this segment has edges other than so and sc, then it has to have one of the following forms:

- $f_1 \xrightarrow{\mathsf{so}} w_1 \xrightarrow{\mathsf{ar}'} w_2 \xrightarrow{\mathsf{vis}'} r \xrightarrow{\mathsf{so}} f_2$;
- $f_1 \xrightarrow{\mathsf{so}} w_1 \xrightarrow{\mathsf{ar}'} w_2 \xrightarrow{\mathsf{so}} f_2$;
- $f_1 \xrightarrow{\mathsf{so}} w_1 \xrightarrow{\mathsf{so}} f_2$.

Assume $(f_2, f_1) \in \mathsf{sc}$. Then the second configuration contradicts COCA, and the last one THINAIR. By POCV and the transitivity of vis, the first configuration yields $(w_2, w_1) \in \mathsf{vis}$, which contradicts POCA. Hence $(f_1, f_2) \in \mathsf{sc}$. We can thus convert the cycle into one in $\mathsf{so} \cup \mathsf{sc}$, contradicting THINAIR.

Let ar'' be any relation that is total on write actions to the same object and contains $\mathsf{ar}' \cup \mathsf{vis}' \cup \mathsf{so} \cup \mathsf{sc}$. Then $\mathsf{ar}'' \cup \mathsf{vis}' \cup \mathsf{so} \cup \mathsf{sc}$ is acyclic. Let

$$\begin{aligned}\mathsf{fr} = \{(r, w) \mid \exists w'.\, (w', r) \in \mathsf{vis} \wedge (w', w) \in \mathsf{ar} \wedge \\ \neg \exists w''.\, (w'', r) \in \mathsf{vis} \wedge (w', w'') \in \mathsf{ar}\}.\end{aligned}$$

We now show that

$$\mathsf{ar}'' \cup \mathsf{vis}' \cup \mathsf{so} \cup \mathsf{sc} \cup \mathsf{fr} \tag{3}$$

is acyclic. Assume the contrary. Then the cycle contains at least one fr edge.

Consider first the case when the cycle does not contain so or sc edges. We can assume that the cycle does not have any ar

edges, as they can only follow fr edges and can be merged with them.

Let us show that $\mathsf{fr};\mathsf{vis}';\mathsf{fr} \subseteq \mathsf{fr}$. Take $(r_1, w_4) \in \mathsf{fr};\mathsf{vis}';\mathsf{fr}$, then for some w_1, w_2, r_2, w_3 we have

$$r_1 \xleftarrow{\mathsf{vis}'} w_1 \xrightarrow{\mathsf{ar}''} w_2 \xrightarrow{\mathsf{vis}'} r_2 \xleftarrow{\mathsf{vis}'} w_3 \xrightarrow{\mathsf{ar}''} w_4$$

and $(w_2, w_3) \in \mathsf{ar}$, $\neg\exists w'.\,(w', r_1) \in \mathsf{vis}' \wedge (w_1, w') \in \mathsf{ar}$. Then $(w_1, w_4) \in \mathsf{ar}$ and hence, $(r_1, w_4) \in \mathsf{fr}$.

Thus, we can assume that the cycle has only a single fr edge. Then it also has a single vis' edge. Hence, for some r, w we have $(r, w) \in \mathsf{fr}$ and $(w, r) \in \mathsf{vis}'$. But the former implies that for some w' we have $(w', r) \in \mathsf{vis}'$, $(w', w) \in \mathsf{ar}$ and $\neg\exists w''.\,(w'', r) \in \mathsf{vis}' \wedge (w', w'') \in \mathsf{ar}$, which is a contradiction.

Assume now that the cycle in (3) has at least one so or sc edge. Consider any segment of the cycle starting and ending by a fence that does not contain fences in the middle. Then the segment does not contain any so or sc edges, except the first and the last one. Any ar edges except possibly the second one on the segment can only follow fr edges and can be merged with them. Since $\mathsf{fr};\mathsf{vis}';\mathsf{fr} \subseteq \mathsf{fr}$, we can thus assume that the segment has a single fr edge. Hence, the segment can only be of one of the following forms:

1. $f_1 \xrightarrow{\mathsf{so}} w_1 \xrightarrow{(\mathsf{ar}'')^*} w_2 \xrightarrow{\mathsf{vis}'} r_1 \xrightarrow{\mathsf{fr}} w_3 \xrightarrow{\mathsf{vis}'} r_2 \xrightarrow{\mathsf{so}} f_2$;
2. $f_1 \xrightarrow{\mathsf{so}} w_1 \xrightarrow{(\mathsf{ar}'')^*} w_2 \xrightarrow{\mathsf{vis}'} r_1 \xrightarrow{\mathsf{fr}} w_3 \xrightarrow{\mathsf{so}} f_2$;
3. $f_1 \xrightarrow{\mathsf{so}} r_1 \xrightarrow{\mathsf{fr}} w_3 \xrightarrow{\mathsf{vis}'} r_2 \xrightarrow{\mathsf{so}} f_2$;
4. $f_1 \xrightarrow{\mathsf{so}} r_1 \xrightarrow{\mathsf{fr}} w_3 \xrightarrow{\mathsf{so}} f_2$;
5. $f_1 \xrightarrow{\mathsf{so}} f_2$;
6. $f_1 \xrightarrow{\mathsf{sc}} f_2$;
7. $f_1 \xrightarrow{\mathsf{so}} w_1 \xrightarrow{(\mathsf{ar}'')^*} w_2 \xrightarrow{\mathsf{so}} f_2$;
8. $f_1 \xrightarrow{\mathsf{so}} w_1 \xrightarrow{(\mathsf{ar}'')^*} w_2 \xrightarrow{\mathsf{vis}'} r_1 \xrightarrow{\mathsf{so}} f_2$.

We now show that in all cases we must have $(f_1, f_2) \in \mathsf{sc}$. Note that this means that we can convert the cycle into one in $\mathsf{so} \cup \mathsf{sc}$, contradicting THINAIR and thus implying the acyclicity of (3).

Assume the contrary, i.e., $(f_2, f_1) \in \mathsf{sc}$. Then cases 5 and 6 contradict THINAIR, and case 7 contradicts COCA. By POCV and the transitivity of vis, case 8 contradict POCA. By the definition of hbo, in cases 1 and 2 the segment has the form

$$f_1 \xrightarrow{\mathsf{so}} w_1 \xrightarrow{(\mathsf{ar}'')^*} w_2 \xrightarrow{\mathsf{vis}'} r_1 \xleftarrow{\mathsf{vis}'} w_4 \xrightarrow{\mathsf{ar}''} w_3 \xrightarrow{(\mathsf{vis}';\mathsf{so})\cup\mathsf{so}} f_2$$

and $(w_2, w_4) \in \mathsf{ar}''$. Hence,

$$f_1 \xrightarrow{\mathsf{so}} w_1 \xrightarrow{\mathsf{ar}''} w_3 \xrightarrow{(\mathsf{vis}';\mathsf{so})\cup\mathsf{so}} f_2$$

By POCV, COCV and the transitivity of vis, $(f_2, f_1) \in \mathsf{sc}$ entails $(w_3, w_1) \in \mathsf{vis} \subseteq \mathsf{hbo}$, which contradicts POCA.

In cases 3 and 4 the segment has the form

$$f_1 \xrightarrow{\mathsf{so}} r_1 \xleftarrow{\mathsf{vis}'} w_4 \xrightarrow{\mathsf{ar}''} w_3 \xrightarrow{(\mathsf{vis}';\mathsf{so})\cup\mathsf{so}} f_2$$

and $\neg\exists w'.\,(r_1, w') \in \mathsf{vis}' \wedge (w_4, w') \in \mathsf{ar}''$. By POCV, COCV and the transitivity of vis, $(f_2, f_1) \in \mathsf{sc}$ entails $(w_3, r_1) \in \mathsf{vis}'$, which yields a contradiction.

Hence, (3) is acyclic. Let us take any total relation including (3) as the desired r. Assume that for some $r, w, w' \in A$ such that $\mathsf{obj}(w) = \mathsf{obj}(w')$ we have $(w, r) \in \mathsf{vis}$ and $(w, w') \in \mathsf{r}$. Then $(w, r) \in \mathsf{r}$, $(w, w') \in \mathsf{ar}$, and hence, $(r, w') \in \mathsf{fr}$. From this it follows that $(r, w') \in \mathsf{r}$. Thus, every read reads the most recent value according to r. □

A.3 Proof of Proposition 9

Consider an execution $X = (A, \mathsf{so}, \mathsf{vis}, \mathsf{ar}, \mathsf{sc})$. Due to the use of factoring in the definition of hb, from the assumption of the theorem we have

$$\begin{aligned}&\forall a, b \in A.\, a \not\sim b \implies\\ &\quad((\forall a', b' \in A.\, a \sim a' \wedge b \sim b' \implies (a', b') \in \mathsf{hb}) \vee\\ &\qquad(\forall a', b' \in A.\, a \sim a' \wedge b \sim b' \implies (b', a') \in \mathsf{hb})).\end{aligned}$$

Hence, hb is total on A and every transaction is contiguous in it, which for the case of integer registers implies serializability.

B. Abstract Implementation and Correspondence Theorem

THEOREM 10. *Our abstract implementation generates executions satisfying axioms in the main text of the paper, when its parameters satisfy appropriate conditions. These conditions are described in Propositions 21, 22, 23, 24, 25, 26, 27 and 28.*

In this section, we describe an abstract implementation of an eventually-consistent database system, and show that it corresponds to the axiomatic specification given in the main text of the paper. Our implementation is highly parameterised. When instantiated with appropriate parameters, it meets different levels of eventual consistency, which we have described axiomatically in the paper. The implementation is defined in terms of rules for transforming sessions and replicas. One of such rules is concerned with the interaction between a session and a replica, and it supports the availability of a replica to a session in the sense that the rule does not involve any communication among replicas.

Throughout this section, we assume a finite totally-ordered set of replica ids:

$$(\mathsf{DId}, <), \text{ ranged over by } d.$$

Also, we call a relation r is ***functional*** if

$$\begin{aligned}&\forall x, y_1, \ldots, y_n, y_1', \ldots, y_n'.\\ &\quad(x, y_1, \ldots, y_n) \in r \wedge (x, y_1', \ldots, y_n') \in r\\ &\qquad\implies y_1 = y_1' \wedge \ldots \wedge y_n = y_n'.\end{aligned}$$

Finally, for any set X, we let $\mathcal{P}_{\mathsf{fin}}(X)$ be the collection of all finite subsets of X.

B.1 Three Main Components

We start with three main components of our implementation: distributed time, message and data type implementation.

A ***distributed time*** is a totally-ordered set of the form

$$\mathsf{LTime} \times \mathsf{DId}.$$

Here LTime is a countably-infinite totally-ordered set with a function $\mathsf{next} : \mathsf{LTime} \to \mathsf{LTime}$ satisfying

$$\forall t, t' \in \mathsf{LTime}.\ \mathsf{next}(t) = t' \implies t < t'$$

and $(\mathsf{LTime} \times \mathsf{DId})$ is ordered lexicographically. We call elements in $(\mathsf{LTime} \times \mathsf{DId})$ ***distributed timestamps***.

Our abstract implementation assumes that the set of action ids is defined by a distributed time:

$$\mathsf{AId} = \mathsf{LTime} \times \mathsf{DId}.$$

A ***message*** is a tuple $m = (x, e, f, W, E)$ in the following set:

$$\mathsf{Obj} \times \mathsf{AId} \times (\bigcup_\tau \mathsf{Op}_\tau) \times \mathcal{P}_{\mathsf{fin}}(\mathsf{AId}) \times \mathcal{P}_{\mathsf{fin}}(\mathsf{AId}).$$

In our intended usage, a message (x, e, f, W, E) represents an operation f performed on the object x at the timestamp e. When this operation was originally issued in a replica, the message says, the actions with ids in W were visible in the replica. When the operation is later propagated to another remote replica, it will stay in the target replica without being executed until all the actions with ids in E are performed on the replica. We will write Msg for the set of messages.

A ***data type implementation*** is a family of tuples

$$\begin{aligned}(&\mathsf{St}_\tau,\ \mathsf{init}_\tau : \mathsf{St}_\tau,\\ &\quad \mathsf{eval}_\tau : \mathsf{AId} \times \mathsf{Op}_\tau \times \mathcal{P}_{\mathsf{fin}}(\mathsf{AId}) \times \mathsf{St}_\tau \to \mathsf{Val} \times \mathsf{St}_\tau,\\ &\quad \mathsf{id}_\tau : \mathsf{St}_\tau \to \mathcal{P}_{\mathsf{fin}}(\mathsf{AId}))\end{aligned}$$

indexed by each data type τ, such that for every τ,

$$\begin{aligned}&(\mathsf{id}_\tau(\mathsf{init}_\tau) = \emptyset) \wedge\\ &(\forall e, f, W.\, \forall \sigma, \sigma' \in \mathsf{St}_\tau.\\ &\quad \mathsf{eval}_\tau(e, f, W, \sigma) = (\_, \sigma') \implies \mathsf{id}_\tau(\sigma') = \mathsf{id}_\tau(\sigma) \cup \{e\}).\end{aligned}$$

The init_τ is the initial state of a data type τ, the next $\mathsf{eval}_\tau(e, f, W, \sigma)$ describes the outcome of running the operation f on the state σ, where the id of the associated action is e and this action is issued when actions in W are visible from the replica. The last $\mathsf{id}_\tau(\sigma)$ returns the ids of all the actions that have influenced on the computation leading to σ.

B.2 Database Implementation

Our implementation of a distributed database assumes the three components that we have just described. It defines the information stored in sessions and replicas, and specifies a protocol that these replicas and sessions need to follow in order to guarantee a desired level of eventual consistency. In the following, we describe the implementation step-by-step:

1. Firstly, we assume a distributed time, a next operator and a data type implementation:

   $$\begin{aligned}&\mathsf{LTime} \times \mathsf{DId}, \qquad \mathsf{next} : \mathsf{LTime} \to \mathsf{LTime},\\ &\{(\mathsf{St}_\tau, \mathsf{init}_\tau, \mathsf{eval}_\tau, \mathsf{id}_\tau)\}_\tau.\end{aligned}$$

2. Secondly, we define configurations for replicas, sessions and entire systems.

   - The set of replica configurations, DB, is given by

     $$\begin{aligned}&\mathsf{Table} = \mathsf{Obj} \to \bigcup_\tau \mathsf{St}_\tau,\\ &\mathsf{DB} = \mathsf{DId} \times \mathsf{LTime} \times \mathsf{Table} \times \mathcal{P}_{\mathsf{fin}}(\mathsf{Msg}) \times \mathcal{P}_{\mathsf{fin}}(\mathsf{Msg}).\end{aligned}$$

     A tuple $(d, t, \rho, M, N) \in \mathsf{DB}$ represents a status of a replica. The first component d is the id of the replica, and (t, d) is the timestamp that the replica maintains for itself. The next ρ is a table for storing objects. The following M consists of messages about actions that were performed on the replica and need to be propagated to other remote replicas. The last component N contains the other kind of messages, those that describe operations performed in other remote replicas and are propagated from them to the current replica.

   - The set of session configurations, $\mathsf{Session}$, is defined as follows:

     $$\mathsf{Session} = \mathsf{SId} \times \mathsf{SCtx},\ \text{ ranged over by } \theta \text{ or } (s, \kappa),$$

     Here SCtx is an unspecified set and it contains session contexts, which store session-specific information. We do not fix what goes into a session context. Various choices of SCtx will emerge as we later consider axioms of eventual consistency.

   - A system configuration is a pair

     $$\mathcal{S} \mid \mathcal{D}$$

     where $\mathcal{S}$ is a subset of $\mathsf{Session}$ and $\mathcal{D}$ is a subset of DB. We require that both $\mathcal{S}$ and $\mathcal{D}$ be functional, and they meet the following condition:

     $$\begin{aligned}\forall e \in \mathsf{id}(\mathcal{S}|\mathcal{D}).\, \exists t.\, \exists (d, t', \_, \_, \_) \in \mathcal{D}.\\ e = (t, d) \wedge e < (t', d),\end{aligned}$$

     where the set $\mathsf{id}(\mathcal{S}|\mathcal{D})$ consists of all action ids appearing in $\mathcal{S}|\mathcal{D}$ and is formally defined by:

     $$\begin{aligned}&\mathsf{id}(\mathcal{S}|\mathcal{D}) =\\ &\{e \mid \exists (\_, \_, \rho, M, N) \in \mathcal{D}.\\ &\quad (\exists x \in \mathsf{Obj}.\, e \in \mathsf{id}_{\mathsf{type}(x)}(\rho(x)))\\ &\quad \vee \exists (\_, e', \_, W, E) \in M \cup N.\, e \in W \cup E \cup \{e'\}\}.\end{aligned}$$

     This condition means that according to the distributed time, every action in $\mathcal{D}$ are performed in the past, if we use timestamps of replicas as our baseline.

