Composite Replicated Data Types

Alexey Gotsman[1] and Hongseok Yang[2]

[1]IMDEA Software Institute [2]University of Oxford

Abstract. Modern large-scale distributed systems often rely on eventually consistent replicated stores, which achieve scalability in exchange for providing weak semantic guarantees. To compensate for this weakness, researchers have proposed various abstractions for programming on eventual consistency, such as replicated data types for resolving conflicting updates at different replicas and weak forms of transactions for maintaining relationships among objects. However, the subtle semantics of these abstractions makes using them correctly far from trivial.

To address this challenge, we propose composite replicated data types, which formalise a common way of organising applications on top of eventually consistent stores. Similarly to a class or an abstract data type, a composite data type encapsulates objects of replicated data types and operations used to access them, implemented using transactions. We develop a method for reasoning about programs with composite data types that reflects their modularity: the method allows abstracting away the internals of composite data type implementations when reasoning about their clients. We express the method as a denotational semantics for a programming language with composite data types. We demonstrate the effectiveness of our semantics by applying it to verify subtle data type examples and prove that it is sound and complete with respect to a standard non-compositional semantics.

1 Introduction

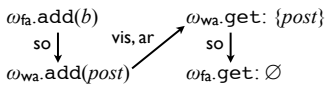
Background. To achieve availability and scalability, many modern networked systems use *replicated stores*, which maintain multiple *replicas* of shared data. Clients can access the data at any of the replicas, and these replicas communicate changes to each other using message passing. For example, large-scale Internet services use data replicas in geographically distinct locations, and applications for mobile devices keep replicas locally as well as in the cloud to support offline use. Ideally, we would like replicated stores to provide *strong consistency*, i.e., to behave as if a single centralised replica handles all operations. However, achieving this ideal usually requires synchronisation among replicas, which slows down the store and even makes it unavailable if network connections between replicas fail [13, 2]. For this reason, modern replicated stores often provide weaker guarantees, described by the umbrella term of *eventual consistency* [4].

Eventually consistent stores adopt an architecture where a replica performs an operation requested by a client locally without any synchronisation with others and immediately returns to the client; the effects of the operation are propagated to other replicas only *eventually*. As a result, different replicas may find out about an operation at different points in time. This leads to *anomalies*, one of which is illustrated by the outcome

Fig. 1. Anomalies illustrating the semantics of causal consistency and causally consistent transactions. The outcomes of operations are shown in comments. The variables v and w are local to clients. The structures shown on the right are explained in §3.2.

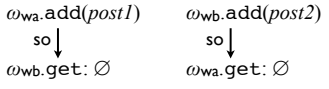
(a) Disallowed

R1: $friends[a].\texttt{add}(b)$ ‖ R2: $v = wall[a].\texttt{get}$ // $\{post\}$
$wall[a].\texttt{add}(post)$ ‖ $w = friends[a].\texttt{get}$ // $\emptyset$



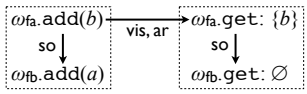
(b) Allowed, even when using transactions

R1: $wall[a].\texttt{add}(post1)$ ‖ R2: $wall[b].\texttt{add}(post2)$
$v = wall[b].\texttt{get}$ // $\emptyset$ ‖ $w = wall[a].\texttt{get}$ // $\emptyset$



(c) Disallowed

R1: atomic { ‖ R2: atomic {
$friends[a].\texttt{add}(b)$ ‖ $v = friends[a].\texttt{get}$ // $\{b\}$
$friends[b].\texttt{add}(a)$ } ‖ $w = friends[b].\texttt{get}$ // $\emptyset$ }



in Figure 1(a). The program shown there consists of two clients operating on set objects $friends[a]$ and $wall[a]$, which represent information about a user a in a social network application. The first client, connected to replica 1, makes b a friend of a's and then posts b's message on a's wall. After each of these operations, replica 1 might send a message with an update to replica 2. If the messages carrying the additions of b to $friends[a]$ and $post$ to $wall[a]$ arrive at replica 2 out of order, the second client can see b's $post$, but does not know that b has become a's friend. This outcome cannot be produced by any interleaving of the operations shown in Figure 1(a) and, hence, is not strongly consistent.

The *consistency model* of a replicated store restricts the anomalies that it exhibits. In this paper, we consider the popular model of *causal consistency* [17], a variant of eventual consistency that strikes a reasonable balance between programmability and efficiency. A causally consistent store disallows the anomaly in Figure 1(a), because it respects causal dependencies between operations: if the programmer sees b's $post$ to a's wall, she is also guaranteed to see all events that led to this posting, such as the addition of b to the set of a's friends. Causal consistency is weaker than strong consistency; in particular, it allows reading stale data. This is illustrated by the outcome in Figure 1(b), which cannot be produced by any interleaving of the operations shown. In a causally consistent store it may be produced because each message about an addition sent by the replica performing it may be slow to get to the other replica.

Due to such subtle semantics, writing correct applications on top of eventually consistent stores is very difficult. In fact, finding a good programming model for eventual consistency is considered one of the major research challenges in the systems community [4]. We build on two programming abstractions proposed by researchers to address this challenge, which we now describe.

One difficulty of programming for eventually consistent stores is that their clients can concurrently issue conflicting operations on the same data item at different replicas. For example, spouses sharing a shopping cart in an online store can add and concur-

rently remove the same item. To deal with these situations, eventually consistent stores provide *replicated data types* [22] that implement *objects*, such as registers, counters or sets, with various strategies for resolving conflicting updates to them. The strategies can be as simple as establishing a total order on all operations using timestamps and letting the last writer win, but can also be much more subtle. For example, a set data type, which can be used to implement a shopping cart, can process concurrent operations trying to add and concurrently remove the same element so that ultimately the element ends up in the set.

Another programming abstraction that eventually consistent stores are starting to provide is *transactions*, which make it easier to maintain relationships among different objects. In this paper we focus on *causally consistent transactions*, implemented (with slight variations) by a number of stores [25, 17, 18, 23, 16, 1, 3]. When a causally consistent transaction performs several updates at a replica, we are guaranteed that these will be delivered to every other replica together. For example, consider the execution in Figure 1(c), where at replica 1 two users befriend each other by adding their identifiers to set objects in the array $friends$. If we did not use transactions, the outcome shown would be allowed by causal consistency, as replica 2 might receive the addition of b to $friends[a]$, but not that of a to $friends[b]$. This would break the expected invariant that the friendship relation encoded by $friends$ is symmetric. Causally consistent transactions disallow this anomaly, but nevertheless provide weaker guarantees than the classical serialisable ACID transactions. The latter guarantee that operations done within a transaction can be viewed as taking effect instantaneously at all replicas. With causally consistent transactions, even though each separate replica sees updates done by a transaction together, different replicas may see them at different times. For example, the outcome in Figure 1(b) could occur even if we executed the pair of commands at each replica in a transaction, again because of delays in message delivery.

A typical way of using replicated data types and transactions for writing applications on top of an eventually consistent store is to keep the application data as a set of objects of replicated data types, and update them using transactions over these objects [25, 23, 16, 1]. Then replicated data types ensure sensible conflict resolution, and transactions ensure the maintenance of relationships among objects. However, due to the subtle semantics of these abstractions, reasoning about the behaviour of applications organised in this way is far from trivial. For example, it is often difficult to trace how the choice of conflict-resolution policies on separate objects affects the policy for the whole application: as we show in §5, a wrong choice can lead to violations of integrity invariants across objects, resulting in undesirable behaviour.

Contributions. To address this challenge, we propose a new programming concept of a *composite replicated data type* that formalises the above way of organising applications using eventually consistent stores. Similarly to a class or an abstract data type, a composite replicated data type encapsulates *constituent objects* of replicated data types and *composite operations* used to access them, each implemented using a transaction. For example, a composite data type representing the friendship relation in a social network may consist of a number of set objects storing the friends of each user, with transactions used to keep the relation symmetric. Composite data types can also capture

the modular structure of applications, since we can construct complex data types from simpler ones in a nested fashion.

We further propose a method for reasoning about programs with composite data types that reflects their modularity: the method allows one to abstract from the internals of composite data type implementations when reasoning about the clients of these data types. Technically, we express our reasoning method as a denotational semantics for a programming language that allows defining composite data types (§4). As any denotational semantics, ours is compositional and is thus able to give a denotation to every composite data type separately. This denotation abstracts from the internal data type structure using what we term *granularity abstraction*: it does not record *fine-grained* events describing operations on the constituent objects that are performed by composite operations, but represents every invocation of a composite operation by a single *coarse-grained* event. Thereby, the denotation allows us to pretend that the composite data type represents a single monolithic object, no different from an object of a *primitive* data type implemented natively by the store. The denotation then describes the data type behaviour using a mechanism recently proposed for specifying primitive replicated data types [10]. The granularity abstraction achieved by this *coarse-grained* denotational semantics is similar (but not identical, as we discuss in §7) to atomicity abstraction, which has been extensively investigated in the context of shared-memory concurrency [12, 24].

Our coarse-grained semantics enables modular reasoning about programs with composite replicated data types. Namely, it allows us to prove a property of a program by: *(i)* computing the denotations of the composite data types used in it; and *(ii)* proving that the program satisfies the property assuming that it uses *primitive* replicated data types with the specifications equal to the denotations of the composite ones. We thus never have to reason about composite data type implementations and their clients together.

Since we use an existing specification mechanism [10] to represent a composite data type denotation, our technical contribution lies in identifying *which* specification to pick. We show that the choice we make is correct by proving that our coarse-grained semantics is sound with respect to a *fine-grained semantics* of the programming language (§6), which records the internal execution of composite operations and follows the standard way of defining language semantics on weak consistency models [10, 6]. We also establish that the coarse-grained semantics is complete with respect to the fine-grained one: we do not lose precision by reasoning with denotations of composite data types instead of their implementations. The soundness and completeness results also imply that our coarse-grained denotational semantics is *adequate*, i.e., can be used for proving the observational equivalence of two composite data type implementations.

We demonstrate the usefulness of the coarse-grained semantics by applying the composite data type denotation it defines to specify and verify small but subtle data types, such as a social graph (§5). In particular, we show how our semantics lets one understand the consequences of different design decisions in the implementation of a composite data type on its behaviour.

Technical challenges. Coming up with a composite data type denotation that would be sound and complete with respect to the standard fine-grained semantics is far from straightforward. The reason is that the semantics of causally consistent transactions is

inherently non-compositional, as is common for weak consistency models. It is defined by so-called *consistency axioms—global* constraints on certain structures over events and relations that represent *all* events occurring in a store execution [11, 10, 9] (§3). Then proving the soundness of our coarse-grained semantics requires us to show that an execution in the fine-grained semantics can be transformed by collapsing fine-grained events inside each composite operation into a single coarse-grained event while preserving the validity of the consistency axioms. This is delicate: e.g., merging two vertices in a DAG can create a cycle. Our main contribution is to craft the denotation of a composite data type so that such problems do not arise. The subtle semantics of causally consistent transactions also makes the proofs of soundness and completeness of the resulting denotational semantics highly nontrivial.

2 Programming language and composite replicated data types

Store data model. We consider a replicated store organised as a collection of ***primitive objects***. Clients interact with the store by invoking ***operations*** on objects from a set Op, ranged over by o. Every object in the store belongs to one of the ***primitive replicated data types*** $B \in \mathsf{PrimType}$, implemented by the store natively. The ***signature*** $\mathsf{sig}(B) \subseteq \mathsf{Op}$ determines the set of operations allowed on objects of the type B. As we explain in §3, the data type also determines the semantics of the operations and, in particular, the conflict-resolution policies implemented by them. For uniformity of definitions, we assume that each operation takes a single parameter and returns a single value from a set of ***values*** Val, whose elements are ranged over by a, b, c, d. We assume that Val includes at least Booleans and integers, their sets and tuples thereof. We use a special value $\bot \in \mathsf{Val}$ to model operations that take no parameter or return no value. For example, primitive data types can include sets with operations `add`, `remove`, `contains` and `get` (the latter returning the set contents).

Composite replicated data types. We develop our results for a language of client programs interacting with the replicated store, whose syntax we show in Figure 2. We consider only programs well-typed according to the rules also shown in the figure. The interface to the store provided by the language is typical of existing implementations [25, 1]. It allows programs to declare objects of primitive replicated data types, residing in the store, invoke operations on them, and combine these into transactions. Crucially, the language also allows declaring ***composite replicated data types*** from the given primitive ones and ***composite objects*** of these types. These composite objects do not actually reside in the store, but serve as client-side anchors for compositions of primitive objects. A declaration D of a composite data type includes several ***constituent objects*** of specified types T_j, which can be primitive types, composite data type declarations or ***data-type variables*** $\alpha \in \mathsf{DVar}$, bound to either. The constituent objects are bound to distinct ***object variables*** x_j, $j = 1..m$ from a set OVar. The declaration D also defines a set of ***composite operations*** O (the type's ***signature***), with each $o \in O$ implemented by a ***command*** C_o executed as a ***transaction*** accessing the objects x_j. We emphasise the use of transactions by wrapping C_o into an atomic block. Since a store implementation executes a transaction at a replica without synchronising with other replicas, transactions never abort.

Fig. 2. Programming language and sample typing rules.

Primitive data types	B	$\in$	$\mathsf{PrimType}$	Data-type variables	α, β	$\in$	DVar
Ordinary variables	v, w	$\in$	$\mathsf{Var} = \{v_{\mathsf{in}}, v_{\mathsf{out}}, \ldots\}$	Object variables	x, y	$\in$	OVar
Data-type contexts	Γ	$::=$	$\alpha_1 : O_1, \ldots, \alpha_k : O_k$	Ordinary contexts	Σ	$::=$	$v_1, \ldots, v_k$
Object contexts	Δ	$::=$	$x_1 : O_1, \ldots, x_k : O_k$				

$D ::= \mathsf{let}\ \{x_j = \mathsf{new}\, T_j\}_{j=1..m}\ \mathsf{in}\ \{o = \mathsf{atomic}\ \{C_o\}\}_{o \in O} \qquad T ::= B \mid D \mid \alpha$

$G ::= v \mid G + G \mid G \wedge G \mid G \vee G \mid \neg G \mid \ldots$

$C ::= \mathsf{var}\ v.\, C \mid v = x.o(G) \mid v = G \mid C; C \mid \mathsf{if}\ G\ \mathsf{then}\ C\ \mathsf{else}\ C \mid \mathsf{while}\ G\ \mathsf{do}\ C \mid \mathsf{atomic}\ \{C\}$

$P ::= C_1 \parallel \ldots \parallel C_n \mid \mathsf{let}\ \alpha = T\ \mathsf{in}\ P \mid \mathsf{let}\ x = \mathsf{new}\, T\ \mathsf{in}\ P$

$\boxed{\Delta \mid \Sigma \vdash C}$ $\quad \dfrac{\mathsf{FV}(G) \cup \{v\} \subseteq (\Sigma - \{v_{\mathsf{in}}, v_{\mathsf{out}}\})}{\Delta, x : \{o\} \cup O \mid \Sigma \vdash v = x.o(G)} \quad \dfrac{\Delta \mid \Sigma, v \vdash C}{\Delta \mid \Sigma \vdash \mathsf{var}\ v.\, C} \quad \dfrac{\Delta \mid \Sigma \vdash C}{\Delta \mid \Sigma \vdash \mathsf{atomic}\ \{C\}}$

$\boxed{\Gamma \vdash T : O}$ $\quad \dfrac{}{\Gamma \vdash B : \mathsf{sig}(B)} \quad \dfrac{\begin{array}{c}\Gamma \vdash T_j : O_j \ \text{ for all } j = 1..m \\ x_1 : O_1, \ldots, x_m : O_m \mid v_{\mathsf{in}}, v_{\mathsf{out}} \vdash C_o \ \text{ for all } o \in O\end{array}}{\Gamma \vdash \mathsf{let}\ \{x_j = \mathsf{new}\, T_j\}_{j=1..m}\ \mathsf{in}\ \{o = \mathsf{atomic}\ \{C_o\}\}_{o \in O} : O}$

$\boxed{\Gamma \mid \Delta \vdash P}$ $\quad \dfrac{\Gamma \vdash T : O \qquad \Gamma \mid \Delta, x : O \vdash P}{\Gamma \mid \Delta \vdash \mathsf{let}\ x = \mathsf{new}\, T\ \mathsf{in}\ P} \quad \dfrac{\Delta \mid \emptyset \vdash C_j \ \text{ for all } j = 1..n}{\Gamma \mid \Delta \vdash C_1 \parallel \ldots \parallel C_n}$

The syntax of commands includes the form $\mathsf{var}\ v.\, C$ for declaring ***ordinary variables*** $v, w \in \mathsf{Var}$, to be used by C, which store values from Val and are initialised to $\perp$. Commands C_o in composite data type declarations D can additionally access two distinguished ordinary variables v_{in} and v_{out} (never declared explicitly), used to pass parameters and return values of operations: the parameter gets assigned to v_{in} at the beginning of the execution of C_o and the return value is read from v_{out} at the end. The command $v = x.o(G)$ executes the operation o on the object bound to the variable x with parameter G and assigns the result to v.[1]

Our type system enforces that commands only invoke operations on objects consistent with the signatures of their types and that all variables be used within the correct scope; in particular, constituent objects of composite types can only be accessed by their composite operations. For simplicity, we do not adopt a similar type discipline for values and treat all expressions as untyped. Finally, for convenience of future definitions, the typing rule for $v = x.o(G)$ requires that v_{in} and v_{out} do not appear in v or G.

Example: social graph. Figure 3 gives our running example of a composite data type `soc`, which maintains friendship relationships and requests between accounts in a toy social network application. To concentrate on core issues of composite data type correctness, we consider a language that does not allow creating unboundedly many objects; hence, we assume a fixed number of accounts N. Using syntactic sugar, the

[1] Since the object bound to x may itself be composite, this may result in atomic blocks being nested. Their semantics is the same as the one obtained by discarding all blocks except the top-level one. In particular, the atomic blocks that we include into the syntax of commands have no effect inside operations of composite data types.

Fig. 3. A social graph data type `soc`.

$$
\begin{array}{l}
D_{\mathsf{soc}} \;=\; \mathsf{let}\ \{\ \mathit{friends} = \mathsf{new}\ \mathtt{RWset}[N];\ \mathit{requesters} = \mathsf{new}\ \mathtt{RWset}[N]\ \}\ \mathsf{in}\ \{ \\
\quad \mathtt{request}(\mathit{from}, \mathit{to}) = \mathsf{atomic}\ \{ \\
\quad\quad \mathsf{if}\ (\mathit{friends}[\mathit{to}].\mathtt{contains}(\mathit{from}) \vee \mathit{requesters}[\mathit{to}].\mathtt{contains}(\mathit{from}))\ \mathsf{then}\ v_{\mathsf{out}} = \mathsf{false} \\
\quad\quad \mathsf{else}\ \{\ \mathit{requesters}[\mathit{to}].\mathtt{add}(\mathit{from}); v_{\mathsf{out}} = \mathsf{true}\ \}\ \}; \\
\quad \mathtt{accept}(\mathit{from}, \mathit{to}) = \mathsf{atomic}\ \{ \\
\quad\quad \mathsf{if}\ (\neg \mathit{requesters}[\mathit{to}].\mathtt{contains}(\mathit{from}))\ \mathsf{then}\ v_{\mathsf{out}} = \mathsf{false} \\
\quad\quad \mathsf{else}\ \{\ \mathit{requesters}[\mathit{to}].\mathtt{remove}(\mathit{from}); \mathit{requesters}[\mathit{from}].\mathtt{remove}(\mathit{to}); \\
\quad\quad\quad\quad \mathit{friends}[\mathit{to}].\mathtt{add}(\mathit{from}); \mathit{friends}[\mathit{from}].\mathtt{add}(\mathit{to}); v_{\mathsf{out}} = \mathsf{true}\ \}\ \}; \\
\quad \mathtt{reject}(\mathit{from}, \mathit{to}) = \mathsf{atomic}\ \{ \\
\quad\quad \mathsf{if}\ (\neg \mathit{requesters}[\mathit{to}].\mathtt{contains}(\mathit{from}))\ \mathsf{then}\ v_{\mathsf{out}} = \mathsf{false} \\
\quad\quad \mathsf{else}\ \{\ \mathit{requesters}[\mathit{to}].\mathtt{remove}(\mathit{from}); \mathit{requesters}[\mathit{from}].\mathtt{remove}(\mathit{to}); v_{\mathsf{out}} = \mathsf{true}\ \}\ \}; \\
\quad \mathtt{breakup}(\mathit{from}, \mathit{to}) = \mathsf{atomic}\ \{ \\
\quad\quad \mathsf{if}\ (\neg \mathit{friends}[\mathit{to}].\mathtt{contains}(\mathit{from}))\ \mathsf{then}\ v_{\mathsf{out}} = \mathsf{false} \\
\quad\quad \mathsf{else}\ \{\ \mathit{friends}[\mathit{to}].\mathtt{remove}(\mathit{from}); \mathit{friends}[\mathit{from}].\mathtt{remove}(\mathit{to}); v_{\mathsf{out}} = \mathsf{true}\ \}\ \}; \\
\quad \mathtt{get}(\mathit{id}) = \mathsf{atomic}\ \{v_{\mathsf{out}} = (\mathit{friends}[\mathit{id}].\mathtt{get}, \mathit{requesters}[\mathit{id}].\mathtt{get})\ \}\ \}
\end{array}
$$

constituent objects are grouped into arrays $\mathit{friends}$ and $\mathit{requesters}$ and have the type `RWset` of sets with a particular conflict-resolution policy (defined in §3.1). We use these sets to store account identifiers: $\mathit{friends}[a]$ gives the set of a's friends, and $\mathit{requesters}[a]$ the set of accounts with pending friendship requests to a. The implementation maintains the expected integrity invariants that the friendship relation is symmetric and the friend and requester sets of any account are disjoint:

$$\forall a, b.\ \mathit{friends}[a].\mathtt{contains}(b) \Leftrightarrow \mathit{friends}[b].\mathtt{contains}(a); \tag{1}$$

$$\forall a.\ \mathit{friends}[a].\mathtt{get} \cap \mathit{requesters}[a].\mathtt{get} = \emptyset. \tag{2}$$

The composite operations allow issuing a friendship request, accepting or rejecting it, breaking up and getting the information about a given account. For readability, we use some further syntactic sugar in the operations. Thus, we replace v_{in} with more descriptive names, recorded after the operation name and, in the case when the parameter is meant to be a tuple, introduce separate names for its components. Thus, from and to desugar to $\mathsf{fst}(v_{\mathsf{in}})$ and $\mathsf{snd}(v_{\mathsf{in}})$. We also allow invoking operations on objects inside expressions and omit unimportant parameters to operations.

The code of the composite operations is mostly as expected. For example, `request` adds the user sending the request to the requester set of the user being asked, after checking, e.g., that the former is not already a friend of the latter. However, this simplicity is deceptive: when reasoning about the behaviour of the data type, we need to consider the possibility of operations being issued concurrently at different replicas. For example, what happens if two users concurrently issue friendship requests to each other? What if two users managing the same institutional account take conflicting decisions, such as concurrently accepting and rejecting a request? As we argue in §5, it is nontrivial to implement the data type so that the behaviour in the above situations be

acceptable. Using the results in this paper, we can specify the desired social graph behaviour and prove that the composite data type in Figure 3 satisfies such a specification. Our specification abstracts from the internal structure of the data type, thereby allowing us to view it as no different from the primitive set data types it is constructed from. This facilitates reasoning about programs using the data type, which we describe next.

Programs. A ***program*** P consists of a series of data type and object variable declarations followed by a ***client***. The latter consists of several commands $C_1, \ldots, C_n$, each representing a user ***session*** accessing the store concurrently with others; a session is thus an analogue of a thread in shared-memory languages. An implementation would connect each session to one of the store replicas (as in examples in Figure 1), but this is transparent on the language level. Data type variables declared in P are used to specify the types of objects declared afterwards, and object variables are used inside sessions C_j, as per the typing rules. Sessions can thus invoke operations on a number of objects of primitive or composite types. By default, every such operation is executed within a dedicated transaction. However, like in composite data type implementations, we allow sessions to group multiple operations into transactions using atomic blocks included into the syntax of commands. We consider data types T and programs P up to the standard alpha-equivalence, adjusted so that v_{in} and v_{out} are not renamed.

Technical restriction. To simplify definitions, we assume that commands inside atomic blocks always terminate and, thus, so do all operations of composite data types. We formalise this restriction when presenting the semantics of the language in §4. It can be lifted at the expense of complicating the presentation. Note that the sessions C_j do not have to terminate, thereby allowing us to model the reactive nature of store clients.

3 Replicated store semantics

A replicated store holds objects of primitive replicated data types and implements operations on these objects. The language of §2 allows us to write programs that interact with the store by invoking the operations while grouping primitive objects into composite ones to achieve modularity. The main contribution of this paper is a denotational semantics of the language that allows the reasoning about a program to reflect this modularity. But before presenting it (in §4), we need to define the semantics of the store itself: which values can operations on primitive objects return in an execution of the store? This is determined by the consistency model of causally consistent transactions [25, 17, 18, 23, 16, 11, 3], which we informally described in §1. To formalise it, we use a variant of the framework proposed by Burckhardt et al. [10, 11, 9], which defines the store semantics declaratively, without referring to implementation-level concepts such as replicas or messages. The framework models store executions using structures on events and relations in the style of weak memory models and allows us to define the semantics of the store in two stages. We first specify the semantics of single operations on primitive objects using *replicated data type specifications* (§3.1), which are certain functions on events and relations. We then specify allowed executions of the store, including multiple operations on different objects, by constraining the events and relations using *consistency axioms* (§3.2).

A correspondence between the declarative store specification and operational models closer to implementations was established elsewhere [10, 11, 9]. Although we do not present an operational model in this paper, we often explain various features of the store specification framework by referring to the implementation-level concepts they are meant to model.

The granularity abstraction of the denotational semantics we define in §4 allows us to pretend that a composite data type is a primitive one. Hence, when defining the semantics, we reuse the replicated data type specifications introduced here to specify the behaviour of a composite data type, such as the one in Figure 3, while abstracting from the internals of its implementation.

3.1 Semantics of primitive replicated data types

In a strongly consistent system, there is a total order on all operations on an object, and each operation takes into account the effects of all operations preceding it in this order. In an eventually consistent system, the result of an operation o is determined in a more complex way:

1. The result of o depends on the set of operations information about which has been delivered to the replica performing o—those *visible* to o. For example, in Figure 1(a) the operation $friends[a].\texttt{get}$ returns $\emptyset$ because the message about $friends[a].\texttt{add}(b)$ has not yet been delivered to the replica performing the `get`.
2. The result of o may also depend on additional information used to order some events. For example, we may decide to order concurrent updates to an object using timestamps, as is the case when we use the last-writer-wins conflicts resolution policy mentioned in §1.

Hence, we specify the semantics of a replicated data type by a function F that computes the return value of an operation o given its *operation context*, which includes all we need to know about the store execution to determine the value: the set of events visible to o, together with a pair of relations on them that specify the above relationships.

Assume a countably-infinite set Event of ***events***, representing operations issued to the store. A relation is a ***strict partial order*** if it is transitive and irreflexive. A ***total order*** is a strict partial order such that for every two distinct elements e and f, the order relates e to f or f to e. We call a pair $p \in \mathsf{Op} \times \mathsf{Val} = \mathsf{AOp}$ of an operation o together with its parameter a an ***applied operation***, written as $o(a)$.

DEFINITION 1 *An **operation context** is a tuple $N = (p, E, \mathsf{aop}, \mathsf{vis}, \mathsf{ar})$, where $p \in \mathsf{AOp}$, E is a finite subset of Event, $\mathsf{aop} : E \to \mathsf{AOp}$, and vis (**visibility**) and ar (**arbitration**) are strict partial orders on E such that $\mathsf{vis} \subseteq \mathsf{ar}$.*

*We call the tuple $M = (E, \mathsf{aop}, \mathsf{vis}, \mathsf{ar})$ a **partial operation context**.*

We write Ctxt for the set of all operation contexts and denote components of N and similar structures as in $N.E$. For a relation R we write $(e, f) \in R$ and $e \xrightarrow{R} f$ interchangeably. Informally, the orders vis and ar record the relationships between events in E motivated by the above points 1 and 2, respectively. In implementation terms, the requirement $\mathsf{vis} \subseteq \mathsf{ar}$ guarantees that timestamps are consistent with message delivery: if

e is visible to f, then e has a lower timestamp than f. We define where vis and ar come from formally in §3.2; for now we just assume that they are given and define *replicated data type specifications* as certain functions of operation contexts including them.

DEFINITION 2 *A **replicated data type specification** is a partial function* $F : \mathsf{Ctxt} \rightharpoonup \mathsf{Val}$ *that returns the same value on isomorphic operation contexts and preserves it on arbitration extensions. Formally, let us order operation contexts by the pre-order* $\sqsubseteq$*:*

$$(p, E, \mathsf{aop}, \mathsf{vis}, \mathsf{ar}) \sqsubseteq (p', E', \mathsf{aop}', \mathsf{vis}', \mathsf{ar}') \iff$$
$$p = p' \wedge \exists \pi \in E \to_{\text{bijective}} E'.\, \pi(\mathsf{aop}) = \mathsf{aop}' \wedge \pi(\mathsf{vis}) = \mathsf{vis}' \wedge \pi(\mathsf{ar}) \subseteq \mathsf{ar}',$$

where we use the expected lifting of π *to relations. Then we require*

$$\forall N, N' \in \mathsf{Ctxt}.\, N \sqsubseteq N' \wedge N \in \mathsf{dom}(F) \implies N' \in \mathsf{dom}(F) \wedge F(N) = F(N'). \quad (3)$$

Let Spec be the set of data type specifications F and assume a fixed F_B for every primitive type $B \in \mathsf{PrimType}$ provided by the store. The requirement (3) states that, once arbitration gives all the information that is needed in addition to visibility to determine the outcome of an operation, arbitrating more events does not change this outcome.

Replicated sets. We illustrate the above definitions by specifying replicated set data types with different conflict-resolution policies. The semantics of a replicated set is straightforward when it is ***add-only***, i.e., its signature is $\{\mathtt{add}, \mathtt{contains}, \mathtt{get}\}$. An element a is in the set if there is an $\mathtt{add}(a)$ event in the context, or informally, if the replica performing $\mathtt{contains}(a)$ has received a message about the addition of a:

$$F_{\mathtt{AOset}}(\mathtt{contains}(a), E, \mathsf{aop}, \mathsf{vis}, \mathsf{ar}) \;=\; (\exists e \in E.\, \mathsf{aop}(e) = \mathtt{add}(a)).$$

We define the result to be $\bot$ for add operations and define the result of get as expected.[2]

Things become more subtle if we allow removing elements, since we need to define the outcome of concurrent operations adding and removing the same element, as in the context $N = (\mathtt{contains}(42), \{e, f\}, \mathsf{aop}, \mathsf{vis}, \mathsf{ar})$, where $\mathsf{aop}(e) = \mathtt{add}(42)$ and $\mathsf{aop}(f) = \mathtt{remove}(42)$. There are several possible ways of resolving this conflict [7]: in ***add-wins sets*** (AWset) adds always win against concurrent removes (so that the element ends up in the set), ***remove-wins sets*** (RWset) act vice versa, and ***last-writer-wins sets*** (LWWset) apply operations in the order of their timestamps. We specify the result of contains in these cases using the vis and ar orders in the operation context:

$$F_{\mathtt{AWset}}(\mathtt{contains}(a), E, \mathsf{aop}, \mathsf{vis}, \mathsf{ar}) =$$
$$\exists e \in E.\, \mathsf{aop}(e) = \mathtt{add}(a) \wedge (\forall f \in E.\, \mathsf{aop}(f) = \mathtt{remove}(a) \implies \neg(e \xrightarrow{\mathsf{vis}} f));$$
$$F_{\mathtt{RWset}}(\mathtt{contains}(a), E, \mathsf{aop}, \mathsf{vis}, \mathsf{ar}) =$$
$$\exists e \in E.\, \mathsf{aop}(e) = \mathtt{add}(a) \wedge (\forall f \in E.\, \mathsf{aop}(f) = \mathtt{remove}(a) \implies f \xrightarrow{\mathsf{vis}} e);$$
$$F_{\mathtt{LWWset}}(\mathtt{contains}(a), E, \mathsf{aop}, \mathsf{vis}, \mathsf{ar}) =$$
$$\exists e \in E.\, \mathsf{aop}(e) = \mathtt{add}(a) \wedge (\forall f \in E.\, \mathsf{aop}(f) = \mathtt{remove}(a) \implies f \xrightarrow{\mathsf{ar}} e),$$
$$\text{if } \mathsf{ar} \text{ is total on } \{e \in E \mid \mathsf{aop}(e) \in \{\mathtt{add}(\_), \mathtt{remove}(\_)\}\};$$
$$F_{\mathtt{LWWset}}(\mathtt{contains}(a), E, \mathsf{aop}, \mathsf{vis}, \mathsf{ar}) = \text{undefined, otherwise.}$$

[2] $F_{\mathtt{AOset}}$ is undefined on contexts with operations other than those from the signature. The type system of our language ensures that such contexts do not arise in its semantics.

Thus, the add-wins semantics is formalised by mandating that `remove` operations cancel only the `add` operations that are visible to them; the remove-wins semantics additionally mandates that they cancel concurrent `add` operations, but not those that follow them in visibility. On the above context N, the operation $\texttt{contains}(42)$ returns `true` iff: $\neg(e \xrightarrow{\mathsf{vis}} f)$ for `AWset`; $f \xrightarrow{\mathsf{vis}} e$ for `RWset`; and $f \xrightarrow{\mathsf{ar}} e$ for `LWWset`. As we show in §5, using a remove-wins set for *requesters* in Figure 3 is crucial for preserving the integrity invariant (2); *friends* could well be add-wins, which would lead to different, but also sensible, data type behaviour.

3.2 Whole-store semantics

We define the semantics of a causally consistent store by the set of its *histories*, which are certain structures on events recording all client-store interactions that can be produced during a run of the store; these include operations invoked on all objects and their return values. The store has no control over the operations occurring in histories, since these are chosen by the client; hence, the semantics only constrains return values. Replicated data type specifications define return values of operations in terms of visibility and arbitration, but where do these orders come from? As we explained in §3.1, intuitively, they are determined by the way messages are delivered and timestamps assigned in a run of a store implementation. Since this highly non-deterministic, in general, visibility and arbitration orders are arbitrary, but not entirely. A causally consistent store provides to its clients a guarantee that these orders in the contexts of different operations in the same run are related in certain ways, and this guarantee disallows anomalies such as the one in Figure 1(a).

We formalise the guarantee using the notion of an *execution*, which extends a history with visibility and arbitration orders on its events. A history is allowed by the store semantics if there is a way to extend it to an execution such that: *(i)* the return values of operations in the execution are obtained by applying replicated data type specifications to contexts extracted from it; and *(ii)* the execution satisfies certain *consistency axioms*, which constrain visibility and arbitration and, therefore, operation contexts.

Histories, executions and the satisfaction of data type specifications. We identify objects (primitive or composite) by elements of the set Obj, ranged over by ω. A strict partial order R is ***prefix-finite*** if $\{f \mid (f, e) \in R\}$ is finite for every e.

DEFINITION 3 *A **history** is a tuple $H = (E, \mathsf{label}, \mathsf{so}, \sim)$, where:*

- $E \subseteq \mathsf{Event}$.
- $\mathsf{label} : E \to \mathsf{Obj} \times \mathsf{AOp} \times \mathsf{Val}$ *describes the events in E: if $\mathsf{label}(e) = (\omega, p, a)$, then the event e describes the applied operation p on the object ω returning the value a.*
- $\mathsf{so} \subseteq E \times E$ *is a **session order**, ordering events in the same session according to the order in which they were submitted to the store. We require that* so *be prefix-finite and be the union of finitely many total orders defined on disjoint subsets of E, which correspond to events in different sessions.*
- $\sim \subseteq E \times E$ *is an equivalence relation grouping events in the same transaction. Since all transactions terminate (§2), we require that every equivalence class of $\sim$*

be a finite set. Since every transaction is performed by a single session, we require that any two distinct events by the same transaction be related by so *one way or another:*

$$\forall e, f.\, e \sim f \wedge e \neq f \implies e \xrightarrow{\mathsf{so}} f \vee f \xrightarrow{\mathsf{so}} e.$$

We also require that a transaction be contiguous in so*:*

$$\forall e, f, g.\, e \xrightarrow{\mathsf{so}} f \xrightarrow{\mathsf{so}} g \wedge e \sim g \implies e \sim f \sim g.$$

An ***execution*** *is a triple* $X = (H, \mathsf{vis}, \mathsf{ar})$ *of a history* H *and prefix-finite strict partial orders* vis *and* ar *on* $H.E$*, such that* $\mathsf{vis} \cup \mathsf{ar} \subseteq \{(e, f) \mid H.\mathsf{obj}(e) = H.\mathsf{obj}(f)\}$ *and* $\mathsf{vis} \subseteq \mathsf{ar}$.

We denote the sets of all histories and executions by Hist and Exec. We write $H.\mathsf{obj}(e)$, $H.\mathsf{aop}(e)$ and $H.\mathsf{rval}(e)$ for the components of $H.\mathsf{label}(e)$ and shorten, e.g., $X.H.\mathsf{so}$ to $X.\mathsf{so}$. Note that the set $H.E$ can be infinite, which models infinite runs. Figure 1(a) graphically represents an execution corresponding to the causality violation anomaly explained in §1. The relation $\sim$ is an identity in this case, and the objects in this and other executions in Figure 1 are add-only sets (`AOset`, §3.1).

Given an execution X, we extract the operation context of an event $e \in X.E$ by selecting all events visible to it according to $X.\mathsf{vis}$:

$$\mathsf{ctxt}(X, e) = (X.\mathsf{aop}(e), E, (X.\mathsf{aop})|_E, (X.\mathsf{vis})|_E, (X.\mathsf{ar})|_E), \tag{4}$$

where $E = (X.\mathsf{vis})^{-1}(e)$ and $\cdot|_E$ is the restriction to events in E. Then, given a function $\mathbb{F} : \mathsf{Obj} \rightharpoonup \mathsf{Spec}$ that associates data type specifications with some objects, we say that an execution X satisfies $\mathbb{F}$ if the return value of every event in X is computed on its context according to the specification that $\mathbb{F}$ gives for the accessed object.

DEFINITION 4 *An execution* X ***satisfies*** $\mathbb{F}$*, written* $X \models \mathbb{F}$*, if*

$$\forall e \in X.E.\, (X.\mathsf{obj}(e) \in \mathsf{dom}(\mathbb{F}) \implies X.\mathsf{rval}(e) = \mathbb{F}(X.\mathsf{obj}(e))(\mathsf{ctxt}(X, e))).$$

Since a context does not include return values, the above equation determines them uniquely for the events e satisfying the premise. For example, in the execution in Figure 1(a) the context of the `get` from ω_{fa} is empty. Hence, to satisfy $\mathbb{F} = (\lambda\omega.\, F_{\mathtt{AOset}})$, the `get` returns $\emptyset$. If we had a vis edge from the $\mathtt{add}(b)$ to the `get`, then the latter would have to return $\{b\}$.

Consistency axioms. We now formulate additional constraints that executions have to satisfy. They restrict the anomalies allowed by the consistency model we consider and, in particular, rule out the execution in Figure 1(a).

To define the semantics of transactions, we use the following operation. For a relation R on a set of events E and an equivalence relation $\sim$ on E (meant to group events in the same transaction), we define the ***factoring*** $R/\!\sim$ of R over $\sim$ as follows:

$$R/\!\sim \;=\; R \cup ((\sim; R; \sim) - (\sim)), \tag{5}$$

where ; composes relations. Thus, $R/\!\sim$ includes all edges from R and those obtained from such edges by relating any actions coming from the same transactions as their endpoints, excluding the case when the endpoints themselves are from the same transaction. We also let $\mathsf{sameobj}(X)(e, f) \iff X.\mathsf{obj}(e) = X.\mathsf{obj}(f)$.

DEFINITION 5 *An execution* $X = ((E, \mathsf{label}, \mathsf{so}, \sim), \mathsf{vis}, \mathsf{ar})$ *is* ***causally consistent*** *if it satisfies the following* ***consistency axioms****:*

CAUSALVIS. $((\mathsf{so} \cup \mathsf{vis})/{\sim})^+ \cap \mathsf{sameobj}(X) \subseteq \mathsf{vis}$*;*
CAUSALAR. $(\mathsf{so} \cup \mathsf{ar})/{\sim}$ *is acyclic;*
EVENTUAL. $\forall e \in E. \left|\{f \in E \mid \mathsf{sameobj}(X)(e, f) \wedge \neg(e \xrightarrow{\mathsf{vis}} f)\}\right| < \infty$.

We write $X \models_{\mathsf{CC}} \mathbb{F}$ *if* $X \models \mathbb{F}$ *and* X *is causally consistent.*

The axioms follow the informal description of the consistency model we gave in §1. We explain them below; however, their details are not crucial for understanding the rest of the paper. Before explaining the axioms, we note that Definitions 4 and 5 allow us to define the semantics of a store with object specifications given by $\mathbb{F} : \mathsf{Obj} \rightharpoonup \mathsf{Spec}$ as the set of histories that can be extended to a causally consistent execution satisfying $\mathbb{F}$:

$$\mathsf{HistCC}(\mathbb{F}) = \{H \mid \exists \mathsf{vis}, \mathsf{ar}.\ (H, \mathsf{vis}, \mathsf{ar}) \models_{\mathsf{CC}} \mathbb{F}\}. \tag{6}$$

To prove that a particular store implementation satisfies this specification, for every history H the implementation produces we have to come up with vis and ar that satisfy the constraint in (6); this is usually done by constructing them from message delivery and timestamps in the run of the implementation producing H. Here we rely on previous correctness proofs of store implementations [10, 11, 9] and use the above declarative specification of the store semantics without fixing the store implementation.

Causal consistency. The axioms CAUSALVIS and CAUSALAR in Definition 5 ensure that visibility and arbitration respect causality between operations. CAUSALVIS guarantees that an event sees all events on the same object that causally affect it, i.e., those preceding it in a chain of session order and visibility edges (ignore the use of factoring over $\sim$ for now). Thus, CAUSALVIS disallows the execution in Figure 1(a). CAUSALAR similarly requires that arbitration be consistent with session order on all objects (recall that $X.\mathsf{vis} \subseteq X.\mathsf{ar}$). EVENTUAL formalises the liveness property that every replica eventually sees every update: it ensures that an event cannot be invisible to infinitely many other events on the same object.

Transactions. The use of factoring over the $\sim$ relation in CAUSALVIS formalises the guarantee provided by causally consistent transactions that we noted in §1: updates done by a transaction get delivered to replicas together. According to CAUSALVIS, a causal dependency established between two actions of different transactions results in a dependency also being established between any other actions in the two transactions. Thus, CAUSALVIS disallows the execution in Figure 1(c), where the dashed rectangles group events into transactions. The axioms allow the execution in Figure 1(b) even when the operations by the same session are done within a transaction—an outcome that would not be allowed with serialisable transactions.

4 Coarse-grained language semantics

We now describe our main contribution—a coarse-grained denotational semantics of programs in the language of §2 that enables modular reasoning. We establish a correspondence between this semantics and the reference fine-grained semantics in §6.

Fig. 4. Key clauses of the session-local semantics of commands. Here FHist and IHist are respectively sets of histories with finite and infinite event sets; $\sigma[v \mapsto a]$ denotes the function that has the same value as σ everywhere except v, where it has the value a; and $[\,]$ is a nowhere-defined function. We assume a standard semantics of expressions $[\![G]\!] : \mathsf{LState}(\Sigma) \to \mathsf{Val}$.

$$\langle \Delta \mid \Sigma \vdash C \rangle \;:\; (\mathsf{dom}(\Delta) \to_{\text{inj}} \mathsf{Obj}) \times \mathsf{LState}(\Sigma) \to \mathcal{P}((\mathsf{FHist} \times \mathsf{LState}(\Sigma)) \cup \mathsf{IHist})$$

$$\langle v = G \rangle(obj, \sigma) = \{(H_{\text{emp}}, \sigma[v \mapsto [\![G]\!]\sigma]) \mid H_{\text{emp}} = (\emptyset, [\,], \emptyset, \emptyset)\}$$

$$\langle v = x.o(G) \rangle(obj, \sigma) = \{(H_e, \sigma[v \mapsto a]) \mid e \in \mathsf{Event} \wedge a \in \mathsf{Val} \wedge H_e = (\{e\}, [e \mapsto (obj(x), o([\![G]\!]\sigma), a)], \emptyset, \{(e, e)\})\}$$

$$\langle \mathsf{atomic}\ \{C\} \rangle(obj, \sigma) = \{((E, \mathsf{label}, \mathsf{so}, E \times E), \sigma') \mid ((E, \mathsf{label}, \mathsf{so}, \sim), \sigma') \in \langle C \rangle(obj, \sigma)\}$$

4.1 Session-local semantics of commands

The semantics of the replicated store defined by (6) in §3 describes the store behaviour under any client and thus produces histories with all possible sets of client operations. However, a particular command C in the language of §2 generates only histories with certain sequences of operations. Thus, our first step is to define a ***session-local*** semantics that, for each (sequential) command C, gives the set of histories that C can possibly generate. This semantics takes into account only the structure of the command C and operations on local variables; the return values of operations executed on objects in the store are chosen arbitrarily. Later (§4.3), we intersect the set of histories produced by the session-local semantics with (6) to take the store semantics into account.

To track the values of local variables Σ in the session-local semantics of a command $\Delta \mid \Sigma \vdash C$ (Figure 2), we use ***local states*** $\sigma \in \mathsf{LState}(\Sigma) = \Sigma \to \mathsf{Val}$. The semantics interprets commands by the function $\langle \Delta \mid \Sigma \vdash C \rangle$ in Figure 4. Its first parameter obj determines the identities of objects bound to object variables in Δ. Given an initial local state σ as the other parameter, $\langle \Delta \mid \Sigma \vdash C \rangle$ returns the set of histories produced by C when run from σ, together with final local states when applicable. The semantics is mostly standard and therefore we give only key clauses; see §A for the remaining ones. Recall that, to simplify our formalism, we require every transaction to terminate (§2). To formalise this assumption, the clause for atomic filters out infinite histories.

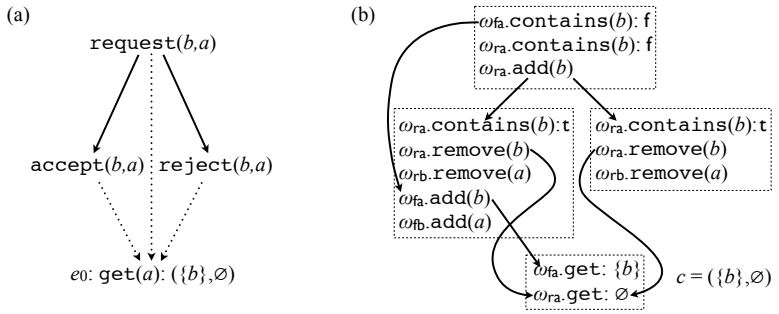
4.2 Composite data type semantics

The distinguishing feature of our coarse-grained semantics is its support for granularity abstraction: the denotation of a composite data type abstracts from its internal structure. Technically, this means that composite data types are interpreted in terms of replicated data type specifications, which we originally used for describing the meaning of primitive data types (§3.1). Thus, type variable environments Γ and data types $\Gamma \vdash T : O$ (Figure 2) are interpreted over the following domains:

$$[\![\Gamma]\!] = \mathsf{dom}(\Gamma) \to \mathsf{Spec}; \qquad [\![\Gamma \vdash T : O]\!] = [\![\Gamma]\!] \to \mathsf{Spec}.$$

We use *type* to range over elements of $[\![\Gamma]\!]$. Two cases in the definition of $[\![\Gamma \vdash T : O]\!]$ are simple. We interpret a primitive data type $B \in \mathsf{PrimType}$ as the corresponding data type specification F_B, which is provided as part of the store specification (§3.1):

Fig. 5. (a) A context N of coarse-grained events for the social graph data type `soc` in Figure 3, with an event e_0 added to represent the operation $N.p$. Solid edges denote both visibility and arbitration (equal, since the data type does not use arbitration). The dashed edges show the additional edges in vis' and ar' introduced in Definition 7. **(b)** An execution X belonging to the concretisation of N. The objects ω_{fa}, ω_{fb}, ω_{ra}, ω_{rb} correspond to the variables $friends[a]$, $friends[b]$, $requesters[a]$, $requesters[b]$ of type `RWset`. Solid edges denote both visibility and arbitration. We have omitted the session order inside transactions, the visibility and arbitration edges it induces and the transitive consequences of the edges shown. Dashed rectangles group events into transactions. The function β maps events in X to the horizontally aligned events in N.



$[\![B]\!]type = F_B$. We define the denotation of a type variable α by looking it up in the environment $type$: $[\![\alpha]\!]type = type(\alpha)$.

The remaining and most interesting case is the interpretation $[\![\Gamma \vdash D : O]\!]$ of a composite data type

$$D \;=\; \mathsf{let}\ \{x_j = \mathsf{new}\ T_j\}_{j=1..m}\ \mathsf{in}\ \{o = \mathsf{atomic}\ \{C_o\}\}_{o \in O}. \tag{7}$$

For $type \in [\![\Gamma]\!]$, the data type specification $F = [\![\Gamma \vdash D : O]\!]type$ returns a value given a context consisting of ***coarse-grained*** events that represent composite operations on an object of type D (e.g., the one in Figure 5(a)). This achieves granularity abstraction, because, once a denotation of this form is computed, it can be used to determine the return value of a composite operation without knowing the operations on the constituent objects x_j that were done by the implementations C_o of the composite operations in its context (e.g., the ones in Figure 3). We call events describing the operations on x_j ***fine-grained***.

Informally, our approach to defining the denotation F of D is to determine the value that F has to return on a context N of coarse-grained events by "running" the implementations C_o of the composite operations invoked in N. This produces an execution X over fine-grained events that describes how C_o acts on the constituent objects x_j—a *concretisation* of N. The execution X has to be causally consistent and satisfy the data type specifications for the objects x_j. We then define $F(N)$ to be the return value that the implementation of the composite operation $N.p$ gives in X. However, concretising N into X is easier said than done: while the history part of X is determined by the session-local semantics of the implementations C_o (§4.1), determining the visibility and arbitration orders so that the resulting denotation be sound (in the sense described in §6) is nontrivial and represents our main insight.

To define the denotation of (7) formally, we first gather all histories that an implementation C_o of a composite operation can produce in the session-local semantics $\langle\cdot\rangle$ into a *summary*: given an applied composite operation and a return value, a summary defines the set of histories that its implementation produces when returning the value.

DEFINITION 6 *A **summary** ρ is a partial map $\rho : \mathsf{AOp} \times \mathsf{Val} \rightharpoonup \mathcal{P}(\mathsf{FHist})$ such that for every $(p, a) \in \mathsf{dom}(\rho)$, $\rho(p, a)$ is closed under the renaming of events, and for every $H \in \rho(p, a)$, $H.\mathsf{so}$ is a total order on $H.E$ and $H.{\sim} = H.E \times H.E$.*

For a family of commands $\{\Delta \mid v_{\mathsf{in}}, v_{\mathsf{out}} \vdash C_o\}_{o \in O}$ and $obj : \mathsf{dom}(\Delta) \to_{\mathsf{inj}} \mathsf{Obj}$, we define the corresponding summary $[\![\{C_o\}_{o \in O}]\!](obj) : \mathsf{AOp} \times \mathsf{Val} \rightharpoonup \mathcal{P}(\mathsf{FHist})$ as follows: for $o' \in O$ and $a, b \in \mathsf{Val}$, we let

$$[\![\{C_o\}_{o \in O}]\!](obj)(o'(a), b) = \\ \{H \mid (H, [v_{\mathsf{in}} \mapsto \_, v_{\mathsf{out}} \mapsto b]) \in \langle \mathsf{atomic}\ \{C_{o'}\}\rangle(obj, [v_{\mathsf{in}} \mapsto a, v_{\mathsf{out}} \mapsto \bot])\}.$$

For example, the method bodies C_o in Figure 3 and an appropriate obj define the summary $\rho_{\mathtt{soc}} = [\![\{C_o\}_{o \in \{\mathtt{request},\mathtt{accept},\ldots\}}]\!](obj)$. This maps the `get` operation in Figure 5(a) to a set of histories including the one shown to the right of it in Figure 5(b).

We now define the executions X that may result from "running" the implementations of composite operations in a coarse-grained context N given by a summary ρ. The definition below pairs these executions X with the value c returned in them by the implementation of $N.p$, since this is what we are ultimately interested in. We first state the formal definition, and then explain it in detail. We write id for the identity relation.

DEFINITION 7 *A pair $(X, c) \in \mathsf{Exec} \times \mathsf{Val}$ is a **concretisation** of a context N with respect to a summary $\rho : \mathsf{AOp} \times \mathsf{Val} \rightharpoonup \mathcal{P}(\mathsf{FHist})$ if for some event $e_0 \notin N.E$ and function $\beta : X.E \to N.E \uplus \{e_0\}$ we have*

$$(\forall f \in (N.E).\ (X.H)|_{\beta^{-1}(f)} \in \rho(N.\mathsf{aop}(f), \_)) \wedge ((X.H)|_{\beta^{-1}(e_0)} \in \rho(N.p, c)); \quad (8)$$

$$\beta(X.\mathsf{so}) \subseteq \mathsf{id}; \quad (9)$$

$$\beta(X.\mathsf{vis}) - \mathsf{id} \subseteq \mathsf{vis}'; \quad (10)$$

$$\beta^{-1}(\mathsf{vis}') \cap \mathsf{sameobj}(X) \subseteq X.\mathsf{vis}; \quad (11)$$

$$\beta(X.\mathsf{ar}) - \mathsf{id} \subseteq \mathsf{ar}', \quad (12)$$

where $\mathsf{vis}' = N.\mathsf{vis} \cup \{(f, e_0) \mid f \in N.E\}$ and $\mathsf{ar}' = N.\mathsf{ar} \cup \{(f, e_0) \mid f \in N.E\}$.

We write $\gamma(N, \rho)$ for the set of all concretisations of N with respect to ρ.

For example, the pair of the execution and the value in Figure 5(b) belongs to $\gamma(N, \rho_{\mathtt{soc}})$ for N in Figure 5(a). When X concretises N with respect to ρ, the history $X.H$ is a result of expanding every composite operation in N into a history of its implementation according to ρ. The function β maps every event in $X.E$ to the event from N it came from, with an event e_0 added to $N.E$ to represent the operation $N.p$; this is formalised by (8). The condition (9) further requires that the implementation of every composite operation be executed in a dedicated session. As it happens, it is enough to consider concretisations of this form to define the denotation.

The conditions (10)–(12) represent the main insight of our definition of the denotation: they tell us how to select the visibility and arbitration orders in X given those in N. They are best understood by appealing to the intuition about how an implementation of the store operates. Recall that, from this perspective, visibility captures message delivery: an event is visible to another event if and only if the information about the former has been delivered to the replica of the latter (§3.1). Also, in implementations of causally consistent transactions, updates done by a transaction are delivered to every replica together (§1). Since composite operations execute inside transactions, the visibility order in N can thus be intuitively thought of as specifying the delivery of groups of updates made by them: we have an edge $e' \xrightarrow{\mathsf{vis}'} f'$ between coarse-grained events e' and f' in N (e.g., `request` and `accept` in Figure 5(a)) if and only if the updates performed by the transaction denoted by e' have been delivered to the replica of f'. Now consider fine-grained events $e, f \in X.E$ on the same constituent object describing updates made inside the transactions of e' and f', so that $\beta(e) = e'$ and $\beta(f) = f'$ (e.g., $\omega_{\mathsf{ra}}.\mathtt{add}(b)$ and $\omega_{\mathsf{ra}}.\mathtt{contains}(b)$ in Figure 5(b)). Then we can have $e \xrightarrow{X.\mathsf{vis}} f$ if and only if $e' \xrightarrow{\mathsf{vis}'} f'$. This is formalised by (10) and (11).

To explain (12), recall that arbitration captures the order of timestamps assigned to events by the store implementation. Also, in implementations the timestamps of all updates done by a transaction are contiguous in this order. Thus, arbitration in N can be thought of as specifying the timestamp order on the level of whole transactions corresponding to the composite operations in N. Then (12) states that the order of timestamps of fine-grained events in X is consistent with that over transactions these events come from.

To define the denotation, we need to consider only those executions concretising N that are causally consistent and satisfy data type specifications. Hence, for $\mathbb{F} : \mathsf{Obj} \rightharpoonup \mathsf{Spec}$ we let

$$\gamma(N, \rho, \mathbb{F}) = \{(X, c) \in \gamma(N, \rho) \mid X \models_{\mathsf{CC}} \mathbb{F}\}.$$

For example, the execution in Figure 5(b) belongs to $\gamma(N, \rho_{\mathtt{soc}}, \mathbb{F})$ for N in Figure 5(a) and $\mathbb{F} = (\lambda\omega.\, F_{\mathtt{RWset}})$. As the following theorem shows, the constraints (8)–(12) are so tight that the set of concretisations defined in this way never contains two different return values; this holds even if we allow choosing object identities differently.

THEOREM 8 *Given a family* $\{\Delta \mid v_{\mathsf{in}}, v_{\mathsf{out}} \vdash C_o\}_{o \in O}$*, we have:*

$$\begin{array}{l}
\forall N.\, \forall obj_1, obj_2 \in [\mathsf{dom}(\Delta) \rightarrow_{\mathsf{inj}} \mathsf{Obj}].\\
\forall \mathbb{F}_1 \in [\mathsf{range}(obj_1) \rightarrow \mathsf{Spec}].\, \forall \mathbb{F}_2 \in [\mathsf{range}(obj_2) \rightarrow \mathsf{Spec}].\\
\quad (\forall x \in \mathsf{dom}(\Delta).\, \mathbb{F}_1(obj_1(x)) = \mathbb{F}_2(obj_2(x))) \implies\\
\qquad \forall (X_1, c_1) \in \gamma(N, [\![\{C_o\}_{o \in O}]\!](obj_1), \mathbb{F}_1).\\
\qquad \forall (X_2, c_2) \in \gamma(N, [\![\{C_o\}_{o \in O}]\!](obj_2), \mathbb{F}_2).\, c_1 = c_2.
\end{array}$$

This allows us to define the denotation of (7) according to the outline we gave before.

DEFINITION 9 *For (7) we let* $[\![\Gamma \vdash D]\!]type = F$*, where* $F : \mathsf{Ctxt} \rightharpoonup \mathsf{Val}$ *is defined as follows: for* $N \in \mathsf{Ctxt}$ *and* $c \in \mathsf{Val}$*, if*

$$\begin{array}{l}
\exists obj \in [\{x_j \mid j = 1..m\} \rightarrow_{\mathsf{inj}} \mathsf{Obj}].\, \exists \mathbb{F} \in [\mathsf{range}(obj) \rightarrow \mathsf{Spec}].\\
\quad (\forall j = 1..m.\, \mathbb{F}(obj(x_j)) = [\![T_j]\!]type) \wedge (_, c) \in \gamma(N, [\![\{C_o\}_{o \in O}]\!](obj), \mathbb{F}),
\end{array}$$

Fig. 6. Semantics of $\Gamma \mid \Delta \vdash P$. Here $H \uplus H' = (H.E \uplus H'.E,\ H.\mathsf{label} \uplus H'.\mathsf{label},\ H.\mathsf{so} \cup H'.\mathsf{so},\ H.{\sim} \cup H'.{\sim})$; undefined if so is $H.E \uplus H'.E$.

$$[\![\Gamma \mid \Delta \vdash P]\!] \ :\ [\![\Gamma]\!] \to \textstyle\prod_{obj \in [\mathsf{dom}(\Delta) \to_{\mathsf{inj}} \mathsf{Obj}]} ((\mathsf{range}(obj) \rightharpoonup \mathsf{Spec}) \to \mathcal{P}(\mathsf{Hist}))$$

$$[\![\mathsf{let}\ \alpha = T\ \mathsf{in}\ P]\!](type, obj, \mathbb{F}) \ =\ [\![P]\!](type[\alpha \mapsto [\![T]\!]type], obj, \mathbb{F})$$

$$[\![\mathsf{let}\ x = \mathsf{new}\ T\ \mathsf{in}\ P]\!](type, obj, \mathbb{F}) \ =\ \bigcup\{[\![P]\!](type, obj[x \mapsto \omega], \mathbb{F}[\omega \mapsto [\![T]\!]type]) \mid \omega \notin \mathsf{range}(obj)\}$$

$$[\![C_1 \parallel \ldots \parallel C_n]\!](type, obj, \mathbb{F}) \ =\ \mathsf{HistCC}(\mathbb{F}) \cap \big\{ \biguplus_{j=1}^{n} H_j \mid \forall j = 1..n.\ (H_j, \_) \in \langle C_j \rangle(obj, [\,]) \vee H_j \in \langle C_j \rangle(obj, [\,]) \big\}$$

then $F(N) = c$; otherwise $F(N)$ is undefined.

The existence and uniqueness of F in the definition follow from Theorem 8. It is easy to check that F defined above satisfies all the properties required in Definition 2 and, hence, $F \in \mathsf{Spec}$. According to the above definition, the denotation of the data type in Figure 3 has to give $(\{b\}, \emptyset)$ on the context in Figure 5(a).

4.3 Program semantics

Having defined the denotations of composite data types, we give the semantics to a program in the language of §2 by instantiating (6) with an $\mathbb{F}$ computed from these denotations and by intersecting the result with the set of histories that can be produced by the program according to the session-local semantics of its sessions (§4.1). A program $\Gamma \mid \Delta \vdash P$ is interpreted with respect to environments $type$, obj and $\mathbb{F}$, which give the semantics of data type variables in Γ, the identities of objects in Δ and the specifications associated with these objects (Figure 6). A data type variable declaration extends the $type$ environment with the specification of the data type computed from its declaration as described in §4.2. An object variable declaration extends obj with a fresh object and $\mathbb{F}$ with the specification corresponding to its type. A client is interpreted by combining all histories its sessions produce in the session-local semantics with respect to obj and intersecting the result with (6). Note that we originally defined the store semantics (6) under the assumption that all replicated data types are primitive. Here we are able to reuse the definition because our denotations of composite data types have the same form as those of primitive ones.

Using the semantics. Our denotational semantics enables modular reasoning about programs with composite replicated data types. Namely, it allows us to check if a program P can produce a given history H by: *(i)* computing the denotations $\mathbb{F}$ of the composite data types used in P; and *(ii)* checking if the client of P can produce H assuming it uses *primitive* data types with the specifications $\mathbb{F}$. Due to the granularity abstraction in our denotation, it represents every invocation of a composite operation by a single event and thereby abstracts from its internal structure. In particular, different composite data type implementations can have the same denotation describing the data type behaviour. As a consequence, in *(ii)* we can pretend that composite data types are primitive and thus do not have to reason about the behaviour of their implementations

and the client together. For example, we can determine how a program using the social graph data type behaves in the situation shown in Figure 5(a) using the result the data type denotation gives on this context, without considering how its implementation behaves (cf. Figure 5(b)). We get the same benefits when reasoning about a complex composite data type D constructed from simpler composite data types T_j as in (7): we can first compute the denotations of T_j and then use the results in reasoning about D.

In practice, we do not compute the denotation of a composite data type D using Definition 9 directly. Instead, we typically invent a specification F that describes the desired behaviour of D, and then prove that F is equal to the denotation of D, i.e., that D is *correct* with respect to F. Definition 9 and, in particular, constraints (8)–(12), give a *proof method* for establishing this. The next section illustrates this on an example.

5 Example: social graph

We have applied the composite data type denotation in §4 to specify and prove the correctness of three composite data types: *(i)* the social graph data type in Figure 3; *(ii)* a shopping cart data type implemented using an add-wins set, which resolves conflicts between concurrent changes to the quantity of the same product; *(iii)* a data type that uses transactions to simultaneously update several objects that resolve conflicts using the last-writer-wins policy (cf. `LWWset` from §3.1). The latter example uses arbitration in a nontrivial way. Due to space constraints, we focus here on the social graph data type and defer the others to §E.

Below we give a specification F_{soc} to the social graph data type, which we have proved to be the denotation of its implementation D_{soc} in Figure 3. The proof is done by considering an arbitrary context N and its concretisation (X, c) according to Definition 7 and showing that $F_{\mathsf{soc}}(N) = c$. The constraints (8)–(12) make the required reasoning mostly mechanical and therefore we defer the easy proof to §E and only illustrate the correspondence between D_{soc} and F_{soc} on examples.

The function F_{soc} is defined recursively using the following operation that selects a subcontext of a given event in a context, analogously to the ctxt operation on executions (4) from §3.2. For a partial context M and an event $e \in M.E$, we let

$$\mathsf{ctxt}(M, e) = (M.\mathsf{aop}(e), E, (M.\mathsf{aop})|_E, (M.\mathsf{vis})|_E, (M.\mathsf{ar})|_E),$$

where $E = (M.\mathsf{vis})^{-1}(e)$. Then

$$\begin{aligned}
&F_{\mathsf{soc}}(\mathtt{get}(a), M) = \\
&(\{b \mid \exists e \in (M.E).\, (M.\mathsf{aop}(e) = \mathtt{accept}((b,a) \mid (a,b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M,e)) \wedge \\
&\qquad \forall f \in (M.E).\, (M.\mathsf{aop}(f) \in \mathtt{breakup}((b,a) \mid (a,b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M,e)) \\
&\qquad\qquad \implies f \xrightarrow{\mathsf{vis}} e\}, \\
&\{b \mid \exists e \in (M.E).\, (M.\mathsf{aop}(e) = \mathtt{request}(b,a)) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M,e)) \wedge \\
&\qquad \forall f \in (M.E).\, (M.\mathsf{aop}(f) \in (\mathtt{accept} \mid \mathtt{reject})((b,a) \mid (a,b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M,e)) \\
&\qquad\qquad \implies f \xrightarrow{\mathsf{vis}} e\}); \\
&F_{\mathsf{soc}}(\mathtt{accept}(b,a), M) = (b \in \mathsf{snd}(F_{\mathsf{soc}}(\mathtt{get}(a), M))).
\end{aligned}$$

The results of `request`, `reject` and `breakup` are defined similarly to `accept`. For brevity, we use the notation $(G_1 \mid G_2)$ above to denote the set arising from picking either G_1 or G_2 as the subexpression of the expression where it occurs. Even though the definition looks complicated, its conceptual idea is simple and has a temporal flavour. Our definition takes into account that: after breaking up, users can become friends again; and sometimes data type operations are unsuccessful, in which case they return false. According to the two components of $F_{\mathtt{soc}}(\mathtt{get}(a), M)$:

1. a's friends are the accounts b with a successful `accept` operation between a and b such that any successful `breakup` between them was in its past, as formalised by visibility. We determine whether an operation was successful by calling $F_{\mathtt{soc}}$ recursively on its subcontext.
2. a's requesters are the accounts b with a successful $\mathtt{request}(b, a)$ operation such that any successful `accept` or `reject` between a and b was in its past.