     We use comma to mean the disjoint union, and represent a singleton set $\{x\}$ simply by its element x. For instance, $(\mathcal{S}, \theta)$ means the union of $\mathcal{S}$ and the singleton set $\{\theta\}$ where θ does not belong to $\mathcal{S}$. Also, we use the following operation on system configurations:

     $$\mathsf{time}(\mathcal{S}|\mathcal{D}) = \{(t, d) \mid (d, t, \_, \_, \_) \in \mathcal{D}\}.$$

3. Thirdly, we assume the presence of the following operations

for all data types τ:

$$\begin{aligned}
\mathsf{instr} &: \mathsf{Session} \to \mathsf{Obj} \times \mathsf{Op} \\
\mathsf{update}_\tau &: \mathsf{Obj}_\tau \times \mathsf{AId} \times \mathsf{Op}_\tau \times \mathsf{St}_\tau \times \mathsf{Table} \times \mathsf{SCtx} \to \mathsf{SCtx} \\
\mathsf{depend}_\tau &: \mathsf{AId} \times \mathsf{Op}_\tau \times \mathsf{St}_\tau \times \mathsf{Table} \times \mathsf{SCtx} \to \mathcal{P}_{\mathsf{fin}}(\mathsf{AId}) \\
\mathsf{enable} &: \mathsf{Obj} \times \mathsf{Session} \times \mathsf{DB} \to \{\mathsf{true}, \mathsf{false}\}
\end{aligned}$$

where $\mathsf{Obj}_\tau = \{x \in \mathsf{Obj} \mid \mathsf{type}(x) = \tau\}$ and the depend operator is required to satisfy the condition below:

$$\mathsf{depend}_\tau(e, f, \sigma, \rho, \kappa) \subseteq (\mathsf{id}_\tau(\sigma) \cup \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x))).$$

The first operation selects the next instruction to be executed by a session, and the second describes how running this instruction changes the session's status. The third operation takes the id of an action, and computes the ids of other actions that should be performed on a replica before the given action. The last is a predicate that checks whether a session can access a replica without violating a desired level of eventual consistency.

The last three operations are the knobs on our abstract implementation, which can be adjusted to achieve a different level of eventual consistency. The details will be given in the rest of this section.

4. Finally, we define our implementation in terms of rules for transforming system configurations:

$$\frac{\begin{array}{l}\delta_i = (d_i, t_i, \rho_i, M_i, N_i) \quad m = (x, e, f, W, E) \in M_1 \\ e \notin \mathsf{id}_{\mathsf{type}(x)}(\rho_2(x)) \\ \delta_2' = (d_2, \mathsf{max}(\mathsf{next}(t_1), t_2), \rho_2, M_2, N_2 \cup \{m\})\end{array}}{\mathcal{S} \mid \mathcal{D}, \delta_1, \delta_2 \xrightarrow{m} \mathcal{S} \mid \mathcal{D}, \delta_1, \delta_2'} \textsc{Prop}$$

$$\frac{\begin{array}{l}\delta = (d, t, \rho, M, N) \qquad (x, e, f, W, E) \in N \\ \tau = \mathsf{type}(x) \qquad E \subseteq \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y)) \\ e \notin \mathsf{id}_\tau(\rho(x)) \qquad (\_, \sigma') = \mathsf{eval}_\tau(e, f, W, \rho(x)) \\ \delta' = (d, t, \rho[x \mapsto \sigma'], M, N)\end{array}}{\mathcal{S} \mid \mathcal{D}, \delta \xrightarrow{()} \mathcal{S} \mid \mathcal{D}, \delta'} \textsc{OpD}$$

$$\frac{\begin{array}{l}\theta = (s, \kappa) \qquad \delta = (d, t, \rho, M, N) \qquad e = (t, d) \\ \mathsf{instr}(\theta) = (x, f) \quad \tau = \mathsf{type}(x) \quad \mathsf{enable}(x, \theta, \delta) \\ \kappa' = \mathsf{update}_\tau(x, e, f, \rho(x), \rho, \kappa) \qquad \theta' = (s, \kappa') \\ W' = \mathsf{id}_\tau(\rho(x)) \qquad E' = \mathsf{depend}_\tau(e, f, \rho(x), \rho, \kappa) \\ \mathsf{eval}_\tau(e, f, W', \rho(x)) = (w, \sigma') \quad a = (e, s, [x.f : w]) \\ \delta' = (d, \mathsf{next}(t), \rho[x \mapsto \sigma'], M \cup \{(x, e, f, W', E')\}, N)\end{array}}{\mathcal{S}, \theta \mid \mathcal{D}, \delta \xrightarrow{(W', a)} \mathcal{S}, \theta' \mid \mathcal{D}, \delta'} \textsc{OpS}$$

The first rule describes the communication between two replicas, which propagates a message from one replica to another. When such a message arrives, the receiving replica waits until an associated causality condition for the message is met. Once this waiting condition is met, the replica updates its local object table according to the second rule. The last rule describes the access to a replica by a session. This generates a new action, which is later propagated to other replicas in the form of a message. Also, it changes the configurations of session and replica that interact, as described by the rule.

LEMMA 11. *If $\mathcal{S}|\mathcal{D} \xrightarrow{\iota} \mathcal{S}'|\mathcal{D}'$ and $\mathcal{S}|\mathcal{D}$ is well-formed, so is $\mathcal{S}'|\mathcal{D}'$.*

Proof: Assume that $\mathcal{S}|\mathcal{D}$ is well-formed and $\mathcal{S}|\mathcal{D} \xrightarrow{\iota} \mathcal{S}'|\mathcal{D}'$. We should show that $\mathcal{S}'|\mathcal{D}'$ is also well-formed. This means two properties of $\mathcal{S}'|\mathcal{D}'$. First, both $\mathcal{S}'$ and $\mathcal{D}'$ are functional. Second,

$$\begin{aligned}\forall e \in \mathsf{id}(\mathcal{S}'|\mathcal{D}').\, \exists t.\, \exists (d, t', \_, \_, \_) \in \mathcal{D}'. \\ e = (t, d) \wedge e < (t', d).\end{aligned}$$

The first property follows from the well-formedness of $\mathcal{S}|\mathcal{D}$ and the fact that all three rules in our abstract implementation do not create any session configurations nor replica configurations. For the second property, we show it by case analysis on the rule used to obtain $\mathcal{S}'|\mathcal{D}'$.

- If the rule is PROP, we have

$$\begin{aligned}&\mathsf{id}(\mathcal{S}|\mathcal{D}) = \mathsf{id}(\mathcal{S}'|\mathcal{D}') \\ &\wedge (\forall (t, d) \in \mathsf{time}(\mathcal{S}|\mathcal{D}).\, \exists t'. \\ &\qquad (t', d) \in \mathsf{time}(\mathcal{S}'|\mathcal{D}') \wedge (t, d) \leq (t', d')).\end{aligned}$$

  The desired property of $\mathcal{S}'|\mathcal{D}'$ follows from these two conjuncts and the well-formedness of $\mathcal{S}|\mathcal{D}$.
- If the rule is OPD, we have

$$\mathsf{id}(\mathcal{S}|\mathcal{D}) = \mathsf{id}(\mathcal{S}'|\mathcal{D}') \wedge \mathsf{time}(\mathcal{S}|\mathcal{D}) = \mathsf{time}(\mathcal{S}'|\mathcal{D}'). \quad (4)$$

  Here the second conjunct holds because the rule does not change the timestamps of any replicas. The first conjunct holds because the OPD rule changes only one replica by running eval, but

$$\begin{aligned}(\_, \sigma') = \mathsf{eval}_\tau(e, f, W, \rho(x)) \\ \implies \mathsf{id}_\tau(\sigma') = \mathsf{id}_\tau(\rho(x)) \cup \{e\}.\end{aligned}$$

  The second property of $\mathcal{S}'|\mathcal{D}'$ follows from (4) and the well-formedness of $\mathcal{S}|\mathcal{D}$.
- The remaining case is that the rule is OPS. By the definition of OPS, we have

$$\begin{aligned}\forall (t, d) \in \mathsf{time}(\mathcal{S}|\mathcal{D}).\, \exists t'. \\ (t', d) \in \mathsf{time}(\mathcal{S}'|\mathcal{D}') \wedge (t, d) \leq (t', d').\end{aligned} \quad (5)$$

  Pick $e' \in \mathsf{id}(\mathcal{S}'|\mathcal{D}')$. Let (x, e, f, W', E') be a message generated by the rule. If e' does not belong to $\{e\} \cup W' \cup E'$, it should be in $\mathsf{id}(\mathcal{S}|\mathcal{D})$, and the desired property of $\mathcal{S}'|\mathcal{D}'$ follows from (5) and the well-formedness of $\mathcal{S}|\mathcal{D}$. If $e' = e$, by the definition of the rule, we have

$$\exists d, t.\, (t, d) = e' \wedge (d, \mathsf{next}(t), \_, \_, \_) \in \mathcal{D}',$$

  from which follows the desired property of $\mathcal{S}'|\mathcal{D}'$. Finally, if $e' \in (W' \cup E') \setminus \{e\}$, we use the condition on the depend operator and the definition of W', and derive that

$$\begin{aligned}e' \in ((W' \cup E') \setminus \{e\}) &\subseteq \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y)) \\ &\subseteq \mathsf{id}(\mathcal{S}|\mathcal{D}),\end{aligned}$$

  where ρ is the fourth argument used for computing E' in the OPS rule. Now the well-formedness of $\mathcal{S}|\mathcal{D}$ and the formula in (5) give the desired property of $\mathcal{S}'|\mathcal{D}'$.

□

Let $\mathsf{id}(\iota) = \{\mathsf{id}(a) \mid (\_, a) = \iota\}$. We show that the rules of our implementation increase the set of action ids occurring in a system configuration.

LEMMA 12. *If* $\mathcal{S}|\mathcal{D} \xrightarrow{\iota} \mathcal{S}'|\mathcal{D}'$, *then*

$$\mathsf{id}(\mathcal{S}|\mathcal{D}) \subseteq \mathsf{id}(\mathcal{S}'|\mathcal{D}') \wedge \mathsf{id}(\mathcal{S}|\mathcal{D}) \cup \mathsf{id}(\iota) = \mathsf{id}(\mathcal{S}'|\mathcal{D}').$$

Proof: We do the case analysis on the rule used in the transition:

$$\mathcal{S}|\mathcal{D} \xrightarrow{\iota} \mathcal{S}'|\mathcal{D}'.$$

The first case is that the rule is PROP. The rule PROP only delivers an existing message to a new target replica, and does not change the object table of any replica. Furthermore, $\mathsf{id}(\iota) = \emptyset$. The lemma follows from these properties of PROP.

The next case is OPS. As in the previous case, OPS only adds a new message (x, e, f, W', E') to some replica, where all action ids in $W' \cup E'$ already appear in the same replica before the rule. Also, although OPS changes the object table of a replica, this change happens only via the execution of eval, which satisfies the following property:

$$(\_, \sigma') = \mathsf{eval}_\tau(e, f, W, \sigma) \implies \mathsf{id}_\tau(\sigma) \cup \{e\} = \mathsf{id}_\tau(\sigma').$$

Since $\mathsf{id}(\iota) = \{e\}$, the claim of this lemma follows from what we have just described.

The final case is OPD. In this case, no message is removed from or added to a replica. The object table of some replica is changing by OPD, but this is done via the execution of eval. Hence, as explained in the previous case, this change only increases the set of action ids in the object table of a replica, and this increment is the action id e of the message (x, e, f, W, E) that is selected by the rule. Note that this message already exists in the replica before the application of the rule, so e is not a new action id. From the observations that we have just made follows that $\mathsf{id}(\mathcal{S}|\mathcal{D}) \subseteq \mathsf{id}(\mathcal{S}'|\mathcal{D}')$. Since $\mathsf{id}(\iota) = \emptyset$, the claim of this lemma holds. □

A message (x, e, f, W, E) ***respects time*** if

$$\forall e' \in E \cup W.\, e' < e.$$

A system configuration $\mathcal{S}|\mathcal{D}$ ***respects time*** if all messages in the configuration respect time and the following condition is met for every replica $(d, t, \rho, M, N) \in \mathcal{D}$:

$$(\forall x \in \mathsf{Obj}.\, \forall e \in \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).\, e < (t, d)) \wedge$$
$$(\forall (\_, e, \_, W, E) \in M \cup N.\, \forall e' \in \{e\} \cup W \cup E.\, e' < (t, d)).$$

LEMMA 13. *If* $\mathcal{S}|\mathcal{D} \xrightarrow{\iota} \mathcal{S}'|\mathcal{D}'$ *and* $\mathcal{S}|\mathcal{D}$ *respects time, then* $\mathcal{S}'|\mathcal{D}'$ *also respects time.*

Proof: We do the case analysis on the rule used in the transition:

$$\mathcal{S}|\mathcal{D} \xrightarrow{\iota} \mathcal{S}'|\mathcal{D}'.$$

The first case is PROP. The rule delivers a message (x, e, f, W, E) from one replica to another, but it does not create any new messages. Thus, all the messages in $\mathcal{S}'|\mathcal{D}'$ are also in $\mathcal{S}|\mathcal{D}$, so they should respect time. It remains to prove that the condition on a replica's timestamp holds for all replicas in $\mathcal{D}'$. If a replica in $\mathcal{D}'$ is not modified by the application of PROP, this condition follows from our assumption on $\mathcal{S}|\mathcal{D}$. Let $\delta = (d, t, \rho, M, N) \in \mathcal{D}'$ be the replica modified by PROP, and let $m = (x, e, f, W, E)$ be the message delivered by PROP. Pick an action id e_0 such that

$$(\exists (\_, e', \_, W', E') \in M \cup N.\, e_0 \in \{e'\} \cup W' \cup E')$$
$$\vee\, (\exists x \in \mathsf{Obj}.\, e_0 \in \mathsf{id}_{\mathsf{type}(x)}(\rho(x))).$$

If the second disjunct holds or the first disjunct is true with a witness different from m, the condition on the timestamp of δ follows from our assumption on $\mathcal{S}|\mathcal{D}$. Otherwise,

$$e_0 \in \{e\} \cup W \cup E.$$

Let (d', t') be the timestamp of a replica that sent the message m in our application of PROP. By our assumption on $\mathcal{S}|\mathcal{D}$,

$$e_0 < (t', d').$$

But by the definition of PROP, $(t', d') < (t, d)$. Hence, $e_0 < (t, d)$ as desired.

The second case is OPD. This rule does not create any new messages. Since every message in $\mathcal{S}|\mathcal{D}$ respects time, so do the ones in $\mathcal{S}'|\mathcal{D}'$. We can thus complete the proof of this case if we show that the condition on the timestamp of a replica holds for every replica in $\mathcal{D}'$. Note that the OPD rule changes only one replica. For all the other unchanged replicas in $\mathcal{D}'$, the condition on their timestamps holds, because these replicas are already present in $\mathcal{S}|\mathcal{D}$ and every replica in $\mathcal{S}|\mathcal{D}$ satisfies the condition on its timestamp. Let δ, δ' be replicas in $\mathcal{D}$ and $\mathcal{D}'$, respectively, such that the OPD rule changes δ to δ'. Then, there are $d, t, \rho, M, N, x, e, f, W, E, \sigma'$ such that

$$\delta = (d, t, \rho, M, N)$$
$$\wedge\, \delta' = (d, t, \rho[x \mapsto \sigma'], M, N)$$
$$\wedge\, (x, e, f, W, E) \in N \wedge (\_, \sigma') = \mathsf{eval}_{\mathsf{type}(x)}(e, f, W, \rho(x)).$$

By the condition imposed on eval, the above formula implies that every action id e' appearing in δ' also occurs in δ. By the assumption on $\mathcal{S}|\mathcal{D}$, this implies that

$$e' < (t, d)$$

which is precisely what we have to prove.