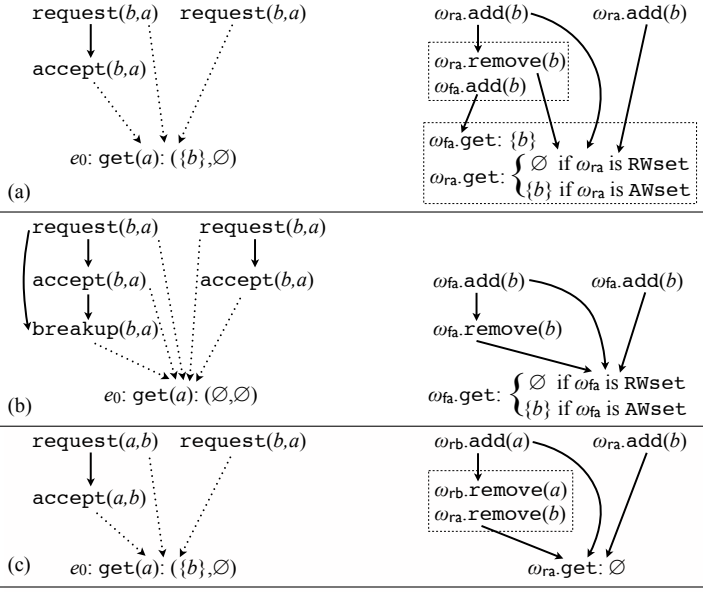
This specifies the behaviour of the data type while abstracting from its implementation, thereby enabling modular reasoning about programs using it (§4.3).

Our specification $F_{\mathtt{soc}}$ can be used to analyse the behaviour of the implementation in Figure 3. By a simple unrolling of the definition of $F_{\mathtt{soc}}$, it is easy to check that the two sets returned by $F_{\mathtt{soc}}(\mathtt{get}(a), M)$ are disjoint and, hence, the invariant (2) in §2 holds; (1) can be checked similarly. Also, since $F_{\mathtt{soc}}$ returns $(\{b\}, \emptyset)$ on the context in Figure 5(a), when the same friendship request is concurrently accepted and rejected, the accept wins. Different behaviour could also be reasonable; the decision ultimately depends on application requirements.

We now illustrate the correspondence between $D_{\mathtt{soc}}$ and $F_{\mathtt{soc}}$ on examples and, on the way, show that our coarse-grained semantics lets one understand how the choice of conflict-resolution policies on constituent objects affects the policy of the composite data type. First, we argue that making *requesters* remove-wins in Figure 3 is crucial for preserving the integrity invariant (2) and satisfying $F_{\mathtt{soc}}$. Indeed, consider the scenario shown in Figure 7(a). Here two users managing the same account b concurrently issue friendship requests to a, which initially sees only one of them. If *requesters* were add-wins, the `accept` by a would affect only the request that it sees. The remaining request would eventually propagate to all replicas in the system, and the calls to `get` in the implementation would thus return b as being both a friend and a requester of a's, violating (2). The remove-wins policy of *requesters* ensures that, when a user accepts or rejects a request, this also removes all identical requests issued concurrently.

If we made *friends* add-wins, this would make the data type behave differently, but sensibly, as illustrated in Figure 7(b). Here we again have two concurrently issued requests from b to a. The account a may also be managed by multiple users, which concurrently accept the requests they happen to see. One of the users then immediately breaks up with a. Since *friends* are remove-wins, this cancels the addition of b to $\mathit{friends}[a]$ (i.e., ω_{fa}) resulting from the concurrent `accept` by the other user; thus, b ends up not being a's friend, as prescribed by $F_{\mathtt{soc}}$. Making *friends* add-wins would result in the reverse outcome, and $F_{\mathtt{soc}}$ would have to change accordingly. Thus, the conflict-resolution policy on *friends* determines the way conflicts between `accept` and `breakup` are resolved.

Fig. 7. (Left) Coarse-grained contexts of the social graph data type together with the result that F_{soc} gives on them. (Right) Relevant events of the fine-grained executions of the implementation in Figure 3 resulting from concretising the contexts according to Definition 7. We use the same conventions as in Figure 5.



Finally, if users a and b issue friendship requests to each other concurrently, a decision such as an `accept` taken on one of them will also affect the other, as illustrated in Figure 7(c). To handle this situation without violating (2), `accept` removes not only the request it is resolving, but also the symmetric one.

6 Fine-grained language semantics, soundness and completeness

To justify that the coarse-grained semantics from §4 is sensible, we relate it to a *fine-grained semantics* that follows the standard way of defining language semantics on weak consistency models [10, 6]. Unlike the coarse-grained semantics, the fine-grained one is defined non-compositionally: it considers only certain *complete* programs and defines the denotation of a program as a whole, without separately defining denotations of composite data types in it. This denotation is computed using histories that record all operations on all primitive objects comprising the composite data types in the program; hence, the name fine-grained. The semantics includes those histories that can be produced by the the program in the session-local semantics (§4.1) and are allowed by the semantics of the store managing the primitive objects the program uses (§3).

We state the correspondence between the coarse-grained and fine-grained semantics as an equivalence of the *externally-observable behaviour* of a program in the two semantics. Let us fix a variable $x_{\mathsf{io}} \in \mathsf{OVar}$ and an object $\mathsf{io} \in \mathsf{Obj}$ used to interpret x_{io}. A program P is ***complete*** if $\emptyset \mid x_{\mathsf{io}} : \{o_{\mathsf{io}}\} \vdash P$. The operation o_{io} on x_{io}

models a combined user input-output action, rather than an operation on the store, and the externally-observable behaviour of a complete program P is given by operations on x_{io} it performs. Formally, for a history H let $\mathsf{observ}(H)$ be its projection to events on io: $\{e \in H \mid H.\mathsf{obj}(e) = \mathsf{io}\}$. We lift observ to sets of histories pointwise. Then we define the set of externally-observable behaviours of a complete program P in the coarse-grained semantics of §4 as $[\![P]\!]_{\mathsf{CG}} = \mathsf{observ}([\![P]\!]([\,], [x_{\mathsf{io}} : \mathsf{io}], [\,]))$. Note that our semantics does not restrict the values returned by o_{io}, thus accepting any input.

To define the fine-grained semantics of a complete program P, we flatten P by inlining composite data type definitions using a series of reductions $\longrightarrow$ on programs (defined shortly). Applying the reductions exhaustively yields programs with only objects of primitive data types, which have the following normal form:

$$\bar{P} \ ::= \ C_1 \parallel \ldots \parallel C_n \ \mid \ \mathsf{let}\ x = \mathsf{new}\ B\ \mathsf{in}\ \bar{P}$$

Given a complete program P, consider the unique $\bar{P}$ such that $P \longrightarrow^* \bar{P}$ and $\bar{P} \not\longrightarrow$ _. Then we define the denotation of P in the fine-grained semantics by the set of externally-observable behaviours that $\bar{P}$ produces when interacting with a causally consistent store managing the primitive objects it uses. To formalise this, we reuse the definition of the coarse-grained semantics and define the denotation of P in the fine-grained semantics as $[\![P]\!]_{\mathsf{FG}} = [\![\bar{P}]\!]_{\mathsf{CG}}$. Since $\bar{P}$ contains only primitive data types, this does not use the composite data type denotation of §4.2.

We now define the reduction $\longrightarrow$. Let Comm be the set of commands C in Figure 2. We use an operator $subst$ that takes a mapping $S : \mathsf{OVar} \times \mathsf{Op} \rightharpoonup \mathsf{Comm}$ and a command C or a program P, and replaces invocations of object operations in C or P according to S. The key clauses defining $subst$ are as follows:

$$\begin{aligned}
subst(S, v = x.o(G)) \ &= \ \mathsf{if}\ ((x,o) \notin \mathsf{dom}(S))\ \mathsf{then}\ (v = x.o(G)) \\
&\qquad \mathsf{else}\ (\mathsf{atomic}\ \{\mathsf{var}\ v_1.\, \mathsf{var}\ v_2.\, v_1 = G; (S(x,o)[v_1/v_{\mathsf{in}}, v_2/v_{\mathsf{out}}]); v = v_2\}) \\
subst(S, \mathsf{let}\ x = \mathsf{new}\ T\ \mathsf{in}\ P) \ &= \ \mathsf{let}\ x = \mathsf{new}\ T\ \mathsf{in}\ subst(S|_{\neg x}, P) \\
subst(S, \mathsf{let}\ \alpha = T\ \mathsf{in}\ P) \ &= \ \mathsf{let}\ \alpha = T\ \mathsf{in}\ subst(S, P) \\
subst(S, C_1 \parallel \ldots \parallel C_n) \ &= \ subst(S, C_1) \parallel \ldots \parallel subst(S, C_n)
\end{aligned}$$

Here v_1, v_2 are fresh ordinary variables, and $S|_{\neg x}$ denotes S with its domain restricted to $(\mathsf{OVar} \setminus \{x\}) \times \mathsf{Op}$. Applying $subst$ to an assignment command does not change the command, and applying it to all others results in recursive applications of $subst$ to their subexpressions. Then the relation $\longrightarrow$ is defined as follows:

$$\begin{aligned}
&\mathcal{P} \ ::= \ [-] \mid \mathsf{let}\ x = \mathsf{new}\ T\ \mathsf{in}\ \mathcal{P} \mid \mathsf{let}\ \alpha = T\ \mathsf{in}\ \mathcal{P} \\
&\mathcal{P}[\mathsf{let}\ \alpha = T\ \mathsf{in}\ P] \ \longrightarrow \ \mathcal{P}[P[T/\alpha]] \\
&\mathcal{P}[\mathsf{let}\ x = \mathsf{new}\ (\mathsf{let}\ \{x_j = \mathsf{new}\ T_j\}_{j=1..m}\ \mathsf{in}\ \{o = \mathsf{atomic}\ \{C_o\}\}_{o \in O})\ \mathsf{in}\ P] \\
&\qquad \longrightarrow \ \mathcal{P}[\mathsf{let}\ \{x_j = \mathsf{new}\ T_j\}_{j=1..m}\ \mathsf{in}\ subst(\{(x,o) \mapsto C_o \mid o \in O\}, P)],
\end{aligned}$$

where x_j do not occur in P. The first reduction rule replaces data-type variables by their definitions, and the second defines the semantics of composite operations via inlining.

Our central technical result is that the coarse-grained semantics of §4 is sound and complete with respect to the fine-grained semantics presented here: the sets of externally-observable behaviours of programs in the two semantics coincide.

THEOREM 10 *For every complete program P we have $[\![P]\!]_{\mathsf{FG}} = [\![P]\!]_{\mathsf{CG}}$.*

We give a (highly nontrivial) proof in §D. The theorem allows us to reason about programs using the coarse-grained semantics, which enables granularity abstraction and modular reasoning (§4.3). It also implies that our denotational semantics is *adequate*, i.e., can be used to prove the observational equivalence of two data type implementations D_1 and D_2: if $[\![D_1]\!] = [\![D_2]\!]$, then $[\![\mathcal{C}[D_1]]\!]_{\mathsf{FG}} = [\![\mathcal{C}[D_2]]\!]_{\mathsf{FG}}$ for all contexts $\mathcal{C}$ of the form $\mathcal{P}[\mathsf{let}\ \alpha = [-]\ \mathsf{in}\ P]$. Note that both soundness and completeness are needed to imply this property.

7 Related work

One of the classical questions of data abstraction is: how can we define the semantics of a data type implementation that abstracts away the implementation details, including a particular choice of data representation? Our results can be viewed as revisiting this question, which has so far been investigated in the context of sequential [14] and shared-memory concurrent [12, 24] programs, in the emerging domain of eventually consistent distributed systems. Most of the work on data abstraction for concurrency has considered a strongly consistent setting [12, 24]. Thus, it typically aimed to achieve *atomicity abstraction*, which allows one to pretend that a composite command takes effect atomically throughout the system. Here we consider data abstraction in the more challenging setting of weak consistency and achieve a weaker and more subtle guarantee of *granularity abstraction*: even though our coarse-grained semantics represents composite operations by single events, these events are still subject to anomalies of causal consistency, with different replicas being able to see the events at different times.

We are aware of only a few previous data abstraction results for weak consistency [15, 8, 5]. The most closely related is the one for the C/C++ memory model [6] by Batty et al. [5]. Like the consistency model we consider, the C/C++ model is defined axiomatically, which leads to some similarities in the general approach followed in [5] and in this paper. However, other features of the settings considered are different. First, we consider arbitrary replicated data types, whereas, as any model of a shared-memory language, the C/C++ one considers only registers with the last-writer-wins conflict-resolution policy. Second, the artefacts related during abstraction in [5] and in this paper are different. Instead of composite replicated data types, [5] considers *libraries*, which encapsulate last-writer-wins registers and operations accessing them implemented by arbitrary code without using transactions. A specification of a library is then just another library, but with operations implemented using atomic blocks reminiscent of our transactions. Hence, a single invocation of an operation of a specification library is still represented by multiple events and therefore [5] does not support granularity abstraction to the extent achieved here. Our work can roughly be viewed as starting where [5] left off, with composite constructions whose operations are implemented using transactions, and specifying their behaviour more declaratively with replicated data type specifications over contexts of coarse-grained events. It is thus possible that our approach can be adapted to give more declarative specifications to C/C++ libraries.

Researchers and developers have often implemented complex objects with domain-specific conflict resolution policies inside replicated stores [21], which requires dealing

with low-level details, such as message exchange between replicas. Our results show that, using causally consistent transactions, such complex domain-specific objects can often be implemented as composite replicated data types, using a high-level programming model to compose replicated objects and their conflict-resolution policies. Furthermore, due to the granularity abstraction we established, the resulting objects can be viewed as no different from those implemented inside the store.

We specify composite replicated data types using the formalism proposed for primitive replicated data types by Burckhardt et al. [10]. Thus, the novelty of our results lies not in the specification formalism, but in achieving granularity abstraction that lets us consider a composite data type as primitive and thereby specify it in this way. Burckhardt et al. also proposed a method for proving the correctness of data type implementations. This method considers only primitive data types implemented inside the store in a low-level way (e.g., using message exchanges), whereas we consider composite data types implemented using transactions in a higher-level model. Thus, the technical challenges addressed by the two methods are different.

Partial orders, such as event structures [19] and Mazurkiewicz traces [20], have been used to define semantics of concurrent or distributed programs by explicitly expressing the dependency relationships among events such programs generate. Our results extend this line of semantics research by considering new kinds of relations among events, describing computations of eventually consistent replicated stores, and studying how consistency axioms on these relations interact with the granularity abstraction for composite replicated data types.

8 Conclusion

In this paper we have proposed the concept of composite replicated data types, which formalises a common way of organising applications on top of eventually consistent stores. We have also presented a coarse-grained denotational semantics for these data types that supports granularity abstraction: the semantics allows us to abstract from the internals of a composite data type implementation and pretend that it represents a single monolithic object, which simplifies reasoning about client programs. Using a nontrivial example, we have illustrated how the denotation of a data type in our semantics specifies its behaviour in tricky situations and thereby lets one understand the consequences of different design decisions in its implementation. Finally, we have shown our semantics is sound and complete with respect to a standard non-compositional semantics.

As we explained in §1, developing correct programs on top of eventually consistent stores is a challenging yet unavoidable task. Our results mark the first step towards providing developers with methods and tools for specifying and verifying programs in this new programming environment and expanding the rich theories of programming languages, such as data abstraction, to this environment. Even though our results were developed for a particular popular variant of eventual consistency—causally consistent transactions—we hope that in the future the results can be generalised to other consistency models with similar formalisations [9, 3].

References

1. Microsoft TouchDevelop. https://www.touchdevelop.com/.
2. D. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, 2012.
3. P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Highly Available Transactions: virtues and limitations. In *VLDB*, 2014.
4. P. Bailis and A. Ghodsi. Eventual consistency today: Limitations, extensions, and beyond. *CACM*, 56(5), 2013.
5. M. Batty, M. Dodds, and A. Gotsman. Library abstraction for C/C++ concurrency. In *POPL*, 2013.
6. M. Batty, S. Owens, S. Sarkar, P. Sewell, and T. Weber. Mathematizing C++ concurrency. In *POPL*, 2011.
7. A. Bieniusa, M. Zawirski, N. M. Preguiça, M. Shapiro, C. Baquero, V. Balegas, and S. Duarte. Semantics of eventually consistent replicated sets. In *DISC*, 2012.
8. S. Burckhardt, A. Gotsman, M. Musuvathi, and H. Yang. Concurrent library correctness on the TSO memory model. In *ESOP*, 2012.
9. S. Burckhardt, A. Gotsman, and H. Yang. Understanding eventual consistency. Technical Report MSR-TR-2013-39, Microsoft, 2013.
10. S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski. Replicated data types: specification, verification, optimality. In *POPL*, 2014.
11. S. Burckhardt, D. Leijen, M. Fähndrich, and M. Sagiv. Eventually consistent transactions. In *ESOP*, 2012.
12. I. Filipovic, P. W. O'Hearn, N. Rinetzky, and H. Yang. Abstraction for concurrent objects. *Theor. Comput. Sci.*, 411(51-52), 2010.
13. S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2), 2002.
14. C. A. R. Hoare. Proof of correctness of data representations. *Acta Inf.*, 1, 1972.
15. R. Jagadeesan, G. Petri, C. Pitcher, and J. Riely. Quarantining weakness - compositional reasoning under relaxed memory models (extended abstract). In *ESOP*, 2013.
16. 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.
17. 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.
18. W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Stronger semantics for low-latency geo-replicated storage. In *NSDI*, 2013.
19. M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains. In *Semantics of Concurrent Computation*, 1979.
20. M. Nielsen, V. Sassone, and G. Winskel. Relationships between models of concurrency. In *REX School/Symposium*, 1993.
21. M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Technical Report 7506, INRIA, 2011.
22. M. Shapiro, N. M. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In *SSS*, 2011.
23. Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *SOSP*, 2011.
24. A. Turon, D. Dreyer, and L. Birkedal. Unifying refinement and hoare-style reasoning in a logic for higher-order concurrency. In *ICFP*, 2013.
25. M. Zawirski, A. Bieniusa, V. Balegas, S. Duarte, C. Baquero, M. Shapiro, and N. Preguiça. SwiftCloud: Fault-tolerant geo-replication integrated all the way to the client machine. Technical Report 8347, INRIA, 2013.

A Additional clauses of the session-local semantics

$$\langle \mathsf{var}\ v.\, C\rangle(obj,\sigma) = (\mathsf{IHist} \cap \langle C\rangle(obj, \sigma[v \mapsto \bot])) \cup \{(H, \sigma'|_{\mathsf{dom}(\Sigma)}) \mid (H,\sigma') \in \langle C\rangle(obj,\sigma[v\mapsto\bot])\};$$

$$\langle \mathsf{if}\ G\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\rangle(obj,\sigma) = \begin{cases} \langle C_1\rangle(obj,\sigma), \text{ if } [\![G]\!]\sigma \neq 0; \\ \langle C_2\rangle(obj,\sigma), \text{ otherwise;} \end{cases}$$

$$\langle C_1; C_2\rangle(obj,\sigma) = \mathsf{seq}(\langle C_1\rangle(obj,\sigma),\ \lambda\sigma'.\, \langle C_2\rangle(obj,\sigma'));$$

$$\langle \mathsf{while}\ G\ \mathsf{do}\ C\rangle(obj,\sigma) = (\mathsf{greatestFix}\ K)(\sigma),$$

where

$$K = \lambda W.\, \lambda\sigma'.\, \begin{cases} \{((\emptyset,[\,],\emptyset,\emptyset),\sigma')\} & \text{if } [\![G]\!]\sigma' = 0; \\ \mathsf{seq}(\langle C\rangle(obj,\sigma'), W) & \text{otherwise;} \end{cases}$$

$$\begin{aligned} &\mathsf{seq}(U,W) = \\ &\{((E_1 \uplus E_2, \mathsf{label}_1 \uplus \mathsf{label}_2, \mathsf{so}_1 \cup \mathsf{so}_2 \cup (E_1 \times E_2), \sim_1 \cup \sim_2), \sigma_2) \mid \\ &\qquad ((E_1,\mathsf{label}_1,\mathsf{so}_1,\sim_1),\sigma_1) \in U \wedge ((E_2,\mathsf{label}_2,\mathsf{so}_2,\sim_2),\sigma_2) \in W(\sigma_1)\} \cup \\ &\{(E_1 \uplus E_2, \mathsf{label}_1 \uplus \mathsf{label}_2, \mathsf{so}_1 \cup \mathsf{so}_2 \cup (E_1 \times E_2), \sim_1 \cup \sim_2) \mid \\ &\qquad ((E_1,\mathsf{label}_1,\mathsf{so}_1,\sim_1),\sigma_1) \in U \wedge (E_2,\mathsf{label}_2,\mathsf{so}_2,\sim_2) \in W(\sigma_1)\} \cup \\ &(\mathsf{IHist} \cap U). \end{aligned}$$

Note that in the clause for local-variable declarations we assume an appropriate alpha-renaming of bound variables.

B Characterisation of $\gamma(N, \rho, \mathbb{F})$ in terms of abstraction

In the rest of the appendix, we use an alternative, equivalent characterisation of $\gamma(N,\rho,\mathbb{F})$ given in the main text. Our characterisation is based on the notion of abstraction. This characterisation will help us to manage the complexity of reasoning involved in the proofs of the theorems.

DEFINITION 11 *An **abstraction** from a history H to a tuple (N, e_0, c) of a context N, an event $e_0 \notin N.E$ and a value $c \in \mathsf{Val}$ is a pair (β,ρ) of a function $\beta : H.E \to N.E \uplus \{e_0\}$ and an summary $\rho : \mathsf{AOp} \times \mathsf{Val} \rightharpoonup \mathcal{P}(\mathsf{FHist})$ such that*

$$\begin{gathered} (\forall f \in (N.E).\, H|_{\beta^{-1}(f)} \in \rho(N.\mathsf{aop}(f), \_)) \ \wedge \\ (H|_{\beta^{-1}(e_0)} \in \rho(N.p, c)) \ \wedge \ \beta(H.\mathsf{so}) \subseteq \mathsf{id} \ \wedge \ \beta(H.\sim) \subseteq \mathsf{id}. \end{gathered}$$

We write $(\beta,\rho) : H \to (N, e_0, c)$ to mean that (β,ρ) is an abstraction of H into (N, e_0, c).

DEFINITION 12 *An **abstraction** from an execution X to (N, e_0, c) is an abstraction $(\beta, \rho)\colon X.H \to (N, e_0, c)$ such that*

$$\beta(X.\mathsf{vis}) - \mathsf{id} \subseteq \mathsf{vis}';$$
$$\beta^{-1}(\mathsf{vis}') \cap \mathsf{sameobj}(X) \subseteq X.\mathsf{vis};$$
$$\beta(X.\mathsf{ar}) - \mathsf{id} \subseteq \mathsf{ar}',$$

where

$$\mathsf{vis}' = N.\mathsf{vis} \cup \{(f, e_0) \mid f \in N.E\};$$
$$\mathsf{ar}' = N.\mathsf{ar} \cup \{(f, e_0) \mid f \in N.E\}.$$

We write $(\beta, \rho) : X \to (N, e_0, c)$ to mean that (β, ρ) is an abstraction of X into (N, e_0, c).

It is immediate from our definition that

$$\gamma(N, \rho) = \{(X, c) \mid \exists e_0 \notin (N.E).\, \exists \beta.\, (\beta, \rho) : X \to (N, e_0, c)\}.$$

From this observation follows the following lemma:

Lemma 1.

$$\gamma(N, \rho, \mathbb{F}) = \{(X, c) \mid \exists e_0 \notin (N.E).\, \exists \beta.(\beta, \rho) : X \to (N, e_0, c) \wedge X \models_{\mathsf{CC}} \mathbb{F}\}.$$

C Proof of Theorem 8

In the following we use $H.\mathsf{op}(e)$ and $H.\mathsf{arg}(e)$ to denote the components of $H.\mathsf{aop}(e)$.

We define the notions of morphisms between histories and executions, which generalise the abstraction from histories or executions to contexts in Definitions 11 and 12. For $\omega \in \mathsf{Obj}$ we let

$$\mathsf{Hist}(\omega) = \{H \in \mathsf{Hist} \mid H.\mathsf{obj}(H.E) = \{\omega\}\};$$
$$\mathsf{Exec}(\omega) = \{X \in \mathsf{Exec} \mid X.H \in \mathsf{Hist}(\omega)\}.$$

DEFINITION 13 *A **morphism** from a history H to another history H', such that $H \in \mathsf{Hist}$ and $H' \in \mathsf{Hist}(\omega)$ for some ω, is a pair of a function $\beta : H.E \to H'.E$ and a summary $\rho : \mathsf{AOp} \times \mathsf{Val} \rightharpoonup \mathcal{P}(\mathsf{FHist})$ such that*

$$(\forall e \in (H'.E).\, H|_{\beta^{-1}(e)} \in \rho(H'.\mathsf{aop}(e), H'.\mathsf{rval}(e))) \quad \wedge$$
$$\beta(H.\mathsf{so}) - \mathsf{id} = (H'.\mathsf{so})|_{\beta(H.E)} \quad \wedge \quad \beta(H.\sim) = (H'.\sim)|_{\beta(H.E)}.$$

We write $(\beta, \rho) : H \to H'$ to mean that (β, ρ) is a morphism from H to H'.

LEMMA 14 *For every morphism $(\beta, \rho) : H \to H'$, we have that*

$$\forall e, f \in H.E.\ (e, f) \in H.\sim \iff (\beta(e), \beta(f)) \in H'.\sim.$$

PROOF. Let (β, ρ) be a morphism from H to H'. Consider $e, f \in H.E$. By the definition of morphism, $\beta(H.\sim) = (H'.\sim)|_{\beta(H.E)}$. Hence,

$$(e, f) \in H.\sim \implies (\beta(e), \beta(f)) \in H'.\sim.$$

Let us move on to the other implication. Assume that $(\beta(e), \beta(f)) \in H'.\sim$. Since $\beta(H.\sim) = (H'.\sim)|_{\beta(H.E)}$, there exist $e', f' \in H.E$ such that

$$\beta(e) = \beta(e') \quad \wedge \quad \beta(f) = \beta(f') \quad \wedge \quad (e', f') \in H.\sim.$$

Let $e'' = \beta(e')$ and $f'' = \beta(f')$. Since (β, ρ) is a morphism,

$$H|_{\beta^{-1}(e'')} \in \rho(H'.\mathsf{aop}(e''), H'.\mathsf{rval}(e'')) \wedge H|_{\beta^{-1}(f'')} \in \rho(H'.\mathsf{aop}(f''), H'.\mathsf{rval}(f'')).$$

Recall that every history in the image of ρ uses the complete relation (i.e., a relation that relates all pairs of events) as its equivalence relation. Also, $e, e' \in (H|_{\beta^{-1}(e'')}).E$ and $f, f' \in (H|_{\beta^{-1}(f'')}).E$. Hence,

$$(e, e') \in H.\sim \quad \wedge \quad (f', f) \in H.\sim.$$

Since $(e', f') \in H.\sim$, by the transitivity of $H.\sim$, we have that $(e, f) \in H.\sim$, as desired. □

DEFINITION 15 *A **morphism** from an execution $X \in \mathsf{Exec}$ to another execution $X' \in \mathsf{Exec}(\omega)$ is a morphism $(\beta, \rho) : X.H \to X'.H$ such that*

$$\begin{aligned} \beta(X.\mathsf{vis}) - \mathsf{id} \subseteq X'.\mathsf{vis} \quad &\wedge \quad \beta^{-1}(X'.\mathsf{vis}) \cap \mathsf{sameobj}(X) \subseteq X.\mathsf{vis} \\ &\wedge \quad \beta(X.\mathsf{ar}) - \mathsf{id} \subseteq X'.\mathsf{ar}. \end{aligned}$$

We write $(\beta, \rho) : X \to X'$ to mean that (β, ρ) is a morphism from X to X'.

DEFINITION 16 *An operation context N can be **embedded** into an execution X at an event $e \notin N.E$, denoted $(N, e) \hookrightarrow X$, if and only if for some $\omega \in \mathsf{Obj}$,*

$$\begin{aligned} X.E = N.E \uplus \{e\} \ \wedge \ X.\mathsf{obj}(X.E) = \{\omega\} \ \wedge \ X.\mathsf{so} = \emptyset \quad \wedge \quad X.\sim = \mathsf{id} \\ \wedge \ X.\mathsf{vis} = N.\mathsf{vis} \cup \{(f, e) \mid f \in N.E\} \ \wedge \ X.\mathsf{ar} = N.\mathsf{ar} \cup \{(f, e) \mid f \in N.E\}. \end{aligned}$$

PROPOSITION 17 *For all β, ρ, X, N, e such that $e \notin N.E, a$,*

$$\begin{aligned} (\beta, \rho) : X \to (N, e, a) \iff \\ (\exists X'.\, (N, e) \hookrightarrow X' \wedge (\beta, \rho) : X \to X' \wedge a = X'.\mathsf{rval}(e)). \end{aligned}$$

Hence, γ from §4 can be equivalently defined as follows:

$$\begin{aligned} \gamma(N, \rho, \mathbb{F}) = \{(X, a) \mid \exists e \notin N.E.\, \exists \beta.\, \exists X'.\, (N, e) \hookrightarrow X' \wedge (\beta, \rho) : X \to X' \\ \wedge\, a = X'.\mathsf{rval}(e) \wedge X \models_{\mathsf{CC}} \mathbb{F}\}. \end{aligned}$$

In the following we sometimes use this fact even without mentioning it explicitly.

For $H_1, H_2 \in \mathsf{Hist}$ and $\pi : H_1.E \to_{\mathsf{bij}} H_2.E$ we let

$$H_1 \approx_\pi H_2 \iff \pi(H_1.\mathsf{label}) = H_2.\mathsf{label} \wedge \pi(H_1.\mathsf{so}) = H_2.\mathsf{so} \wedge \pi(H_1.\sim) = H_2.\sim;$$

$$H_1 \approx H_2 \iff \exists \pi.\, H_1 \approx_\pi H_2.$$

For $X_1, X_2 \in \mathsf{Exec}$ and $\pi : X_1.E \to_{\mathsf{bij}} X_2.E$ we let:

$$X_1 \approx_\pi X_2 \iff X_1.H \approx_\pi X_2.H \wedge \pi(X_1.\mathsf{vis}) = X_2.\mathsf{vis} \wedge (\pi(X_1.\mathsf{ar}) \cup X_2.\mathsf{ar} \text{ is acyclic});$$

$$\begin{aligned} X_1 \approx_{\mathsf{rval}} X_2 \iff\ & X_1.E = X_2.E \wedge H_1.\mathsf{obj} = H_2.\mathsf{obj} \wedge H_1.\mathsf{aop} = H_2.\mathsf{aop} \wedge \\ & X_1.\mathsf{so} = X_2.\mathsf{so} \wedge X_1.\sim = X_2.\sim \wedge X_1.\mathsf{vis} = X_2.\mathsf{vis} \wedge X_1.\mathsf{ar} = X_2.\mathsf{ar}. \end{aligned}$$

We also let

$$\begin{aligned} \mathsf{CExec} = \{X \in \mathsf{Exec} \mid\ & (|X.E| < \infty) \wedge \\ & \exists \omega \in \mathsf{Obj}.\, X.H \in \mathsf{Hist}(\omega) \wedge X.\mathsf{so} = \emptyset \wedge X.\sim = \mathsf{id}\}. \end{aligned}$$

A history H is ***sequential*** if $H.\mathsf{so}$ is total on $H.E$. We write SHist for the set of sequential histories. It is easy to see that all histories produced by the session-local semantics are sequential. For $H \in \mathsf{SHist}$ we write $H(n)$ for the n-th event in $H.\mathsf{so}$, undefined when there is not one. We also write $H|_n$ for the projection of H to the first n events in $H.\mathsf{so}$; if $n > |H.E|$, we let $H|_n = H$.

PROPOSITION 18 (DETERMINACY) *For $\Delta \mid \Sigma \vdash C$ we have:*

$$\begin{aligned} & \forall obj : \mathsf{dom}(\Delta) \to_{\mathsf{inj}} \mathsf{Obj}.\, \forall H_1, H_2 \in \mathsf{Hist}.\, \forall \sigma, \sigma_1, \sigma_2 \in \mathsf{LState}(\Sigma). \\ & \quad (H_1, \sigma_1), (H_2, \sigma_2) \in \langle C \rangle(obj, \sigma) \wedge H_1 \approx H_2 \implies \sigma_1 = \sigma_2 \end{aligned}$$

and

$$\begin{aligned} & \forall obj : \mathsf{dom}(\Delta) \to_{\mathsf{inj}} \mathsf{Obj}.\, \forall H_1, H_2 \in \mathsf{Hist}.\, \forall \sigma \in \mathsf{LState}(\Sigma).\, \forall n. \\ & (H_1, \_), (H_2, \_) \in \langle C \rangle(obj, \sigma) \wedge H_1|_n \approx H_2|_n \wedge (1 \le n \le |H_1|, |H_2|) \implies \\ & \quad (n+1 \le |H_1| \iff n+1 \le |H_2|) \wedge \\ & \quad (n+1 \le |H_1| \implies H_1.\mathsf{obj}(H_1(n+1)) = H_2.\mathsf{obj}(H_2(n+1)) \\ & \qquad\qquad \wedge H_1.\mathsf{aop}(H_1(n+1)) = H_1.\mathsf{aop}(H_2(n+1))). \end{aligned}$$

THEOREM 19 *Given a family of well-typed commands $\{\Delta \mid v_{\mathsf{in}}, v_{\mathsf{out}} \vdash C_o\}_{o \in O}$, we have:*

$$\begin{aligned} & \forall obj \in [\mathsf{dom}(\Delta) \to_{\mathsf{inj}} \mathsf{Obj}].\, \forall \mathbb{F} \in [\mathsf{range}(obj) \to \mathsf{Spec}]. \\ & \forall X_1', X_2' \in \mathsf{CExec}.\, \forall X_1, X_2 \in \mathsf{Exec}.\, \forall \beta_1, \beta_2. \\ & (\beta_1, [\![\{C_o\}_{o \in O}]\!](obj)) : X_1 \to X_1' \wedge (\beta_2, [\![\{C_o\}_{o \in O}]\!](obj)) : X_2 \to X_2' \wedge \\ & X_1' \approx_{\mathsf{rval}} X_2' \wedge X_1 \models_{\mathsf{CC}} \mathbb{F} \wedge X_2 \models_{\mathsf{CC}} \mathbb{F} \implies \\ & \quad X_1' = X_2' \wedge \exists \pi : X_1.E \to_{\mathsf{bij}} X_2.E.\, X_1 \approx_\pi X_2 \wedge \forall f \in X_1.E.\, \beta_1(f) = \beta_2(\pi(f)). \end{aligned}$$

PROOF. Consider

$$\{x_j : O_j \mid j = 1..m\} \mid v_{\mathsf{in}}, v_{\mathsf{out}} \vdash C_o, \quad o \in O$$

and

$$obj \in [\{x_j \mid j = 1..m\} \to_{\mathsf{inj}} \mathsf{Obj}]; \qquad \mathbb{F} \in [\mathsf{range}(obj) \to \mathsf{Spec}]$$

and let $\rho = [\![\{C_o\}_{o \in O}]\!](obj)$.

We prove that the following holds for all $X_1', X_2' \in \mathsf{CExec}$ such that $X_1' \approx_{\mathsf{rval}} X_2'$ by induction on $|X_1'.E| = |X_2'.E|$:

$$\begin{array}{l}\forall X_1, X_2 \in \mathsf{Exec}.\, \forall \beta_1, \beta_2.\\ (\beta_1, \rho) : X_1 \to X_1' \wedge (\beta_2, \rho) : X_2 \to X_2' \wedge X_1 \models_{\mathsf{CC}} \mathbb{F} \wedge X_2 \models_{\mathsf{CC}} \mathbb{F} \implies\\ \quad X_1' = X_2' \wedge \exists \pi : X_1.E \to_{\mathsf{bij}} X_2.E.\, X_1 \approx_\pi X_2 \wedge \forall f \in X_1.E.\, \beta_1(f) = \beta_2(\pi(f)).\end{array}$$

The base case when $|X_1'.E| = |X_2'.E| = 0$ is trivial.

Consider $X_1', X_2' \in \mathsf{CExec}$, $X_1, X_2 \in \mathsf{Exec}$ and β_1, β_2 such that

$$X_1' \approx_{\mathsf{rval}} X_2' \wedge (\beta_1, \rho) : X_1 \to X_1' \wedge (\beta_2, \rho) : X_2 \to X_2' \wedge X_1 \models_{\mathsf{CC}} \mathbb{F} \wedge X_2 \models_{\mathsf{CC}} \mathbb{F}.$$

Let us choose an event $e \in X_1'.E$ that does not have successors in $X_1'.\mathsf{so} \cup X_1'.\mathsf{vis} \cup X_1'.\mathsf{ar}$; this is possible since $X_1' \in \mathsf{CExec}$. Then $e \in X_2'.E$ and it does not have successors in $X_2'.\mathsf{so} \cup X_2'.\mathsf{vis} \cup X_2'.\mathsf{ar}$. Let

$$Y_1' = X_1'|_{X_1'.E - \{e\}} \quad \wedge \quad Y_2' = X_2'|_{X_2'.E - \{e\}};$$

then $Y_1' \approx_{\mathsf{rval}} Y_2'$. Let

$$Y_1 = X_1|_{X_1.E - \beta_1^{-1}(e)} \quad \wedge \quad Y_2 = X_2|_{X_2.E - \beta_2^{-1}(e)}.$$

It is easy to see that Y_1 and Y_2 still satisfy CAUSALVIS and CAUSALAR. If $(f, g) \in X_1.\mathsf{vis}$ for some $f \in \beta_1^{-1}(e)$ and $g \in X_1.E - \beta_1^{-1}(e)$, then $(e, \beta_1(g)) \in X_1'.\mathsf{vis}$, contradicting the fact that e does not have successors in $X_1'.\mathsf{vis}$. Hence, such f and g do not exist, which implies that $Y_1 \models \mathbb{F}$. We similarly establish $Y_2 \models \mathbb{F}$.

Let

$$\beta_1' = \beta_1|_{X_1.E - \beta_1^{-1}(e)} \quad \wedge \quad \beta_2' = \beta_2|_{X_2.E - \beta_2^{-1}(e)}.$$

Is easy to check that

$$(\beta_1', \rho) : Y_1 \to Y_1' \quad \wedge \quad (\beta_2', \rho) : Y_2 \to Y_2'.$$

We have thus established all the premisses of the induction hypothesis for Y_1' and Y_2'. Applying it, we get that $Y_1' = Y_2'$ and for some $\pi : Y_1.E \to_{\mathsf{bij}} Y_2.E$ we have $Y_1 \approx_\pi Y_2$ and $\forall f \in Y_1'.E.\, \beta_1'(f) = \beta_2'(\pi(f))$.

Let $(o, a) = X_1'.\mathsf{aop}(e) = X_2'.\mathsf{aop}(e)$. Then

$$\begin{array}{c}((X_1.H)|_{\beta_1^{-1}(e)}, [v_{\mathsf{in}} \mapsto \_, v_{\mathsf{out}} \mapsto X_1'.\mathsf{rval}(e)]) \in \langle \mathsf{atomic}\ \{C_o\}\rangle(obj, [v_{\mathsf{in}} \mapsto a, v_{\mathsf{out}} \mapsto \bot])\\ \wedge\\ ((X_2.H)|_{\beta_2^{-1}(e)}, [v_{\mathsf{in}} \mapsto \_, v_{\mathsf{out}} \mapsto X_2'.\mathsf{rval}(e)]) \in \langle \mathsf{atomic}\ \{C_o\}\rangle(obj, [v_{\mathsf{in}} \mapsto a, v_{\mathsf{out}} \mapsto \bot])\end{array} \tag{13}$$

and, in particular, $(X_2.H)|_{\beta_2^{-1}(e)}, (X_2.H)|_{\beta_2^{-1}(e)} \in \mathsf{SHist}$.

For $n \geq 0$ let

$$E_1^n = (((X_1.H)|_{\beta_1^{-1}(e)})|_n).E \quad \wedge \quad E_2^n = (((X_2.H)|_{\beta_2^{-1}(e)})|_n).E;$$

$$X_1^n = X_1|_{Y_1.E \cup E_1^n} \quad \wedge \quad X_2^n = X_2|_{Y_2.E \cup E_2^n}.$$

We prove by induction on n that

$$\exists \pi' : E_1^n \to_{\mathsf{bij}} E_2^n.\, X_1^n \approx_{\pi \uplus \pi'} X_2^n \wedge (\forall f \in X_1^n.E.\, \beta_1(f) = \beta_2((\pi \uplus \pi')(f))).$$

The base case of $n = 0$ follows from $Y_1 \approx_\pi Y_2$ and $\forall f \in Y_1'.E.\, \beta_1'(f) = \beta_2'(\pi(f))$.

Assume that for some $n \geq 0$ and $\pi' : E_1^n \to_{\mathsf{bij}} E_2^n$ we have $X_1^n \approx_{\pi \uplus \pi'} X_2^n$ and $\forall f \in X_1^n.E.\, \beta_1(f) = \beta_2((\pi \uplus \pi')(f))$. Then

$$((X_1.H)|_{\beta_1^{-1}(e)})|_n \approx_{\pi'} ((X_2.H)|_{\beta_2^{-1}(e)})|_n. \tag{14}$$

If $n \geq |\beta_1^{-1}(e)|$, then the above implies

$$((X_1.H)|_{\beta_1^{-1}(e)}) \approx_{\pi'} ((X_2.H)|_{\beta_2^{-1}(e)})|_{|\beta_1^{-1}(e)|}.$$

Due to (13), by Proposition 18 we get $|\beta_2^{-1}(e)| = |\beta_1^{-1}(e)|$ (so that $X_1^n = X_1^{n+1}$ and $X_2^n = X_2^{n+1}$) and

$$\begin{aligned}((X_1.H)|_{\beta_1^{-1}(e)})|_{n+1} &= ((X_1.H)|_{\beta_1^{-1}(e)}) \\ &\approx_{\pi'} ((X_2.H)|_{\beta_2^{-1}(e)}) = ((X_2.H)|_{\beta_2^{-1}(e)})|_{n+1},\end{aligned}$$

as required. We obtain the same when $n \geq |\beta_2^{-1}(e)|$ in a similar way.

Now assume that $n < |\beta_1^{-1}(e)|$ and $n < |\beta_2^{-1}(e)|$ and let

$$e_1 = ((X_1.H)|_{\beta_1^{-1}(e)})(n+1) \quad \wedge \quad e_2 = ((X_2.H)|_{\beta_2^{-1}(e)})(n+1).$$

Due to (13) and (14), by Proposition 18 we get

$$X_1.\mathsf{obj}(e_1) = X_2.\mathsf{obj}(e_2) \quad \wedge \quad X_1.\mathsf{aop}(e_1) = X_2.\mathsf{aop}(e_2). \tag{15}$$

Let $\pi'' = \pi'[e_1 : e_2]$; then $X_1^{n+1}.H \approx_{\pi \uplus \pi''} X_2^{n+1}.H$ and

$$\beta_1(e_1) = e = \beta_2(e_2) = \beta_2((\pi \uplus \pi'')(e_1)),$$

as required.

We now show $(\pi \uplus \pi'')(X_1^{n+1}.\mathsf{vis}) = X_2^{n+1}.\mathsf{vis}$. Consider $(f, g) \in X_1^{n+1}.\mathsf{vis}$:

– If $f \neq e_1$ and $g \neq e_1$, then $f, g \in X_1^n.E$. Then $X_1^n \approx_{\pi \uplus \pi'} X_2^n$ implies

$$((\pi \uplus \pi')(f), (\pi \uplus \pi')(g)) \in X_2^n.\mathsf{vis}$$

and, hence,

$$((\pi \uplus \pi'')(f), (\pi \uplus \pi'')(g)) \in X_2^{n+1}.\mathsf{vis}.$$

– We have previously shown that there are no $f \in \beta_1^{-1}(e)$ and $g \in X_1.E - \beta_1^{-1}(e)$ such that $(f, g) \in X_1.\mathsf{vis}$. Hence, we are left with the case when $g = e_1$, which we consider in the following.
– If $f \in E_1^n$, then $(f, e_1) \in X_1.\mathsf{so}$. But then $X_1^n \approx_{\pi \uplus \pi'} X_2^n$ implies

$$((\pi \uplus \pi')(f), e_2) \in X_2.\mathsf{so}$$

and, by CAUSALVIS for X_2, we have

$$((\pi \uplus \pi')(f), e_2) \in X_2.\mathsf{vis}.$$

This implies

$$((\pi \uplus \pi'')(f), (\pi \uplus \pi'')(e_1)) \in X_2.\mathsf{vis}.$$

– Assume now that $f \in X_1.E - \beta_1^{-1}(e)$. Then $\beta_1(f) \neq \beta_1(e_1) = e$ and $(\beta_1, \rho) : X_1 \to X_1'$ implies

$$(\beta_1(f), e) \in X_1'.\mathsf{vis}.$$

Since $X_1' \approx_{\mathsf{rval}} X_2'$, we also have

$$(\beta_1(f), e) \in X_2'.\mathsf{vis}.$$

Since $e = \beta_2(e_2)$ and $\beta_1(f) = \beta_2((\pi \uplus \pi')(f))$, this is equivalent to

$$(\beta_2((\pi \uplus \pi')(f)), \beta_2(e_2)) \in X_2'.\mathsf{vis}. \tag{16}$$

We have $X_1.\mathsf{obj}(f) = X_1.\mathsf{obj}(e_1)$ and $X_1^n \approx_{\pi \uplus \pi'} X_2^n$ implies $X_2.\mathsf{obj}((\pi \uplus \pi')(f)) = X_1.\mathsf{obj}(f)$. From this and (15) we get

$$X_2.\mathsf{obj}((\pi \uplus \pi')(f)) = X_2.\mathsf{obj}(e_2).$$

Given this and (16), from $(\beta_2, \rho) : X_2 \to X_2'$ we get

$$((\pi \uplus \pi')(f), e_2) \in X_2.\mathsf{vis},$$

which implies

$$((\pi \uplus \pi'')(f), (\pi \uplus \pi'')(e_1)) \in X_2^{n+1}.\mathsf{vis}.$$

We have thus shown that $(\pi \uplus \pi'')(X_1^{n+1}.\mathsf{vis}) = X_2^{n+1}.\mathsf{vis}$.

We now show that

$$(\pi \uplus \pi'')(X_1^{n+1}.\mathsf{ar}) \cup X_2^{n+1}.\mathsf{ar}$$

is acyclic. Assume that there is a cycle in this relation. Since $X_1^n \approx_{\pi \uplus \pi'} X_2^n$, we know that

$$(\pi \uplus \pi')(X_1^n.\mathsf{ar}) \cup X_2^n.\mathsf{ar}$$

is acyclic. Hence, the above cycle contains e_2 and an edge

$$(e_2, f) \in (\pi \uplus \pi'')(X_1^{n+1}.\mathsf{ar}) \cup X_2^{n+1}.\mathsf{ar}$$

for some f. Then

$$(e_2, f) \in X_2^{n+1}.\mathsf{ar} \vee \exists g.\, (e_1, g) \in X_1^{n+1}.\mathsf{ar}.$$

Using the choice of e_1 and e_2, similarly to how it was done previously, we can show that none of these cases is possible, which establishes the desired acyclicity guarantee.

Similarly to how we previously showed that $Y_1 \models \mathbb{F}$ and $Y_2 \models \mathbb{F}$, we can show $X_1^{n+1} \models \mathbb{F}$ and $X_2^{n+1} \models \mathbb{F}$. Let

$$Z_1 = (X_1^{n+1}.H, X_1^{n+1}.\mathsf{vis}, (X_1^{n+1}.\mathsf{ar} \cup (\pi \uplus \pi'')^{-1}(X_2^{n+1}.\mathsf{ar}))^+) \quad \wedge$$
$$Z_2 = (X_2^{n+1}.H, X_2^{n+1}.\mathsf{vis}, (X_2^{n+1}.\mathsf{ar} \cup (\pi \uplus \pi'')(X_1^{n+1}.\mathsf{ar}))^+).$$

From the above-established acyclicity guarantee, Z_1 and Z_2 are executions. Since data type specifications preserve their value on arbitration extensions, we still have $Z_1 \models \mathbb{F}$ and $Z_2 \models \mathbb{F}$. Furthermore, from what we have shown so far and (15) it easily follows that $\mathsf{ctxt}(Z_1, e_1) \approx \mathsf{ctxt}(Z_2, e_2)$. Since data type specifications give the same values on isomorphic contexts,

$$X_1^{n+1}.\mathsf{rval}(e_1) = Z_1.\mathsf{rval}(e_1) = Z_2.\mathsf{rval}(e_2) = X_2^{n+1}.\mathsf{rval}(e_2).$$

This finally establishes $X_1^{n+1} \approx_{\pi \uplus \pi''} X_2^{n+1}$, completing the induction.

Now choosing n such that $n \geq |\beta_1^{-1}(e)|$ and $n \geq |\beta_2^{-1}(e)|$, we get $X_1^n = X_1$ and $X_2^n = X_2$. Then the statement just proved gives us that for some π' : $((X_1.H)|_{\beta_1^{-1}(e)}).E \rightarrow_{\mathsf{bij}} ((X_2.H)|_{\beta_2^{-1}(e)}).E$ we have $X_1 \approx_{\pi \uplus \pi'} X_2$. As a consequence,

$$(X_1.H)|_{\beta_1^{-1}(e)} \approx_{\pi'} (X_2.H)|_{\beta_2^{-1}(e)}.$$

Then by (13) and Proposition 18 we get $X_1'.\mathsf{rval}(e) = X_2'.\mathsf{rval}(e)$, as required. □

PROPOSITION 20 *Given a family of well-typed commands* $\{\Delta \mid v_{\mathsf{in}}, v_{\mathsf{out}} \vdash C_o\}_{o \in O}$*, we have:*

$\forall N.\, \forall obj_1, obj_2 \in [\mathsf{dom}(\Delta) \rightarrow_{\mathsf{inj}} \mathsf{Obj}]$.
$\forall \mathbb{F}_1 \in [\mathsf{range}(obj_1) \rightarrow \mathsf{Spec}].\, \forall \mathbb{F}_2 \in [\mathsf{range}(obj_2) \rightarrow \mathsf{Spec}]$.
$(\forall x \in \mathsf{dom}(\Delta).\, \mathbb{F}_1(obj_1(x)) = \mathbb{F}(obj_2(x))) \implies$
$\forall (X_1, a) \in \gamma(N, [\![\{C_o\}_{o \in O}]\!](obj_1), \mathbb{F}_1).\, \exists X_2.\, (X_2, a) \in \gamma(N, [\![\{C_o\}_{o \in O}]\!](obj_2), \mathbb{F}_2)$.

PROOF OF THEOREM 8. Assume $N, obj_1, obj_2, \mathbb{F}_1, \mathbb{F}_2, X_1, a_1, X_2, a_2$ satisfying the conditions of the theorem. Since

$$(X_1, a_1) \in \gamma(N, [\![\{C_o\}_{o \in O}]\!](obj_1), \mathbb{F}_1),$$

for some $e_1 \notin N.E$ we have

$$\exists \beta_1.\, (\beta_1, [\![\{C_o\}_{o \in O}]\!](obj_1)) : X_1 \rightarrow (N, e_1, a_1) \wedge X_1 \models_{\mathsf{CC}} \mathbb{F}_1.$$

By Proposition 20 there exists $Y_2 \in \mathsf{Exec}$ such that

$$(Y_2, a_2) \in \gamma(N, [\![\{C_o\}_{o \in O}]\!](obj_1), \mathbb{F}_1),$$

i.e.,

$$\exists e_2 \notin N.E.\, \exists \beta_2.\, (\beta_2, [\![\{C_o\}_{o \in O}]\!](obj_1)) : Y_2 \rightarrow (N, e_2, a_2) \wedge Y_2 \models_{\mathsf{CC}} \mathbb{F}_1.$$

Then it is easy to see that

$$\exists \beta_2'.\, (\beta_2', [\![\{C_o\}_{o \in O}]\!](obj_1)) : Y_2 \to (N, e_1, a_2) \wedge Y_2 \models_{\mathsf{CC}} \mathbb{F}_1.$$

By Proposition 17, there exist X_1', X_2' such that

$$(\beta_1, [\![\{C_o\}_{o \in O}]\!](obj)) : X_1 \to X_1' \ \wedge \ (\beta_2, [\![\{C_o\}_{o \in O}]\!](obj)) : Y_2 \to X_2' \ \wedge$$
$$(N, e_1) \hookrightarrow X_1' \ \wedge \ (N, e_1) \hookrightarrow X_2' \ \wedge \ a_1 = X_1'.\mathsf{rval}(e_1) \ \wedge \ a_2 = X_2'.\mathsf{rval}(e_1).$$

We can ensure that we have $X_1', X_2' \in \mathsf{CExec}$; then $X_1' \approx_{\mathsf{rval}} X_2'$. By Theorem 19 we get $X_1' = X_2'$, so that $a_1 = a_2$, as required. □

D Proof of Theorem 10

In this section we prove the following statement, which implies Theorem 10.

THEOREM 21 *If $P \longrightarrow P'$, then $[\![P']\!]_{\mathsf{CG}} = [\![P]\!]_{\mathsf{CG}}$.*

The case of the first reduction out of the two given in §6 is standard. We therefore only consider the case of the second, discharged by Theorem 42 in §D (the $\supseteq$ direction, i.e., soundness) and Theorem 47 in §D.5 (the $\subseteq$ direction, i.e., completeness). To prove the former result, we first need to reformulate the data type denotation from Definition 9, which is the subject of §D.3.

D.1 Properties of the factoring operation $(-/\sim)$

We prove a few useful properties of the factoring operation $(-/\sim)$, which will be used in the proofs of our main results later. Let E be a set of events and $\sim$ an equivalence relation on E.

LEMMA 22 *For all relations R on E and events $e, e_0, f, f_0 \in E$,*

$$(e \not\sim f \ \wedge \ e \sim e_0 \xrightarrow{R/\sim} f_0 \sim f) \implies e \xrightarrow{R/\sim} f.$$

PROOF. Let R, e, e_0, f, f_0 be a relation and events satisfying the assumption of the lemma. If $e_0 \xrightarrow{R} f_0$, by the definition of the factoring operation,

$$e \xrightarrow{R/\sim} f.$$

Otherwise, there exist $e_1, f_1 \in E$ such that

$$(e_0 \not\sim f_0 \quad \wedge \quad e_0 \sim e_1 \xrightarrow{R} f_1 \sim f_0).$$

Since $\sim$ is transitive,

$$e \sim e_1 \quad \wedge \quad f_1 \sim f.$$

Furthermore, $e \not\sim f$ and $e_1 \xrightarrow{R} f_1$. Hence,

$$e \xrightarrow{R/\sim} f,$$

as desired. □

LEMMA 23 *The factoring operation $(-/\sim)$ is a monotone closure operator. That is, for all relations R, S on E,*

$$R \subseteq (R/\sim) \quad \wedge \quad ((R/\sim)/\sim) = (R/\sim) \quad \wedge \quad (R \subseteq S \implies (R/\sim) \subseteq (S/\sim)).$$

PROOF. The first conjunct $R \subseteq (R/\sim)$ is an immediate consequence of the definition of factoring. For the second, by the same consequence again, it suffices to prove that

$$((R/\sim)/\sim) \subseteq (R/\sim).$$

For the sake of contradiction, suppose that $((R/\sim)/\sim) \not\subseteq (R/\sim)$. Then, there exist e, e', f, f' such that

$$e \not\sim f \quad \wedge \quad e \sim e' \xrightarrow{R/\sim} f' \sim f \quad \wedge \quad (e, f) \notin (R/\sim).$$

The first two conjuncts imply $e \xrightarrow{R/\sim} f$ by Lemma 22. This contradicts the third conjunct.

It remains to show the monotonicity of the factoring operation. Assume that $R \subseteq S$. Pick $e, f \in E$ such that $e \xrightarrow{R/\sim} f$. If $e \xrightarrow{R} f$, then

$$e \xrightarrow{S/\sim} f.$$

Otherwise, there exist $e_0, f_0 \in E$ such that

$$e \not\sim f \quad \wedge \quad e \sim e_0 \xrightarrow{R} f_0 \sim f.$$

Since $R \subseteq S$, the above formula implies $e_0 \xrightarrow{S} f_0$. Hence, by the formula again, we have that $e \xrightarrow{S/\sim} f$. □

LEMMA 24 *The factoring operation preserves union: for all relations R, S on E,*

$$(R \cup S)/(\sim) = (R/\sim) \cup (S/\sim).$$

PROOF. By the monotonicity of the factoring operation (Lemma 23), we have that

$$(R/\sim) \cup (S/\sim) \subseteq (R \cup S)/(\sim).$$

Hence, it suffices to prove the other subset relationship. Pick $e, f \in E$ such that

$$e \xrightarrow{(R \cup S)/\sim} f.$$

If $e \xrightarrow{R \cup S} f$, by the definition of factoring,

$$e \xrightarrow{(R/\sim) \cup (S/\sim)} f.$$

Otherwise, there exist $e_0, f_0 \in E$ such that

$$e \not\sim f \quad \wedge \quad e \sim e_0 \xrightarrow{R \cup S} f_0 \sim f.$$

Then, $e_0 \xrightarrow{R} f_0$ or $e_0 \xrightarrow{S} f_0$. In the former case, we have

$$(e, f) \in (R/\sim) \subseteq (R/\sim) \cup (S/\sim).$$

Similarly, in the latter case, we have

$$(e, f) \in (S/\sim) \subseteq (R/\sim) \cup (S/\sim).$$

□

LEMMA 25 *For all relations R on E,*

$$(R/\sim)^+/(\sim) = (R/\sim)^+.$$

This means that the transitive closure is well-defined over the set of $\sim$-factored relations.

PROOF. It suffices to show that for all events $e, e_0, e_1, \ldots, e_n, f$ with $n > 0$,

$$(e \not\sim f \wedge e \sim e_0 \wedge e_n \sim f \wedge \forall i \in \{0, \ldots, n{-}1\}.\ e_i \xrightarrow{R/\sim} e_{i+1}) \implies e \xrightarrow{(R/\sim)^+} f.$$

Let $e, e_0, e_1, \ldots, e_n, f$ be the events satisfying the assumption of the above implication. Then,

$$e \not\sim e_n. \tag{17}$$

This is because otherwise $e \sim f$ by the transitivity of $\sim$, but this would contradict the assumption that $e \not\sim f$. Furthermore, $e \sim e_0$ by assumption. Hence, there exists $m \in \{0, \ldots, n{-}1\}$ such that

$$e \sim e_m \quad \wedge \quad e \not\sim e_{m+1}.$$

By Lemma 22,

$$e \xrightarrow{R/\sim} e_{m+1}. \tag{18}$$

Furthermore, by a similar argument now applied to f and e_m (instead of e and e_n), we get $k \in \{m+1, \ldots, n\}$ such that

$$e_k \sim f \quad \wedge \quad e_{k-1} \not\sim f.$$

By Lemma 22 again,

$$e_{k-1} \xrightarrow{R/\sim} f. \tag{19}$$

If $k - 1 = m$, then $e \sim e_{k-1}$. Since $e \not\sim f$, the relationship in (19) implies the desired

$$e \xrightarrow{R/\sim} f,$$

because of Lemma 22. Assume now that $k - 1 \neq m$. Then, $k - 1 \geq m + 1$. From (18) and (19), it follows that

$$e \xrightarrow{R/\sim} e_{m+1} \xrightarrow{(R/\sim)^*} e_{k-1} \xrightarrow{R/\sim} f.$$

This implies the desired $e \xrightarrow{(R/\sim)^+} f$. $\square$

Finally, we show a few properties that describe interactions between factoring and a function on events. Let E_0, E_1 be sets of events, and $\sim_0, \sim_1$ equivalence relations on E_0 and E_1, respectively. Also, consider a function $\beta : E_0 \to E_1$ such that

$$\forall e, f \in E_0.\ e \sim_0 f \iff \beta(e) \sim_1 \beta(f).$$

The following lemmas hold for these $E_0, E_1, \sim_0, \sim_1$ and β.

LEMMA 26 *For all relations R_0 on E_0, and R_1 on E_1, if*

$$R_0/\sim_0 = R_0 \quad \wedge \quad R_1/\sim_1 = R_1 \quad \wedge \quad R_1^+ = R_1,$$

then for all $e, f \in E_0$ such that $e \not\sim_0 f$,

$$(\beta(e) \xrightarrow{((\beta(R_0)/\sim_1)\cup R_1)^+} \beta(f)) \iff (e \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} f).$$

PROOF. First, we prove the right-to-left implication. Consider $e, f \in E_0$ such that

$$e \xrightarrow{R_0 \cup (\beta^{-1}(R_1)/\sim_0)} f.$$

We will prove that

$$\beta(e) \xrightarrow{(\beta(R_0)/\sim_1)\cup R_1} \beta(f). \tag{20}$$

Note that from this follows the right-to-left implication. By the choice of e and f,

$$e \xrightarrow{R_0} f \quad \vee \quad e \xrightarrow{\beta^{-1}(R_1)/\sim_0} f.$$

Since $\beta(R_0) \subseteq (\beta(R_0)/\sim_1)$, the first disjunct above implies the desired (20). Now suppose that the second disjunct holds. Note that $(\beta \circ \beta^{-1})(R_1) \subseteq R_1$. Hence, if

$$e \xrightarrow{\beta^{-1}(R_1)} f,$$

we get

$$\beta(e) \xrightarrow{R_1} \beta(f).$$

This gives the desired relationship in (20). On the other hand, if for some $e', f' \in E_0$,

$$e \sim_0 e' \xrightarrow{\beta^{-1}(R_1)} f' \sim_0 f \quad \wedge \quad e \not\sim_0 f,$$

then

$$\beta(e) \sim_1 \beta(e') \xrightarrow{(\beta\circ\beta^{-1})(R_1)} \beta(f') \sim_1 \beta(f) \quad \wedge \quad \beta(e) \not\sim_1 \beta(f),$$

because β preserves $\sim_0$ and reflects $\sim_1$. Since $(\beta \circ \beta^{-1})(R_1) \subseteq R_1$, the above relationship implies that

$$\beta(e) \xrightarrow{R_1/\sim_1} \beta(f).$$

Recall that $R_1/{\sim_1} = R_1$ by the assumption of the lemma. Hence, this relationship between $\beta(e)$ and $\beta(f)$ gives the desired (20).

Next, we prove the left-to-right implication claimed in the lemma. Consider $e, f \in E_0$ such that

$$\beta(e) \xrightarrow{((\beta(R_0)/{\sim_1})\cup R_1)^+} \beta(f).$$

We have to show that

$$e \xrightarrow{(R_0\cup(\beta^{-1}(R_1)/{\sim_0}))^+} f. \tag{21}$$

Let $R'_0 = \beta(R_0)/{\sim_1}$. By our choice of e and f, there exist a sequence $(e_0, e_1, \ldots, e_n)$ in E_1 with $n \geq 1$ such that

$$\beta(e) = e_0 \quad \wedge \quad \beta(f) = e_n \quad \wedge \quad \forall 0 \leq i < n.\ (e_i \xrightarrow{R'_0} e_{i+1} \vee e_i \xrightarrow{R_1} e_{i+1}).$$

We will show (21) by induction on the number of times that the first disjunct involving R'_0 holds.

The base case is that it never holds for any element in the sequence $(e_0, \ldots, e_n)$. In this case, we have

$$\beta(e) = e_0 \xrightarrow{R_1^+} e_n = \beta(f).$$

But R_1 is transitive by assumption. Hence, $\beta(e) \xrightarrow{R_1} \beta(f)$, which then gives the desired (21).

Now let us consider the inductive case. In this case, there exists $0 \leq j < n$ such that $e_j \xrightarrow{R'_0} e_{j+1}$. Unpacking this, we get $e_j \xrightarrow{\beta(R_0)/{\sim_1}} e_{j+1}$, which is equivalent to the following:

$$\begin{array}{l}\exists e'_j, e'_{j+1} \in E_1.\\ \quad \big(e_j \sim_1 e'_j \xrightarrow{\beta(R_0)} e'_{j+1} \sim_1 e_{j+1} \wedge \big((e_j = e'_j \wedge e'_{j+1} = e_{j+1}) \vee (e_j \not\sim_1 e_{j+1})\big)\big).\end{array} \tag{22}$$

The first conjunct above implies that

$$\exists e''_j, e''_{j+1} \in E_0.\ \beta(e''_j) = e'_j \wedge \beta(e''_{j+1}) = e'_{j+1} \wedge e''_j \xrightarrow{R_0} e''_{j+1}.$$

Let

$$\begin{array}{l} k = \min\{i \mid e_i \sim_1 \beta(e''_j) \wedge 0 \leq i \leq j\} \quad \text{and}\\ l = \max\{i \mid \beta(e''_{j+1}) \sim_1 e_i \wedge j+1 \leq i \leq n\}.\end{array}$$

Because of Lemmas 23 and 24,

$$(R'_0 \cup R_1)/(\sim_1) = ((\beta(R_0)/{\sim_1})/{\sim_1}) \cup (R_1/{\sim_1}) = (\beta(R_0)/{\sim_1}) \cup R_1 = (R'_0 \cup R_1).$$

Hence,

$$\begin{array}{l}((k > 0 \wedge e_{k-1} \xrightarrow{R'_0\cup R_1} \beta(e''_j)) \vee (k = 0 \wedge \beta(e) \sim_1 \beta(e''_j)))\\ \quad \wedge \quad ((l < n \wedge \beta(e''_{j+1}) \xrightarrow{R'_0\cup R_1} e_{l+1}) \vee (l = n \wedge \beta(e''_{j+1}) \sim_1 \beta(f)))\end{array}$$

Since β reflects $\sim_1$, the above formula implies

$$((\beta(e) \xrightarrow{(R_0' \cup R_1)^+} \beta(e_j'') \wedge e \not\sim_0 e_j'') \ \vee \ (e \sim_0 e_j'')) \\ \wedge \quad ((\beta(e_{j+1}'') \xrightarrow{(R_0' \cup R_1)^+} \beta(f) \wedge e_{j+1}'' \not\sim_0 f) \ \vee \ (e_{j+1}'' \sim_0 f)).$$

By the induction hypothesis, the above formula implies that

$$((e \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} e_j'') \ \vee \ (e \sim_0 e_j'')) \\ \wedge \quad ((e_{j+1}'' \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} f) \ \vee \ (e_{j+1}'' \sim_0 f)).$$

We do the case analysis depending on which disjunct holds in the two conjuncts above.