The final case is OPS. Let δ, δ' be replicas in $\mathcal{D}$ and $\mathcal{D}'$ such that the OPS rule transforms δ to δ'. Then, there exist $d, t, \rho, M, N, x, \sigma', f, W', E'$ satisfying the following properties:

$$(\delta = (d, t, \rho, M, N)) \wedge$$
$$((\_, \sigma') = \mathsf{eval}_{\mathsf{type}(x)}(\_, \_, \_, \rho(x))) \wedge$$
$$(W' = \mathsf{id}_{\mathsf{type}(x)}(\rho(x))) \wedge$$
$$(E' = \mathsf{depend}_{\mathsf{type}(x)}(\_, \_, \rho(x), \rho, \_)) \wedge$$
$$(\delta' = (d, \mathsf{next}(t), \rho[x \mapsto \sigma'], M \cup \{(x, (t, d), f, W', E')\}, N))$$

Every message in $\mathcal{D}'$ other than $m = (x, (t, d), f, W', E')$ exists in $\mathcal{D}$, so it reflects time. By the condition imposed on depend and the definition of W',

$$(E' \cup W') \subseteq \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y))$$

Hence, by our assumption on $\mathcal{S}|\mathcal{D}$,

$$\forall e' \in E' \cup W'.\, e' < (t, d).$$

Hence, m also respects time. It remains to show the condition on the timestamp of each replica in $\mathcal{D}'$. This condition is met for all replicas in $\mathcal{D}'$ other than δ', because they are already present in $\mathcal{D}$ and every replica in $\mathcal{D}$ satisfies the same condition on its timestamp. Pick an action id e' in δ'. Then, e' already appears in δ, or

$$\begin{aligned} e' &\in \mathsf{id}_{\mathsf{type}(x)}(\sigma') \cup \{(t, d)\} \cup E' \cup W' \\ &= \{(t, d)\} \cup \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y)) \end{aligned}$$

where the equality above uses the conditions on id and depend. If $e' = (t, d)$, then

$$e' = (t, d) < (\mathsf{next}(t), d).$$

Otherwise, e' appears in δ already. So in this case, by our assumption on $\mathcal{S}|\mathcal{D}$, we have

$$e' < (t, d) < (\mathsf{next}(t), d).$$

In both cases, we proved that the condition on the timestamp of δ' holds, as desired. □

A configuration $\mathcal{S}|\mathcal{D}$ is ***empty*** if for all $(\_, \_, \rho, M, N) \in \mathcal{D}$,

$$(M = N = \emptyset) \wedge (\forall x \in \mathsf{Obj}.\, \rho(x) = \mathsf{init}_{\mathsf{type}(x)}).$$

COROLLARY 14. *If $\mathcal{S}_0|\mathcal{D}_0$ is empty and*

$$\mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \dots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n$$

then $\mathcal{S}_n|\mathcal{D}_n$ respects time.

Proof: This corollary follows from Lemma 13 and the fact that all empty system configurations respect time. □

B.3 Fair Trace

We represent a computation of our implementation in terms of a trace:

DEFINITION 15. *A* **trace** *is a tuple*

$$(\mathcal{S}_0, \mathcal{D}_0, \{(\iota_k, \mathcal{S}_k, \mathcal{D}_k)\}_{1 \le k \le n})$$

for $n \in \mathbb{N} \cup \{\omega\}$, such that $\mathcal{S}_0|\mathcal{D}_0$ is empty and

$$\forall k.\, (1 \le k \le n) \implies (\mathcal{S}_{k-1}|\mathcal{D}_{k-1} \xrightarrow{\iota_k} \mathcal{S}_k|\mathcal{D}_k).$$

We will often present traces in the following more readable form:

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \dots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n$$

LEMMA 16. *Every trace*

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \dots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n,$$

satisfies the following properties:

1. *For every i,*

$$\mathsf{id}(\mathcal{S}_i|\mathcal{D}_i) = \{\mathsf{id}(a) \mid \exists k.\, (\_, a) = \iota_k \wedge k \le i\}.$$

   This implies that $\mathsf{id}(\mathcal{S}_i|\mathcal{D}_i) \subseteq \mathsf{id}(\mathcal{S}_j|\mathcal{D}_j)$ for all i, j with $i < j$.
2. *For all $i \in \{1, \dots, n\}$ and $a \in \mathsf{Act}$, if $(\_, a) = \iota_i$, then*

$$\mathsf{id}(a) \notin \mathsf{id}(\mathcal{S}_{i-1}|\mathcal{D}_{i-1}) \wedge \mathsf{id}(a) \in \mathsf{id}(\mathcal{S}_i|\mathcal{D}_i).$$

3. *For all $i, j \in \{1, \dots, n\}$ and $a, b \in \mathsf{Act}$, if*

$$(\_, a) = \iota_i \wedge (\_, b) = \iota_j \wedge i \neq j$$

   then $\mathsf{id}(a) \neq \mathsf{id}(b)$.

Proof: The first property follows from Lemma 12 and the fact that $\mathsf{id}(\mathcal{S}_0|\mathcal{D}_0) = \emptyset$. To prove the second, consider i, a such that $(\_, a) = \iota_i$. By the definition of our rules, $\mathsf{id}(a)$ should come from the timestamp and the id of a replica in the configuration $\mathcal{S}_{i-1}|\mathcal{D}_{i-1}$. By the well-formedness of $\mathcal{S}_{i-1}|\mathcal{D}_{i-1}$, this means that $\mathsf{id}(a)$ is not in $\mathsf{id}(\mathcal{S}_{i-1}|\mathcal{D}_{i-1})$. Also, by the definition of our rules, $\mathsf{id}(a)$ should be in $\mathsf{id}(\mathcal{S}_i|\mathcal{D}_i)$. We have just shown that the second property holds. The last property in the lemma is a consequence of the other two properties. □

LEMMA 17. *Consider a trace*

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \dots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n.$$

For all i, $(\_, \_, \_, M, N) \in \mathcal{D}_i$ and $(\_, e, \_, W, \_) \in M \cup N$, there exist a, j such that

$$j \le i \wedge (W, a) = \iota_j \wedge \mathsf{id}(a) = e.$$

Proof: By our condition on the trace, there are no messages in the initial configuration $\mathcal{S}_0|\mathcal{D}_0$. Hence, all messages in some $\mathcal{D}_i$ are generated by the application of the OPS rule. So, for every replica $(\_, \_, \_, M, N) \in \mathcal{D}_i$ and every message $m = (\_, e, \_, W, \_) \in M \cup N$ in this replica, there is an index k such that the message is generated by OPS in the k-th step. This k and the action generated at this step are the desired j and a in this lemma. □

LEMMA 18. *Consider a trace*

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \dots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n.$$

For all i, $e \in \mathsf{id}(\mathcal{S}_i|\mathcal{D}_i)$, $(\_, \_, \rho, M, N) \in \mathcal{D}_i$ and $x \in \mathsf{Obj}$, if

$$e \in \mathsf{id}_{\mathsf{type}(x)}(\rho(x)) \vee (\exists W.\, (x, \_, \_, W, \_) \in M \cup N \wedge e \in W)$$

then

$$\exists k \le i.\, \exists a.\, (\_, a) = \iota_k \wedge \mathsf{id}(a) = e \wedge \mathsf{obj}(a) = x.$$

Proof: Pick i, ρ, M, N, e, x such that

$$((\_, \_, \rho, M, N) \in \mathcal{D}_i) \wedge$$
$$(e \in \mathsf{id}_{\mathsf{type}(x)}(\rho(x)) \vee \exists W.\, (x, \_, \_, W, \_) \in M \cup N \wedge e \in W).$$

By definition, the initial configuration of the trace τ should be empty. This implies that $\mathsf{id}(\mathcal{S}_0|\mathcal{D}_0) = \emptyset$. Hence, by Lemma 12, there exist k, a such that

$$k \leq i \wedge (\_, a) = \iota_k \wedge \mathsf{id}(a) = e.$$

It remains to show that $\mathsf{obj}(a) = x$. We show this by induction on $i - k$. Suppose that $i = k$. Then, by the well-formedness of $\mathcal{S}_{i-1}|\mathcal{D}_{i-1}$, e is not in $\mathsf{id}(\mathcal{S}_{i-1}|\mathcal{D}_{i-1})$. This means that the state $\rho(x)$ or the message $(x, \_, \_, W, \_)$ is produced by the rule at the k-th step. This can happen only when $x = \mathsf{obj}(a)$. Now suppose that $i > k$. If e appears in a message, this message must already exist in $\mathcal{S}_{i-1}|\mathcal{D}_{i-1}$. In this case, by induction hypothesis, we get $x = \mathsf{obj}(a)$. Otherwise, e is in $\mathsf{id}_{\mathsf{type}(x)}(\rho(x))$. Let ρ' be the object table of the same replica in $\mathcal{S}_{i-1}|\mathcal{D}_{i-1}$. If e is already in $\mathsf{id}_{\mathsf{type}(x)}(\rho'(x))$, we can use induction hypothesis and deduce that $x = \mathsf{obj}(a)$ as desired. If not, this means that the OPD rule was applied at the i-th step to $\rho'(x)$ and some message (x, e, f', W', E'), and produced $\rho(x)$. Now we can apply the induction hypothesis on the message (x, e, f', W', E'), and obtain the desired conclusion that $x = \mathsf{obj}(a)$. □

LEMMA 19. *Consider a trace*

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \ldots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n.$$

Every configuration $\mathcal{S}_i|\mathcal{D}_i$ respects time.

Proof: Every empty configuration respects time. Furthermore, by Lemma 13, all the rules in our abstract implementation transforms time-respecting configurations to time-respecting ones. This lemma follows from these two observations. □

DEFINITION 20. *A trace*

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \ldots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n$$

is **fair** *if for every action a such that $(\_, a) = \iota_k$ for some k, there is a threshold $1 \leq k_0 \leq n$ satisfying the following condition:*

$$\forall i \geq k_0.\, \forall (\_, \_, \rho, \_, \_) \in \mathcal{D}_i.$$
$$\exists x \in \mathsf{Obj}.\, x = \mathsf{obj}(a) \wedge \mathsf{id}(a) \in \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

B.4 Correspondence: Basic Axioms

We now relate the abstract implementation with the axiomatic description given in the main part of this paper. Our plan is to show that every fair trace generates an execution in our axiomatic semantics that satisfies all the well-formedness axioms, the data type axiom and the basic eventual consistency axioms in Figure 1. Once this basic correspondence is established, we describe further conditions on the parameters of our implementation, and show that they validate axioms for session guarantees and causality.

Consider a fair trace

$$\tau = (\mathcal{S}_0, \mathcal{D}_0, \{(\iota_k, \mathcal{S}_k, \mathcal{D}_k)\}_k).$$

We generate an execution $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$ from this trace τ as follows:

$$A = \{a \mid \exists k.\, (\_, a) = \iota_k\},$$
$$\mathsf{vis} = \{(a, b) \in A \times A \mid \exists k, W.\, (W, b) = \iota_k \wedge \mathsf{id}(a) \in W\},$$
$$\mathsf{so} = \{(a, b) \in A \times A \mid \exists k, l.\, k < l \wedge (\_, a) = \iota_k \wedge (\_, b) = \iota_l \wedge \mathsf{ses}(a) = \mathsf{ses}(b)\},$$
$$\mathsf{ar} = \{(a, b) \in A \times A \mid \mathsf{id}(a) < \mathsf{id}(b) \wedge \mathsf{obj}(a) = \mathsf{obj}(b)\}.$$

This execution assumes a replicated data type specification $\mathcal{F}$ that is compatible with a data type implementation

$$\{(\mathsf{St}_\tau, \mathsf{init}_\tau, \mathsf{eval}_\tau, \mathsf{id}_\tau)\}_\tau$$

in the following sense: for every type τ, there exists a relation $\mathcal{R}_\tau$ between the last three components of contexts for the type τ and data type states in St_τ such that

1. $((\emptyset, \emptyset, \emptyset), \mathsf{init}_\tau) \in \mathcal{R}_\tau$;
2. if

$$((V, r_0, r_1), \sigma) \in \mathcal{R}_\tau$$
$$\wedge\, \mathsf{eval}_\tau(e, f, W, \sigma) = (w, \sigma') \wedge (\forall g \in \mathsf{Op}_\tau.\, (e, g) \notin V)$$
$$\wedge\, r_0' \supseteq (r_0 \cup \{((e', g), (e, f)) \mid e' \in W \wedge (e', g) \in V\})$$
$$\wedge\, r_1' = r_1 \cup \{((e', g), (e, f)) \mid e' < e \wedge (e', g) \in V\} \cup \{((e, f), (e', g)) \mid e < e' \wedge (e', g) \in V\}$$

then

$$((V \cup \{(e, f)\}, r_0', r_1'), \sigma') \in \mathcal{R}_\tau \wedge \mathcal{F}_\tau(f, V, r_0, r_1) = w;$$

PROPOSITION 21. *All the fair traces generate executions satisfying* SOWF, VISWF, ARWF, EVENTUAL *and* THINAIR.

Proof: Pick a fair trace τ:

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \ldots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n$$

where $n \in \mathbb{N} \cup \{\omega\}$. Let $(A, \mathsf{vis}, \mathsf{so}, \mathsf{ar})$ be an execution constructed from this trace according to our recipe described above. We will show that this execution satisfies all the axioms mentioned in the proposition.

Firstly, we show that SOWF holds. By the definition of so, if $a \xrightarrow{\mathsf{so}} b$, then $\mathsf{ses}(a) = \mathsf{ses}(b)$. Furthermore, in this case, $a \neq b$ because of Lemma 16. Hence, so is irreflexive. Furthermore, for every $a, b \in A$, if $a \neq b$ but $\mathsf{ses}(a) = \mathsf{ses}(b)$, we have that

$$(a \xrightarrow{\mathsf{so}} b) \vee (a \xrightarrow{\mathsf{so}} b)$$

by the definition of so. It remains to show that so is transitive. Consider $a, b, c \in A$ such that $a \xrightarrow{\mathsf{so}} b$ and $b \xrightarrow{\mathsf{so}} c$. By the definition of so, there are i, j, k, l such that

$$i < j \wedge k < l \wedge \iota_i = (\_, a) \wedge \iota_j = (\_, b) \wedge \iota_k = (\_, b) \wedge \iota_l = (\_, c).$$

But $j = k$ because of Lemma 16. Hence, $a \xrightarrow{\mathsf{so}} c$ by the definition of so.

Secondly, we consider the VISWF axiom. This follows from Lemma 18 and the definition of the OPS rule.

Thirdly, we prove the ARWF axiom. Suppose that $a \xrightarrow{\mathsf{ar}} b$. By the definition of ar, $\mathsf{obj}(a) = \mathsf{obj}(b)$. The irreflexivity and the transitivity of ar follow from the same properties of $<$ on action ids. Since $<$ is total and the ids of different actions in A are different (Lemma 16), ar relates any two actions in A that work on the same object. Furthermore, for every $a \in A$, $\mathsf{vis}^{-1}(a)$ contains only those actions working on $\mathsf{obj}(a)$, because of the VISWF axiom that we just proved above. Hence, $\mathsf{ar}|_{\mathsf{vis}^{-1}(a)}$ is a total order for every $a \in A$, as required in the last part of the ARWF axiom.

Fourthly, we show the EVENTUAL axiom. If A is finite, the axiom becomes vacuous and it holds. Suppose that A is infinite. Note that this supposition implies $n = \omega$. Pick an action $a \in A$. Let $x = \mathsf{obj}(a)$. Then, there exists k such that

$$(\_, a) = \iota_k.$$

Since the trace is fair, there exists k_0 such that

$$\forall i \geq k_0.\, \forall (\_, \_, \rho, \_, \_) \in \mathcal{D}_i.\, \mathsf{id}(a) \in \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

To show the axiom holds for a, we should prove that there are at most finitely many $b \in A$ satisfying the condition below:

$$\mathsf{obj}(b) = x \wedge \neg(a \xrightarrow{\mathsf{vis}} b).$$

Notice that if an action $b \in A$ satisfies the condition above,

$$\exists k_1.\, k_1 \leq k_0 \wedge (\_, b) = \iota_{k_1}.$$

But the number of possible witnesses k_1 in the above formula is at most k_0. So only finitely many b's can satisfy the condition above.

Fifthly, we prove that the THINAIR axiom is satisfied. We define a total order $<_\tau$ on A by

$$a <_\tau b \iff \exists i, j.\, i < j \wedge \iota_i = (\_, a) \wedge \iota_j = (\_, b).$$

By Lemma 16, $<_\tau$ is an irreflexive and transitive total order on A. We will prove that

$$\forall a, b \in A.\, (a \xrightarrow{\mathsf{vis}} b \vee a \xrightarrow{\mathsf{so}} b) \implies a <_\tau b. \tag{6}$$

To see why the THINAIR axiom follows from this formula, note that the formula implies

$$(\mathsf{vis} \cup \mathsf{so})^+ \subseteq (<_\tau)^+$$

but the RHS of this subset relationship is included in $<_\tau$ because $<_\tau$ is transitive. Now the irreflexivity of $<_\tau$ implies that $(\mathsf{vis} \cup \mathsf{so})^+$ is irreflexive as well, as required by the THINAIR axiom. Let us go back to the proof of the formula in (6). Pick $a, b \in A$. If $a \xrightarrow{\mathsf{so}} b$, then $a <_\tau b$ by the definition of so. Now suppose that $a \xrightarrow{\mathsf{vis}} b$. This means that there exists i such that

$$a \in \mathsf{id}(\mathcal{S}_i|\mathcal{D}_i) \wedge (\_, b) = \iota_{i+1}.$$

By Lemma 12, there should be k such that

$$a \notin \mathsf{id}(\mathcal{S}_k|\mathcal{D}_k) \wedge (\_, a) = \iota_{k+1}.$$

By Lemma 16,

$$k + 1 < i + 1.$$

Hence, $a <_\tau b$, as desired. □

PROPOSITION 22. *All the fair traces generate executions satisfying* RVAL.

Proof: Pick a trace τ:

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \ldots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n$$

where $n \in \mathbb{N} \cup \{\omega\}$. Let $(A, \mathsf{vis}, \mathsf{so}, \mathsf{ar})$ be an execution constructed from this trace according to our recipe described above.

We will first show that

$$\begin{aligned}
&\forall i.\, \forall (_, _, \rho, _, _) \in \mathcal{D}_i.\, \forall x \in \mathsf{Obj}.\\
&\quad \exists \tau, \sigma, B.\, \tau = \mathsf{type}(x) \wedge \sigma = \rho(x)\\
&\qquad \wedge B = \{a \in A \mid \mathsf{id}(a) \in \mathsf{id}_\tau(\sigma)\}\\
&\qquad \wedge ((\mathsf{event}(B), \mathsf{event}(\mathsf{vis}|_B), \mathsf{event}(\mathsf{ar}|_B)), \sigma) \in \mathcal{R}_\tau.
\end{aligned} \tag{7}$$

Our proof is by induction on i. Pick ρ and x as described in the formula above, and define τ, σ, B again as described in the formula. We need to show that the last conjunct in the formula holds. When $i = 0$,

$$\sigma = \mathsf{init}_\tau \wedge B = \emptyset.$$

Hence, the claimed relationship by $\mathcal{R}_\tau$ in the formula becomes

$$((\emptyset, \emptyset, \emptyset), \mathsf{init}_\tau) \in \mathcal{R}_\tau,$$

which holds because it is precisely one of the conditions assumed on $\mathcal{R}_\tau$. Now assume that $i > 0$, and that the formula in (7) holds for all $0 \leq j < i$. We do the case analysis on the rule used at the i-th step. If the rule is PROP, no object tables of replicas change during the i-th step. The last conjunct in (7) follows from induction hypothesis. The other cases are that OPS and OPD are used. In these cases, the induction hypothesis gives the desired conclusion except when ρ is the table of a replica changed by the rule and x is the object affected by the rule. To take care of this exception, assume that ρ and x are the table and the object updated by the rule. Then, there exist e, f, W_0, σ_0, B_0 and $a_0 \in A$ such that

$$\begin{aligned}
&(\mathsf{eval}_\tau(e, f, W_0, \sigma_0) = (_, \sigma))\\
&\wedge B_0 = \{a \in A \mid \mathsf{id}(a) \in \mathsf{id}_\tau(\sigma_0)\}\\
&\wedge (\forall g.\, (e, g) \notin \mathsf{event}(B_0))\\
&\wedge ((\mathsf{event}(B_0), \mathsf{event}(\mathsf{vis}|_{B_0}), \mathsf{event}(\mathsf{ar}|_{B_0})), \sigma_0) \in \mathcal{R}_\tau\\
&\wedge \mathsf{event}(a_0) = (e, f)\\
&\wedge (B_0 \cup \{a_0\} = B)\\
&\wedge \{b \in B_0 \mid \mathsf{id}(b) \in W_0\} \subseteq \mathsf{vis}^{-1}(a_0).
\end{aligned} \tag{8}$$

The third conjunct comes from the well-formedness of configurations (OPS) or the conditions in the rule (OPD). The fourth

conjunct holds because of induction hypothesis. The fifth conjunct is true since for every action id such as e in a fair trace, there are a, k with $\iota_k = (\_, a) \wedge \mathsf{id}(a) = e$ (Lemma 12). The sixth conjunct follows from our assumption on eval_τ, the fact that σ is obtained from σ' and the following properties of B and B_0:

$$(B = \{a \in A \mid \mathsf{id}(a) \in \mathsf{id}_\tau(\sigma)\}) \\ \wedge (B_0 = \{a \in A \mid \mathsf{id}(a) \in \mathsf{id}_\tau(\sigma_0)\}).$$

The seventh conjunct holds because of Lemma 17 (OpD) or the definition of the rule (OpS). We note two consequences of the formula in (8):

$$\begin{aligned} &\mathsf{event}(\mathsf{vis}|_{B_0}) \cup \{((e', f'), (e, f)) \mid (e', f') \in \mathsf{event}(B_0) \\ &\qquad \wedge e' \in W_0\} \\ &= \mathsf{event}(\mathsf{vis}|_{B_0}) \cup \mathsf{event}(\{(b, a_0) \mid b \in B_0 \wedge \mathsf{id}(b) \in W_0\}) \\ &\subseteq \mathsf{event}(\mathsf{vis}|_{B_0 \cup \{a_0\}}) \\ &= \mathsf{event}(\mathsf{vis}|_B). \end{aligned}$$

And

$$\begin{aligned} &\mathsf{event}(\mathsf{ar}|_{B_0}) \\ &\quad \cup \{((e', f'), (e, f)) \mid (e', f') \in \mathsf{event}(B_0) \wedge e' < e\} \\ &\quad \cup \{((e, f), (e', f')) \mid (e', f') \in \mathsf{event}(B_0) \wedge e < e'\} \\ &= \mathsf{event}(\mathsf{ar}|_{B_0}) \\ &\qquad \cup \mathsf{event}(\{(b, a_0) \mid b \in B_0 \wedge \mathsf{id}(b) < \mathsf{id}(a_0)\}) \\ &\qquad \cup \mathsf{event}(\{(a_0, b) \mid b \in B_0 \wedge \mathsf{id}(a_0) < \mathsf{id}(b)\}) \\ &= \mathsf{event}(\mathsf{ar}|_{B_0 \cup \{a_0\}}) = \mathsf{event}(\mathsf{ar}|_B). \end{aligned}$$

By the second condition on $\mathcal{R}_\tau$, what we have shown so far imply that

$$((\mathsf{event}(B), \mathsf{event}(\mathsf{vis}|_B), \mathsf{event}(\mathsf{ar}|_B)), \sigma_0) \in \mathcal{R}_\tau,$$

as desired.

Next we use the formula in (7) and prove that this execution satisfies the RVAL axiom. Pick $a \in A$. By the definition of A, there exists a unique k such that

$$\iota_k = (\_, a).$$

Let ρ and x be the table and the object that the OpS rule has used for creating the action a in the k-th step. Define τ, σ, B as follows:

$$\tau = \mathsf{type}(x) \wedge \sigma = \rho(x) \wedge B = \{b \in A \mid \mathsf{id}(b) \in \mathsf{id}_\tau(\sigma)\}.$$

Then, by (7),

$$((\mathsf{event}(B), \mathsf{event}(\mathsf{vis}|_B), \mathsf{event}(\mathsf{ar}|_B)), \sigma) \in \mathcal{R}_\tau. \quad (9)$$

Meanwhile, $B = \mathsf{vis}^{-1}(a)$ by the definition of vis, so that

$$\mathsf{ctxt}(a) = (\mathsf{op}(a), \mathsf{event}(B), \mathsf{event}(\mathsf{vis}|_B), \mathsf{event}(\mathsf{ar}|_B)).$$

Furthermore, $\mathsf{id}(a) \notin \{\mathsf{id}(b) \mid b \in B\}$ by the well-formedness of the configuration at the $(k-1)$-th step and the definition of the OpS rule. Hence, the relationship in (9) implies

$$(\mathcal{F}_\tau(\mathsf{ctxt}(a)), \_) = \mathsf{eval}_\tau(\mathsf{id}(a), \mathsf{op}(a), \mathsf{id}_\tau(\sigma), \sigma).$$

Since $\mathsf{rval}(a)$ is defined to be the value computed by the RHS function above in the definition of OpS, this in turn gives $\mathsf{rval}(a) = \mathcal{F}_\tau(\mathsf{ctxt}(a))$, as desired. □

B.5 Correspondence: Axioms for Session Guarantees

Next, we consider axioms about session guarantees. To do so, we make a few assumptions:

1. We assume that SCtx has the following form:

   $$\mathsf{SCtx} = (\mathsf{Obj} \to \mathcal{P}_{\mathsf{fin}}(\mathsf{AId})) \times (\mathsf{Obj} \to \mathcal{P}_{\mathsf{fin}}(\mathsf{AId})) \times \mathsf{Code}$$

   A session context is a tuple of two tables and code, (R, U, K), where R records all the actions read for each object, U does the same for actions performed by the session and K is the piece of the remaining code to be executed by the session.

2. We require the following condition on update:

   $$\mathsf{update}_\tau(x, e, f, \sigma, \rho, (R, U, K)) \sqsupseteq \\ (R[x \mapsto R(x) \cup \mathsf{id}_\tau(\sigma)], U[x \mapsto U(x) \cup \{e\}], K).$$

   where $(R', U', K') \sqsupseteq (R'', U'', K'')$ means that

   $$\forall x \in \mathsf{Obj}.\, R'(x) \supseteq R''(x) \wedge U'(x) \supseteq U''(x).$$

PROPOSITION 23. *The following conditions on the* enable *predicate imply corresponding session-guarantee axioms as described:*

1. For every fair trace τ, the RYW *axiom holds for the execution generated from τ if*

$$\forall x, U, \rho.\, \mathsf{enable}(x, (\_, (\_, U, \_)), (\_, \_, \rho, \_, \_)) \\ \implies U(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

2. For every fair trace τ, the MR *axiom holds for the execution generated from τ if*

$$\forall x, R, \rho.\, \mathsf{enable}(x, (\_, (R, \_, \_)), (\_, \_, \rho, \_, \_)) \\ \implies R(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

3. For every fair trace τ, the WFRA *axiom holds for the execution generated from τ if*

$$\forall x, R, \rho.\, \mathsf{enable}(x, (\_, (R, \_, \_)), (\_, \_, \rho, \_, \_)) \\ \implies R(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

4. For every fair trace τ, the MWA *axiom holds for the execution generated from τ if*

$$\forall x, U, \rho.\, \mathsf{enable}(x, (\_, (\_, U, \_)), (\_, \_, \rho, \_, \_)) \\ \implies U(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

Proof: Consider a fair trace

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \dots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n.$$

Let $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$ be the execution constructed from this trace τ according to our recipe. We go through each item of the list in the proposition, and show that if the condition in the item holds, this execution satisfies the axiom mentioned in the item.

Let us start with the first item. Suppose that

$$\forall x, U, \rho.\, \mathsf{enable}(x, (\_, (\_, U, \_)), (\_, \_, \rho, \_, \_)) \\ \implies U(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

Pick $a, b \in A$ such that

$$a \xrightarrow{\mathsf{soo}} b.$$

By the definition of soo,

$$(a \xrightarrow{\mathsf{so}} b) \wedge (\mathsf{obj}(a) = \mathsf{obj}(b)).$$

We unpack the definition of so in the first conjunct:

$$\exists k, l.\, k < l \wedge (\_, a) = \iota_k \wedge (\_, b) = \iota_l \wedge \mathsf{ses}(a) = \mathsf{ses}(b).$$

Then, there are

$$\theta \in \mathcal{S}_{l-1} \;\text{ and }\; \delta \in \mathcal{D}_{l-1}$$

such that the OpS rule is applied for θ and δ at the l step in the trace τ. We note one simple consequence:

$$(\mathsf{ses}(a), \_) = (\mathsf{ses}(b), \_) = \theta.$$

Let U and ρ be the write table and the object table of θ and δ, respectively:

$$(\_, (\_, U, \_)) = \theta \wedge (\_, \_, \rho, \_, \_) = \delta.$$

The rules in our abstract implementation only increase sets stored in the U tables of all sessions. Furthermore, the U table of a session contains the ids of all the operations performed in the session. These two facts imply that

$$\mathsf{id}(a) \in U(\mathsf{obj}(a)) = U(\mathsf{obj}(b)).$$

Since the OpS rule is applied at the l step, the following check by the enable predicate should hold:

$$\mathsf{enable}(\mathsf{obj}(b), \theta, \delta).$$

Hence, by our supposition on the predicate,

$$\mathsf{id}(a) \in U(\mathsf{obj}(b)) \subseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(b))}(\rho(\mathsf{obj}(b))).$$

Recall that the definition of the vis relation says that

$$\forall a' \in A.\ (\mathsf{id}(a') \in \mathsf{id}_{\mathsf{type}(\mathsf{obj}(b))}(\rho(\mathsf{obj}(b)))) \implies (a' \xrightarrow{\mathsf{vis}} b).$$

Since $\mathsf{id}(a) \in \mathsf{id}_{\mathsf{type}(\mathsf{obj}(b))}(\rho(\mathsf{obj}(b)))$, we have $(a \xrightarrow{\mathsf{vis}} b)$, as desired.

Next, we prove the item about the MR axiom. Suppose that

$$\forall x, R, \rho.\, \mathsf{enable}(x, (\_, (R, \_, \_)), (\_, \_, \rho, \_, \_)) \implies R(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

Consider actions $a, b, c \in A$ such that

$$a \xrightarrow{\mathsf{vis}} b \xrightarrow{\mathsf{soo}} c.$$

By the definition of the soo relation, there are k, l such that

$$k < l \wedge (\_, b) = \iota_k \wedge (\_, c) = \iota_l \wedge \mathsf{ses}(b) = \mathsf{ses}(c) \wedge \mathsf{obj}(b) = \mathsf{obj}(c).$$

Let θ_k and δ_k be the session in $\mathcal{S}_{k-1}$ and the replica in $\mathcal{D}_{k-1}$ used in the k-th step of our trace. Similarly, let θ_l and δ_l be the session in $\mathcal{S}_{l-1}$ and the replica in $\mathcal{D}_{l-1}$ engaged in the l-th step of our trace. Let R_k, R_l and ρ_k, ρ_l be tables such that

$$(\_, (R_k, \_, \_)) = \theta_k \wedge (\_, \_, \rho_k, \_, \_) = \delta_k \wedge (\_, (R_l, \_, \_)) = \theta_l \wedge (\_, \_, \rho_l, \_, \_) = \delta_l.$$

By our supposition on the enable predicate,

$$R_l(\mathsf{obj}(c)) \subseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho_l(\mathsf{obj}(c))).$$

Hence, to complete this case, we only need to show that

$$\mathsf{id}(a) \in R_l(\mathsf{obj}(c)),$$

because every action with its id in $\rho_l(\mathsf{obj}(c))$ becomes related to c by the vis relation. Since $a \xrightarrow{\mathsf{vis}} b$, by the definition of the OpS rule,

$$\mathsf{id}(a) \in \mathsf{id}_{\mathsf{type}(\mathsf{obj}(b))}(\rho_k(\mathsf{obj}(b))).$$

This means that a gets included in the $\mathsf{obj}(b)$ entry of the R table of the session $\mathsf{ses}(b)$ after the k-th step. But $\mathsf{ses}(b) = \mathsf{ses}(c)$ and the R table of each session only grows by the rules of our abstract implementation. Hence, $\mathsf{id}(a) \in R_l(\mathsf{obj}(b))$. Since $\mathsf{obj}(b) = \mathsf{obj}(c)$, we get the desired membership of a.

We move on to the WFRA axiom. Suppose that

$$\forall x, R, \rho.\, \mathsf{enable}(x, (\_, (R, \_, \_)), (\_, \_, \rho, \_, \_)) \implies R(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

Consider actions $a, b, c \in A$ such that

$$a \xrightarrow{\mathsf{vis}} b(\xrightarrow{\mathsf{soo}})^* c.$$

We should show that $a \xrightarrow{\mathsf{ar}} c$. That is,

$$\mathsf{id}(a) < \mathsf{id}(c) \wedge \mathsf{obj}(a) = \mathsf{obj}(c).$$

By the definition of vis and Lemma 18,

$$\mathsf{obj}(a) = \mathsf{obj}(b).$$

Also, by the definition of soo,

$$\mathsf{obj}(b) = \mathsf{obj}(c).$$

Hence, $\mathsf{obj}(a) = \mathsf{obj}(c)$. It remains to show that

$$\mathsf{id}(a) < \mathsf{id}(c).$$

Let k, l be indices of the trace τ such that

$$k \leq l \wedge (\_, b) = \iota_k \wedge (\_, c) = \iota_l.$$

Also, let

$$(\_, (R, \_, \_)) = \theta_l \in \mathcal{S}_{l-1} \;\text{ and }\; (\_, \_, \rho, \_, \_) = \delta_l \in \mathcal{D}_{l-1}$$

be the session and the replica that get transformed by the l-th step of the trace τ. Since the OpS rule is applied on these θ_l and δ_l with respect to the object $\mathsf{obj}(c)$, we should have

$$\mathsf{enable}(\mathsf{obj}(c), \theta_l, \delta_l).$$

Because of our supposition on the enable predicate, this implies that

$$R(\mathsf{obj}(c)) \subseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho_l(\mathsf{obj}(c))).$$

Note that since all configurations in τ respect time (Lemma 19) and $\mathsf{id}(c)$ is the timestamp of δ_l, the above subset relationship entails that

$$\forall a' \in R(\mathsf{obj}(c)).\, \mathsf{id}(a') < \mathsf{id}(c). \tag{10}$$

Since b and c are related by soo, they should have the same session id. This means that $R(\mathsf{obj}(c))$ includes the ids of all actions read in the k-th step, which has produced the action b. Since $a \xrightarrow{\mathsf{vis}} b$, the action a is one of those read actions, so its id should be in $R(\mathsf{obj}(c))$. Now from (10), we can infer that $\mathsf{id}(a) < \mathsf{id}(c)$, as desired.