1. **Subcase** $e \sim_0 e_j''$ **and** $e_{j+1}'' \sim_0 f$**:** Since $e_j'' \xrightarrow{R_0} e_{j+1}''$ and $e \not\sim_0 f$ by assumption,

   $$e \xrightarrow{R_0/\sim_0} f.$$

   Since R_0 is $\sim_0$-factored, this gives the desired (21).
2. **Subcase** $e \sim_0 e_j''$ **and** $e_{j+1}'' \not\sim_0 f$**:** Then,

   $$e_{j+1}'' \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} f. \tag{23}$$

   We consider two cases depending on whether $e \not\sim_0 e_{j+1}''$. Suppose that $e \not\sim_0 e_{j+1}''$. Then, since $e_j'' \xrightarrow{R_0} e_{j+1}''$,

   $$e \xrightarrow{R_0/\sim_0} e_{j+1}''.$$

   Since R_0 is $\sim_0$-factored (i.e., $R_0/\sim_0 = R_0$), this and the relationship in (23) imply that

   $$e \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} f.$$

   Now suppose that $e \sim_0 e_{j+1}''$. Then, since $e \not\sim_0 f$ by assumption and $(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+$ is $\sim_0$-factored, the relationship in (23) implies that

   $$e \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} f.$$

3. **Subcase** $e \not\sim_0 e_j''$ **and** $e_{j+1}'' \sim_0 f$**:** This subcase is symmetric to the previous one.
4. **Subcase** $e \not\sim_0 e_j''$ **and** $e_{j+1}'' \not\sim_0 f$**:** Then,

   $$e \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} e_j'' \quad \wedge \quad e_{j+1}'' \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} f.$$

   Since $e_j'' \xrightarrow{R_0} e_{j+1}''$, we get the desired

   $$e \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} f.$$

This concludes the proof of this lemma. □

LEMMA 27 *For all relations R_0 on E_0, and R_1 on E_1, if*

$$\begin{array}{lll} R_0/{\sim_0} = R_0 & \wedge & R_1/{\sim_1} = R_1 \quad \wedge \quad R_1^+ = R_1 \\ & \wedge & \textit{both } R_1 \textit{ and } (R_0 \cup (\beta^{-1}(R_1)/{\sim_0})) \textit{ are acyclic} \\ & \wedge & \beta(R_0 \cap {\sim_0}) - \mathsf{id} \subseteq R_1 \end{array}$$

then $(((\beta(R_0) - \mathsf{id})/{\sim_1}) \cup R_1)$ *is acyclic.*

PROOF. Let $R_0' = (\beta(R_0) - \mathsf{id})/{\sim_1}$. For the sake of contradiction, suppose that $(R_0' \cup R_1)$ is cyclic. Then, there exist

$$e_0, \ldots, e_n \in E_1 \text{ for some } n \geq 1$$

such that

$$(\forall 0 \leq i < n.\ e_i \xrightarrow{R_0' \cup R_1} e_{i+1}) \quad \wedge \quad e_n \xrightarrow{R_0' \cup R_1} e_0.$$

We prove that the existence of such a cycle leads to contradiction, using induction on the number of times that R_0' is used in the cycle.

The base case is that R_0' is not used at all. In this case, the sequence $(e_0, \ldots, e_n)$ demonstrates that R_1 is cyclic. This contradicts our acyclicity assumption on R_1.

Let us move on to the inductive case. Suppose that R_0' is used $m > 0$ times in the cycle $(e_0, \ldots, e_n)$. Then, there exists $0 \leq k \leq n$ such that

$$e_k \xrightarrow{R_0'} e_{(k+1) \bmod (n+1)}.$$

By the definition of R_0', there exist $e, f \in E_0$ such that

$$\begin{array}{l} \beta(e) \neq \beta(f) \quad \wedge \quad e \xrightarrow{R_0} f \quad \wedge \\ ((\beta(e) = e_k \wedge \beta(f) = e_{(k+1) \bmod (n+1)}) \vee \\ \ (e_k \sim_1 \beta(e) \wedge \beta(f) \sim_1 e_{(k+1) \bmod (n+1)} \wedge e_k \not\sim_1 e_{(k+1) \bmod (n+1)})). \end{array}$$

The second disjunct in the last conjunct implies that

$$\beta(e) \not\sim_1 \beta(f)$$

Hence, regardless of whether the first or second disjunct holds in the last conjunct, we have that

$$(\beta(f) \xrightarrow{(R_0' \cup R_1)^+/{\sim_1}} \beta(e)). \tag{24}$$

We do the case analysis on whether $e \sim_0 f$. Suppose that $e \sim_0 f$. Then,

$$\beta(e) \xrightarrow{\beta(R_0 \cap {\sim_0}) - \mathsf{id}} \beta(f).$$

Thus, by the assumption that $\beta(R_0 \cap {\sim_0}) - \mathsf{id} \subseteq R_1$,

$$\beta(e) \xrightarrow{R_1} \beta(f).$$

This means that the cycle $(e_0, \ldots, e_n)$ can be formed by using R_0' in $m-1$ times as well. This lets us use the induction hypothesis and obtain contradiction.

Suppose now that $e \not\sim_0 f$. Since $\sim_0$ is symmetric, $f \not\sim_0 e$. Using this fact, we start from (24) and reason as follows:

$$\begin{aligned}(\beta(f) \xrightarrow{(R_0' \cup R_1)^+/\sim_1} \beta(e)) &\implies (\beta(f) \xrightarrow{(R_0' \cup R_1)^+} \beta(e)) \\ &\implies (\beta(f) \xrightarrow{(\beta(R_0)/\sim_1 \cup R_1)^+} \beta(e)) \\ &\implies (f \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} e).\end{aligned}$$

The first implication holds because all of R_0', R_1 and $R_0' \cup R_1$ are $\sim_1$-factored (Lemmas 23 and 24), and the set of such $\sim_1$-factored relations is closed under the transitive closure operation (Lemma 25). The second implication unrolls the definition of R_0' and uses the monotonicity of $\lambda R'.\ (R'/\sim_1 \cup R_1)^+$ (Lemma 23). The last implication holds because of Lemma 26. Since

$$e \xrightarrow{R_0} f,$$

we have that

$$e \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} e,$$

which contradicts the acyclicity of $(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+$. □

We call a relation R (not necessarily a strict partial order) ***prefix-finite*** if for every e, $\{f \mid (f, e) \in R^+\}$ is finite.

LEMMA 28 *For all relations R, S on a set E, if R is prefix-finite but $(R \cup S)$ is not, there exists $e \in E$ such that*

$$\{f \mid f \xrightarrow{S \cup (S;(R \cup S)^*;S)} e\}$$

is infinite. Here $(-;-)$ is the usual operation for relational composition.

PROOF. Consider relations R, S on a set E such that R is prefix-finite but $(R \cup S)$ is not. Let

$$R_0 = (R \cup S).$$

Since R_0 is not prefix-finite, there exists $e \in E$ such that

$$E' = \{f \mid f \xrightarrow{R_0^+} e\}$$

is infinite. Since R is prefix-finite, there should be infinitely-many elements of E' that are not related to e via R^+. This means that the following subset of E' is infinite:

$$E'' = \{f \mid \exists e_f \in E.\ f \xrightarrow{R_0^*;S} e_f \xrightarrow{R^*} e\}.$$

Pick a witness e_f for each f in E'' that satisfies the condition in the definition of E''. Because of the prefix-finiteness of R, the set of these chosen witnesses e_f has to be

finite. This means that one witness e_f is reused infinitely-many times so that it is related to infinitely-many different f's in the set E'' above. Let us denote this witness by e_w. By the reasoning just given, the set

$$E''' = \{f \mid f \xrightarrow{R_0^*;S} e_w\}.$$

is infinite. Now notice that E''' satisfies the following equality:

$$E''' = \{f \mid \exists e'_f \in E.\ f \xrightarrow{R^*} e'_f \xrightarrow{S \cup (S;R_0^*;S)} e_w\}.$$

Choose a witness e'_f for each $f \in E'''$ that satisfies the condition in the definition of E''' above. This time the set of chosen witnesses e'_f is infinite. This is because otherwise there would exist one witness e'_f that is related to infinitely-many elements in E''' by R^*, but the existence of such a witness contradicts the prefix-finiteness of R. Since there are infinitely-many witnesses e'_f, the following set is also infinite:

$$\{e'_f \mid e'_f \xrightarrow{S \cup (S;R_0^*;S)} e_w\},$$

as claimed by the lemma. □

LEMMA 29 *For all relations R_0 on E_0, and R_1 on E_1, if*

$$\begin{array}{lll} R_0/{\sim_0} = R_0 & \wedge & R_1/{\sim_1} = R_1 \quad \wedge \quad R_1^+ = R_1 \\ & \wedge & \textit{both } R_1 \textit{ and } (R_0 \cup (\beta^{-1}(R_1)/{\sim_0})) \textit{ are prefix-finite} \\ & \wedge & \textit{every equivalence class of } \sim_1 \textit{ is finite} \end{array}$$

then $((\beta(R_0)/{\sim_1}) \cup R_1)$ is prefix-finite.

PROOF. Let $R'_0 = \beta(R_0)/{\sim_1}$. For the sake of contradiction, suppose that $(R'_0 \cup R_1)$ is not prefix-finite. Note that R_1 is prefix-finite by assumption. By Lemma 28, there exists $e \in E_1$ such that

$$E'_1 = \{f \mid f \xrightarrow{R'_0 \cup (R'_0;(R'_0 \cup R_1)^*;R'_0)} e\}$$

is infinite. Since every equivalence class of $\sim_1$ is finite, the following subset of E'_1 is also infinite:

$$E''_1 = \{f \mid f \xrightarrow{R'_0 \cup (R'_0;(R'_0 \cup R_1)^*;R'_0)} e \wedge f \not\sim_1 e\}.$$

Now we can pick countably many elements $f_1, \ldots, f_n, \ldots$ from E''_1 such that

$$\forall i, j \geq 1.\ i \neq j \implies f_i \not\sim_1 f_j,$$

because, again, every equivalence class of $\sim_1$ is finite. Since f_i is in the domain of the relation R'_0 and $R'_0 = \beta(R_0)/{\sim_1}$, there exist $f'_1, \ldots, f'_n, \ldots \in E_0$ such that

$$(\forall i, j \geq 1.\ i \neq j \implies f'_i \neq f'_j) \quad \wedge \quad (\forall i \geq 1.\ \beta(f'_i) \sim_1 f_i).$$

Also, since e is in the range of the relation R_0', there exists $e' \in E_0$ such that

$$e \sim_1 \beta(e').$$

Recall that $f_i \not\sim_1 e$ for every $i \geq 1$, and that β preserves $\sim_0$. Hence, for every $i \geq 1$,

$$\beta(f_i') \not\sim_1 \beta(e') \quad \wedge \quad f_i' \not\sim_0 e'. \tag{25}$$

Also, for every i,

$$f_i \xrightarrow{R_0' \cup (R_0';(R_0' \cup R_1)^*;R_0')} e,$$

so that

$$f_i \xrightarrow{(R_0' \cup R_1)^+} e.$$

Since $(R_0' \cup R_1)^+$ is $\sim_1$-factored (Lemmas 23, 24 and 25), the above relationship and the property in (25) imply that

$$\beta(f_i') \xrightarrow{(R_0' \cup R_1)^+} \beta(e') \text{ for every } i \geq 1.$$

Because of (25), we can apply Lemma 26 here and derive

$$f_i' \xrightarrow{(R_0 \cup (\beta^{-1}(R_1)/\sim_0))^+} e' \text{ for every } i \geq 1.$$

This contradicts the prefix-finiteness of $(R_0 \cup (\beta^{-1}(R_1)/\sim_0))$, the assumption of this lemma. □

D.2 Generalised axioms and lifting operation

For $X \in \mathsf{Exec}$ and $R \subseteq (X.E)^2$ we write $(X, R) \models_{\mathsf{CCS}} \mathbb{F}$ if $X \models \mathbb{F}$ and X satisfies the following version of the axioms CAUSALVIS and CAUSALAR, and the additional axiom PREFIXFINITEAR′:

CAUSALVIS′. $((X.\mathsf{so} \cup X.\mathsf{vis} \cup R)/(X.\sim))^+ \cap \mathsf{sameobj}(X) \subseteq X.\mathsf{vis}$.
CAUSALAR′. $(X.\mathsf{so} \cup X.\mathsf{ar} \cup R)/(X.\sim)$ is acyclic.
PREFIXFINITEAR′. $(X.\mathsf{so} \cup X.\mathsf{ar} \cup R)/(X.\sim)$ is prefix-finite.

We write $X \models_{\mathsf{CCS}} \mathbb{F}$ if $X \models \mathbb{F}$ and X satisfies CAUSALVIS and CAUSALAR (but not necessarily EVENTUAL).

For

$$X \in \mathsf{Exec}, \qquad \omega \in \mathsf{Obj}, \qquad H' \in \mathsf{Hist}(\omega), \qquad R \subseteq (H'.E)^2, \qquad \beta : X.E \to H'.E$$

we define

$$\mathsf{lift}(X, H', R, \beta) = (H', \ ((H'.\mathsf{so} \cup (\beta(X.\mathsf{vis}) - \mathsf{id}) \cup R)/(H'.\sim))^+, \\ ((H'.\mathsf{so} \cup (\beta(X.\mathsf{ar}) - \mathsf{id}) \cup R)/(H'.\sim))^+).$$

LEMMA 30 *For all*

$$X \in \mathsf{Exec}, \quad \omega \in \mathsf{Obj}, \quad H' \in \mathsf{Hist}(\omega), \quad R \subseteq (H'.E)^2, \quad (\beta, \rho) : X.H \to H',$$

if $R' = H'.\mathsf{so} \cup (R/(H'.\sim))$ *is acyclic and prefix-finite, and* $(X, \beta^{-1}(R'^+))$ *satisfies* CAUSALVIS′, CAUSALAR′ *and* PREFIXFINITEAR′, *then* $\mathsf{lift}(X, H', R, \beta)$ *is an execution satisfying* CAUSALVIS *and* CAUSALAR.

PROOF. Let

$$\begin{aligned}
&X = (H, \mathsf{vis}, \mathsf{ar}) = ((E, \mathsf{label}, \mathsf{so}, \sim), \mathsf{vis}, \mathsf{ar});\\
&H' = (E', \mathsf{label}', \mathsf{so}', \sim');\\
&\mathsf{vis}' = ((\mathsf{so}' \cup (\beta(\mathsf{vis}) - \mathsf{id}) \cup R)/\sim')^+;\\
&\mathsf{ar}' = ((\mathsf{so}' \cup (\beta(\mathsf{ar}) - \mathsf{id}) \cup R)/\sim')^+.
\end{aligned}$$

Recall that β preserves $\sim$ and reflects $\sim'$ by Lemma 14. We will use this fact without mentioning it explicitly in this proof.

We need to discharge the following four requirements:

1. $\mathsf{vis}' \subseteq \mathsf{ar}'$;
2. vis' and ar' are defined on the same object (in this case ω);
3. vis' and ar' are prefix-finite strict partial orders on E'; and
4. $(H', \mathsf{vis}', \mathsf{ar}')$ satisfies CAUSALVIS and CAUSALAR.

The first requirement holds because $\mathsf{vis} \subseteq \mathsf{ar}$, and vis' and ar' are obtained by applying the following monotone operation on those vis and ar:

$$\lambda R_0.\ ((\mathsf{so}' \cup (\beta(R_0) - \mathsf{id}) \cup R)/\sim')^+.$$

This operation is monotone because it uses only monotone operators; for the monotonicity of $(-/\sim')$, see Lemma 23. The second requirement is an immediate consequence of the assumption that $H' \in \mathsf{Hist}(\omega)$. Since vis' and ar' are both transitive by definition and ar' includes vis', the third requirement follows if we show that ar' is acyclic and prefix-finite. We will prove the acyclicity of ar' later when we discharge the last requirement, in particular, the CAUSALAR axiom. For the prefix-finiteness of ar', we first notice that

$$\begin{aligned}
\mathsf{ar}' &= ((\mathsf{so}' \cup (\beta(\mathsf{ar}) - \mathsf{id}) \cup R)/\sim')^+\\
&= ((\mathsf{so}'/\sim') \cup ((\beta(\mathsf{ar}) - \mathsf{id})/\sim') \cup (R/\sim'))^+\\
&= (\mathsf{so}' \cup ((\beta(\mathsf{ar}) - \mathsf{id})/\sim') \cup (R/\sim'))^+\\
&= (((\beta(\mathsf{ar}) - \mathsf{id})/\sim') \cup (\mathsf{so}' \cup (R/\sim')))^+\\
&= (((\beta(\mathsf{ar}) - \mathsf{id})/\sim') \cup R')^+\\
&\subseteq ((\beta(\mathsf{ar}/\sim)/\sim') \cup R'^+)^+.
\end{aligned}$$

The second equality holds because the factoring distributes over union (Lemma 24), and the third follows from the fact that so' is $\sim'$-factored. The fifth equality just rolls the definition of R', and the last subset relationship uses the monotonicity of factoring (Lemma 23). Note that since R' is $\sim'$-factored, so is R'^+ (Lemma 25). Furthermore, $\sim'$ is the equivalence relation from the history H', all of its equivalence classes are finite sets. Thus, by Lemma 29, it suffices to prove the prefix-finiteness of

$$R'^+ \quad \text{and} \quad (\mathsf{ar}/\sim) \cup (\beta^{-1}(R'^+)/\sim).$$

By assumption, R' is prefix-finite, so R'^+ is prefix-finite as well. For the latter relation, we have

$$(\mathsf{ar}/\sim) \cup (\beta^{-1}(R'^+)/\sim) \;=\; (\mathsf{ar} \cup \beta^{-1}(R'^+))/\sim$$

since the factoring operation preserves union (Lemma 24). The RHS of the equation above is prefix-finite because $(X, \beta^{-1}(R'^{+}))$ satisfies the PREFIXFINITEAR$'$ axiom.

The rest of the proof focuses on showing the fourth requirement.

CAUSALVIS**.** We prove the axiom as follows:

$$\begin{aligned}
((\mathsf{so}' \cup \mathsf{vis}')/\sim')^{+} &= ((\mathsf{so}'/\sim') \cup (\mathsf{vis}'/\sim'))^{+} \\
&= ((\mathsf{so}'/\sim') \cup (((\mathsf{so}' \cup (\beta(\mathsf{vis}) - \mathsf{id}) \cup R)/\sim')^{+}/\sim'))^{+} \\
&= ((\mathsf{so}'/\sim') \cup ((\mathsf{so}' \cup (\beta(\mathsf{vis}) - \mathsf{id}) \cup R)/\sim')^{+})^{+} \\
&= ((\mathsf{so}'/\sim') \cup ((\mathsf{so}' \cup (\beta(\mathsf{vis}) - \mathsf{id}) \cup R)/\sim'))^{+} \\
&= ((\mathsf{so}' \cup (\beta(\mathsf{vis}) - \mathsf{id}) \cup R)/\sim')^{+} \\
&= \mathsf{vis}'.
\end{aligned}$$

The first equality holds because factoring distributes over union (Lemma 24), and the second simply unrolls the definition of vis'. The third equality uses the fact that the set of factored relations is closed under transitive closure (Lemma 25), and the next equality is a simple fact on transitive closure. The fifth equality uses the distributivity of factoring over union (Lemma 24). The last equality is just the rolling of the definition of vis'.

CAUSALAR**.** We need to prove the acyclicity of the following relation:

$$(\mathsf{so}' \cup \mathsf{ar}')/\sim' = (\mathsf{so}' \cup ((\mathsf{so}' \cup (\beta(\mathsf{ar}) - \mathsf{id}) \cup R)/\sim')^{+})/\sim'.$$

We simplify the RHS of this equation using properties of $\sim'$:

$$\begin{aligned}
&(\mathsf{so}' \cup ((\mathsf{so}' \cup (\beta(\mathsf{ar}) - \mathsf{id}) \cup R)/\sim')^{+})/\sim' \\
&= (\mathsf{so}'/\sim') \cup (((\mathsf{so}' \cup (\beta(\mathsf{ar}) - \mathsf{id}) \cup R)/\sim')^{+}/\sim') \\
&= (\mathsf{so}'/\sim') \cup ((\mathsf{so}' \cup (\beta(\mathsf{ar}) - \mathsf{id}) \cup R)/\sim')^{+} \\
&= (\mathsf{so}'/\sim') \cup ((\mathsf{so}'/\sim') \cup ((\beta(\mathsf{ar}) - \mathsf{id})/\sim') \cup (R/\sim'))^{+} \\
&= ((\mathsf{so}'/\sim') \cup ((\beta(\mathsf{ar}) - \mathsf{id})/\sim') \cup (R/\sim'))^{+} \\
&= (\mathsf{so}' \cup ((\beta(\mathsf{ar}) - \mathsf{id})/\sim') \cup (R/\sim'))^{+} \\
&= (((\beta(\mathsf{ar}) - \mathsf{id})/\sim') \cup R')^{+} \\
&= (((\beta(\mathsf{ar}) - \mathsf{id})/\sim') \cup (R'^{+}))^{+} \\
&= (((\beta(\mathsf{ar}/\sim) - \mathsf{id})/\sim') \cup (R'^{+}))^{+}.
\end{aligned}$$

The first equality uses the distributivity of factoring over union, the second follows from Lemma 25, and the third uses the same distributivity again. The fourth is an immediate consequence of the transitive closure operation, and the fifth equality holds because $\mathsf{so}'/\sim' = \mathsf{so}'$. The sixth equality rolls the definition of R'. The next equality is a simple consequence of transitive closure. The last equality holds because β preserves $\sim$ and reflects $\sim'$, so that

$$((\beta(\mathsf{ar}) - \mathsf{id})/\sim') = ((\beta(\mathsf{ar}/\sim) - \mathsf{id})/\sim').$$

By our simplification above, it is sufficient to prove the acyclicity of the following relation:

$$S = ((\beta(\mathsf{ar}/\sim) - \mathsf{id})/\sim' \cup (R'^{+})).$$

Note that

$$(R'^{+})/\sim' = (\mathsf{so}' \cup (R/\sim'))^{+}/\sim' = (\mathsf{so}' \cup (R/\sim'))^{+} = R'^{+}.$$

The second equality holds because $(\mathsf{so}' \cup (R/\sim'))$ is $\sim'$-factored (due to $\mathsf{so}'/\sim' = \mathsf{so}'$ and Lemmas 23 and 24) and the set of $\sim'$-factored relations is closed under the transitive closure operator (Lemma 25). Furthermore, $\mathsf{ar}/\sim$ is $\sim$-factored (Lemma 23). Hence, by Lemma 27, to show the acyclicity of S, it suffices to prove the facts below:

1. The relations R'^{+} and $((\mathsf{ar}/\sim) \cup (\beta^{-1}(R'^{+})/\sim))$ are acyclic.
2. $\beta((\mathsf{ar}/\sim) \cap \sim) - \mathsf{id} \subseteq R'^{+}$.

The acyclicity of R'^{+} is one of the assumptions of this lemma. The other relation $((\mathsf{ar}/\sim) \cup (\beta^{-1}(R'^{+})/\sim))$ is same as

$$(\mathsf{ar} \cup \beta^{-1}(R'^{+}))/\sim$$

by Lemma 24. Hence, its acyclicity follows from the fact that $(X, \beta^{-1}(R'^{+}))$ satisfies CAUSALAR$'$. For the second condition, consider $e, f \in X.E$ such that

$$\beta(e) \neq \beta(f) \quad \wedge \quad e \xrightarrow{(\mathsf{ar}/\sim)\cap\sim} f.$$

By the definition of factoring, the latter condition implies that $e \xrightarrow{\mathsf{ar}} f$ and $e \sim f$. Since $e \sim f$, we have

$$e \xrightarrow{\mathsf{so}} f \quad \vee \quad f \xrightarrow{\mathsf{so}} e.$$

The second disjunct here is not possible because it would contradict the fact that X satisfies the CAUSALAR axiom. Thus, the first disjunct holds, from which it follows that

$$\beta(e) \xrightarrow{\beta(\mathsf{so})} \beta(f).$$

Since $\beta(e) \neq \beta(f)$ and (β, ρ) is a morphism from $X.H$ to H', the above relationship implies

$$(\beta(e), \beta(f)) \in \mathsf{so}' \subseteq R'^{+},$$

as desired. □

LEMMA 31 *For all*

$$X \in \mathsf{Exec}, \quad \omega \in \mathsf{Obj}, \quad H' \in \mathsf{Hist}(\omega), \quad R \subseteq (H'.E)^2, \quad (\beta, \rho) : X.H \to H',$$

if $R' = H'.\mathsf{so} \cup (R/(H'.\sim))$ *is acyclic and prefix-finite, and* $(X, \beta^{-1}(R'^{+}))$ *satisfies* CAUSALVIS$'$ *and* CAUSALAR$'$*, then* (β, ρ) *is a morphism from* X *to* $\mathsf{lift}(X, H', R, \beta)$.

PROOF. Let $X' = \mathsf{lift}(X, H', R, \beta)$. By assumption, (β, ρ) is a morphism from $X.H$ to $X'.H$. Furthermore, X' is an execution by Lemma 30. Thus, it is sufficient to prove that

$$\begin{array}{c} \beta(X.\mathsf{vis}) - \mathsf{id} \subseteq X'.\mathsf{vis}, \qquad \beta^{-1}(X'.\mathsf{vis}) \cap \mathsf{sameobj}(X) \subseteq X.\mathsf{vis}, \\ \beta(X.\mathsf{ar}) - \mathsf{id} \subseteq X'.\mathsf{ar}. \end{array} \tag{26}$$

The first and the last requirements hold because by the definition of the lift operator, we have

$$\begin{array}{rcl} X'.\mathsf{vis} & = & ((H'.\mathsf{so} \cup (\beta(X.\mathsf{vis}) - \mathsf{id}) \cup R)/(H'.\sim))^+ \supseteq \beta(X.\mathsf{vis}) - \mathsf{id}, \\ X'.\mathsf{ar} & = & ((H'.\mathsf{so} \cup (\beta(X.\mathsf{ar}) - \mathsf{id}) \cup R)/(H'.\sim))^+ \supseteq \beta(X.\mathsf{ar}) - \mathsf{id}. \end{array}$$

The rest of the proof will focus on showing the second requirement in (26). During the proof, we will use the fact that β preserves $\sim$ and reflects $\sim'$ (Lemma 14), without mentioning it explicitly.

Let $R' = H'.\mathsf{so} \cup (R/H'.\sim)$. To discharge the requirement, we use one important assumption that $(X, \beta^{-1}(R'^+))$ satisfies CAUSALVIS$'$:

$$X.\mathsf{vis} \supseteq ((X.\mathsf{so} \cup X.\mathsf{vis} \cup \beta^{-1}(R'^+))/(X.\sim))^+ \cap \mathsf{sameobj}(X).$$

Hence, the requirement follows if we show that

$$((X.\mathsf{so} \cup X.\mathsf{vis} \cup \beta^{-1}(R'^+))/(X.\sim))^+ \supseteq \beta^{-1}(X'.\mathsf{vis}).$$

By the definition of X', this proof obligation is equivalent to

$$\begin{array}{l} ((X.\mathsf{so} \cup X.\mathsf{vis} \cup \beta^{-1}(R'^+))/(X.\sim))^+ \\ \qquad \supseteq \beta^{-1}(((H'.\mathsf{so} \cup (\beta(X.\mathsf{vis}) - \mathsf{id}) \cup R)/(H'.\sim))^+). \end{array} \tag{27}$$

But the RHS is the same as

$$\beta^{-1}(((H'.\mathsf{so}/H'.\sim) \cup ((\beta(X.\mathsf{vis}) - \mathsf{id})/H'.\sim) \cup (R/H'.\sim))^+)$$

because of the distributivity of the factoring $-/H'.\sim$ with respect to the union operator (Lemma 24). Since $H'.\mathsf{so}/H'.\sim = H'.\mathsf{so}$ and $R' = (H'.\mathsf{so} \cup (R/H'.\sim))$,

$$\begin{array}{l} \beta^{-1}((H'.\mathsf{so} \cup ((\beta(X.\mathsf{vis}) - \mathsf{id})/H'.\sim) \cup (R/H'.\sim))^+) \\ = \beta^{-1}((((\beta(X.\mathsf{vis}) - \mathsf{id})/H'.\sim) \cup R')^+) \\ = \beta^{-1}((((\beta(X.\mathsf{vis}) - \mathsf{id})/H'.\sim) \cup R'^+)^+). \end{array}$$

Meanwhile, the LHS of (27) is the same as

$$((X.\mathsf{so}/X.\sim) \cup (X.\mathsf{vis}/X.\sim) \cup (\beta^{-1}(R'^+)/X.\sim))^+,$$

because $(-/X.\sim)$ distributes over union (Lemma 24). Hence, our proof obligation can be further simplified to

$$\begin{array}{l} ((X.\mathsf{so}/X.\sim) \cup (X.\mathsf{vis}/X.\sim) \cup (\beta^{-1}(R'^+)/X.\sim))^+ \\ \qquad \supseteq \beta^{-1}((((\beta(X.\mathsf{vis}) - \mathsf{id})/H'.\sim) \cup R'^+)^+). \end{array} \tag{28}$$

Pick $e, f \in X.E$ such that

$$\beta(e) \xrightarrow{(((\beta(X.\mathsf{vis})-\mathsf{id})/H'.\sim)\cup R'^{+})^{+}} \beta(f). \tag{29}$$

We have to show that

$$e \xrightarrow{((X.\mathsf{so}/X.\sim)\cup(X.\mathsf{vis}/X.\sim)\cup(\beta^{-1}(R'^{+})/X.\sim))^{+}} f. \tag{30}$$

The rest of our proof uses case split on whether $e \xrightarrow{X.\sim} f$.

Assume that $e \xrightarrow{X.\sim} f$. Then,

$$e \xrightarrow{X.\mathsf{so}} f \quad \vee \quad f \xrightarrow{X.\mathsf{so}} e.$$

Note that the first disjunct implies the desired relationship in (30). The second disjunct also does, because it never holds. To see this, suppose that it did hold. Then, we would have

$$\beta(f) = \beta(e) \quad \vee \quad \beta(f) \xrightarrow{H'.\mathsf{so}} \beta(e).$$

Both disjuncts here imply the presence of cycle in

$$(((\beta(X.\mathsf{vis}) - \mathsf{id})/H'.\sim) \cup R'^{+})^{+} \;=\; (X'.\mathsf{vis}).$$

This contradicts the fact that X' is an execution (so its visibility relation is a strict partial order) (Lemma 30).

Now assume that $(e, f) \notin X.\sim$. We weaken the relationship in (29) slightly and derive:

$$\beta(e) \xrightarrow{((\beta(X.\mathsf{vis}/X.\sim)/H'.\sim)\cup R'^{+})^{+}} \beta(f) \tag{31}$$

Note that $(X.\mathsf{vis}/X.\sim)$ is $X.\sim$-factored (Lemma 23), and also that R'^{+} is $X'.\sim$-factored (Lemma 25). Hence, we can apply Lemma 26 to the relationship in (31), and obtain

$$e \xrightarrow{((X.\mathsf{vis}/X.\sim)\cup(\beta^{-1}(R'^{+})/X.\sim))^{+}} f,$$

which gives the desired relationship in (30). □

D.3 Reformulation of data type denotations

Recall that a relation R on a set E is a ***partial equivalence relation*** if it is an equivalence relation on a subset E_0 of E. This subset E_0 is called the domain of R, denoted $\mathsf{dom}(R)$.

For $X \in \mathsf{Exec}$ and a partial equivalence relation R on $X.E$ such that $R \subseteq X.\sim$, we let

$$\begin{aligned}&\mathsf{proj}(X, R) = \\ &((X.E|_{\mathsf{dom}(R)},\ X.\mathsf{label}|_{\mathsf{dom}(R)},\ X.\mathsf{so} \cap R,\ X.\sim \cap R),\ X.\mathsf{vis}|_{\mathsf{dom}(R)}, X.\mathsf{ar}|_{\mathsf{dom}(R)}).\end{aligned}$$

LEMMA 32 *For all $X \in \mathsf{Exec}$ and partial equivalence relations R on $X.E$ such that $R \subseteq X.\sim$ and $\mathsf{dom}(R)$ is finite, the projection $\mathsf{proj}(X, R)$ is also an execution.*

PROOF. Consider an execution $X \in \mathsf{Exec}$ and a partial equivalence relation R on $X.E$ such that $R \subseteq X.\sim$. Let

$$((E, \mathsf{label}, \mathsf{so}, \sim), \mathsf{vis}, \mathsf{ar}) = X, \qquad ((E', \mathsf{label}', \mathsf{so}', \sim'), \mathsf{vis}', \mathsf{ar}') = \mathsf{proj}(X, R).$$

First, we discharge the requirements on $\sim'$. Since $R \subseteq \sim$, we have that

$$R = (\sim \cap R) = \sim'.$$

Hence, $\sim'$ is an equivalence relation on $\mathsf{dom}(R)$, which is equal to $(E \cap \mathsf{dom}(R)) = E'$. Also, since $\sim'$ is included in $\sim$ and every equivalence class of $\sim$ is finite, equivalence classes of $\sim'$ are also all finite. Finally, for all $e, f \in E'$,

$$\begin{aligned}
(e \sim' f) &\implies ((e \sim f) \wedge (e \xrightarrow{R} f)) \\
&\implies (((e \xrightarrow{\mathsf{so}} f) \vee (f \xrightarrow{\mathsf{so}} e)) \wedge (e \xrightarrow{R} f)) \\
&\implies ((e \xrightarrow{\mathsf{so} \cap R} f) \vee (f \xrightarrow{\mathsf{so} \cap R} e)).
\end{aligned}$$

The last implication uses the symmetry of the partial equivalence relation R.