Finally, we prove the item on the MWA axiom. Suppose that

$$\forall x, U, \rho.\, \mathsf{enable}(x, (\_, (\_, U, \_)), (\_, \_, \rho, \_, \_)) \implies U(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

Consider $a, b \in A$ such that

$$a \xrightarrow{\mathsf{soo}} b.$$

By the definition of soo, there exist indices k, l of the trace τ such that

$$k < l \wedge (\_, a) = \iota_k \wedge (\_, b) = \iota_l \wedge \mathsf{ses}(a) = \mathsf{ses}(b) \wedge \mathsf{obj}(a) = \mathsf{obj}(b).$$

Let

$$(\_, (\_, U, \_)) = \theta_l \in \mathcal{S}_{l-1} \text{ and } (\_, \_, \rho, \_, \_) = \delta_l \in \mathcal{D}_{l-1}$$

be the session and the replica that are transformed by the l-th step of the trace τ. Since the OPS rule applies to these session and replica, we should have that

$$\mathsf{enable}(\mathsf{obj}(b), \theta_l, \delta_l).$$

By the supposition on the enable predicate, this implies that

$$U(\mathsf{obj}(b)) \subseteq \rho(\mathsf{obj}(b)).$$

Furthermore, since all configurations in the trace τ respect time (Lemma 19) and the timestamp of δ_l is $\mathsf{id}(b)$, we have that

$$\forall a' \in U(\mathsf{obj}(b)).\, \mathsf{id}(a') < \mathsf{id}(b).$$

Recall that all the rules in our implementation only increase the sets stored in the U part of each session, and they store all actions performed by the session in its U component. These properties and the fact that $\mathsf{ses}(a) = \mathsf{ses}(b)$ imply that

$$a \in U(\mathsf{obj}(a)).$$

Since $\mathsf{obj}(a) = \mathsf{obj}(b)$, this in turn entails $a \in U(\mathsf{obj}(b))$. Hence,

$$\mathsf{id}(a) < \mathsf{id}(b),$$

from which follows the desired $a \xrightarrow{\mathsf{ar}} b$. □

PROPOSITION 24. *The following conditions on the* enable *predicate imply corresponding session-guarantee axioms as described:*

1. *For every fair trace τ, the* WFRV *axiom holds for the execution generated from τ if*

$$(\forall x, e, f, \sigma, \rho, \kappa.\, \mathsf{id}_{\mathsf{type}(x)}(\sigma) \subseteq \mathsf{depend}_{\mathsf{type}(x)}(e, f, \sigma, \rho, \kappa)) \wedge (\forall x, R, \rho.\, \mathsf{enable}(x, (\_, (R, \_, \_)), (\_, \_, \rho, \_, \_)) \implies R(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x))).$$

2. *For every fair trace τ, the* MWV *axiom holds for the execution generated from τ if*

$$(\forall x, e, f, \sigma, \rho, \kappa.\, \mathsf{id}_{\mathsf{type}(x)}(\sigma) \subseteq \mathsf{depend}_{\mathsf{type}(x)}(e, f, \sigma, \rho, \kappa)) \wedge (\forall x, U, \rho.\, \mathsf{enable}(x, (\_, (\_, U, \_)), (\_, \_, \rho, \_, \_)) \implies U(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x))).$$

Proof: Consider a fair trace

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \dots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n.$$

Let $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$ be an execution generated by this trace τ. We go through both items in the proposition, and show that if the condition in the item holds, this generated execution satisfies the axiom mentioned in the item.

First, we prove the case of the WFRV axiom. Suppose that

$$(\forall x, e, f, \sigma, \rho, \kappa.\, \mathsf{id}_{\tau}(\sigma) \subseteq \mathsf{depend}_{\mathsf{type}(x)}(e, f, \sigma, \rho, \kappa)) \wedge (\forall x, R, \rho.\, \mathsf{enable}(x, (\_, (R, \_, \_)), (\_, \_, \rho, \_, \_)) \implies R(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x))).$$

Consider actions $a, b, c, d \in A$ such that

$$a \xrightarrow{\mathsf{vis}} b(\xrightarrow{\mathsf{soo}})^* c \xrightarrow{\mathsf{vis}} d$$

Let i, j, k, l be indices of τ and W_i, W_j, W_k, W_l sets of actions such that

$$(W_i, a) = \iota_i \wedge (W_j, b) = \iota_j \wedge (W_k, c) = \iota_k \wedge (W_l, d) = \iota_l.$$

We should show $a \xrightarrow{\mathsf{vis}} d$, equivalently,

$$\mathsf{id}(a) \in W_l.$$

Since $a \xrightarrow{\mathsf{vis}} b$ and $c \xrightarrow{\mathsf{vis}} d$,

$$(\mathsf{id}(a) \in W_j) \wedge (\mathsf{id}(c) \in W_l).$$

Hence, it suffices to show that

$$W_j \subseteq W_k \wedge W_k \subseteq W_l. \tag{11}$$

Let

$$(\_, (R_k, \_, \_)) = \theta_k \in \mathcal{S}_{k-1} \text{ and } (\_, \_, \rho_k, \_, \_) = \delta_k \in \mathcal{D}_{k-1}$$

be the session and the replica that are updated by the OPS rule at the k-th step of τ. By the definition of the rule, we should have

$$\mathsf{enable}(\mathsf{obj}(c), \theta_k, \delta_k)$$

which by our supposition implies that

$$R_k(\mathsf{obj}(c)) \subseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho_k(\mathsf{obj}(c))). \tag{12}$$

The set $R_k(\mathsf{obj}(c))$ includes the ids of all the actions on $\mathsf{obj}(c)$ that have been read up to the $(k-1)$-th step. Since $\mathsf{ses}(b) = \mathsf{ses}(c)$ and $\mathsf{obj}(b) = \mathsf{obj}(c)$, this implies that

$$W_j \subseteq R_k(\mathsf{obj}(c)).$$

Furthermore, $\rho_k(\mathsf{obj}(c)) = W_k$, so we have

$$W_j \subseteq W_k,$$

which is the first conjunct of our proof obligation (11). It remains to show the second conjunct in (11). Let

$$(\_, \_, \rho_l, \_, \_) = \delta_l \in \mathcal{D}_{l-1}$$

be the replica that was used to create the action d in the l-th step. Then,

$$W_l = \mathsf{id}_{\mathsf{type}(\mathsf{obj}(d))}(\rho_l(\mathsf{obj}(d))).$$

Since $c \xrightarrow{\mathsf{vis}} d$, we have that $k < l$. Also, by the VISWF axiom (Proposition 21),

$$\mathsf{obj}(c) = \mathsf{obj}(d).$$

If δ_l and δ_k are about the same replica,

$$\begin{aligned} W_k &= \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho_k(\mathsf{obj}(c))) \\ &= \mathsf{id}_{\mathsf{type}(\mathsf{obj}(d))}(\rho_k(\mathsf{obj}(d))) \\ &\subseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(d))}(\rho_l(\mathsf{obj}(d))) = W_l, \end{aligned}$$

because the ids associated with object tables only grow by the rules in our abstract implementation. Suppose that δ_l and δ_k are configurations of different replicas. Recall that $\mathsf{id}(c) \in W_l$ because $c \xrightarrow{\mathsf{vis}} d$. In order for this to happen, a message

$$(x', \mathsf{id}(c), f', W', E')$$

for some x', f', W', E' must have been incorporated into the replica ρ_l by the application of the OPD rule before the l-th step. But the only message with the action id $\mathsf{id}(c)$ in the trace is the one generated by the k-th step in the trace τ. Hence,

$$\begin{aligned} E' &= \mathsf{depend}_{\mathsf{type}(\mathsf{obj}(c))}(\mathsf{id}(c), \_, \rho_k(\mathsf{obj}(c)), \_, \_) \\ &\supseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho_k(\mathsf{obj}(c))) \\ &= W_k. \end{aligned}$$

When the OPD rule was applied, it must have shown that every action id in W_k was incorporated into the $\mathsf{obj}(c)$ entry of the object table in an updated replica. Since the set of actions associated with the object table of a replica only grows and $\mathsf{obj}(c) = \mathsf{obj}(d)$, we can conclude that

$$W_k \subseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(d))}(\rho_l(\mathsf{obj}(d))) = W_l,$$

as desired.

Next, we prove the item on the WMV axiom. Suppose that

$$\begin{aligned} &(\forall x, e, f, \sigma, \rho, \kappa.\, \mathsf{id}_\tau(\sigma) \subseteq \mathsf{depend}_{\mathsf{type}(x)}(e, f, \sigma, \rho, \kappa)) \\ &\wedge (\forall x, U, \rho.\, \mathsf{enable}(x, (\_, (\_, U, \_)), (\_, \_, \rho, \_, \_)) \\ &\qquad \implies U(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x))). \end{aligned}$$

Consider $a, b, c \in A$ such that

$$a \xrightarrow{\mathsf{soo}} b \xrightarrow{\mathsf{vis}} c.$$

Then,

$$\mathsf{ses}(a) = \mathsf{ses}(b) \wedge \mathsf{obj}(a) = \mathsf{obj}(b).$$

Let i, j, k be indices of the trace τ and W_i, W_j, W_k sets of action ids such that

$$(W_i, a) = \iota_i \wedge (W_j, b) = \iota_j \wedge (W_k, c) = \iota_k.$$

We should show that

$$\mathsf{id}(a) \in W_k.$$

Let

$$(\_, (\_, U_j, \_)) = \theta_j \in \mathcal{S}_{j-1} \text{ and } (\_, \_, \rho_j, \_, \_) = \delta_j \in \mathcal{D}_{j-1}$$

be the session and the replica involved in the application of the OPS rule in the j-th step. By our supposition,

$$U_j(\mathsf{obj}(b)) \subseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(b))}(\rho_j(\mathsf{obj}(b))) = W_j.$$

Since $a \xrightarrow{\mathsf{soo}} b$ and U_j stores all the updates on the session θ_j,

$$\mathsf{id}(a) \in U_j(\mathsf{obj}(b)) \subseteq W_j.$$

Hence, we can discharge our proof obligation simply by showing that

$$W_j \subseteq W_k.$$

Let

$$(\_, (\_, U_k, \_)) = \theta_j \in \mathcal{S}_{k-1} \text{ and } (\_, \_, \rho_k, \_, \_) = \delta_k \in \mathcal{D}_{k-1}$$

be the session and the replica involved in the application of the OPS rule in the k-th step. Since $b \xrightarrow{\mathsf{vis}} c$,

$$(k < l) \wedge (\mathsf{obj}(b) = \mathsf{obj}(c)).$$

If δ_l and δ_k are configurations of the same replica, we have

$$\begin{aligned} W_l &= \mathsf{id}_{\mathsf{type}(\mathsf{obj}(b))}(\rho_l(\mathsf{obj}(b))) \\ &\subseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho_k(\mathsf{obj}(c))) = W_k, \end{aligned}$$

because the set of action ids associated with the object table of a replica only grows by the rules of our abstract implementation. Suppose that δ_l and δ_k are configurations of different replicas. In order for $\mathsf{id}(b)$ to appear in W_k, a message of the form

$$(x', \mathsf{id}(b), f', W', E')$$

for some x', f', W', E' must have been incorporated into the replica δ_k before the k-th step. But if a message stores the

action id $\mathsf{id}(b)$, it should be the one generated by the l-th step. Hence,

$$\begin{aligned} E' &= \mathsf{depend}_{\mathsf{type}(\mathsf{obj}(b))}(\mathsf{id}(b), \_, \rho_l(\mathsf{obj}(b)), \_, \kappa) \\ &\supseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(b))}(\rho_l(\mathsf{obj}(b))). \\ &= W_l. \end{aligned}$$

Recall that whenever a message gets incorporated by our OPD rule, we check that every action ids in the message's E part appear in the corresponding entry of the object table of a replica updated by the rule. Furthermore, the action ids associated with any entry in the object table of a replica only increase by the rules of our abstract implementation. Hence,

$$\begin{aligned} W_l &\subseteq E' \\ &\subseteq \mathsf{id}_{\mathsf{type}(\mathsf{obj}(b))}(\rho_k(\mathsf{obj}(b))) \\ &= \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho_k(\mathsf{obj}(c))) = W_k, \end{aligned}$$

which is the conclusion that we are looking for. □

B.6 Correspondence: Causality Axioms

PROPOSITION 25. *For every fair trace τ, the* COCV *and the* COCA *axioms hold for the execution generated from the trace τ if*

$$\begin{aligned} &(\forall x, e, f, \sigma, \rho, R, U, K. \\ &\qquad \mathsf{id}_{\mathsf{type}(x)}(\sigma) \cup \bigcup_{y \in \mathsf{Obj}} U(y) \\ &\qquad\qquad \subseteq \mathsf{depend}_{\mathsf{type}(x)}(e, f, \sigma, \rho, (R, U, K))) \wedge \\ &(\forall U, \rho.\ \mathsf{enable}(\_, (\_, (\_, U, \_)), (\_, \_, \rho, \_, \_)) \\ &\qquad \implies \forall x.\, U(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x))) \end{aligned}$$

Proof: Consider a fair trace

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \ldots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n.$$

Let $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$ be an execution generated by this trace τ. We need to show that the execution satisfies the COCV and the COCA axioms.

For each binary relation r on A, we say that a set X of action ids is ***r*-closed** if

$$\forall a, b \in A.\, (\mathsf{id}(b) \in X \wedge (a, b) \in r \implies \mathsf{id}(a) \in X).$$

The key observation behind our proof is that every configuration $\mathcal{S}_i|\mathcal{D}_i$ in the trace τ satisfies the following property:

$$\begin{aligned} &(\forall(\_, \_, \rho, \_, \_) \in \mathcal{D}_i.\ \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)) \text{ is hb-closed}) \wedge \\ &(\forall(\_, \_, \_, M, N) \in \mathcal{D}_i. \forall a, a' \in A.\, \forall(\_, \mathsf{id}(a), \_, \_, E) \in M \cup N. \\ &\qquad\qquad (a', a) \in (\mathsf{vis} \cup \mathsf{so}) \implies \mathsf{id}(a') \in E). \end{aligned}$$

This property can be proven by induction on i. When $i = 0$, the property holds because $\mathcal{S}_0|\mathcal{D}_0$ is empty. Now consider the case that $i > 0$, and suppose that the property holds for all $j < i$. We do the case analysis on the rule used in the i-th step. If the rule is PROP, no new messages are generated, and the object tables of replicas remain unchanged by the rule. Hence, the property holds by induction hypothesis. If the rule is OPD, the situation is similar to the previous case. The only difference lies in one replica that gets affected by the rule. Let

$$\delta = (\_, \_, \rho, \_, \_) \text{ and } m = (x, e, \_, \_, E)$$

be a replica's configuration and the message that get used by the OPD rule in the i-th step. Also, let σ' be the new state of a data type computed by OPD, and a an action in A that has e as its id. By induction hypothesis, the set

$$\bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y))$$

is hb-closed, and

$$\forall a' \in A.\, (a', a) \in (\mathsf{vis} \cup \mathsf{so}) \implies \mathsf{id}(a') \in E.$$

Furthermore, by the definition of OPD,

$$E \subseteq \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y)).$$

By the condition on eval and the definition of OPD,

$$\mathsf{id}_{\mathsf{type}(x)}(\sigma') = \mathsf{id}_{\mathsf{type}(x)}(\rho(x)) \cup \{e\}.$$

Recall that $\mathsf{hb} = (\mathsf{so} \cup \mathsf{vis})^+$. From what we have proved follows that the below set is hb-closed as well:

$$\bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho[x \mapsto \sigma'](y)).$$

The remaining case is OPS. Let

$$\theta = (\_, (\_, U, \_)), \text{ and } \delta = (d, t, \rho, \_, \_)$$

be the configurations of the session and the replica affected by the OPS rule. Let

$$m = (x, e, f, W', E') \text{ and } a$$

be the message and the action generated by the rule. Note that $\mathsf{id}(a) = e$. By the definition of the rule,

$$\exists y.\, \mathsf{enable}(y, \theta, \delta).$$

But because of the assumption in the proposition, this implies that

$$\forall y.\, U(y) \subseteq \mathsf{id}_{\mathsf{type}(y)}(\rho(y)). \tag{13}$$

Also, because of the assumption on depend in the proposition,

$$\mathsf{id}_{\mathsf{type}(x)}(\rho(x)) \cup \bigcup_{y \in \mathsf{Obj}} U(y) \subseteq E'.$$

This implies that

$$\forall a' \in A.\, (a', a) \in (\mathsf{vis} \cup \mathsf{so}) \implies \mathsf{id}(a') \in E'.$$

Furthermore, from what we have just shown, the condition on eval and the definition of OPS, it follows that the set of action ids associated with the updated replica is the following set:

$$Y = \{(t, d)\} \cup \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y)).$$

By the definition of vis, the ids of actions in $\mathsf{vis}^{-1}(a)$ are included in the second argument of the above union. Also, by (13), those of actions in $\mathsf{so}^{-1}(a)$ are also included in the second argument. These two facts, the definition of hb and the hb-closedness of the second argument imply that the set Y is also hb-closed, as desired.

Let us go back to the proof that the execution $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$ satisfies the COCV and the COCA axioms. We consider the COCV axiom first. Consider $a, c \in A$ such that

$$(a \xrightarrow{\mathsf{hb}} c) \wedge \mathsf{obj}(a) = \mathsf{obj}(c).$$

By the definition of hb, there exists $b \in A$ such that

$$(a(\xrightarrow{\mathsf{hb}})^* b \xrightarrow{\mathsf{so}} c) \vee (a(\xrightarrow{\mathsf{hb}})^* b \xrightarrow{\mathsf{vis}} c). \tag{14}$$

Let i, j, k, W_i, W_j, W_k be indices and sets of action ids such that

$$(W_i, a) = \iota_i \wedge (W_j, b) = \iota_j \wedge (W_k, c) = \iota_k.$$

Let

$$\theta_k = (\_, (\_, U, \_)) \text{ and } \delta_k = (d, t, \rho, \_, \_)$$

be the configurations of the session and the replica used by the k-th step of the trace τ. Assume that the first disjunct of (14) holds. Then,

$$(j < k) \wedge (\mathsf{id}(b) \in \bigcup_{x \in \mathsf{Obj}} U(x)).$$

By the assumption in the proposition,

$$\bigcup_{x \in \mathsf{Obj}} U(x) \subseteq \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

Hence,

$$\mathsf{id}(b) \in \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

By what we proved about the hb closedness in the first half of this proof, the set $\bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x))$ is hb-closed. Hence,

$$\mathsf{id}(a) \in \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

By Lemma 18,

$$\mathsf{id}(a) \in \mathsf{id}_{\mathsf{type}(\mathsf{obj}(a))}(\rho(\mathsf{obj}(a))) = W_k.$$

Hence, $a \xrightarrow{\mathsf{vis}} c$ as desired. Now assume that the second disjunct of (14) holds. Then,

$$\mathsf{id}(b) \in \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho(\mathsf{obj}(c))) = W_k.$$

By what we proved about the hb closedness in the first half of this proof, the set

$$\bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x))$$

is hb-closed. Hence,

$$\mathsf{id}(a) \in \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

But $\mathsf{obj}(a) = \mathsf{obj}(c)$, so by Lemma 18,

$$\mathsf{id}(a) \in \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho(\mathsf{obj}(c))) = W_k,$$

from which follows $a \xrightarrow{\mathsf{vis}} c$ as desired.

Next, we handle the COCA axiom. Define a total order $<_\tau$ on all actions in A as follows:

$$a <_\tau b \iff \mathsf{id}(a) < \mathsf{id}(b).$$

This is an irreflexive transitive total order because so is $<$ on action ids and different actions in A have different action ids (Lemma 16). We will show that

$$\forall a, b \in A.\, (a \xrightarrow{\mathsf{ar}} b \vee a \xrightarrow{\mathsf{vis}} b \vee a \xrightarrow{\mathsf{so}} b) \implies a <_\tau b. \tag{15}$$

Then, from the transitivity and irreflexivity of $<_\tau$ follows that

$$(\mathsf{ar} \cup \mathsf{vis} \cup \mathsf{so})^+ = (\mathsf{ar} \cup \mathsf{hb})^+$$

is irreflexive, as desired. Pick $a, b \in A$ such that

$$a \xrightarrow{\mathsf{ar}} b \vee a \xrightarrow{\mathsf{vis}} b \vee a \xrightarrow{\mathsf{so}} b.$$

Let i, j, W_i, W_j be indices and sets of action ids such that

$$(W_i, a) = \iota_i \wedge (W_j, b) = \iota_j.$$

If $a \xrightarrow{\mathsf{ar}} b$, then $a <_\tau b$ by the definition of ar. If $a \xrightarrow{\mathsf{vis}} b$,

$$\mathsf{id}(a) \in W_j.$$

Since the configuration $\mathcal{S}_{i-1}|\mathcal{D}_{i-1}$ of the trace right before the i-th step respects time (Lemma 19),

$$\mathsf{id}(a) < \mathsf{id}(b).$$

Hence, $a <_\tau b$. The remaining case is that $a \xrightarrow{\mathsf{so}} b$. Let

$$\theta_j = (\_, (\_, U, \_)) \text{ and } \delta_j = (d, t, \rho, \_, \_)$$

be the configurations of the session and the replica used in the j-th step of the trace τ. Then, since $\mathsf{ses}(a) = \mathsf{ses}(b)$,

$$\mathsf{id}(a) \in \bigcup_{x \in \mathsf{Obj}} U(x).$$

By the assumption of the proposition,

$$\bigcup_{x \in \mathsf{Obj}} U(x) \subseteq \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x))$$

Since the configuration $\mathcal{S}_{i-1}|\mathcal{D}_{i-1}$ of the trace right before the i-th step respects time (Lemma 19),

$$\forall a' \in \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).\, \mathsf{id}(a') < (t, d) = \mathsf{id}(b).$$

From what we have proven so far, it follows that

$$\mathsf{id}(a) < \mathsf{id}(b),$$

which gives $a <_\tau b$ as desired. □

B.7 Correspondence: On-demand Cross Object Causality

We assume that each operation f is annotated with $\mu \in \{\mathsf{ORD}, \mathsf{CSL}\}$, as explained in the main text of the paper. Also, we assume the following two operations for each data type τ:

$$\mathsf{idc}_\tau : \mathsf{St}_\tau \to \mathcal{P}_{\mathsf{fin}}(\mathsf{AId}), \qquad \mathsf{ido}_\tau : \mathsf{St}_\tau \to \mathcal{P}_{\mathsf{fin}}(\mathsf{AId})$$

such that for every τ and all $\sigma, \sigma' \in \mathsf{St}_\tau$,

$$\begin{aligned}
&(\mathsf{idc}_\tau(\sigma) \cap \mathsf{ido}_\tau(\sigma) = \emptyset \wedge \mathsf{idc}_\tau(\sigma) \cup \mathsf{ido}_\tau(\sigma) = \mathsf{id}_\tau(\sigma)) \wedge \\
&(\forall e, f_\mu, W.\, \mathsf{eval}_\tau(e, f_\mu, W, \sigma) = (_, \sigma') \implies \\
&\quad ((\mu = \mathsf{ORD} \implies \mathsf{ido}_\tau(\sigma') = \mathsf{ido}_\tau(\sigma) \cup \{e\}) \wedge \\
&\quad (\mu = \mathsf{CSL} \implies \mathsf{ido}_\tau(\sigma') = \mathsf{ido}_\tau(\sigma) \cup \{e\})).
\end{aligned}$$

PROPOSITION 26. *For every fair trace τ, the on-demand-causality versions of the* COCV *and the* COCA *axioms hold for the execution generated from the trace τ if*

$$\begin{aligned}
&(\forall x, e, f_\mu, \sigma, \rho, R, U, K. \\
&\qquad \mathsf{idc}_{\mathsf{type}(x)}(\sigma) \cup \bigcup_{y \in \mathsf{Obj}} U(y) \\
&\qquad\qquad \subseteq \mathsf{depend}_{\mathsf{type}(x)}(e, f, \sigma, \rho, (R, U, K))) \wedge \\
&(\forall U, \rho.\, \mathsf{enable}(_, (_, (_, U, _)), (_, _, \rho, _, _)) \\
&\qquad \implies \forall x.\, U(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x))).
\end{aligned}$$

Proof: The proof is almost identical to that of Proposition 25, except a few changes due to a new definition of hb. We repeat the proof of Proposition 25 here but with these required changes. Consider a fair trace

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \ldots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n.$$

Let $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$ be an execution generated by this trace τ. We need to show that the execution satisfies the versions of the COCV and the COCA axioms for the new definition of hb.

We remind the reader that for each binary relation r on A, a set X of action ids is said to be ***r*-closed** if

$$\forall a, b \in A.\, (\mathsf{id}(b) \in X \wedge (a, b) \in r \implies \mathsf{id}(a) \in X).$$

The key observation behind our proof is that every configuration $\mathcal{S}_i|\mathcal{D}_i$ in the trace τ satisfies the following property:

$$\begin{aligned}
&(\forall (_, _, \rho, _, _) \in \mathcal{D}_i.\ \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)) \text{ is hb-closed}) \wedge \\
&(\forall (_, _, _, M, N) \in \mathcal{D}_i. \forall a, a' \in A. \forall (_, \mathsf{id}(a), f_\mu, _, E) \in M \cup N. \\
&\qquad (a', a) \in (\mathsf{so} \cup \mathsf{cvis}) \implies \mathsf{id}(a') \in E)
\end{aligned}$$

Note that hb here is $(\mathsf{so} \cup \mathsf{cvis})^+$, not $(\mathsf{so} \cup \mathsf{vis})^+$ in Proposition 25. This property can be proven by induction on i. When $i = 0$, the property holds because $\mathcal{S}_0|\mathcal{D}_0$ is empty. Now consider the case that $i > 0$, and suppose that the property holds for all $j < i$. We do the case analysis on the rule used in the i-th step. If the rule is PROP, no new messages are generated, and the object tables of replicas remain unchanged by the rule. Hence, the property holds by induction hypothesis. If the rule is OPD, the situation is similar to the previous case. The only difference lies in one replica that gets affected by the rule. Let

$$\delta = (\_, \_, \rho, \_, \_) \text{ and } m = (x, e, f_\mu, \_, E)$$

be a replica's configuration and the message that get used by the OPD rule in the i-th step. Also, let σ' be the new state of a data type computed by OPD, and a an action in A that has e as its id. By induction hypothesis, the set

$$\bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y))$$

is hb-closed, and

$$\forall a' \in A.\, (a', a) \in (\mathsf{cvis} \cup \mathsf{so}) \implies \mathsf{id}(a') \in E.$$

Furthermore, by the definition of OPD,

$$E \subseteq \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y)).$$

By the condition on eval and the definition of OPD,

$$\mathsf{id}_{\mathsf{type}(x)}(\sigma') = \mathsf{id}_{\mathsf{type}(x)}(\rho(x)) \cup \{e\}.$$

Recall that $\mathsf{hb} = (\mathsf{so} \cup \mathsf{cvis})^+$. From what we have proved follows that the below set is hb-closed as well:

$$\bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho[x \mapsto \sigma'](y)).$$

The remaining case is OPS. Let

$$\theta = (\_, (\_, U, \_)), \text{ and } \delta = (d, t, \rho, \_, \_)$$

be the configurations of the session and the replica affected by the OPS rule. Let

$$m = (x, e, f_\mu, W', E') \text{ and } a$$

be the message and the action generated by the rule. Note that $\mathsf{id}(a) = e$. By the definition of the rule,

$$\exists y.\, \mathsf{enable}(y, \theta, \delta).$$

But because of the assumption in the proposition, this implies that

$$\forall y.\, U(y) \subseteq \mathsf{id}_{\mathsf{type}(y)}(\rho(y)). \tag{16}$$

Also, because of the assumption on depend in the proposition,

$$\left(\mathsf{idc}_{\mathsf{type}(x)}(\rho(x)) \cup \bigcup_{y \in \mathsf{Obj}} U(y)\right) \subseteq E'.$$

Since $\mathsf{idc}_{\mathsf{type}(x)}(\rho(x))$ contains the ids of all actions that are cvis-related to a, the above subset relationship implies that

$$\forall a' \in A.\, (a', a) \in (\mathsf{cvis} \cup \mathsf{so}) \implies \mathsf{id}(a') \in E'.$$

Furthermore, from what we have just shown, the condition on eval and the definition of OPS, it follows that the set of action ids associated with the updated replica is the following set:

$$Y = \{(t, d)\} \cup \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y)).$$

By the definition of vis, the ids of actions in $\mathsf{vis}^{-1}(a)$ are included in the second argument of the above union. This means that those of actions in $\mathsf{cvis}^{-1}(a)$ should also be included, because $\mathsf{cvis}^{-1}(a) \subseteq \mathsf{vis}^{-1}(a)$. Also, by (16), the ids of actions in $\mathsf{so}^{-1}(a)$ are also included in the second argument. These two facts, the definition of hb and the hb-closedness of the second argument imply that the set Y is also hb-closed, as desired.

Let us go back to the proof that the execution $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$ satisfies the COCV and the COCA axioms. We consider the COCV axiom first. Consider $a, c \in A$ such that

$$(a \xrightarrow{\mathsf{hb}} c) \wedge \mathsf{obj}(a) = \mathsf{obj}(c).$$

By the definition of hb, there exists $b \in A$ such that

$$(a(\xrightarrow{\mathsf{hb}})^* b \xrightarrow{\mathsf{so}} c) \vee (a(\xrightarrow{\mathsf{hb}})^* b \xrightarrow{\mathsf{cvis}} c). \tag{17}$$

Let i, j, k, W_i, W_j, W_k be indices and sets of action ids such that

$$(W_i, a) = \iota_i \wedge (W_j, b) = \iota_j \wedge (W_k, c) = \iota_k.$$

Let

$$\theta_k = (\_, (\_, U, \_)) \text{ and } \delta_k = (d, t, \rho, \_, \_)$$

be the configurations of the session and the replica used by the k-th step of the trace τ. Assume that the first disjunct of (17) holds. Then,

$$(j < k) \wedge (\mathsf{id}(b) \in \bigcup_{x \in \mathsf{Obj}} U(x)).$$

By the assumption in the proposition,

$$\bigcup_{x \in \mathsf{Obj}} U(x) \subseteq \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

Hence,

$$\mathsf{id}(b) \in \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

By what we proved about the hb closedness in the first half of this proof, the set $\bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x))$ is hb-closed. Hence,

$$\mathsf{id}(a) \in \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

By Lemma 18,

$$\mathsf{id}(a) \in \mathsf{id}_{\mathsf{type}(\mathsf{obj}(a))}(\rho(\mathsf{obj}(a))) = W_k.$$

Hence, $a \xrightarrow{\mathsf{vis}} c$ as desired. Now assume that the second disjunct of (17) holds. Then,

$$\mathsf{id}(b) \in \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho(\mathsf{obj}(c))) = W_k.$$

By what we proved about the hb closedness in the first half of this proof, the set

$$\bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x))$$

is hb-closed. Hence,

$$\mathsf{id}(a) \in \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

But $\mathsf{obj}(a) = \mathsf{obj}(c)$, so by Lemma 18,

$$\mathsf{id}(a) \in \mathsf{id}_{\mathsf{type}(\mathsf{obj}(c))}(\rho(\mathsf{obj}(c))) = W_k,$$

from which follows $a \xrightarrow{\mathsf{vis}} c$ as desired.