Second, we handle the requirements on so'. Since so' is obtained by restricting so, it inherits the prefix-finiteness of so. By the same reason, so' is irreflexive. It is also transitive, because it is the intersection of two transitive relations so and R. Thus, to prove that so' is a total order on finitely many disjoint subsets of E', it suffices to find a partition $\mathcal{E}$ of E' with finitely many components, such that two different elements in E' are related by so' in one way or the other if and only if they belong to the same part of this partition. Recall that so is the union of total orders on components of some partition $\mathcal{E}_E = \{E_j\}_{j \in J}$ of E. Let $\mathcal{E}_R$ be the collection of all equivalence classes of R, and define a new collection of subsets of $(E \cap \mathsf{dom}(R)) = E'$ as follows:

$$\mathcal{E} = \{E_j \cap E_R \mid E_j \in \mathcal{E}_E \wedge E_R \in \mathcal{E}_R\}.$$

Then, $\mathcal{E}$ forms a partition of E'. By construction, so' relates two different elements e and f from E' in one way or the other if and only if e and f belong to the same component of this partition $\mathcal{E}$. Also, since $\mathsf{dom}(R)$ is finite by assumption, $\mathcal{E}$ has only finitely many components. The remaining requirement on so' is that so' is $\sim'$-factored. This follows from the fact that no $e, e', f, f' \in E'$ satisfy

$$e \sim' e' \xrightarrow{\mathsf{so}'} f' \sim' f \quad \wedge \quad \neg(e \sim' f).$$

To see this impossibility, just notice that the above formula implies

$$e \xrightarrow{R} e' \xrightarrow{R} f' \xrightarrow{R} f \quad \wedge \quad \neg(e \xrightarrow{R} f),$$

which is impossible because R is transitive.

Third, we discharge the conditions on vis' and ar'. Both are the restrictions of vis and ar by the same relation $\mathsf{dom}(R) \times \mathsf{dom}(R)$, so they inherit the prefix-finiteness and irreflexivity of vis and ar, and relate events on the same objects only. Furthermore, by

the same reason and the assumption that $\mathsf{vis} \subseteq \mathsf{ar}$, we have that $\mathsf{vis}' \subseteq \mathsf{ar}'$. It remains to show the transitivity of vis' and ar'. This is easy because vis' and ar' both are the intersection of two transitive relations: vis and $E' \times E'$ in the case of vis'; ar and $E' \times E'$ in the case of ar'. □

The next lemma uses the notation of factoring

$$S/R$$

for partial equivalence relations R. The definition of S/R in this case is the same as that for the original factoring in §3.2.

LEMMA 33 *Let $X \in \mathsf{Exec}$ be an execution and R a partial equivalence relations on $X.E$ such that $R \subseteq X.\sim$.*

1. *If $X.\mathsf{so}$ is R-factored (i.e., $(X.\mathsf{so}/R) \subseteq X.\mathsf{so}$), and X satisfies* CAUSALVIS *and* CAUSALAR*, then $\mathsf{proj}(X, R)$ also satisfies* CAUSALVIS *and* CAUSALAR.
2. *If $\mathsf{dom}(R)$ is closed under taking the inverse image with respect to $X.\mathsf{vis}$, that is,*

$$\{e \mid \exists f \in \mathsf{dom}(R).\ e \xrightarrow{X.\mathsf{vis}} f\} \subseteq \mathsf{dom}(R),$$

then

$$\forall \mathbb{F}.\ (X \models \mathbb{F} \implies \mathsf{proj}(X, R) \models \mathbb{F}).$$

PROOF. Consider an execution $X \in \mathsf{Exec}$ and a partial equivalence relation R on $X.E$ such that $R \subseteq X.\sim$. Let

$$((E, \mathsf{label}, \mathsf{so}, \sim), \mathsf{vis}, \mathsf{ar}) = X, \qquad ((E', \mathsf{label}', \mathsf{so}', \sim'), \mathsf{vis}', \mathsf{ar}') = \mathsf{proj}(X, R).$$

We first prove the claim of the lemma regarding the CAUSALAR and CAUSALVIS axioms. Assume that $X.\mathsf{so}$ is R-factored and the execution X satisfies CAUSALAR and CAUSALVIS.

Note that in order to prove $\mathsf{proj}(X, R)$ satisfies CAUSALAR, we just need to show that

$$((\mathsf{ar}' \cup \mathsf{so}')/\sim') \ \subseteq\ ((\mathsf{ar} \cup \mathsf{so})/\sim), \tag{32}$$

because X already satisfies CAUSALAR and the RHS relation above is already acyclic. Let us simplify the proof obligation in (32) slightly by reasoning about its LHS as follows:

$$((\mathsf{ar}' \cup \mathsf{so}')/\sim') \ =\ ((\mathsf{ar}'/\sim') \cup (\mathsf{so}'/\sim')) \ =\ ((\mathsf{ar}'/\sim') \cup \mathsf{so}') \ \subseteq\ ((\mathsf{ar}'/\sim') \cup \mathsf{so}).$$

Here the first equality uses the distributivity of factoring over union (Lemma 24), the second uses the fact that so' is $\sim'$-factored, and the last subset relation holds because so' is defined as the intersection of so with R. Our transformation above implies that in order to prove (32), it suffices to show

$$(\mathsf{ar}'/\sim') \ \subseteq\ ((\mathsf{ar}/\sim) \cup \mathsf{so}).$$

Since $\mathsf{ar}' \subseteq \mathsf{ar}$ as well, we can further simplify the above subset relationship to the following property:

$$\forall e, e', f, f' \in \mathsf{dom}(R).\ ((e \sim' e' \xrightarrow{\mathsf{ar}'} f' \sim' f) \wedge \neg(e \sim' f)) \implies (e \xrightarrow{(\mathsf{ar}/\sim)\cup\mathsf{so}} f). \tag{33}$$

Pick e, e', f, f' that satisfy the assumption of the implication in (33). If $\neg(e \sim f)$, then $e \xrightarrow{\mathsf{ar}/\sim} f$, because $\sim' \subseteq \sim$ and $\mathsf{ar}' \subseteq \mathsf{ar}$. The desired conclusion in (33) follows from this. If $e \sim f$, we have

$$(e \xrightarrow{\mathsf{so}} f) \quad \vee \quad (f \xrightarrow{\mathsf{so}} e).$$

The first disjunct immediately implies the conclusion of (33). The second disjunct is, on the other hand, not possible. Suppose that it was. Since $e \not\sim' f$, we also have that $e' \not\sim' f'$. Furthermore, so is R-factored and $\sim'$ is the same as R. Hence, we should have that

$$f' \xrightarrow{\mathsf{so}} e'.$$

This means that $(\mathsf{so} \cup \mathsf{ar})$ is cyclic, which contradicts the assumption that X satisfies CAUSALAR.

Our proof that $\mathsf{proj}(X, R)$ satisfies CAUSALVIS has a similar structure as the proof that we have just given. In this case, we should show that

$$((\mathsf{vis}' \cup \mathsf{so}')/\sim')^+ \ \subseteq\ \mathsf{vis}'.$$

By definition, the LHS relation is defined over E', and the RHS relation is the same as $\mathsf{vis} \cap (E' \times E')$. Thus, the above subset relationship is equivalent to

$$((\mathsf{vis}' \cup \mathsf{so}')/\sim')^+ \ \subseteq\ \mathsf{vis}.$$

Meanwhile, since X satisfies CAUSALVIS, we have that

$$((\mathsf{vis} \cup \mathsf{so})/\sim)^+ \ \subseteq\ \mathsf{vis}.$$

Hence, it suffices to prove that

$$((\mathsf{vis}' \cup \mathsf{so}')/\sim') \ \subseteq\ ((\mathsf{vis} \cup \mathsf{so})/\sim).$$

Recall that the factoring distributes over union (Lemma 24), and that so' and so are already factored via $\sim'$ and $\sim$, respectively. Hence, we can further simplify the above subset relationship as follows:

$$((\mathsf{vis}'/\sim') \cup \mathsf{so}') \ \subseteq\ ((\mathsf{vis}/\sim) \cup \mathsf{so}). \tag{34}$$

But $\mathsf{so}' = (\mathsf{so} \cap \sim') \subseteq \mathsf{so}$ and $\mathsf{vis}' = (\mathsf{vis} \cap (E' \times E')) \subseteq \mathsf{vis}$. Hence, one way to prove the subset relationship in (34) is to show that

$$\forall e, e', f, f' \in E'.\ ((e \sim' e' \xrightarrow{\mathsf{vis}} f' \sim' f) \wedge \neg(e \sim' f)) \implies (e \xrightarrow{(\mathsf{vis}/\sim)\cup\mathsf{so}} f). \tag{35}$$

Pick e, e', f, f' from E' that satisfy the assumption of the implication above. If $\neg(e \sim f)$, then $e \xrightarrow{\mathsf{vis}/\sim} f$ and the desired conclusion of (35) follows. If $(e \sim f)$, then

$$(e \xrightarrow{\mathsf{so}} f) \quad \vee \quad (f \xrightarrow{\mathsf{so}} e).$$

If the first disjunct holds, the conclusion of (35) follows immediately. The second disjunct, on the other hand, never holds. This is because otherwise we have $f' \xrightarrow{\mathsf{so}} e'$, so that

$$f' \xrightarrow{\mathsf{vis}} e'.$$

But we already have $e' \xrightarrow{\mathsf{vis}} f'$, so by the transitivity of vis, we get $e' \xrightarrow{\mathsf{vis}} e'$, which contradicts the irreflexivity of vis.

We now move on to the next claim of the lemma. Assume that

$$\{e \mid \exists f \in \mathsf{dom}(R).\ e \xrightarrow{X.\mathsf{vis}} f\} \ \subseteq\ \mathsf{dom}(R).$$

Consider a specification map $\mathbb{F}$ such that $X \models \mathbb{F}$. We should show that $\mathsf{proj}(X, R) \models \mathbb{F}$. Let $X' = \mathsf{proj}(X, R)$. Pick $e \in E'$ such that $X'.\mathsf{obj}(e) \in \mathsf{dom}(\mathbb{F})$. Define

$$N = \mathsf{ctxt}(X, e) \quad \text{and} \quad N' = \mathsf{ctxt}(X', e).$$

We need to prove that

$$X'.\mathsf{rval}(e) = \mathbb{F}(X'.\mathsf{obj}(e))(N').$$

This proof obligation can be discharged if we show that $N = N'$. This is because from this equality follows that

$$X'.\mathsf{rval}(e) = X.\mathsf{rval}(e) = \mathbb{F}(X.\mathsf{obj}(e))(N) = \mathbb{F}(X'.\mathsf{obj}(e))(N').$$

Proving the equality $N = N'$ is easy. Since $\mathsf{dom}(R)$ is closed under taking the inverse image with respect to $X.\mathsf{vis}$ and it is the same as E', we have that

$$\begin{aligned}\{f \in E \mid (f, e) \in X.\mathsf{vis}\} \ &=\ (\{f \in E \mid (f, e) \in X.\mathsf{vis}\} \cap E') \\ &=\ \{f \in E' \mid (f, e) \in X.\mathsf{vis} \times (E' \times E')\} \\ &=\ \{f \in E' \mid (f, e) \in X'.\mathsf{vis}\}.\end{aligned}$$

The desired equality $N = N'$ follows from this. □

For a function $\beta : E \to E'$ and a subset $E'_0 \subseteq E'$, we define a partial equivalence relation on E as follows:

$$\mathsf{per}(\beta, E'_0) \ =\ \{(e, f) \mid e, f \in E \wedge \beta(e) = \beta(f) \wedge \beta(e) \in E'_0\}.$$

COROLLARY 34 *For all objects ω, executions $X \in \mathsf{Exec}$ and $X' \in \mathsf{Exec}(\omega)$, morphisms $(\beta, \rho) : X \to X'$, specification maps $\mathbb{F}$, and subsets E'_0 of $X'.E$, if E'_0 is finite and closed under the inverse image of $X'.\mathsf{vis}$, then the projection $\mathsf{proj}(X, \mathsf{per}(\beta, E'_0))$ is an execution. Furthermore, if X satisfies* CAUSALVIS, CAUSALAR, *and $\mathbb{F}$, then so does the projection.*

We sometimes denote the projection $\mathsf{proj}(X, \mathsf{per}(\beta, E'_0))$ in this corollary using a simpler notation:

$$\mathsf{proj}(X, \beta, E'_0).$$

PROOF OF COROLLARY 34. Pick

$$\omega \in \mathsf{Obj}, \qquad X \in \mathsf{Exec}, \qquad X' \in \mathsf{Exec}(\omega), \qquad (\beta, \rho) : X \to X', \qquad \mathbb{F}$$

that satisfy the assumptions of this corollary. Also choose a finite subset E'_0 of $X'.E$. Let

$$R = \mathsf{per}(\beta, E'_0).$$

We will derive the claimed conclusion of the corollary using Lemmas 32 and 33. Specifically, we will show that

$$\begin{aligned} \mathsf{dom}(R) \text{ is finite} \quad &\wedge \quad R \subseteq X.\sim \quad \wedge \quad (X.\mathsf{so}/R) \subseteq X.\mathsf{so} \\ &\wedge \quad \{e \mid \exists f \in \mathsf{dom}(R).\, e \xrightarrow{X.\mathsf{vis}} f\} \subseteq \mathsf{dom}(R). \end{aligned} \tag{36}$$

The first conjunct of (36) is a consequence of the facts that E'_0 is finite and the inverse image of β for a finite set is also finite—the image should be the finite union of the event sets of histories selected from $\rho(p, b)$ for some (p, b) but every history in $\rho(p, b)$ has a finite event set.

We move on to the second conjunct of (36). Consider $e, f \in E$ such that $e \xrightarrow{R} f$. Then, $\beta(e) = \beta(f)$ by the definition of R. Since $X'.\sim$ is an equivalence relation, this implies that $\beta(e) \xrightarrow{X'.\sim} \beta(f)$. Now we use the fact that β reflects the equivalence relation (Lemma 14), and conclude that $e \xrightarrow{X.\sim} f$, as desired.

Next, we prove the third conjunct of (36). Consider $e, e', f, f' \in X.E$ such that

$$e \xrightarrow{R} e' \xrightarrow{X.\mathsf{so}} f' \xrightarrow{R} f \quad \wedge \quad \neg(e \xrightarrow{R} f).$$

If $\neg(e \xrightarrow{X.\sim} f)$, we have that

$$e \xrightarrow{X.\mathsf{so}/X.\sim} f,$$

because $R \subseteq X.\sim$. But $X.\mathsf{so}$ is already $X.\sim$-factored, so $e \xrightarrow{X.\mathsf{so}} f$, as desired. Now assume that $e \xrightarrow{X.\sim} f$. Then,

$$(e \xrightarrow{X.\mathsf{so}} f) \quad \vee \quad (f \xrightarrow{X.\mathsf{so}} e).$$

The first disjunct is the very conclusion that we look for. The second disjunct, on the other hand, never holds. Suppose that it did. Note that

$$\beta(e) \neq \beta(f) \quad \wedge \quad \beta(e') \neq \beta(f')$$

because $\neg(e \xrightarrow{R} f)$ but $e \xrightarrow{R} e'$ and $f' \xrightarrow{R} f$. Then, by the definition of morphism,

$$\beta(f) \xrightarrow{X'.\mathsf{so}} \beta(e) \quad \wedge \quad \beta(e') \xrightarrow{X'.\mathsf{so}} \beta(f').$$

Since $\beta(e) = \beta(e')$ and $\beta(f) = \beta(f')$, the relationships above imply that $X'.\mathsf{so}$ is reflexive, which contradicts the fact that $X'.\mathsf{so}$ is a strict partial order.

Finally, we show the last conjunct of (36). Pick $e, f \in X.E$ such that

$$f \in \mathsf{dom}(R) \quad \wedge \quad e \xrightarrow{X.\mathsf{vis}} f.$$

If $\beta(e) = \beta(f)$, then $e \xrightarrow{R} f$, so e should be in $\mathsf{dom}(R)$. Otherwise,

$$\beta(e) \xrightarrow{X'.\mathsf{vis}} \beta(f).$$

Recall that by assumption, E'_0 is closed under the inverse image of $X'.\mathsf{vis}$. Furthermore, since $f \in \mathsf{dom}(R)$, we should have that $\beta(f) \in E'_0$. Hence, $\beta(e) \in E'_0$ as well. This means that $e \in \mathsf{dom}(R)$. □

LEMMA 35 *For all objects ω, executions $X \in \mathsf{Exec}$ and $X' \in \mathsf{Exec}(\omega)$, morphisms $(\beta, \rho) : X \to X'$, and events $e \in X'.E$, we have the following well-formed abstraction:*

$$(\beta, \rho) : \mathsf{proj}(X, R) \to (\mathsf{ctxt}(X', e), e, X'.\mathsf{rval}(e)) \quad \textit{where } R = \mathsf{per}(\beta, \mathsf{ctxt}(X', e).E \cup \{e\}).$$

PROOF. Pick $\omega, X, X', (\beta, \rho), e$. Let

$$\begin{array}{lll} N = \mathsf{ctxt}(X', e), & a = X'.\mathsf{rval}(e), & E_N = N.E \cup \{e\}, \\ R = \mathsf{per}(\beta, E_N), & X_0 = \mathsf{proj}(X, R). & \end{array}$$

Since $X'.\mathsf{vis}$ is prefix-finite, $N.E$ is finite. Hence, E_N is also finite. Furthermore, E_N is closed under the inverse image of $X'.\mathsf{vis}$, because $X'.\mathsf{vis}$ is transitive. What we have just shown implies that $\mathsf{proj}(X, R)$ is a well-defined execution (Corollary 34). Let

$$\mathsf{vis}' = N.\mathsf{vis} \cup \{(f, e) \mid f \in N.E\}, \qquad \mathsf{ar}' = N.\mathsf{ar} \cup \{(f, e) \mid f \in N.E\}.$$

It suffices to show that

$$\begin{array}{l} (\forall f \in (N.E).\, (X_0.H)|_{\beta^{-1}(f)} \in \rho(N.\mathsf{aop}(f), X'.\mathsf{rval}(f))) \\ \quad \wedge \quad ((X_0.H)|_{\beta^{-1}(e)} \in \rho(N.p, a)) \quad \wedge \quad \beta(X_0.\mathsf{so}) \subseteq \mathsf{id} \\ \quad \wedge \quad \beta(X_0.{\sim}) \subseteq \mathsf{id} \quad \wedge \quad \beta(X_0.\mathsf{vis}) - \mathsf{id} \subseteq \mathsf{vis}' \\ \quad \wedge \quad \beta(X_0.\mathsf{ar}) - \mathsf{id} \subseteq \mathsf{ar}' \quad \wedge \quad \beta^{-1}(\mathsf{vis}') \cap \mathsf{sameobj}(X_0) \subseteq X_0.\mathsf{vis}. \end{array} \tag{37}$$

The first two conjuncts of (37) hold because (β, ρ) is a morphism from X to X' and so, for every $f \in E_N$,

$$(X_0.H)|_{\beta^{-1}(f)} \;=\; (X.H)|_{\beta^{-1}(f)} \;\in\; \rho(X'.\mathsf{aop}(f), X'.\mathsf{rval}(f)).$$

The next two conjuncts of (37) are immediate consequences of the definitions of R and the projection $\mathsf{proj}(X, R)$, as shown below:

$$\begin{array}{rcl} \beta(X_0.\mathsf{so}) & = & \beta(X.\mathsf{so} \cap R) \;\subseteq\; \beta(R) \;\subseteq\; \mathsf{id}, \\ \beta(X_0.{\sim}) & = & \beta(X.{\sim} \cap R) \;\subseteq\; \beta(R) \;\subseteq\; \mathsf{id}. \end{array}$$

We prove the fifth conjunct of (37) as follows:

$$\begin{aligned}\beta(X_0.\mathsf{vis}) - \mathsf{id} &\subseteq (\beta(X.\mathsf{vis}) - \mathsf{id}) \cap (E_N \times E_N)\\ &\subseteq X'.\mathsf{vis} \cap (E_N \times E_N)\\ &= \mathsf{vis}'.\end{aligned}$$

Here the first subset relationship holds because $X_0.\mathsf{vis}$ is a restriction of $X.\mathsf{vis}$ and it is defined over $\mathsf{dom}(R) = \beta^{-1}(E_N)$. The next subset relationship uses the fact that (β, ρ) is a morphism from X to X'. The following equality is an immediate consequence of the definition of vis'.

The proof of the sixth conjunct is similar:

$$\begin{aligned}\beta(X_0.\mathsf{ar}) - \mathsf{id} &\subseteq (\beta(X.\mathsf{ar}) - \mathsf{id}) \cap (E_N \times E_N)\\ &\subseteq X'.\mathsf{ar} \cap (E_N \times E_N)\\ &= \mathsf{ar}'.\end{aligned}$$

The first subset relationship uses the facts that $X_0.\mathsf{ar}$ is defined over $\mathsf{dom}(R)$ and that it is a restriction of $X.\mathsf{ar}$. The second subset relationship holds because (β, ρ) is a morphism from X to X'. The last equality comes from the fact that $X'.\mathsf{ar}$ is an acyclic relation and includes $X'.\mathsf{vis}$.

Finally, we prove the last conjunct of (37) as follows:

$$\begin{aligned}\beta^{-1}(\mathsf{vis}') \cap \mathsf{sameobj}(X_0) &\subseteq (\beta^{-1}(X'.\mathsf{vis}) \cap \mathsf{sameobj}(X)) \cap (\mathsf{dom}(R) \times \mathsf{dom}(R))\\ &\subseteq X.\mathsf{vis} \cap (\mathsf{dom}(R) \times \mathsf{dom}(R))\\ &= X_0.\mathsf{vis}.\end{aligned}$$

The first subset relationship uses the definitions of vis' and X_0, the second comes from the fact that (β, ρ) is a morphism from X to X', and the following equality follows from the definition of $X_0.\mathsf{vis}$. □

PROPOSITION 36 *For all*

$$\mathbb{F}, \quad X \in \mathsf{Exec}, \quad \omega \in \mathsf{Obj}, \quad H' \in \mathsf{Hist}(\omega), \quad R \subseteq (H'.E)^2, \quad (\beta, \rho) : X.H \to H'$$

if $R' = (H'.\mathsf{so} \cup (R/H'.\sim))$ *is acyclic and prefix-finite, and* $(X, \beta^{-1}(R'^{+})) \models_{\mathsf{CCS}} \mathbb{F}$*, then the following properties hold for* $X' = \mathsf{lift}(X, H', R, \beta)$*:*

1. X' *is an execution, and satisfies* CAUSALVIS *and* CAUSALAR.
2. (β, ρ) *is a morphism from* X *to* X'.
3. *For every* $e \in X'.E$*, if we let* $R_0 = \mathsf{per}(\beta, \{e\} \cup \mathsf{ctxt}(X', e).E)$*, then*

$$\mathsf{proj}(X, R_0) \models_{\mathsf{CC}} \mathbb{F} \wedge (\beta, \rho) : \mathsf{proj}(X, R_0) \to (\mathsf{ctxt}(X', e), e, X'.\mathsf{rval}(e)).$$

PROOF. The first property follows from Lemma 30, and the second from Lemma 31. For the third property, we note that since $(X, \beta^{-1}(R'^{+})) \models_{\mathsf{CCS}} \mathbb{F}$,

$$X \models_{\mathsf{CC}} \mathbb{F}.$$

Pick $e \in X'.E$. Let

$$E_N = \{e\} \cup \mathsf{ctxt}(X', e).E \quad \text{and} \quad R_0 = \mathsf{per}(\beta, E_N).$$

Then, since $X'.\mathsf{vis}$ is prefix-finite, E_N is finite. Also, it is closed under the inverse image of $X'.\mathsf{vis}$. Thus, by Corollary 34, $\mathsf{proj}(X, R_0)$ is an execution such that

$$\mathsf{proj}(X, R_0) \models_{\mathsf{CC}} \mathbb{F}.$$

Furthermore, by Lemma 35, we have the following well-defined abstraction:

$$(\beta, \rho) : \mathsf{proj}(X, R_0) \to (\mathsf{ctxt}(X', e), e, X'.\mathsf{rval}(e)).$$

□

LEMMA 37 *For all F, $\mathbb{F}$ and ρ, we have that*

$$\forall N.\ \forall (X, a) \in \gamma(N, \rho, \mathbb{F}).\ F(N) = a$$

if and only if for all

$$X \in \mathsf{Exec}, \quad \omega \in \mathsf{Obj}, \quad H' \in \mathsf{Hist}(\omega), \quad R \subseteq (H'.E)^2, \quad \beta : X.E \to H'.E,$$

we have that

$$\Big((\beta, \rho) : X.H \to H' \ \wedge\ (H'.\mathsf{so} \cup (R/H'.\sim)) \textit{ is acyclic and prefix-finite}$$
$$\wedge\ (X, \beta^{-1}\big((H'.\mathsf{so} \cup (R/H'.\sim))^+\big)) \models_{\mathsf{CCS}} \mathbb{F}\Big) \implies \mathsf{lift}(X, H', R, \beta) \models_{\mathsf{CC}} [\omega \mapsto F]$$

PROOF. Consider arbitrary F, $\mathbb{F}$ and ρ. We show the equivalence in the proposition for these F, $\mathbb{F}$ and ρ.

"Only if". Pick

$$X \in \mathsf{Exec}, \quad \omega \in \mathsf{Obj}, \quad H' \in \mathsf{Hist}(\omega), \quad R \subseteq (H'.E)^2, \quad \beta \in [X.E \to H'.E]$$

such that

$$\begin{aligned} (\beta, \rho) : X.H \to H' \quad &\wedge \quad (H'.\mathsf{so} \cup (R/H'.\sim)) \text{ is acyclic and prefix-finite} \\ &\wedge \quad (X, \beta^{-1}\big((H'.\mathsf{so} \cup (R/H'.\sim))^+\big)) \models_{\mathsf{CCS}} \mathbb{F}. \end{aligned}$$

We need to show that

$$\mathsf{lift}(X, H', R, \beta) \models [\omega \mapsto F].$$

Let

$$X' = \mathsf{lift}(X, H', R, \beta).$$

By Proposition 36, X' is an execution satisfying CAUSALVIS and CAUSALAR. Hence, it is sufficient to prove that

$$X' \models [\omega \mapsto F].$$

Pick $e \in X'.E$. Then,

$$X'.\mathsf{obj}(e) = \omega.$$

Let

$$N = \mathsf{ctxt}(X', e), \quad a = X'.\mathsf{rval}(e), \quad \text{and} \quad R_0 = \mathsf{per}(\beta, \{e\} \cup N.E).$$

Then by Proposition 36,

$$\mathsf{proj}(X, R_0) \models_{\mathsf{CC}} \mathbb{F} \wedge (\beta, \rho) : \mathsf{proj}(X, R_0) \to (N, e, a).$$

This implies that

$$(\mathsf{proj}(X, R_0), a) \in \gamma(N, \rho, \mathbb{F}).$$

Thus, by the assumption of the proposition,

$$F(N) = a.$$

We have just shown that $X' \models [\omega \mapsto F]$, as required.

"If". Pick

$$N \quad \text{and} \quad (X, a) \in \gamma(N, \rho, \mathbb{F}).$$

We have to prove that

$$F(N) = a.$$

Since $(X, a) \in \gamma(N, \rho, \mathbb{F})$, there exist β and e such that

$$e \notin N.E \quad \wedge \quad (\beta, \rho) : X \to (N, e, a) \quad \wedge \quad X \models_{\mathsf{CC}} \mathbb{F}.$$

By the second conjunct here, for every $f \in N.E$, there exists $a_f \in \mathsf{Val}$ such that

$$(X.H)|_{\beta^{-1}(f)} \in \rho(N.\mathsf{aop}(f), a_f).$$

We abuse the notations slightly and let $a_e = a$ and $N.\mathsf{aop}(e) = N.p$. Pick an arbitrary object $\omega \in \mathsf{Obj}$. Define an execution X' as follows:

$$\begin{aligned} X'.E = N.E \uplus \{e'\} \quad & \wedge \quad X'.\mathsf{label}(f) = (\omega, N.\mathsf{aop}(f), a_f) \quad (\text{for all } f \in N.E \cup \{e\}) \\ & \wedge \quad X'.\mathsf{so} = \emptyset \quad \wedge \quad X'.{\sim} = \mathsf{id} \\ & \wedge \quad X'.\mathsf{vis} = N.\mathsf{vis} \cup \{(f, e) \mid f \in N.E\} \\ & \wedge \quad X'.\mathsf{ar} = N.\mathsf{ar} \cup \{(f, e) \mid f \in N.E\}. \end{aligned}$$

It is relatively easy to check that X' is an execution, if we do not forget to use the following facts: (i) $X'.E = N.E \cup \{e\}$ is finite; (ii) e is not in $N.E$; (iii) both $N.\mathsf{vis}$ and $N.\mathsf{ar}$ are strict partial orders; (iv) $N.\mathsf{vis} \subseteq N.\mathsf{ar}$. Also, since (β, ρ) is an abstraction from X to (N, e, a), we have the following morphism from X to X':

$$(\beta, \rho) : X \to X'.$$

Let

$$H' = X'.H \quad \text{and} \quad R' = H'.\mathsf{so} \cup (X'.\mathsf{vis}/H'.{\sim}).$$

Then, $R' = X'.\mathsf{vis}$. Hence, it is acyclic. This also implies that R' is prefix-finite, because it is defined over the finite set $X'.E$. We will next show that $(X, \beta^{-1}(R'^{+}))$ satisfies CAUSALVIS′, CAUSALAR′ and PREFIXFINITEAR′.

Let us start with CAUSALVIS′. Note that, since $X'.\mathsf{vis}$ is transitive,

$$\beta^{-1}(R'^{+}) = \beta^{-1}((X'.\mathsf{vis})^{+}) = \beta^{-1}(X'.\mathsf{vis}).$$

Using $X'.\mathsf{so} = \emptyset$, $X'.{\sim} = \mathsf{id}$ and properties of β, we get

$$\begin{array}{rcl} \beta(X.\mathsf{so} \cup X.\mathsf{vis}) - \mathsf{id} & = & (\beta(X.\mathsf{so}) - \mathsf{id}) \cup (\beta(X.\mathsf{vis}) - \mathsf{id}) \\ & \subseteq & X'.\mathsf{so} \cup X'.\mathsf{vis} \\ & = & X'.\mathsf{vis}. \end{array}$$

From these observations and the same fact that $X'.\mathsf{so}$ is the empty relation and $X'.{\sim}$ is the identity relation we derive the following equalities:

$$\begin{array}{rcl} & & ((X.\mathsf{so} \cup X.\mathsf{vis} \cup \beta^{-1}(R'^{+}))/X.{\sim})^{+} \\ & = & ((X.\mathsf{so} \cup X.\mathsf{vis} \cup \beta^{-1}(X'.\mathsf{vis}^{+}))/X.{\sim})^{+} \\ & = & ((X.\mathsf{so} \cup X.\mathsf{vis} \cup \beta^{-1}(X'.\mathsf{vis}))/X.{\sim})^{+} \\ & = & (((X.\mathsf{so} \cup X.\mathsf{vis})/X.{\sim}) \cup (\beta^{-1}(X'.\mathsf{vis})/X.{\sim}))^{+} \\ & = & (((X.\mathsf{so} \cup X.\mathsf{vis})/X.{\sim}) \cup \beta^{-1}(X'.\mathsf{vis}))^{+} \\ & = & ((X.\mathsf{so} \cup X.\mathsf{vis})/X.{\sim})^{+} \cup (\beta^{-1}(X'.\mathsf{vis}))^{+} \\ & = & ((X.\mathsf{so} \cup X.\mathsf{vis})/X.{\sim})^{+} \cup \beta^{-1}(X'.\mathsf{vis}). \end{array}$$

The first equality holds because $R' = X'.\mathsf{vis}$, the second follows from the transitivity of $X'.\mathsf{vis}$, and the third uses the distributivity of the quotienting over the relation union. The fourth follows from the fact that for all $e_1, e_2, e_3, e_4 \in X.E$,

$$\begin{array}{r} ((e_1, e_2) \in X.{\sim} \wedge (e_2, e_3) \in \beta^{-1}(X'.\mathsf{vis}) \wedge (e_3, e_4) \in X.{\sim}) \\ \implies (e_1, e_4) \in \beta^{-1}(X'.\mathsf{vis}). \end{array} \tag{38}$$

The last equality holds because the inverse image of a transitive relation is also transitive. The most tricky part is the fifth equality. It holds because for all $e_0, e_1, e_2, e_3, e_4 \in X.E$,

$$\begin{array}{l} ((e_0, e_1) \in X.{\sim} \wedge (e_1, e_2) \in (X.\mathsf{so} \cup X.\mathsf{vis}) \wedge (e_2, e_3) \in X.{\sim} \wedge (e_3, e_4) \in \beta^{-1}(X'.\mathsf{vis})) \\ \quad \implies ((e_0, e_1) \in X.{\sim} \wedge (e_1, e_2) \in (X.\mathsf{so} \cup X.\mathsf{vis}) \wedge (e_2, e_4) \in \beta^{-1}(X'.\mathsf{vis})) \\ \quad \implies ((e_0, e_1) \in X.{\sim} \wedge (e_1, e_4) \in \beta^{-1}(X'.\mathsf{vis})) \\ \quad \implies (e_0, e_4) \in \beta^{-1}(X'.\mathsf{vis}), \end{array}$$

where the first and third steps use (38) and the second step uses the transitivity of $X'.\mathsf{vis}$ and the inclusion $\beta(X.\mathsf{so} \cup X.\mathsf{vis}) \subseteq (\mathsf{id} \cup X'.\mathsf{vis})$. Because of what we have just shown,

$$\begin{array}{rcl} & & ((X.\mathsf{so} \cup X.\mathsf{vis} \cup \beta^{-1}(R'^{+}))/(X.{\sim}))^{+} \cap \mathsf{sameobj}(X) \\ & = & ((X.\mathsf{so} \cup X.\mathsf{vis})/(X.{\sim}))^{+} \cap \mathsf{sameobj}(X) \cup \beta^{-1}(X'.\mathsf{vis}) \cap \mathsf{sameobj}(X) \\ & \subseteq & X.\mathsf{vis}. \end{array}$$

The last subset relationship holds because X satisfies the CAUSALVIS axiom and (β, ρ) is a morphism from X to X'.