Next, we handle the COCA axiom. Define a total order $<_\tau$ on all actions in A as follows:

$$a <_\tau b \iff \mathsf{id}(a) < \mathsf{id}(b).$$

This is an irreflexive transitive total order because so is $<$ on action ids and different actions in A have different action ids (Lemma 16). We will show that

$$\forall a, b \in A.\, (a \xrightarrow{\mathsf{ar}} b \vee a \xrightarrow{\mathsf{cvis}} b \vee a \xrightarrow{\mathsf{so}} b) \implies a <_\tau b. \tag{18}$$

Then, from the transitivity and irreflexivity of $<_\tau$ follows that

$$(\mathsf{ar} \cup \mathsf{cvis} \cup \mathsf{so})^+ = (\mathsf{ar} \cup \mathsf{hb})^+$$

is irreflexive, as desired. Pick $a, b \in A$ such that

$$a \xrightarrow{\mathsf{ar}} b \vee a \xrightarrow{\mathsf{cvis}} b \vee a \xrightarrow{\mathsf{so}} b.$$

Let i, j, W_i, W_j be indices and sets of action ids such that

$$(W_i, a) = \iota_i \wedge (W_j, b) = \iota_j.$$

If $a \xrightarrow{\mathsf{ar}} b$, then $a <_\tau b$ by the definition of ar. If $a \xrightarrow{\mathsf{cvis}} b$,

$$\mathsf{id}(a) \in W_j.$$

Since the configuration $\mathcal{S}_{i-1}|\mathcal{D}_{i-1}$ of the trace right before the i-th step respects time (Lemma 19),

$$\mathsf{id}(a) < \mathsf{id}(b).$$

Hence, $a <_\tau b$. The remaining case is that $a \xrightarrow{\mathsf{so}} b$. Let

$$\theta_j = (\_, (\_, U, \_)) \text{ and } \delta_j = (d, t, \rho, \_, \_)$$

be the configurations of the session and the replica used in the j-th step of the trace τ. Then, since $\mathsf{ses}(a) = \mathsf{ses}(b)$,

$$\mathsf{id}(a) \in \bigcup_{x \in \mathsf{Obj}} U(x).$$

By the assumption of the proposition,

$$\bigcup_{x \in \mathsf{Obj}} U(x) \subseteq \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x))$$

Since the configuration $\mathcal{S}_{i-1}|\mathcal{D}_{i-1}$ of the trace right before the i-th step respects time (Lemma 19),

$$\forall e' \in \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).\, e' < (t, d) = \mathsf{id}(b).$$

From what we have proven so far, it follows that

$$\mathsf{id}(a) < \mathsf{id}(b),$$

which gives $a <_\tau b$ as desired. □

B.8 Correspondence: Fence

To accommodate fences in our abstract implementation, we need to allow sessions to issue fence instructions. This amounts to four changes. The first is that the instr function can return not just operations on objects, but also the fence instruction:

$$\mathsf{instr} : \mathsf{Session} \to (\mathsf{Obj} \times \mathsf{Op}) \cup \{\mathsf{fence}\}.$$

The second change is the addition of a new function on sessions:

$$\mathsf{nextCode} : \mathsf{Code} \to \mathsf{Code}.$$

Given a code part K of a session, this function updates the part so that it is ready to execute the next command of K. The third change is the introduction of a new rule for fence:

$$\frac{\begin{array}{l}\delta = (d, t, \rho, M, N) \qquad \delta_i = (d_i, t_i, \rho_i, M_i, N_i) \\ \theta = (s, (R, U, K)) \qquad \mathsf{instr}(\theta) = \mathsf{fence} \\ \forall y \in \mathsf{Obj}.\, U(y) \subseteq \mathsf{id}_{\mathsf{type}(y)}\rho(y) \\ \forall i.\, \forall y \in \mathsf{Obj}.\, \mathsf{id}_{\mathsf{type}(y)}\rho(y) \subseteq \mathsf{id}_{\mathsf{type}(y)}\rho_i(y) \\ a = ((t, d), s, \mathsf{fence}) \qquad \theta' = (s, (R, U, \mathsf{nextCode}(K))) \\ \delta' = (d, \mathsf{next}(t), \rho, M, N) \\ \delta'_i = (d_i, \mathsf{max}(t_i, \mathsf{next}(t)), \rho_i, M_i, N_i)\end{array}}{\mathcal{S}, \theta \mid \delta, \delta_1, \ldots, \delta_n \xrightarrow{a} \mathcal{S}, \theta' \mid \delta', \delta'_1, \ldots, \delta'_n} \textsc{Opf}$$

The fourth change lies in the construction of an execution from a trace. Consider a fair trace:

$$\tau = (\mathcal{S}_0, \mathcal{D}_0, \{(\iota_k, \mathcal{S}_k, \mathcal{D}_k)\}_k).$$

We generate an execution $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar}, \mathsf{sc})$ from this trace τ as follows:

$$\begin{array}{rl}
A = & \{a \mid \exists k.\, (_, a) = \iota_k \vee a = \iota_k\}, \\
\mathsf{vis} = & \{(a, b) \in A \times A \mid \exists k, W.\, (W, b) = \iota_k \wedge \mathsf{id}(a) \in W\}, \\
\mathsf{so} = & \{(a, b) \in A \times A \mid \exists k, l.\, k < l \wedge \mathsf{ses}(a) = \mathsf{ses}(b) \\
& \qquad \wedge ((_, a) = \iota_k \vee a = \iota_k) \\
& \qquad \wedge ((_, b) = \iota_l \vee b = \iota_l)\}, \\
\mathsf{ar} = & \{(a, b) \in A \times A \mid \exists k, l.\, (_, a) = \iota_k \wedge (_, b) = \iota_l \\
& \qquad \wedge \mathsf{id}(a) < \mathsf{id}(b) \wedge \mathsf{obj}(a) = \mathsf{obj}(b)\}, \\
\mathsf{sc} = & \{(a, b) \in A \times A \mid \exists k, l.\, a = \iota_k \wedge b = \iota_l \wedge k < l\}.
\end{array}$$

The soundness of most axioms can be proved by arguments similar to what we presented so far. Hence, instead of going through all the axioms, we focus on the four most important ones: SCWF, THINAIR, COCV and COCA.

PROPOSITION 27. *Every fair trace generates executions satisfying* SCWF *and* THINAIR.

Proof: Pick a fair trace τ:

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \ldots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n$$

where $n \in \mathbb{N} \cup \{\omega\}$. Let $(A, \mathsf{vis}, \mathsf{so}, \mathsf{ar}, \mathsf{sc})$ be an execution constructed from this trace. We will show that this execution satisfies SCWF and THINAIR.

Firstly, we prove that the SCWF axiom holds for the execution. For all k, l, a, b, if

$$a = \iota_k \wedge b = \iota_l \wedge k \neq l$$

then

$$\mathsf{op}(a) = \mathsf{op}(b) = \mathsf{fence} \wedge a \neq b,$$

because of the definition of the rules in our abstract implementation. This immediately implies that sc is irreflexive. It also implies the transitivity of sc. To see this, consider a, b, c such that

$$a \xrightarrow{\mathsf{sc}} b \wedge b \xrightarrow{\mathsf{sc}} c.$$

By the definition of sc, there exist i, j, k, l such that

$$i < j \wedge k < l \wedge a = \iota_i \wedge b = \iota_j \wedge b = \iota_k \wedge c = \iota_l.$$

By what we have shown above, $j = k$. Hence, $a \xrightarrow{\mathsf{sc}} c$, as desired. Now it remains to show that sc is total. Consider $a, b \in A$ such that $\mathsf{op}(a) = \mathsf{op}(b) = \mathsf{fence}$ and $a \neq b$. By the definition of the rules in our abstract implementation, there must exist i, j such that

$$a = \iota_i \wedge b = \iota_j.$$

Since $a \neq b$, i must be different from j. Then, $i < j$ or $j < i$. Hence,

$$(a \xrightarrow{\mathsf{sc}} b) \vee (b \xrightarrow{\mathsf{sc}} a),$$

as the totality requires.

Secondly, we prove that the execution satisfies the THINAIR axiom. We note that for every action $a \in A$, there exists a unique index k such that

$$(\_, a) = \iota_k \vee a = \iota_k.$$

This is because the timestamps of replicas are only increasing by our rules, and every action in A uses the timestamp of the accessed replica, which is increased right after the generation of the action. We write $\mathsf{index}(a)$ for the unique index k, and using this notation, we define a binary relation $<_\tau$ on A:

$$a <_\tau b \iff \mathsf{index}(a) < \mathsf{index}(b).$$

By definition, $<_\tau$ is a total order on A. Hence, we can prove the required acyclicity of $\mathsf{so} \cup \mathsf{vis} \cup \mathsf{sc}$ by showing that

$$\forall a, b \in A.\, (a \xrightarrow{\mathsf{so}} b \vee a \xrightarrow{\mathsf{vis}} b \vee a \xrightarrow{\mathsf{sc}} b) \implies a <_\tau b.$$

Consider actions $a, b \in A$. Assume that $a \xrightarrow{\mathsf{so}} b$ or $a \xrightarrow{\mathsf{sc}} b$. In this case, by the definition of so and sc, we have that $a <_\tau b$, as required. The remaining case is that $a \xrightarrow{\mathsf{vis}} b$. By the definition of vis, there exists a set of actions W such that

$$((W, b) = \iota_{\mathsf{index}(b)}) \wedge (\mathsf{id}(a) \in W). \tag{19}$$

Recall that the initial configuration $\mathcal{S}_0|\mathcal{D}_0$ has to be empty by the definition of trace. One consequence of this and the definition of our rules is that

$$\begin{array}{l}\forall i.\, \forall (\_, \_, \rho, \_, \_) \in \mathcal{D}_i.\, \forall x \in \mathsf{Obj}.\, \forall a \in A. \\ \qquad \mathsf{id}(a) \in \mathsf{id}_{\mathsf{type}(x)}(\rho(x)) \implies \mathsf{index}(a) \leq i.\end{array}$$

Since W in (19) comes from ρ in $\mathcal{D}_{\mathsf{index}(b)-1}$ and it contains $\mathsf{id}(a)$, it follows from the above implication that $\mathsf{index}(a) < \mathsf{index}(b)$. This means that $a <_\tau b$, as desired. □

PROPOSITION 28. *For every fair trace τ, the fence versions of the* COCV *and the* COCA *axioms hold for the execution generated from the trace τ if*

$$\begin{aligned}&(\forall x, e, f_\mu, \sigma, \rho, R, U, K.\\ &\qquad \mathsf{idc}_{\mathsf{type}(x)}(\sigma) \cup \bigcup_{y \in \mathsf{Obj}} U(y)\\ &\qquad\qquad \subseteq \mathsf{depend}_{\mathsf{type}(x)}(e, f, \sigma, \rho, (R, U, K))) \wedge\\ &(\forall U, \rho.\ \mathsf{enable}(\_, (\_, (\_, U, \_)), (\_, \_, \rho, \_, \_))\\ &\qquad \implies \forall x. U(x) \subseteq \mathsf{id}_{\mathsf{type}(x)}(\rho(x))).\end{aligned}$$

Proof: The proof is similar to that of Proposition 26. Consider a fair trace

$$\tau = \mathcal{S}_0|\mathcal{D}_0 \xrightarrow{\iota_1} \mathcal{S}_1|\mathcal{D}_1 \xrightarrow{\iota_2} \ldots \xrightarrow{\iota_n} \mathcal{S}_n|\mathcal{D}_n.$$

Let $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar}, \mathsf{sc})$ be an execution generated by this trace τ. We need to show that the execution satisfies the fence versions of the COCV and the COCA axioms, which use the notion of hb that accommodates both the fence operator and the on-demand causal operations.

We remind the reader that for each binary relation r on A, a set X of action ids is said to be ***r*-closed** if

$$\forall a, b \in A.\, (\mathsf{id}(b) \in X \wedge (a, b) \in r \implies \mathsf{id}(a) \in X).$$

For every $a \in A$, we write $\mathsf{index}(a)$ for the unique index k such that

$$(a = \iota_k) \vee ((\_, a) = \iota_k).$$

Also, for each index i of the trace τ, we let

$$F_i = \{\mathsf{id}(a) \mid a \in A \wedge \mathsf{op}(a) = \mathsf{fence} \wedge \mathsf{index}(a) \leq i\}.$$

The key observation behind our proof is that every configuration $\mathcal{S}_i|\mathcal{D}_i$ in the trace τ satisfies the following property:

$$\begin{aligned}&(\forall (\_, \_, \rho, \_, \_) \in \mathcal{D}_i.\, F_i \cup \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)) \text{ is } \mathsf{hb}\text{-closed}) \wedge\\ &(\forall (\_, \_, \_, M, N) \in \mathcal{D}_i. \forall a, a' \in A. \forall (\_, \mathsf{id}(a), f_\mu, \_, E) \in M \cup N.\\ &\quad (a', a) \in (\mathsf{so} \cup \mathsf{cvis}) \implies \mathsf{id}(a') \in E \cup F_i).\end{aligned}$$

Note that hb here is $(\mathsf{so} \cup \mathsf{cvis} \cup \mathsf{sc})^+$. This property can be proven by induction on i. When $i = 0$, the property holds because $\mathcal{S}_0|\mathcal{D}_0$ is empty and for every $a \in A$, $\mathsf{index}(a) \geq 1$. We move on to the case that $i > 0$. Suppose that the property holds for all $j < i$. We do the case analysis on the rule used in the i-th step.

If the rule is PROP, no new messages nor fence actions are generated, and the object tables of replicas remain unchanged by the rule. Hence, the property holds by induction hypothesis and the fact that F_k only increases as k gets larger.

If the rule is OPD, the situation is similar to the previous case. The only difference lies in one replica that gets affected by the rule. Let

$$\delta = (\_, \_, \rho, \_, \_) \text{ and } m = (x, e, f_\mu, \_, E)$$

be a replica's configuration and the message that get used by the OPD rule in the i-th step. Also, let σ' be the new state of a data type computed by OPD, and a the action in A with $\mathsf{id}(a) = e$; such an action exists because all messages in a trace are generated together with corresponding actions in our abstract implementation. Since OPD does not generate a fence action,

$$F_{i-1} = F_i.$$

By induction hypothesis, the set

$$F_{i-1} \cup \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y))$$

is hb-closed, and

$$\forall a' \in A.\, (a', a) \in (\mathsf{cvis} \cup \mathsf{so}) \implies \mathsf{id}(a') \in E \cup F_{i-1}$$

Furthermore, by the definition of OPD,

$$E \subseteq \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y)).$$

By the condition on eval and the definition of OPD,

$$\mathsf{id}_{\mathsf{type}(x)}(\sigma') = \mathsf{id}_{\mathsf{type}(x)}(\rho(x)) \cup \{e\}.$$

Recall that $\mathsf{hb} = (\mathsf{so} \cup \mathsf{cvis} \cup \mathsf{sc})^+$, and that $\mathsf{op}(a) \neq \mathsf{fence}$ by the definition of the rules in our abstract implementation. From what we have proved follows that for every $b \in A$, if $(b, a) \in (\mathsf{so} \cup \mathsf{cvis} \cup \mathsf{sc})$, then $\mathsf{id}(b)$ belongs to the following set:

$$F_i \cup \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho[x \mapsto \sigma'](y)). \tag{20}$$

This fact, $F_i = F_{i-1}$ and induction hypothesis imply that the set above is hb-closed.

The next case is OPS. Let

$$\theta = (\_, (\_, U, \_)), \text{ and } \delta = (d, t, \rho, \_, \_)$$

be the configurations of the session and the replica affected by the OPS rule. Let

$$m = (x, e, f_\mu, W', E') \text{ and } a$$

be the message and the action generated by the rule. By the definition of the rule,

$$\exists y.\, \mathsf{enable}(y, \theta, \delta).$$

But because of the assumption in the proposition, this implies that

$$\forall y.\, U(y) \subseteq \mathsf{id}_{\mathsf{type}(y)}(\rho(y)).$$

Also, because of the assumption on depend in the proposition,

$$\left(\mathsf{idc}_{\mathsf{type}(x)}(\rho(x)) \cup \bigcup_{y \in \mathsf{Obj}} U(y)\right) \subseteq E'$$

Since $\mathsf{idc}_{\mathsf{type}(x)}(\rho(x))$ contains the ids of all actions that are causal and vis-related to a, the above subset relationship implies that

$$\begin{aligned}&\forall a' \in A.\\ &\quad \mathsf{level}(a') = \mathsf{CSL} \wedge (a', a) \in (\mathsf{vis} \cup \mathsf{so}) \implies \mathsf{id}(a') \in E'.\end{aligned}$$

Also, for every $a' \in A$, if

$$\exists c.\, \mathsf{op}(a') = \mathsf{fence} \wedge a' \xrightarrow{\mathsf{so}} c \xrightarrow{\mathsf{hbo}} a$$

then

$$\mathsf{index}(a') < \mathsf{index}(a).$$

The details of this consequence are given in the second part of the proof of Proposition 27. Hence, every such a' should be in F_i. From this membership and the fact on E' above follows that

$$\forall a' \in A.\ (a', a) \in (\mathsf{so} \cup \mathsf{cvis}) \implies \mathsf{id}(a') \in E' \cup F_i.$$

This means that the generated message by the rule satisfies the property that we try to show. All the other messages satisfy the property as well because of induction hypothesis and the fact that $F_i = F_{i-1}$. By a similar reason, the object tables in $\mathcal{D}_i$ other than ρ also satisfy the desired property. It remains to show that the object table ρ satisfies the property. By the condition on eval and the definition of OPS, the set of action ids associated with the updated replica is the following set:

$$\{\mathsf{id}(a)\} \cup \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y)).$$

Also,

$$F_{i-1} = F_i.$$

Now consider the set:

$$F_i \cup \{\mathsf{id}(a)\} \cup \bigcup_{y \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(y)}(\rho(y)). \tag{21}$$

By the definition of vis, the ids of actions in $\mathsf{vis}^{-1}(a)$ are included in the above union. Also, for every $a' \in A$, if

$$\exists c.\, \mathsf{op}(a') = \mathsf{fence} \wedge a' \xrightarrow{\mathsf{so}} c \xrightarrow{\mathsf{hbo}} a$$

then

$$a' \in F_{i-1} = F_i.$$

From what we just argued follows that the ids of actions in $\mathsf{cvis}^{-1}(a)$ are included in the set in (21). By the assumption of the proposition, the ids of actions in $\mathsf{so}^{-1}(a)$ are also included in the same set in (21). Furthermore, sc does not relate non-fence actions and $\mathsf{op}(a) \neq \mathsf{fence}$. Hence, the ids of the actions in $(\mathsf{so} \cup \mathsf{cvis} \cup \mathsf{sc})^{-1}(a)$ are contained in the set in (21). This membership and the induction hypothesis imply that the set in (21) is hb-closed, as desired.

The remaining case is OPF. This rule does not generate any messages, and $F_{i-1} \subseteq F_i$. So, the desired property on messages follows from the induction hypothesis. Also, the rule does not alter any object tables. Thus, it suffices to show that the fence action generated by the rule does not lead to the violation of the property on object tables. Let a be the fence action generated by the rule in the i-th step of the trace. Consider $a' \in A$ such that

$$(a', a) \in (\mathsf{so} \cup \mathsf{cvis} \cup \mathsf{sc}).$$

This implies that

$$\mathsf{index}(a') < \mathsf{index}(a). \tag{22}$$

It suffices to show that

$$\forall (\_, \_, \rho_i, \_, \_) \in \mathcal{D}_i.\, \mathsf{id}(a') \in F_i \cup \bigcup_{y \in \mathsf{Obj}} (\mathsf{id}_{\mathsf{type}(y)}(\rho_i(y))).$$

This follows from (22), the assumption of the proposition on the enable predicate, and the definitions of so, cvis and sc.

Let us go back to the proof that the execution

$$(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar}, \mathsf{sc})$$

satisfies the COCV and the COCA axioms. We consider the COCV axiom first. Consider $a, c \in A$ such that

$$(a \xrightarrow{\mathsf{hb}} c) \wedge \mathsf{obj}(a) = \mathsf{obj}(c).$$

Let i, k, W_i, W_k be indices and sets of action ids such that

$$(W_i, a) = \iota_i \wedge (W_k, c) = \iota_k.$$

Then,

$$i < k$$

because $b \xrightarrow{\mathsf{hb}} b' \implies \mathsf{index}(b) < \mathsf{index}(b')$; the details of this implication are given in the second part of the proof of Proposition 27. Let

$$\theta_k = (\_, (\_, U, \_)) \text{ and } \delta_k = (d, t, \rho, \_, \_)$$

be the configurations of the session and the replica used by the k-th step of the trace τ. By what we proved about the hb closedness in the first half of this proof, the set

$$F_k \cup \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x))$$

is hb-closed. Furthermore, a is not a fence action, but $a \xrightarrow{\mathsf{hb}} c$. Hence,

$$\mathsf{id}(a) \in \bigcup_{x \in \mathsf{Obj}} \mathsf{id}_{\mathsf{type}(x)}(\rho(x)).$$

This implies

$$\mathsf{id}(a) \in \mathsf{id}_{\mathsf{type}(\mathsf{obj}(a))}(\rho(\mathsf{obj}(a))) = W_k,$$

where we used the fact that $\mathsf{obj}(a) = \mathsf{obj}(c)$. This gives $a \xrightarrow{\mathsf{vis}} c$, as desired.

Next, we handle the COCA axiom. Define a total order $<_\tau$ on all actions in A as follows:

$$a <_\tau b \iff \mathsf{id}(a) < \mathsf{id}(b).$$

This is an irreflexive transitive total order because so is $<$ on action ids and different actions in A have different action ids. We will show that

$$\begin{aligned} \forall a, b \in A.\ (a \xrightarrow{\mathsf{ar}} b \vee a \xrightarrow{\mathsf{cvis}} b \vee a \xrightarrow{\mathsf{so}} b \vee a \xrightarrow{\mathsf{sc}} b) \\ \implies a <_\tau b. \end{aligned} \tag{23}$$

Then, from the transitivity and irreflexivity of $<_\tau$ follows that

$$(\mathsf{ar} \cup \mathsf{cvis} \cup \mathsf{so} \cup \mathsf{sc})^+ = (\mathsf{ar} \cup \mathsf{hb})^+$$

is irreflexive, as desired. Pick $a, b \in A$ such that

$$a \xrightarrow{\mathsf{ar}} b \vee a \xrightarrow{\mathsf{cvis}} b \vee a \xrightarrow{\mathsf{so}} b \vee a \xrightarrow{\mathsf{sc}} b.$$

If $a \xrightarrow{\mathsf{ar}} b$, then $a <_\tau b$ by the definition of ar. If $a \xrightarrow{\mathsf{sc}} b$, then a and b must be fence actions and we should have that

$$\mathsf{index}(a) < \mathsf{index}(b).$$

The OPF rule ensures that the timestamp of the generated fence action is smaller than the new timestamps of all replicas, and no rules in our implementation decrease the timestamps of replicas. Hence, the above relationship on the indices of a and b implies that

$$\mathsf{id}(a) < \mathsf{id}(b),$$

which gives $a <_\tau b$ as desired. By similar reasoning and the assumption of the proposition on the enable predicate, we can also conclude that if $a \xrightarrow{\mathsf{so}} b$, then $a <_\tau b$. The only remaining case is that

$$a \xrightarrow{\mathsf{cvis}} b.$$

This case gets split into two subcases:

$$(\exists f.\, a \xrightarrow{\mathsf{vis}} b \wedge \mathsf{op}(a) = f_{\mathsf{CSL}}) \vee$$
$$(\exists c.\, \mathsf{op}(a) = \mathsf{fence} \wedge a \xrightarrow{\mathsf{so}} c \xrightarrow{\mathsf{hbo}} b).$$

If the second disjunct above holds, we can again use the similar reasoning on the timestamps of replicas and the OPF rule. Since $\mathsf{index}(a) < \mathsf{index}(b)$ in this subcase, this reasoning shows that $\mathsf{id}(a) < \mathsf{id}(b)$. Hence, $a <_\tau b$. Now let us assume that the first disjunct holds. Then, neither a nor b is a fence action. Let i, j, W_i, W_j be indices and sets of action ids such that

$$(W_i, a) = \iota_i \wedge (W_j, b) = \iota_j.$$

Since $a \xrightarrow{\mathsf{vis}} b$, the set W_j should contain $\mathsf{id}(a)$. The rules of our implementation ensure that if an action is generated from a replica in the trace, it gets an id that is larger than the ids of any actions applied to the replica so far. Hence, $\mathsf{id}(a) < \mathsf{id}(b)$, which gives the desired $a <_\tau b$. □

C. Comparison with the C/C++ Memory Model

Our formalisation allows us to compare different forms of eventual consistency with memory consistency models implemented by common hardware and programming languages. To this end, we specialise the axioms in Figure 1 to the case when all objects are of the type intreg (§2), which corresponds to the setting of the latter models (Figure 6). The notion of an execution can be simplified in this case: we define a ***shared-memory execution*** as a tuple $(A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$, where (A, so) is a history, and vis and ar satisfy VISWF and ARWF from Figure 6. Now write actions do not have incoming vis edges, and the ar relation totally orders writes to the same object. Read actions have a single incoming vis edge from the write action it fetches its value from. According to the RVAL axiom in Figure 6, a read returns the value written by the vis-related write, or the default value 0 in its absence. The EVENTUAL axiom is adjusted to account for the changed type of vis: it prohibits infinitely many reads to read from writes preceding any given one in ar.

The other axioms in Figure 6 are formulated in the 'negative' form, prohibiting certain configurations in an execution, rather than in the 'positive' form, like in Figure 1, since here this is more concise. For example, RYW says that a read r that happened after a write w_2 in a session cannot read from a write w_1 earlier in ar. This follows from the RYW axiom in Figure 1: in an execution in the sense of Definition 5 satisfying that axiom, the read r has to see the write w_2; then according to RVAL in Figure 1 and the definition of $\mathcal{F}_{\mathsf{intreg}}$, r cannot take its value from w_1. Similarly, the COCV axiom says that, if a read r causally happens after a write w_2, then it cannot read from a write w_1 earlier in ar. As before, this follows from COCV in Figure 1, which in this case guarantees that, r sees both writes. There are no axioms corresponding to the WFRV and MWV session guarantees, because for integer registers, the session guarantee axioms in Figure 6 are enough to imply per-object causality: the conjunction of the axioms RYW–MWA in Figure 6 is equivalent to the conjunction of POCV and POCA.

The resulting consistency model maps to that of C/C++ [8] as explained in §3.3. The only major difference is that C/C++ has the following weaker analogue of COCA:

$$\neg\exists w_1, w_2.\quad w_1 \underset{\mathsf{ar}}{\overset{\mathsf{hb}}{\rightleftarrows}} w_2$$

This does not allow ar to contradict hb, but allows ar edges for different objects to form cycles with it. As we explained in §3.3, the stronger version of COCA is validated by common implementations of causal consistency.

PROPOSITION 29. *The system specifications for the axioms of basic eventual consistency, per-object causal consistency and causal consistency as defined in Figure 6 and Figure 1 for* intreg *are identical.*

D. Comparison with Parallel Snapshot Isolation

Figure 7 gives the pseudocode of the abstract implementation of parallel snapshot isolation for replicated data types, similar to the one given by Sovran et al. [30]. We have removed write-write conflict detection and introduced sessions. See [30] for the explanation of the implementation. We assume that each session always accesses the same site. To validate EVENTUAL, we also assume that the **upon** statement is subject to a fairness constraint that every update is eventually delivered to every site.

THEOREM 30. *The set of histories produced by the implementation in Figure 7 is equal to the one produced by the specification in Figure 4, assuming that all objects are of type* intreg *and all actions are causal.*

PROOF SKETCH. We first show that the set of histories produced by the implementation in Figure 7 is included into that specified in Figure 4. Consider a run of the implementation in Figure 7. We now construct a corresponding execution X and show that it satisfies the axioms in 4. To construct the arbitration relation ar, we associate every action a with a timestamp

Figure 6. Eventual consistency axioms specialised to read-write registers. Variables r, r_1, r_2 and w, w_1, w_2 range over read and write actions, respectively; arg selects the argument of a write.

WELL-FORMEDNESS AXIOMS

SOWF: so is the union of transitive, irreflexive and total orders on actions by each session

VISWF: $\forall w, r.\ w \xrightarrow{\mathsf{vis}} r \implies \mathsf{obj}(w) = \mathsf{obj}(r) \wedge \mathsf{op}(w) = \mathsf{wr} \wedge \mathsf{op}(r) = \mathsf{rd} \wedge (\forall w_1, w_2.\ w_1 \xrightarrow{\mathsf{vis}} r \wedge w_2 \xrightarrow{\mathsf{vis}} r \implies w_1 = w_2)$

ARWF: ar is the union of transitive, irreflexive and total orders on writes to each object

DATA TYPE AXIOM

RVAL: $\forall r \in A.\ (\exists w.\ w \xrightarrow{\mathsf{vis}} r \wedge \mathsf{rval}(r) = \mathsf{arg}(w)) \vee ((\neg\exists w.\ w \xrightarrow{\mathsf{vis}} r) \wedge \mathsf{rval}(r) = 0)$

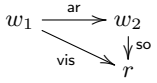
BASIC EVENTUAL CONSISTENCY AXIOMS

THINAIR: $\mathsf{so} \cup \mathsf{vis}$ is acyclic

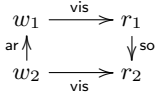
EVENTUAL: $\forall w \in A.\ \neg(\exists \text{ infinitely many } r \in A.\ \exists w'.\ w' \xrightarrow{\mathsf{vis}} r \wedge w' \xrightarrow{\mathsf{ar}} w)$

SESSION GUARANTEES (A.K.A. COHERENCE AXIOMS)

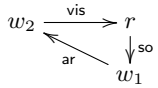
RYW: $\neg\exists r, w_1, w_2.$



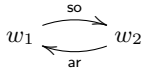
MR: $\neg\exists r_1, r_2, w_1, w_2.$



WFRA: $\neg\exists r, w_1, w_2.$

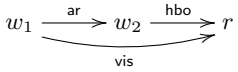


MWA: $\neg\exists w_1, w_2.$



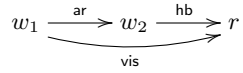
CAUSALITY AXIOMS

POCV: $\neg\exists w_1, w_2, r.$



POCA: $\neg\exists w_1, w_2.$

COCV: $\neg\exists w_1, w_2, r.$



COCA:

$\mathsf{hb} \cup \mathsf{ar}$ is acyclic

Figure 7. Abstract implementation of parallel snapshot isolation by Sovran et al. [30]

```
Sites: 1..N
Log[N]: log of operations per site
Transaction attributes: site, sessionId, startTs, commitTs[N]

operation startTx(x)
  x.startTs := new monotomic timestamp

operation write(x, obj, data)
  append (obj, data) to x.updates

operation read(x, obj)
  return state of obj from x.updates and Log[site(x)]
    up to timestamp x.startTs

operation commitTx(x, obj, data)
  x.commitTs[site(x)] := new monotonic timestamp
  append x.updates to Log[site(x)] with
    timestamp x.commitTs[site(x)]

upon [∃x,s: x.commitTs[site(x)] != ⊥ and
      x.commitTs[s] = ⊥ and ∀y such that
      y.commitTs[site(x)] < x.startTs : y.commitTs[s] != ⊥]
  x.commitTs[s] := new monotonic timestamp
  append x.updates to Log[s] with timestamp x.commitTs[s]
```

$t(a)$, which is a pair of the time when its transaction was commited on its original site and the order of the operation in the transaction. Then ar is constructed by lexicographically ordering the resulting timestamps. Then ARWF and VISWF (for any definition of vis) hold. To construct the visibility relation vis, we first construct vis' relation, relating writes to reads that took them into account when computing their result in read(). We then let $\mathsf{vis} = ((\mathsf{so} \cup \mathsf{vis}')/{\sim})^+$, so that COCV is satisfied automatically. Due to the causality clause in the **upon** statement, it is easy to check that the projection of vis to pairs of writes followed by reads is equal to vis'. Hence, RVAL is satisfied.

For all actions a and b, if $a \xrightarrow{\mathsf{so}} b$ or $a \xrightarrow{\mathsf{vis}} b$, then $t(a) < t(b)$, and thus, THINAIR holds as well. EVENTUAL is satisfied by the fairness condition associated with the **upon** statement. Since all actions in a transaction receive the same timestamp, the clause for ar in TRANSACT holds; since transactions are appended to logs atomically, so does the clause for vis. ISOLATION holds, since, in the implementation, operations performed by an uncommitted transactions are only visible to that transactions.

Assume that COCA does not hold, and thus, there is a cycle in $(\mathsf{so} \cup \mathsf{vis} \cup \mathsf{ar})/{\sim}$. The cycle cannot contain actions from a single transaction only, as this would contradict the way we assign timestamps $t(a)$ to actions a. Hence, the cycle contains actions from multiple transactions. Then every time we move from one transaction to another one on the cycle, the first component of the timestamp of the corresponding actions strictly increases, which yields a contradiction. Hence, COCA holds, which completes the proof.

We now show that the set of histories produced by specification in Figure 4 is included into that of the implementation in Figure 7. To this end, consider an execution $X = (A, \mathsf{so}, \mathsf{vis}, \mathsf{ar})$ satisfying the axioms in 4. We construct the corresponding run of the implementation as follows. Let ar' be any total order on all transactions generated by $(\mathsf{ar} \cup \mathsf{vis} \cup \mathsf{so})/{\sim}$. Such an order exists due to COCA. We assume that every session has its own dedicated site. The action set A and the session order so determine the sequences of operations performed by every session. Every transaction executes atomically and commits straight away in the order given by ar'. Then so, vis and ar edges are consistent with the timestamps assigned to transactions. Propagation in the **upon** statement is driven by the vis edges: when an action a reads an object, we propagate all the writes that it is supposed to see, as well as their causal predecessors determined by the **upon** statement, to the site the action executes on (except the writes that have already been propagated to it). Since so, vis and ar edges are consistent with the timestamps, all such actions have already been executed. Furthermore, it is easy to check that this in fact propagates only the

actions visible to a. It remains to show that the return value of read() in the implementation is consistent with the one specified by the reference execution. Suppose this were not the case. Then at the time of execution of a read a in the implementation, its site would contain a write b such that $\neg(b \xrightarrow{\mathsf{vis}} a)$. The write b could only be propagated to the site as a consequence of an hb dependency from b to a. But then COCV implies that b has to be visible to a, yielding a contradiction. Hence, the run we have constructed is indeed a run of the implementation. $\square$