To show CAUSALAR$'$, we notice that

$$\beta((X.\mathsf{so} \cup X.\mathsf{vis} \cup X.\mathsf{ar})/X.\sim) - \mathsf{id} \quad \subseteq \quad X'.\mathsf{vis} \cup X'.\mathsf{ar} \tag{39}$$

and also that

$$\begin{aligned}
&(X.\mathsf{so} \cup X.\mathsf{vis} \cup X.\mathsf{ar} \cup \beta^{-1}(R'^{+}))/X.\sim \\
&\quad = \quad (X.\mathsf{so} \cup X.\mathsf{vis} \cup X.\mathsf{ar} \cup \beta^{-1}(X'.\mathsf{vis}^{+}))/X.\sim \\
&\quad = \quad (X.\mathsf{so} \cup X.\mathsf{vis} \cup X.\mathsf{ar} \cup \beta^{-1}(X'.\mathsf{vis}))/X.\sim \\
&\quad = \quad ((X.\mathsf{so} \cup X.\mathsf{vis} \cup X.\mathsf{ar})/X.\sim) \cup \beta^{-1}(X'.\mathsf{vis}).
\end{aligned}$$

Furthermore, $(X.\mathsf{so} \cup X.\mathsf{vis} \cup X.\mathsf{ar})/(X.\sim)$ is acyclic. Thus, if $(X.\mathsf{so} \cup X.\mathsf{vis} \cup X.\mathsf{ar} \cup \beta^{-1}(R'^{+}))/(X.\sim)$ is cyclic, this cycle should contain an edge from $\beta^{-1}(X'.\mathsf{vis})$. But in this case, because of (39), β maps this cycle to a cycle in X' that has edges only from $X'.\mathsf{vis}$ and $X'.\mathsf{ar}$. This contradicts the fact that $X'.\mathsf{vis} \cup X'.\mathsf{ar}$ is acyclic.

The last axiom is PREFIXFINITEAR$'$. Since (β, ρ) is a morphism from X to X' and $X'.E$ is finite, $X.E$ should also be finite. This implies the PREFIXFINITEAR$'$ axiom.

We have thus established that $(X, \beta^{-1}(R'^{+}))$ satisfies CAUSALVIS$'$, CAUSALAR$'$ and PREFIXFINITEAR$'$. Since $R' = X'.\mathsf{vis}$ is acyclic and prefix-finite, $\mathsf{lift}(X, H', X'.\mathsf{vis}, \beta)$ is an execution satisfying CAUSALVIS and CAUSALAR by Lemma 30. Furthermore, since $X \models_{\mathsf{CC}} \mathbb{F}$ and $H' \in \mathsf{Hist}(\omega)$, by our assumption we have

$$\mathsf{lift}(X, H', X'.\mathsf{vis}, \beta) \models [\omega \mapsto F]. \tag{40}$$

We now show a certain correspondence between $\mathsf{lift}(X, H', X'.\mathsf{vis}, \beta)$ and X'. First, their histories are the same since we chose $H' = X'.H$.

Next,

$$\begin{aligned}
\mathsf{lift}(X, H', X'.\mathsf{vis}, \beta).\mathsf{vis} \quad &= \quad ((H'.\mathsf{so} \cup (\beta(X.\mathsf{vis}) - \mathsf{id}) \cup X'.\mathsf{vis})/X'.\sim)^{+} \\
&= \quad ((\beta(X.\mathsf{vis}) - \mathsf{id}) \cup X'.\mathsf{vis})^{+} \\
&= \quad (X'.\mathsf{vis})^{+} \quad = \quad X'.\mathsf{vis}.
\end{aligned}$$

The second equality holds because $H'.\mathsf{so} = \emptyset$ and $X'.\sim = \mathsf{id}$. The third equality comes from the fact that (β, ρ) is a morphism from X to X' and so $\beta(X.\mathsf{vis}) - \mathsf{id} \subseteq X'.\mathsf{vis}$. The fourth follows from the transitivity of $X'.\mathsf{vis}$.

Third,

$$\begin{aligned}
\mathsf{lift}(X, H', X'.\mathsf{vis}, \beta).\mathsf{ar} \quad &= \quad ((H'.\mathsf{so} \cup (\beta(X.\mathsf{ar}) - \mathsf{id}) \cup X'.\mathsf{vis})/X'.\sim)^{+} \\
&= \quad ((\beta(X.\mathsf{ar}) - \mathsf{id}) \cup X'.\mathsf{vis})^{+} \\
&\subseteq \quad (X'.\mathsf{ar} \cup X'.\mathsf{vis})^{+} \\
&= \quad (X'.\mathsf{ar})^{+} \quad = \quad X'.\mathsf{ar}.
\end{aligned}$$

The second equality holds because $H'.\mathsf{so} = \emptyset$ and $X'.\sim = \mathsf{id}$. The next inclusion holds because (β, ρ) is a morphism and so $\beta(X.\mathsf{ar}) - \mathsf{id} \subseteq X'.\mathsf{ar}$. The last two equalities use the facts that $X'.\mathsf{vis} \subseteq X'.\mathsf{ar}$ and $X'.\mathsf{ar}$ is transitive.

Thus, $\mathsf{lift}(X, H', X'.\mathsf{vis}, \beta)$ and X' are identical, except the latter may have a bigger arbitration relation. Since F preserves its value on arbitration extensions (Definition 2), from (40) we get

$$X' \models [\omega \mapsto F].$$

This in turn gives

$$a = X'.\mathsf{rval}(e) = F(\mathsf{ctxt}(X', e)) = F(N),$$

as required. □

We compare specifications $F, F' \subseteq \mathsf{Spec}$ using the following order:

$$F \sqsubseteq F'$$

if and only if

$$\forall N \in \mathsf{Ctxt}.\ N \in \mathsf{dom}(F) \implies (N \in \mathsf{dom}(F') \wedge F(N) = F'(N)).$$

Using this order, we define an operation that selects minimal specifications from a given set of specifications: for all $\mathcal{F} \subseteq \mathsf{Spec}$,

$$\mathsf{max}(\mathcal{F}) = \{F \in \mathcal{F} \mid \neg\exists F' \in \mathcal{F}.\ F \neq F' \wedge F \sqsubseteq F'\}.$$

Then from Lemma 37 we get

COROLLARY 38

$$\{[\![\Gamma \vdash \mathsf{let}\ \{x_j = \mathsf{new}\ T_j\}_{j=1..m}\ \mathsf{in}\ \{o(v_{\mathsf{in}}) : v_{\mathsf{out}} = \mathsf{atomic}\ \{C_o\}\}_{o\in O} : O]\!]\eta\} =$$
$$\mathsf{max}\left(\begin{array}{l}\{F \mid \forall obj \in [\{x_j \mid j = 1..m\} \to_{\mathsf{inj}} \mathsf{Obj}].\ \exists \mathbb{F} \in [\mathsf{range}(obj) \to \mathsf{Spec}].\\ (\forall j = 1..m.\, \mathbb{F}(obj(x_j)) \in [\![T_j]\!]\eta) \wedge\\ (\forall X \in \mathsf{Exec}.\, \forall \omega \in \mathsf{Obj}.\, \forall H' \in \mathsf{Hist}(\omega).\, \forall R \subseteq (H'.E)^2.\, \forall \beta : X.E \to H'.E.\\ \quad ((\beta, [\![\{C_o\}_{o\in O}]\!](obj)) : X.H \to H'\\ \quad\ \wedge (H'.\mathsf{so} \cup (R/H'.\sim)\ \textit{is acyclic and prefix-finite})\\ \quad\ \wedge (X, \beta^{-1}((H'.\mathsf{so} \cup (R/H'.\sim))^+)) \models_{\mathsf{CCS}} \mathbb{F})\\ \qquad\qquad \implies \mathsf{lift}(X, H', R, \beta) \models_{\mathsf{CC}} [\omega \mapsto F])\}\end{array}\right).$$

D.4 Soundness

Throughout this section, we fix an object variable environment $\{x_j : O_j \mid j = 1..m\}$ and a collection of commands

$$\{x_j : O_j \mid j = 1..m\} \mid v_{\mathsf{in}}, v_{\mathsf{out}} \vdash C_o, \quad o \in O.$$

DEFINITION 39 *For $H, H' \in \mathsf{Hist}$ we write $(\beta, \rho, \omega) : H \Rightarrow H'$ if the function $\beta : H.E \to H'.E$, summary $\rho : \mathsf{AOp} \times \mathsf{Val} \rightharpoonup \mathcal{P}(\mathsf{FHist})$ and object $\omega \in \mathsf{Obj}$ are such that*

$$\begin{array}{l}(\forall e \in H'.E.\, (H'.\mathsf{obj}(e) \neq \omega \implies e \in H.E \wedge \beta(e) = e \wedge H.\mathsf{label}(e) = H'.\mathsf{label}(e))\\ \qquad \wedge (H'.\mathsf{obj}(e) = \omega \implies e \notin H.E \wedge H|_{\beta^{-1}(e)} \in \rho(H'.\mathsf{aop}(e), H'.\mathsf{rval}(e))))\\ \wedge\ \beta(H.\mathsf{so}) - \mathsf{id} = (H'.\mathsf{so})|_{\beta(H.E)}\\ \wedge\ \beta(H.\sim) = (H'.\sim)|_{\beta(H.E)}.\end{array}$$

We note that $(\beta, \rho, \omega) : H \Rightarrow H'$ implies

$$\forall e, f, g \in H.E.\, e \xrightarrow{H.\mathsf{so}} f \xrightarrow{H.\mathsf{so}} g \wedge \beta(e) = \beta(g) \implies \beta(e) = \beta(f) = \beta(g), \quad (41)$$

for otherwise we would have $\beta(e) \xrightarrow{H'.\mathsf{so}} \beta(f) \xrightarrow{H'.\mathsf{so}} \beta(g)$ and $\beta(e) = \beta(g)$, contradicting the transitivity and irreflexivity of $H'.\mathsf{so}$.

Let

$$(\!|C|\!)(obj, \sigma) = \{H \mid H \in \langle C \rangle(obj, \sigma) \vee (H, \_) \in \langle C \rangle(obj, \sigma)\}.$$

PROPOSITION 40 *Assume*

$$\begin{array}{c} \Delta \cup \{x : O\} \mid \Sigma \vdash C; \\ \Delta \cup \{x_j : O_j \mid j = 1..m\} \mid \Sigma \vdash subst(\{(x, o) \mapsto C_o \mid o \in O\}, C). \end{array}$$

Then

$$\begin{array}{l} \forall obj : \mathsf{dom}(\Delta \cup \{x_j : O_j \mid j = 1..m\}) \to_{\mathsf{inj}} \mathsf{Obj}. \\ \forall \sigma : \Sigma \to \mathsf{Val}. \forall \omega \in \mathsf{Obj} - \mathsf{range}(obj). \\ \forall H \in (\!|subst(\{(x, o) \mapsto C_o \mid o \in O\}, C)|\!)(obj, \sigma). \\ \quad \exists H' \in (\!|C|\!)(obj|_{\mathsf{OVar} - \{x_j \mid j=1..m\}}[x \mapsto \omega], \sigma). \exists \beta : H.E \to H'.E. \\ \qquad ((\beta, [\![\{C_o\}_{o \in O}]\!](obj|_{\{x_j \mid j=1..m\}}), \omega) : H \Rightarrow H'). \end{array}$$

Let P_{C} range over clients:

$$P_{\mathsf{C}} \ ::= \ C_1 \parallel \ldots \parallel C_n.$$

We let

$$\langle C_1 \parallel \ldots \parallel C_n \rangle obj = \left\{ \biguplus_{j=1}^{n} H_j \mid \forall j = 1..n.\, H_j \in (\!|C_j|\!)(obj, [\,]) \right\}.$$

Then the clause for the denotation of a client can be rewritten as

$$\begin{array}{l} [\![C_1 \parallel \ldots \parallel C_n]\!](type, obj, \mathbb{F}) \\ \qquad = \{H \ \mid \ H \in \langle C_1 \parallel \ldots \parallel C_n \rangle obj \wedge \exists \mathsf{vis}, \mathsf{ar}.\, (H, \mathsf{vis}, \mathsf{ar}) \models_{\mathsf{CC}} \mathbb{F}\}. \end{array}$$

From Proposition 40 we get

COROLLARY 41 *Assume*

$$\begin{array}{c} \emptyset \mid \Delta \cup \{x : O\} \vdash P_{\mathsf{C}}; \\ \emptyset \mid \Delta \cup \{x_j : O_j \mid j = 1..m\} \vdash subst(\{(x, o) \mapsto C_o \mid o \in O\}, P_{\mathsf{C}}). \end{array}$$

Then

$$\begin{array}{l} \forall obj : \mathsf{dom}(\Delta \cup \{x_j : O_j \mid j = 1..m\}) \to_{\mathsf{inj}} \mathsf{Obj}. \forall \omega \in \mathsf{Obj} - \mathsf{range}(obj). \\ \forall H \in \langle subst(\{(x, o) \mapsto C_o \mid o \in O\}, P_{\mathsf{C}}) \rangle obj. \\ \quad \exists H' \in \langle P_{\mathsf{C}} \rangle (obj|_{\mathsf{OVar} - \{x_j \mid j=1..m\}}[x \mapsto \omega]). \exists \beta : H.E \to H'.E. \\ \qquad ((\beta, [\![\{C_o\}_{o \in O}]\!](obj), \omega) : H \Rightarrow H'). \end{array}$$

We now discharge the most difficult case in the soundness direction of Theorem 21—that of the data-type-inlining reduction.

THEOREM 42 (SOUNDNESS) *Let*

$$\begin{aligned} D &= (\mathsf{let}\ \{x_j = \mathsf{new}\ T_j\}_{j=1..m}\ \mathsf{in}\ \{o(v_{\mathsf{in}}) : v_{\mathsf{out}} = \mathsf{atomic}\ \{C_o\}\}_{o\in O});\\ P^2 &= (\mathsf{let}\ x = \mathsf{new}\ D\ \mathsf{in}\ P);\\ P^1 &= (\mathsf{let}\ \{x_j = \mathsf{new}\ T_j\}_{j=1..m}\ \mathsf{in}\ \mathit{subst}(\{(x,o)\mapsto C_o \mid o\in O\}, P)), \end{aligned}$$

where P^1 and P^2 are complete. Then $[\![P^1]\!]_{\mathsf{CG}} \subseteq [\![P^2]\!]_{\mathsf{CG}}$.

PROOF. Let

$$\begin{aligned} \emptyset \mid \{x_{\mathsf{io}} : \{o_{\mathsf{io}}\}\} \cup \Delta \cup \{x : O\} &\vdash P^2_{\mathsf{C}};\\ \emptyset \mid \{x_{\mathsf{io}} : \{o_{\mathsf{io}}\}\} \cup \Delta \cup \{x_j : O_j \mid j = 1..m\} &\vdash P^1_{\mathsf{C}} \end{aligned}$$

be the client parts of P^1 and P^2, respectively, so that

$$P^1_{\mathsf{C}} = \mathit{subst}(\{(x,o)\mapsto C_o \mid o\in O\}, P^2_{\mathsf{C}}).$$

Consider any $obj_0 : (\mathsf{dom}(\Delta) \cup \{x_{\mathsf{io}}\} \cup \{x_j \mid j = 1..m\}) \to_{\mathsf{inj}} \mathsf{Obj}$ such that $obj_0(x_{\mathsf{io}}) = \mathsf{io}$ and $\mathbb{F}_0 : (\mathsf{range}(obj_0) - \{\mathsf{io}\}) \to \mathsf{Spec}$ such that $\forall j = 1..m.\, \mathbb{F}_0(obj(x_j)) \in [\![T_j]\!][\,]$. Further, consider any execution

$$X_0 = (H_0, \mathsf{vis}_0, \mathsf{ar}_0) = ((E_0, \mathsf{label}_0, \mathsf{so}_0, \sim_0), \mathsf{vis}_0, \mathsf{ar}_0) \in [\![P^1_{\mathsf{C}}]\!]([\,], obj_0, \mathbb{F}_0).$$

In the following, we also use selectors such as obj_0 for this and other executions. We also take $\omega \in \mathsf{Obj} - \mathsf{range}(obj_0)$ and define $obj'_0 = obj_0|_{\mathsf{OVar}-\{x_j|j=1..m\}}[x \mapsto \omega]$. Let

$$F = [\![D]\!][\,]; \qquad L = \mathsf{range}(obj_0|_{\{x_j|j=1..m\}}); \qquad \mathbb{F}'_0 = \mathbb{F}_0|_{\mathsf{Obj}-L}[\omega \mapsto F].$$

The goal of the following development is to construct X''_0 such that

$$\mathsf{observ}(X_0.H) = \mathsf{observ}(X''_0.H) \wedge X''_0 \in [\![P^2_{\mathsf{C}}]\!]([\,], obj'_0, \mathbb{F}'_0).$$

Since $X_0 \in [\![P^1_{\mathsf{C}}]\!]([\,], obj_0, \mathbb{F}_0)$, we have $H_0 \in \langle P^1_{\mathsf{C}}\rangle obj_0$ and $X_0 \models_{\mathsf{CC}} \mathbb{F}_0$. From the former, by Corollary 41 for some

$$H'_0 = (E'_0, \mathsf{label}'_0, \mathsf{so}'_0, \sim'_0) \in \langle P^2_{\mathsf{C}}\rangle obj'_0 \tag{42}$$

and $\beta_0 : H_0.E \to H'_0.E$ we have $(\beta_0, [\![\{C_o\}_{o\in O}]\!](obj), \omega) : H_0 \Rightarrow H'_0$.

Let

$$\begin{aligned} \mathbb{F} = \mathbb{F}_0|_L; \quad obj &= obj_0|_{\{x_j|j=1..m\}};\\ X = (H, \mathsf{vis}, \mathsf{ar}) = ((E, \mathsf{label}, \mathsf{so}, \sim), \mathsf{vis}, \mathsf{ar}) &= X_0|_E\\ \text{and } \beta = \beta_0|_E \text{ for } E &= \{e \mid \mathsf{obj}'_0(\beta_0(e)) = \omega\};\\ H' = (E', \mathsf{label}', \mathsf{so}', \sim') = H'_0|_{E'} \text{ for } E' &= \{e \mid \mathsf{obj}'_0(e) = \omega\}; \end{aligned}$$

It is easy to see that

$$(\forall e \in E.\, \mathsf{obj}(e) \in L) \wedge (\forall e \in E_0 - E.\, \mathsf{obj}_0(e) \notin L). \tag{43}$$

Then $\mathsf{observ}(H_0) = \mathsf{observ}(H_0')$. We also have $X \in \mathsf{Exec}$, $H' \in \mathsf{Hist}(\omega)$ and $(\beta, [\![\{C_o\}_{o\in O}]\!](obj)) : H \to H'$.

For $r \subseteq E_0 \times E_0$ we let $\mathsf{lib}(r) = r|_E$ and $\mathsf{client}(r) = r|_{E_0 - E}$; for $r \subseteq E_0' \times E_0'$ we let $\mathsf{lib}'(r) = r|_{E'}$.

Let

$$\begin{aligned}
Q &= \beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id};\\
R &= \mathsf{lib}'((\mathsf{so}_0' \cup (Q/{\sim_0'}))^+);\\
X' &= (H', ((\mathsf{so}' \cup (\beta(\mathsf{vis}) - \mathsf{id}) \cup R)/{\sim'})^+, ((\mathsf{so}' \cup (\beta(\mathsf{ar}) - \mathsf{id}) \cup R)/{\sim'})^+).
\end{aligned}$$

Then, since $F = [\![D]\!][\,]$, by Corollary 38 and Lemmas 30 and 31 we get

$$\begin{aligned}
&((((R/{\sim'}) \cup \mathsf{so}') \text{ is acyclic and prefix-finite}) \wedge (X, \beta^{-1}((\mathsf{so}' \cup (R/{\sim'}))^+)) \models_{\mathsf{CCS}} \mathbb{F})\\
&\implies X' \in \mathsf{Exec} \wedge X' \models_{\mathsf{CCS}} [\omega \mapsto F] \wedge (\beta, [\![\{C_o\}_{o\in O}]\!](obj)) : X \to X'.
\end{aligned} \tag{44}$$

Proposition A: *$Q/{\sim_0'}$ is transitive.* Take $e', f', g' \in E_0'$ such that

$$e' \xrightarrow{Q/{\sim_0'}} f' \xrightarrow{Q/{\sim_0'}} g'$$

and consider three cases.

1. $e' \sim_0' f' \sim_0' g'$. Then

   $$e' \xrightarrow{Q} f' \xrightarrow{Q} g'.$$

   Taking into account $(\beta_0, [\![\{C_o\}_{o\in O}]\!](obj), \omega) : H_0 \Rightarrow H_0'$, for some $e, f_1, f_2, g \in E_0$ we have

   $$\beta_0(e) = e'; \quad \beta_0(f_1) = \beta_0(f_2) = f'; \quad \beta_0(g) = g';$$

   $$e \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+} f_1 \sim_0 f_2 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+} g.$$

   Then $e \sim_0 f_1 \sim_0 f_2 \sim_0 g$ and, hence,

   $$e \xrightarrow{\mathsf{so}_0} f_1 \sim_0 f_2 \xrightarrow{\mathsf{so}_0} g$$

   and any pair of the four events is related by so_0. The case of $g \xrightarrow{\mathsf{so}_0} e$ contradicts (41); hence, $e \xrightarrow{\mathsf{so}_0} g$. If $\beta_0(e) = \beta_0(g)$, then

   $$f' \xrightarrow{\mathsf{so}_0'} g' \xrightarrow{\mathsf{so}_0} f',$$

   contradicting the properties of so_0'. Hence, $\beta_0(e) \neq \beta_0(g)$ and $(e', g') \in Q$.

2. $f' \not\sim'_0 g'$. Then for some $e'_1, f'_1, f'_2, g'_1 \in E'_0$ we have

$$e' \sim'_0 e'_1 \xrightarrow{Q} f'_1 \sim'_0 f' \sim'_0 f'_2 \xrightarrow{Q} g'_1 \sim'_0 g'.$$

Hence, for some $e_1, f_1, f_2, g_1 \in E_0$ we have

$$\beta_0(e_1) = e'_1; \quad \beta_0(f_1) = f'_1; \quad \beta_0(f_2) = f'_2; \quad \beta_0(g_1) = g'_1;$$

$$e_1 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} f_1 \sim_0 f_2 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} g_1$$

and $f_2 \not\sim_0 g_1$. Because of the latter, for some $f_3 \in E_0$ we have

$$e_1 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} f_1 \sim_0 f_2 \xrightarrow{(\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0} f_3 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} g_1$$

and $f_2 \not\sim_0 f_3$. By the definition of factoring, this implies

$$e_1 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} f_1 \xrightarrow{(\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0} f_3 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} g_1,$$

and $f_1 \not\sim_0 g_1$, so that $e_1 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} g_1$.
Since $f_1 \not\sim_0 g_1$, we cannot have $e_1 \sim_0 g_1$: if this were the case, then similarly to the above we would be able to show that

$$f_1 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} e_1,$$

contradicting CAUSALAR for X_0. Since $\beta_0(e_1) = \beta_0(g_1)$ implies $e_1 \sim_0 g_1$, we get that $\beta_0(e_1) \neq \beta_0(g_1)$ and, therefore, $(e'_1, g'_1) \in Q$. Since $e_1 \not\sim_0 g_1$, we cannot have $e'_1 \not\sim_0 g'_1$ and, hence, $(e', g') \in Q$.
3. $e' \not\sim'_0 f'$. This case is analogous to the previous one.

■

Proposition B: $(\mathsf{so}'_0 \cup (Q/\sim'_0))^+ = \mathsf{so}'_0 \cup ((\mathsf{so}'_0)^*; (Q/\sim'_0); (\mathsf{so}'_0)^*)$. Consider $(e'_0, f'_0) \in (\mathsf{so}'_0 \cup (Q/\sim'_0))^+$. Then there is a path from e'_0 to f'_0 in $\mathsf{so}'_0 \cup (Q/\sim'_0)$. By Proposition A, $Q/\sim'_0$ and so'_0 are transitive. Hence, we can assume that the edges from these two relations alternate on the path.

Consider first the case when all events on the path are from the same transaction according to $\sim'_0$. Then every edge is in $\mathsf{so}'_0 \cup Q$ and, since $Q \subseteq Q|_{\beta_0(E_0)}$, the path is in

$$\begin{aligned}
&\mathsf{so}'_0 \cup ((\mathsf{so}'_0)^*; (\mathsf{so}'_0|_{\beta_0(E_0)} \cup Q)^+; (\mathsf{so}'_0)^*) \\
\subseteq\ &\mathsf{so}'_0 \cup ((\mathsf{so}'_0)^*; ((\beta_0(\mathsf{so}_0) - \mathsf{id}) \cup Q)^+; (\mathsf{so}'_0)^*) \\
\subseteq\ &\mathsf{so}'_0 \cup ((\mathsf{so}'_0)^*; Q^+; (\mathsf{so}'_0)^*) \\
\subseteq\ &\mathsf{so}'_0 \cup ((\mathsf{so}'_0)^*; Q/\sim'_0; (\mathsf{so}'_0)^*)
\end{aligned}$$

as required.

Assume now that the path contains events from at least two different transactions. We can ensure that the path does not contain edges

$$e_1 \xrightarrow{\mathsf{so}'_0 \cup (Q/\sim'_0)} e_2$$

for $e_1 \sim' e_2$, since, e.g., we can replace any sequence of edges

$$e_1 \xrightarrow{\mathsf{so}'_0 \cup (Q/\sim'_0)} e_2 \xrightarrow{\mathsf{so}'_0 \cup (Q/\sim'_0)} \dots e_n \xrightarrow{\mathsf{so}'_0 \cup (Q/\sim'_0)} e_{n+1}$$

such that $e_1 \sim' e_2 \sim' \dots \sim' e_n$ and $e_n \not\sim' e_{n+1}$ by a single edge

$$e_1 \xrightarrow{\mathsf{so}'_0 \cup (Q/\sim'_0)} e_{n+1}$$

for which $e_1 \not\sim' e_{n+1}$.

Assume the path contains a fragment of the form

$$e' \xrightarrow{Q/\sim'_0} f' \xrightarrow{\mathsf{so}'_0} g' \xrightarrow{Q/\sim'_0} h'; \tag{45}$$

then $e' \not\sim'_0 f'$, $f' \not\sim'_0 g'$ and $g' \not\sim'_0 h'$. Then for some $e'_0, f'_0, g'_0, h'_0 \in E'_0$ we have

$$e' \sim'_0 e'_0 \xrightarrow{Q} f'_0 \sim'_0 f' \xrightarrow{\mathsf{so}'_0} g' \sim'_0 g'_0 \xrightarrow{Q} h'_0 \sim'_0 h',$$

Hence,

$$e' \sim'_0 e'_0 \xrightarrow{Q} f'_0 \xrightarrow{\mathsf{so}'_0} g'_0 \xrightarrow{Q} h'_0 \sim'_0 h',$$

Since $Q \subseteq Q|_{\beta_0(E_0)}$, we have

$$(f'_0, g'_0) \in \mathsf{so}'_0|_{\beta_0(E_0)} = (\beta_0(\mathsf{so}_0) - \mathsf{id}) \subseteq Q,$$

and thus

$$e' \sim'_0 e'_0 \xrightarrow{Q} f'_0 \xrightarrow{Q} g'_0 \xrightarrow{Q} h'_0 \sim'_0 h'.$$

This implies

$$e' \xrightarrow{Q/\sim'_0} f'_0 \xrightarrow{Q/\sim'_0} g'_0 \xrightarrow{Q/\sim'_0} h'.$$

Then by the transitivity of $Q/\sim'_0$ we get

$$e' \xrightarrow{Q/\sim'_0} h'.$$

In this case we can thus replace the fragment (45) by this edge.

Applying the above steps repeatedly, we can ensure that the path does not have fragments of the form (45), which means that $(e'_0, f'_0) \in \mathsf{so}'_0 \cup ((\mathsf{so}'_0)^*; (Q/\sim'_0); (\mathsf{so}'_0)^*)$. ■

Proposition C: $\mathsf{so}'_0 \cup (Q/\sim'_0)$ *is acyclic.* By Proposition B and the facts that so'_0 and $Q/\sim'_0$ are transitive and irreflexive, if there is a cycle in $\mathsf{so}'_0 \cup (Q/\sim'_0)$, then it can be converted into the form

$$e' \xrightarrow{Q/\sim'_0} f' \xrightarrow{\mathsf{so}'_0} e'.$$

If $e' \sim'_0 f'$, then $e' \xrightarrow{Q} f'$ and, hence, $f' \xrightarrow{\mathsf{so}'_0|_{\beta_0(E_0)}} e'$. The latter implies $f' \xrightarrow{Q} e'$, which yields a contradiction with the acyclicity of $Q/\sim'_0$.

Now assume $e' \not\sim'_0 f'$. Then for some $e'_0, f'_0 \in E'_0$ we have

$$e' \sim'_0 e'_0 \xrightarrow{Q} f'_0 \sim'_0 f' \xrightarrow{\mathsf{so}'_0} e'.$$

This implies

$$e'_0 \xrightarrow{Q} f'_0 \xrightarrow{\mathsf{so}'_0} e'_0,$$

which yields a contradiction as above. ■

Proposition D: $\beta^{-1}((\mathsf{so}' \cup (R/\sim'))^+) = \mathsf{lib}(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{sameop}$, *where* $\mathsf{sameop} = \{(e,f) \mid \beta(e) = \beta(f)\}$. By Proposition B we have:

$$\begin{aligned}
&\beta^{-1}((\mathsf{so}' \cup (R/\sim'))^+) \\
&= \beta^{-1}((\mathsf{so}' \cup (\mathsf{lib}'((\mathsf{so}'_0 \cup (Q/\sim'_0))^+)/\sim'))^+) \\
&= \beta^{-1}(\mathsf{lib}'((\mathsf{so}'_0 \cup (Q/\sim'_0))^+)) \\
&= \beta^{-1}((\mathsf{so}'_0 \cup (Q/\sim'_0))^+) \\
&= \beta^{-1}(\mathsf{so}'_0 \cup ((\mathsf{so}'_0)^*; (Q/\sim'_0); (\mathsf{so}'_0)^*)) \\
&= \beta^{-1}(\mathsf{so}') \cup \beta^{-1}((\mathsf{so}'_0)^*; ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{id})/\sim'_0); (\mathsf{so}'_0)^*).
\end{aligned}$$

We have:

$$\beta^{-1}(\mathsf{so}') = \beta^{-1}(\mathsf{so}'|_{\beta(E)}) = \beta^{-1}(\beta(\mathsf{so}) - \mathsf{id}) = \mathsf{so} - \mathsf{sameop}.$$

We now show

$$\beta^{-1}((\mathsf{so}'_0)^*; ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{id})/\sim'_0); (\mathsf{so}'_0)^*) = \mathsf{lib}(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{sameop}.$$

Take $(e,f) \in \mathsf{lib}(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{sameop}$. Then

$$(\beta(e), \beta(f)) \in \beta(\mathsf{lib}(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{sameop}) = \beta(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{id},$$

which implies the required.

Now take

$$(e,f) \in \beta^{-1}((\mathsf{so}'_0)^*; ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{id})/\sim'_0); (\mathsf{so}'_0)^*).$$

Then

$$(\beta(e), \beta(f)) \in (\mathsf{so}'_0)^*; ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{id})/\sim'_0); (\mathsf{so}'_0)^*.$$

Hence, for some $e'_0, f'_0 \in E'_0$ we have

$$\beta(e) \xrightarrow{(\mathsf{so}'_0)^*} e'_0 \xrightarrow{(\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{id})/\sim'_0} f'_0 \xrightarrow{(\mathsf{so}'_0)^*} \beta(f).$$

Assume $e'_0 \sim'_0 f'_0$; then

$$\beta(e) \xrightarrow{(\mathsf{so}'_0)^*} e'_0 \xrightarrow{\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+) - \mathsf{id}} f'_0 \xrightarrow{(\mathsf{so}'_0)^*} \beta(f).$$

Therefore, for some $e_0, f_0 \in E$ such that $\beta(e_0) = e'_0$ and $\beta(f_0) = f'_0$ we have

$$e \xrightarrow{(\mathsf{so}_0)^*} e_0 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} f_0 \xrightarrow{(\mathsf{so}_0)^*} f.$$

This implies

$$e \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} f.$$

By Proposition C we have $\beta(e) \neq \beta(f)$, which, together with the above, establishes the required.

Consider now the case when $e'_0 \not\sim'_0 f'_0$. Then for some $e'_1, f'_1 \in E'_0$ we have

$$\beta(e) \xrightarrow{(\mathsf{so}'_0)^*} e'_0 \sim'_0 e'_1 \xrightarrow{\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+)} f'_1 \sim'_0 f'_0 \xrightarrow{(\mathsf{so}'_0)^*} \beta(f)$$

and $e'_1 \not\sim'_0 f'_1$.

If $\beta(e) \sim'_0 e'_0$, then $\beta(e) \xrightarrow{(\mathsf{so}'_0)^*|_{\beta_0(E_0)}} \beta(e) \sim'_0 e'_1$.

If $\beta(e) \not\sim'_0 e'_0$, then $\beta(e) \neq e'_0$ and $\beta(e) \xrightarrow{(\mathsf{so}'_0)^*|_{\beta_0(E_0)}} e'_1 \sim'_0 e'_1$.

We can make a similar argument for $\beta(f)$ and f'_0. Thus, for some $e'_0, f'_0 \in E'_0$ we have that

$$\beta(e) \xrightarrow{(\mathsf{so}'_0)^*|_{\beta_0(E_0)}} e'_0 \sim'_0 e'_1 \xrightarrow{\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+)} f'_1 \sim'_0 f'_0 \xrightarrow{(\mathsf{so}'_0)^*|_{\beta_0(E_0)}} \beta(f).$$

Then for some $e_0, e_1, f_0, f_1 \in E_0$ we have

$$\beta(e_0) = e'_0; \quad \beta(e_1) = e'_1; \quad \beta(f_0) = f'_0; \quad \beta(f_1) = f'_1;$$

$$e \xrightarrow{(\mathsf{so}_0)^*} e_0 \sim_0 e_1 \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} f_1 \sim_0 f_0 \xrightarrow{(\mathsf{so}_0)^*} f,$$

and $e_1 \not\sim_0 f_1$. This implies

$$e \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/\sim_0)^+} f.$$

By Proposition C we have $\beta(e) \neq \beta(f)$, which, together with the above, establishes the required. ■

Proposition E: $\mathsf{so}_0 \cup \mathsf{ar}_0/\sim$ *is prefix-finite.* Assume there is $e \in E_0$ such that the set

$$W_e = \{f \mid f \xrightarrow{(\mathsf{so}_0 \cup \mathsf{ar}_0/\sim_0)^+} e\}$$

is infinite. Recall that so_0 partitions events in E_0 into finitely many sessions. Let U_e be the set of events in the same session as e. Since so_0 is prefix-finite and X_0 satisfies CAUSALAR, we can assume without loss of generality that $W_e \cap U_e = \emptyset$ and for every $f \in W_e$ there exists $e_f \notin U_e$ such that

$$f \xrightarrow{((\mathsf{so}_0 \cup \mathsf{ar}_0/\sim_0) \cap (E_0 - U_e)^2)^*} e_f \xrightarrow{(\mathsf{ar}_0/\sim_0)} e.$$

Since ar_0 is prefix-finite, the set $\{e_f \mid f \in W_e\}$ is finite. Hence, there exists e_{f_0} such that

$$\{f \mid f \xrightarrow{((\mathsf{so}_0 \cup \mathsf{ar}_0/\sim_0)^*) \cap (E_0 - U_e)^2} e_{f_0}\}$$

is infinite. Continuing repeatedly as above, after finitely many steps we can eliminate all the sessions except one, which contradicts the prefix-finiteness of so_0. ■

Proposition F: $(X, \beta^{-1}((\mathsf{so}_0' \cup (R/{\sim'}))^+) \models_{\mathsf{CCS}} \mathbb{F}$. We first show that X satisfies the required axioms using the fact that $X_0 \models_{\mathsf{CC}} \mathbb{F}_0$.

- CAUSALVIS′. Consider

$$\begin{aligned}
&(e, f)\\
\in\ &((\mathsf{so} \cup \mathsf{vis} \cup \beta^{-1}((\mathsf{so}_0' \cup (R/{\sim'}))^+))/{\sim})^+ \cap \mathsf{sameobj}(X)\\
=\ &((\mathsf{so} \cup \mathsf{vis} \cup (\mathsf{lib}(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{sameop}))/{\sim})^+ \cap \mathsf{sameobj}(X)\\
\subseteq\ &((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+ \cap \mathsf{sameobj}(X_0),
\end{aligned}$$

  (we used Proposition D). Then by CAUSALVIS for X_0 we have $(e, f) \in \mathsf{vis}_0$. We also know $\mathsf{obj}(e) = \mathsf{obj}(f) \in L$, so that by (43) we get $(e, f) \in \mathsf{vis}$.
- CAUSALAR′. Using Proposition D, we get:

$$\begin{aligned}
&((\mathsf{so} \cup \mathsf{ar} \cup \beta^{-1}((\mathsf{so}_0' \cup (R/{\sim'}))^+))/{\sim})^+\\
=\ &((\mathsf{so} \cup \mathsf{ar} \cup \mathsf{lib}(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{sameop})/{\sim})^+\\
\subseteq\ &((\mathsf{so}_0 \cup \mathsf{ar}_0)/{\sim_0})^+.
\end{aligned}$$

  Hence, the validity of CAUSALAR for X_0 implies the validity of CAUSALAR′ for X.
- PREFIXFINITEAR′. We need to show that $(\mathsf{so} \cup \mathsf{ar} \cup \beta^{-1}((\mathsf{so}_0' \cup (R/{\sim'}))^+))/{\sim}$ is prefix-finite. It is sufficient to prove that $((\mathsf{so} \cup \mathsf{ar} \cup \beta^{-1}((\mathsf{so}_0' \cup (R/{\sim'}))^+))/{\sim})^+$ is prefix-finite. Above we showed that

$$((\mathsf{so} \cup \mathsf{ar} \cup \beta^{-1}((\mathsf{so}_0' \cup (R/{\sim'}))^+))/{\sim})^+ \subseteq ((\mathsf{so}_0 \cup \mathsf{ar}_0)/{\sim_0})^+,$$

  which is prefix-finite by Proposition E.

Let us now show that $X \models \mathbb{F}$. Consider $e \in E$; then $\mathsf{obj}(e) \in L$ and, in particular, $\mathsf{obj}(e) \neq \mathsf{io}$. From (43) it follows that $\mathsf{ctxt}(X_0, e) = \mathsf{ctxt}(X, e)$ and

$$\mathsf{rval}(e) = \mathsf{rval}_0(e) = \mathbb{F}_0(\mathsf{obj}_0(e))(\mathsf{ctxt}(X_0, e)) = \mathbb{F}(\mathsf{obj}(e))(\mathsf{ctxt}(X, e)),$$

as required. ■

Proposition G: $((R/{\sim'}) \cup \mathsf{so}')$ *is prefix-finite.* Similarly to how it was done in Proposition D we get

$$((R/{\sim'}) \cup \mathsf{so}')^+ = \mathsf{so}_0' \cup ((\mathsf{so}_0')^*; (Q/{\sim_0'}); (\mathsf{so}_0')^*).$$

Since so_0' is prefix-finite, it is sufficient to show that for any e, the set

$$\{f \mid f \xrightarrow{Q/{\sim_0'}} e\}$$

is finite. Since $Q = \beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id}$, this follows from Proposition E. ■

From Propositions C, F, G and (44) we then get $X' \models_{\mathsf{CCS}} [\omega \mapsto F]$ and

$$(\beta, [\![\{C_o\}_{o \in O}]\!](obj)) : X \to X'. \tag{46}$$

Let

$$\begin{aligned} \mathsf{vis}'_0 &= \mathsf{client}(\mathsf{vis}_0) \cup ((\mathsf{so}' \cup (\beta(\mathsf{vis}) - \mathsf{id}) \cup R)/{\sim'})^+; \\ \mathsf{ar}'_0 &= \mathsf{client}(\mathsf{ar}_0) \cup ((\mathsf{so}' \cup (\beta(\mathsf{ar}) - \mathsf{id}) \cup R)/{\sim'})^+; \\ X'_0 &= (H'_0, \mathsf{vis}'_0, \mathsf{ar}'_0). \end{aligned}$$

Proposition H: *For* $S = \beta_0(((\mathsf{so}_0 \cup \mathsf{ar}_0)/{\sim_0})^+) - \mathsf{id}$,

- $S/{\sim'_0}$ *is transitive;*
- $(\mathsf{so}'_0 \cup (S/{\sim'_0}))^+ = \mathsf{so}'_0 \cup ((\mathsf{so}'_0)^*; (S/{\sim'_0}); (\mathsf{so}'_0)^*)$*;*
- $\mathsf{so}'_0 \cup (S/{\sim'_0})$ *is acyclic.*

Proved the same way as Propositions A, B, C. ■

Proposition I: $X'_0 \in \mathsf{Exec}$, $X'_0 \models_{\mathsf{CCS}} \mathbb{F}'_0$ *and* X'_0 *satisfies the following version of* EVENTUAL*:*

$$\begin{aligned} &\forall e \in E'_0.\, \mathsf{obj}'_0(e) \neq \omega \implies \\ &\qquad \neg(\exists \textit{ infinitely many } f \in E'_0.\, \mathsf{sameobj}(X'_0)(e, f) \wedge \neg(e \xrightarrow{\mathsf{vis}'_0} f)). \end{aligned} \tag{47}$$

Since ar_0 and $X'.\mathsf{ar}$ are prefix-finite, we have that so is ar'_0. We now show that X'_0 satisfies the required axioms.

- CAUSALAR. We have:

$$\begin{aligned} &((\mathsf{so}'_0 \cup \mathsf{ar}'_0)/{\sim'_0})^+ \\ &= ((\mathsf{so}'_0 \cup \mathsf{client}(\mathsf{ar}_0) \cup ((\mathsf{so}' \cup (\beta(\mathsf{ar}) - \mathsf{id}) \cup R)/{\sim'})^+)/{\sim'_0})^+ \\ &= ((\mathsf{so}'_0 \cup \beta_0(\mathsf{client}(\mathsf{ar}_0)) \cup \\ &\quad ((\mathsf{so}' \cup (\beta(\mathsf{ar}) - \mathsf{id}) \cup \\ &\qquad \mathsf{lib}'((\mathsf{so}'_0 \cup ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim'_0}))^+))/{\sim'})^+)/{\sim'_0})^+ \\ &\subseteq \mathsf{so}'_0 \cup ((\beta_0(\mathsf{client}(\mathsf{ar}_0)) \cup (\beta_0(((\mathsf{so}_0 \cup \mathsf{ar}_0)/{\sim_0})^+) - \mathsf{id}))/{\sim'_0})^+ \\ &= (\mathsf{so}'_0 \cup ((\beta_0(((\mathsf{so}_0 \cup \mathsf{ar}_0)/{\sim_0})^+) - \mathsf{id})/{\sim'_0}))^+. \end{aligned}$$

  Let

$$S = \beta_0(((\mathsf{so}_0 \cup \mathsf{ar}_0)/{\sim_0})^+) - \mathsf{id}.$$

  It is thus enough to show that the relation $\mathsf{so}'_0 \cup (S/{\sim'_0})$ is acyclic. This follows from Proposition H.
- CAUSALVIS. We need to prove

$$((\mathsf{so}'_0 \cup \mathsf{vis}'_0)/{\sim'_0})^+ \cap \mathsf{sameobj}(X'_0) \subseteq \mathsf{vis}'_0.$$

Using Proposition B we obtain:

$$
\begin{aligned}
&\mathsf{vis}_0' \\
&= \mathsf{client}(\mathsf{vis}_0) \cup ((\mathsf{so}' \cup (\beta(\mathsf{vis}) - \mathsf{id}) \cup R)/{\sim'})^+ \\
&= \mathsf{client}(\mathsf{vis}_0) \cup ((\mathsf{so}' \cup (\beta(\mathsf{vis}) - \mathsf{id}) \cup \\
&\qquad\qquad \mathsf{lib}'((\mathsf{so}_0' \cup ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}))^+))/{\sim'})^+ \\
&= \mathsf{client}(\mathsf{vis}_0) \cup ((\mathsf{lib}'((\mathsf{so}_0' \cup ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}))^+))/{\sim'})^+ \\
&= \mathsf{client}(\mathsf{vis}_0) \cup \mathsf{lib}'((\mathsf{so}_0' \cup ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}))^+) \\
&= \mathsf{client}(\mathsf{vis}_0) \cup \mathsf{lib}'(\mathsf{so}_0' \cup ((\mathsf{so}_0')^*; ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}); (\mathsf{so}_0')^*));
\end{aligned}
$$

$$
\begin{aligned}
&((\mathsf{so}_0' \cup \mathsf{vis}_0')/{\sim_0'})^+ \\
&= ((\mathsf{so}_0' \cup \mathsf{client}(\mathsf{vis}_0) \cup \\
&\qquad \mathsf{lib}'((\mathsf{so}_0' \cup ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}))^+))/{\sim_0'})^+ \\
&\subseteq ((\mathsf{so}_0' \cup \mathsf{client}(\mathsf{vis}_0) \cup (\mathsf{so}_0' \cup ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}))^+)/{\sim_0'})^+ \\
&= (\mathsf{so}_0' \cup ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}))^+. \\
&= \mathsf{so}_0' \cup ((\mathsf{so}_0')^*; ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}); (\mathsf{so}_0')^*).
\end{aligned}
$$

Hence, we need to show

$$
(\mathsf{so}_0' \cup ((\mathsf{so}_0')^*; ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}); (\mathsf{so}_0')^*)) \cap \mathsf{sameobj}(X_0') \subseteq
$$
$$
\mathsf{client}(\mathsf{vis}_0) \cup \mathsf{lib}'(\mathsf{so}_0' \cup ((\mathsf{so}_0')^*; ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}); (\mathsf{so}_0')^*)).
$$

Let

$$
\begin{aligned}
(e', f') \in\ & (\mathsf{so}_0' \cup ((\mathsf{so}_0')^*; ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}); (\mathsf{so}_0')^*)) \\
& \cap \mathsf{sameobj}(X_0').
\end{aligned}
$$

Then either $e', f' \in E'$ or $e', f' \in E_0' - E'$. The former case is obvious. Consider the case when $e', f' \in E_0'' - E'$. If $(e', f') \in \mathsf{so}_0' \cap \mathsf{sameobj}(X_0')$, then $(e', f') \in \mathsf{so}_0 \cap \mathsf{sameobj}(X_0) \subseteq \mathsf{vis}_0$, and thus $(e', f') \in \mathsf{client}(\mathsf{vis}_0)$. The remaining case is that

$$
(e', f') \in ((\mathsf{so}_0')^*; ((\beta_0(((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+) - \mathsf{id})/{\sim_0'}); (\mathsf{so}_0')^*) \cap \mathsf{sameobj}(X_0').
$$

As in the proof of Proposition D, we can show that this implies

$$
e' \xrightarrow{((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+} f'.
$$

and, hence,

$$
(e', f') \in ((\mathsf{so}_0 \cup \mathsf{vis}_0)/{\sim_0})^+ \cap \mathsf{sameobj}(X_0') \subseteq \mathsf{vis}_0,
$$

as required.

– (47). Assume that for some $e \in E_0'$ such that $\mathsf{obj}_0'(e) \neq \omega$ we have

$$
\exists \text{ infinitely many } f \in E_0'.\, \mathsf{sameobj}(X_0')(e, f) \wedge \neg(e \xrightarrow{\mathsf{vis}_0'} f).
$$

Then all fs are in $E_0 - E$ and, correspondingly, $\neg(e \xrightarrow{\mathsf{vis}_0} f)$, yielding a contradiction with EVENTUAL for X_0.

Finally, let us show that $X'_0 \models \mathbb{F}'_0$. Consider $e \in E'_0$ such that $\mathsf{obj}'_0(e) \neq \mathsf{io}$. Given (43), it is easy to see that $\mathsf{ctxt}(X'_0, e) = \mathsf{ctxt}(X_0, e)$, if $\mathsf{obj}'_0(e) \neq \omega$, and $\mathsf{ctxt}(X'_0, e) = \mathsf{ctxt}(X', e)$, otherwise, from which the required follows. ■

Proposition J: *There exists* $X''_0 \in \mathsf{Exec}$ *such that*

$$X''_0 \models_{\mathsf{CC}} \mathbb{F}'_0 \quad \wedge \quad X''_0.H = H'_0 \quad \wedge \quad X''_0.\mathsf{vis} \supseteq X'_0.\mathsf{vis} \quad \wedge \quad X''_0.\mathsf{ar} \supseteq X'_0.\mathsf{ar}.$$

By Proposition I we have $X'_0 \models_{\mathsf{CCS}} \mathbb{F}'_0$ and (47). Hence, the following holds for $Y = X'_0$:

$$\begin{aligned} Y \models_{\mathsf{CCS}} \mathbb{F}'_0 \quad \wedge \quad Y.H = H'_0 \quad \wedge \quad Y.\mathsf{vis} \supseteq X'_0.\mathsf{vis} \quad \wedge \quad Y.\mathsf{ar} \supseteq X'_0.\mathsf{ar} \\ \wedge \quad Y|_{\{e \mid \mathsf{obj}'_0(e) \neq \omega\}} = X'_0|_{\{e \mid \mathsf{obj}'_0(e) \neq \omega\}}; \end{aligned} \tag{48}$$

$$\begin{aligned} \forall e \in E'_0.\, \mathsf{obj}'_0(e) \neq \omega \implies \\ \neg(\exists \text{ infinitely many } f \in E'_0.\, \mathsf{sameobj}(X'_0)(e, f) \wedge \neg(e \xrightarrow{Y.\mathsf{vis}} f)); \end{aligned} \tag{49}$$

$$(\beta, [\![\{C_o\}_{o \in O}]\!](obj)) : X \to Y|_{\{e \mid \mathsf{obj}'_0(e) = \omega\}}. \tag{50}$$

Consider an arbitrary execution Y satisfying these conditions. If Y satisfies EVENTUAL, then we can let $X''_0 = Y$. Otherwise, there exist events $e \in E'_0$ such that

$$\exists \text{ infinitely many } f \in E'_0.\, \mathsf{obj}'_0(e) = \mathsf{obj}'_0(f) \wedge \neg(e \xrightarrow{Y.\mathsf{vis}} f). \tag{51}$$

From (49) it follows that for all such events e we have $\mathsf{obj}'_0(e) \neq \omega$. Out of all events e satisfying (51), let us take an event e_0 that is minimal in $((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^+$, i.e., such that there does not exist an event $e \in E'_0$ for which (51) holds and

$$e \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^+} e_0.$$

Let

$$\begin{gathered} J_0 = \{e \mid e \sim'_0 e_0 \wedge \text{(51) holds}\}; \\ J = \{f \in E'_0 \mid \exists e \in J_0.\, \mathsf{obj}'_0(e) = \mathsf{obj}'_0(f) \wedge \neg(e \xrightarrow{Y.\mathsf{vis}} f)\}. \end{gathered}$$

As in Proposition E, we can show that $((\mathsf{so}'_0 \cup Y.\mathsf{ar})/\sim'_0)^+$ is prefix-finite. Hence, there are at most finitely many $f \in J$ such that

$$\exists e \in J_0.\, f \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{ar})/\sim'_0)^+} e.$$

Let J_1 be the subset of J resulting from removing all such f as well as all f such that $f \sim'_0 e_0$; then J_1 is infinite.

Next, let

$$G = \{e \in E'_0 \mid e \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^+} e_0 \vee (e \sim'_0 e_0 \wedge e \notin J_0)\};$$

then G is finite and (51) is false for every $e \in G$ by (47) and the choice of e_0 and J_0. Let

$$G' = \{g \in E'_0 \mid \exists e \in G.\, \mathsf{obj}'_0(e) = \mathsf{obj}'_0(g) \wedge \neg(e \xrightarrow{Y.\mathsf{vis}} g)\};$$

then G' is finite. Hence, there are only finitely many $f \in J_1$ such that

$$\exists g \in G'.\, f \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/{\sim'_0})^+ \cup {\sim'_0}} g.$$

Let J_2 be the subset of J_1 resulting from removing all such f. Then J_2 is infinite.

Since $Y.\mathsf{vis} \supseteq X'_0.\mathsf{vis} \supseteq \beta(\mathsf{vis})$ and X satisfies EVENTUAL, there are at most finitely many $f \in J_2$ such that $\mathsf{obj}(\beta^{-1}(f)) \cap \mathsf{obj}(\beta^{-1}(J_0)) \neq \emptyset$. Let J' be the set of such f and

$$J_3 = J_2 - \{f' \mid \exists f \in J'.\, f' \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/{\sim'_0})^+ \cup {\sim'_0}} f\}.$$

Then J_3 is still infinite. Finally, take some element $e \in J_3$ that is minimal in $((\mathsf{so}'_0 \cup Y.\mathsf{vis})/{\sim'_0})^+$, i.e.,

$$\neg\exists e' \in J_2.\, e' \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/{\sim'_0})^+} e.$$

Let

$$J_4 = J_3 - \{e' \mid e \sim'_0 e'\}. \tag{52}$$

To summarise, J_4 includes all but finitely many $f \in E'_0$ such that

$$\exists e \in J_0.\, \mathsf{obj}'_0(e) = \mathsf{obj}'_0(f) \wedge \neg(e \xrightarrow{Y.\mathsf{vis}} f)$$

and for any $f \in J_4$ we have:

$$\mathsf{obj}(\beta^{-1}(f)) \cap \mathsf{obj}(\beta^{-1}(J_0)) \neq \emptyset; \tag{53}$$

$$\neg(f \sim'_0 e_0); \tag{54}$$

$$\neg\exists e \in J_0.\, f \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{ar})/{\sim'_0})^+} e; \tag{55}$$

$$\begin{aligned}\forall e, g \in E'_0.\, (\mathsf{obj}'_0(e) = \mathsf{obj}'_0(g) \wedge (e \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/{\sim'_0})^+} e_0 \vee (e \sim'_0 e_0 \wedge e \notin J_0)) \\ \wedge\, f \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/{\sim'_0})^+ \cup {\sim'_0}} g) \implies e \xrightarrow{Y.\mathsf{vis}} g.\end{aligned} \tag{56}$$

Let

$$U = \{(e, f) \mid e \in J_0 \wedge f \in J_4 \wedge \mathsf{obj}'_0(e) = \mathsf{obj}'_0(f)\}.$$
$$Y' = (H'_0, Y.\mathsf{vis} \cup U, (Y.\mathsf{ar} \cup U)^+).$$

We now show that $Y' \models_{\mathsf{CCS}} \mathbb{F}'_0$. We start by showing that Y' satisfies CAUSALAR. Assume there is a cycle in

$$(\mathsf{so}'_0 \cup (Y.\mathsf{ar} \cup U)^+)/{\sim'_0}.$$

Then there is also a cycle in

$$(\mathsf{so}'_0 \cup Y.\mathsf{ar} \cup U)/{\sim'_0}.$$

Since $Y \models_{\mathsf{CCS}} \mathbb{F}'_0$, the cycle has edges from $U/{\sim'_0}$. Without loss of generality we can assume that there is a single such edge on the cycle (the cycle can be transformed to ensure this), so that the cycle is of the form

$$e' \xrightarrow{U/\sim'_0} f' \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{ar})/\sim'_0)^+} e'$$

for some $e', f' \in E'_0$. Then by (54) we have

$$f \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{ar})/\sim'_0)^+} e_0$$

for some $f \in J_4$, which contradicts (55). Hence, Y' satisfies CAUSALAR.

We now prove that Y' satisfies CAUSALVIS. Consider $e, g \in E'_0$ such that $\mathsf{obj}'_0(e) = \mathsf{obj}'_0(g)$ and

$$e \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis} \cup U)/\sim'_0)^+} g.$$

If the path from e to g contains only edges from $(\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0$, then, since Y satisfies CAUSALVIS, $(e, g) \in Y.\mathsf{vis}$ and we are done. Now assume that the path contains edges from $U/\sim'_0$. Without loss of generality we can assume that there is a single such edge on the path, so that the path is of the form

$$e \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^*} e' \xrightarrow{U/\sim'_0} f' \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^*} g$$

for some $e', f' \in E'_0$. Hence, we have

$$e \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^*} e' \sim'_0 e'' \xrightarrow{U} f'' \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^+ \cup \sim'_0} g$$

for some $e'' \in J_0$ and $f'' \in J_4$. If $e \not\sim'_0 e'$, then the above implies

$$e \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^+} e_0$$

and by (56) we have $(e, g) \in Y.\mathsf{vis}$. If $e \sim'_0 e'$ and $e \notin J_0$, then by (56) we again get $(e, g) \in Y.\mathsf{vis}$. The only remaining case is $e \in J_0$, so that

$$e \sim'_0 e'' \xrightarrow{U} f'' \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^+ \cup \sim'_0} g.$$

Assume $(e, g) \notin Y.\mathsf{vis}$. We show that $g \in J_4$, so that $(e, g) \in U \subseteq Y'.\mathsf{vis}$. This follows from the following claims:

- We cannot have $g \sim'_0 e_0$. For in this case we would have

  $$e_0 \sim'_0 e'' \xrightarrow{U} f'' \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^+ \cup \sim'_0} g \sim'_0 e_0.$$

  By (54), this implies

  $$f'' \xrightarrow{((\mathsf{so}'_0 \cup Y.\mathsf{vis})/\sim'_0)^+} e_0,$$

  which contradicts (55).

– We cannot have

$$g \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{ar})/\sim_0')^+} e'''$$

for any $e''' \in J_0$. For in this case we would have

$$e''' \sim_0' e'' \xrightarrow{U} f'' \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{vis})/\sim_0')^+ \cup \sim_0'} g \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{ar})/\sim_0')^+} e'''$$

By (54), this implies

$$f'' \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{vis})/\sim_0')^+} e''',$$

which contradicts (55). Hence, $g \in J_1$.

– There cannot exist e_1, e_2 such that

$$(e_1 \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{vis})/\sim_0')^+} e_0 \vee (e_1 \sim_0' e_0 \wedge e_1 \notin J_0)) \wedge$$
$$\mathsf{obj}_0'(e_1) = \mathsf{obj}_0'(e_2) \wedge \neg(e_1 \xrightarrow{Y.\mathsf{vis}} e_2) \wedge g \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{vis})/\sim_0')^+ \cup \sim_0'} e_2.$$

For in this case we would have

$$f'' \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{vis})/\sim_0')^+ \cup \sim_0'} g \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{vis})/\sim_0')^+ \cup \sim_0'} e_2,$$

which implies

$$f'' \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{vis})/\sim_0')^+ \cup \sim_0'} e_2.$$

Hence,

$$(e_1 \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{vis})/\sim_0')^+} e_0 \vee (e_1 \sim_0' e_0 \wedge e_1 \notin J_0)) \wedge$$
$$\mathsf{obj}_0'(e_1) = \mathsf{obj}_0'(e_2) \wedge \neg(e_1 \xrightarrow{Y.\mathsf{vis}} e_2) \wedge f'' \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{vis})/\sim_0')^+ \cup \sim_0'} e_2.$$

which contradicts $f'' \in J_4$. Hence, $g \in J_2$.

Since

$$f'' \xrightarrow{((\mathsf{so}_0' \cup Y.\mathsf{vis})/\sim_0')^+ \cup \sim_0'} g \wedge f'' \in J_4,$$

we also have $g \in J_3$ and $g \in J_4$. This shows that Y' indeed satisfies CAUSALVIS.

Finally, from (53) and (50) we get that

$$(\beta, [\![\{C_o\}_{o \in O}]\!](obj)) : X \to Y'|_{\{e \mid \mathsf{obj}_0'(e) = \omega\}}.$$

From this and $Y \models \mathbb{F}_0'$, by Corollary 34 and Lemma 35 we get $Y' \models \mathbb{F}_0'$.

We have thus shown how to convert an execution Y satisfying (48) and (49) into an execution Y' satisfying the same properties, but containing fewer events invalidating EVENTUAL. Continuing this ad infinitum, in the limit we obtain the desired X_0''. In particular, the prefix-finiteness of $X_0''.\mathsf{vis}$ and $X_0''.\mathsf{ar}$ is ensured by the removal of finitely many elements from J_3 in (52). ■

By (42) we have $H'_0 \in \langle P^2_{\mathsf{C}} \rangle obj'_0$; therefore, by Proposition J, $X''_0 \in [\![P^2_{\mathsf{C}}]\!]([\,], obj'_0, \mathbb{F}'_0)$. We have also previously established $\mathsf{observ}(H_0) = \mathsf{observ}(H'_0)$. We have thus shown the following:

$\forall obj_0 : (\mathsf{dom}(\Delta) \cup \{x_{\mathsf{io}}\} \cup \{x_j \mid j = 1..m\}) \to_{\mathsf{inj}} \mathsf{Obj}.$
$\forall \mathbb{F}_0 : (\mathsf{range}(obj_0) - \{\mathsf{io}\}) \to \mathsf{Spec}.$
$\forall \omega \in \mathsf{Obj} - \mathsf{range}(obj_0).\, (obj_0(x_{\mathsf{io}}) = \mathsf{io}) \wedge (\forall j = 1..m.\, \mathbb{F}_0(obj(x_j)) = [\![T_j]\!][\,]) \implies$
$\forall X_0 \in [\![P^1_{\mathsf{C}}]\!]([\,], obj_0, \mathbb{F}_0).\, \exists X''_0.$
$X''_0 \in [\![P^2_{\mathsf{C}}]\!]([\,], obj_0|_{\mathsf{OVar} - \{x_j \mid j=1..m\}}[x \mapsto \omega], \mathbb{F}_0|_{\mathsf{Obj} - \mathsf{range}(obj_0|_{\{x_j \mid j=1..m\}})}[\omega \mapsto [\![D]\!][\,]]).$
$\wedge\, \mathsf{observ}(X_0.H) = \mathsf{observ}(X''_0.H).$

It is easy to see that this implies the statement of Theorem 42. □

D.5 Completeness

Let us again fix an object variable environment $\{x_j : O_j \mid j = 1..m\}$ and a collection of commands

$$\{x_j : O_j \mid j = 1..m\} \mid v_{\mathsf{in}}, v_{\mathsf{out}} \vdash C_o, \quad o \in O.$$

LEMMA 43 *Consider executions X, X', X'_0, Y', object ω and abstraction (β, ρ) such that*

$$X' = X'_0|_{\{e \mid X'_0.\mathsf{obj}(e) = \omega\}} \quad \wedge \quad Y' = X'_0|_{\{e \mid X'_0.\mathsf{obj}(e) \neq \omega\}} \quad \wedge \quad X.E \cap Y'.E = \emptyset \quad \wedge$$
$$X.\mathsf{obj}(X.E) \cap X'_0.\mathsf{obj}(X'_0.E) = \emptyset \quad \wedge \quad (\beta, \rho) : X \to X'.$$

Let

$$\begin{aligned} X_0 = ((&Y'.E \uplus X.E, Y'.\mathsf{label} \uplus X.\mathsf{label},\\ &Y'.\mathsf{so} \cup X.\mathsf{so} \cup \{(e,f) \mid e \in Y'.E \wedge f \in X.E \wedge (e, \beta(f)) \in X'_0.\mathsf{so}\} \cup\\ &\quad \{(e,f) \mid e \in X.E \wedge f \in Y'.E \wedge (\beta(e), f) \in X'_0.\mathsf{so}\},\\ &Y'.{\sim} \cup X.{\sim} \cup \{(e,f) \mid e \in Y'.E \wedge f \in X.E \wedge (e, \beta(f)) \in X'_0.{\sim}\} \cup\\ &\quad \{(e,f) \mid e \in X.E \wedge f \in Y'.E \wedge (\beta(e), f) \in X'_0.{\sim}\}),\\ &Y'.\mathsf{vis} \cup X.\mathsf{vis}, Y'.\mathsf{ar} \cup X.\mathsf{ar}). \end{aligned} \tag{57}$$

If X'_0 and X are causally consistent, then X_0 is a causally consistent execution.

PROOF. We first show CAUSALAR. Then it is easy to check that X_0 is indeed an execution. Let $\beta' : X_0.E \to X'_0.E$ be defined as follows:

$$\beta'(e) = (\text{if } e \in X.E \text{ then } \beta(e) \text{ else } e).$$

From $(\beta, \rho) : X \to X'$ and the constraint on ρ in the assumptions of the theorem, we get that for all $e, f \in X_0.E$:

$$\begin{aligned} (e,f) \in X_0.\mathsf{ar} &\implies (\beta'(e), \beta'(f)) \in X'_0.\mathsf{ar} \cup (\mathsf{id} \cap (X'.E)^2);\\ (e,f) \in X_0.\mathsf{so} &\implies (\beta'(e), \beta'(f)) \in X'_0.\mathsf{so} \cup (\mathsf{id} \cap (X'.E)^2);\\ (e,f) \in X_0.{\sim} &\iff (\beta'(e), \beta'(f)) \in X'_0.{\sim}. \end{aligned}$$

Hence,

$$\begin{array}{l}\forall e, f.\, (e,f) \in (X_0.\mathsf{so} \cup X_0.\mathsf{ar})/(X_0.\sim) \\ \quad \Longrightarrow\ (\beta'(e), \beta'(f)) \in (X_0'.\mathsf{so} \cup X_0'.\mathsf{ar})/(X_0'.\sim) \cup (\mathsf{id} \cap (X'.E)^2).\end{array} \tag{58}$$

Assume there is a cycle in $(X_0.\mathsf{so} \cup X_0.\mathsf{ar})/(X_0.\sim)$. Consider first the case when β' maps all events on the cycle to the same event in X_0'; then all of these events belong to $X.E$. Consider two adjacent events $e, f \in X$ on the cycle. Since $\beta(e) = \beta(f)$, we have $(e,f) \in X.\sim$ and e and f must be related by $X.\mathsf{so}$. But then $(e,f) \in X.\mathsf{so} \cup X.\mathsf{ar}$ and, since X is causally consistent, it must be that $(e,f) \in X.\mathsf{so}$. Thus, the existence of the cycle contradicts the acyclicity of $X.\mathsf{so}$.

If β' maps at least two events on the cycle to different events in X_0', then (58) implies that β' maps the cycle to one in $(X_0.\mathsf{so} \cup X_0.\mathsf{ar})/(X_0.\sim)$, contradicting causal consistency of X_0.

We now show CAUSALVIS. Consider $(e,f) \in (X_0.\mathsf{so} \cup X_0.\mathsf{vis})/(X_0.\sim)$. Similarly to the above, we can show that either

$$(\beta'(e), \beta'(f)) \in (X_0'.\mathsf{so} \cup X_0'.\mathsf{vis})/(X_0'.\sim)$$

or $\beta'(e) = \beta'(f)$ and $(e,f) \in X.\mathsf{so}$. Now consider

$$(e_0, f_0) \in ((X_0.\mathsf{so} \cup X_0.\mathsf{vis})/(X_0.\sim))^+ \wedge \mathsf{sameobj}(X_0).$$

Then either

$$(\beta'(e_0), \beta'(f_0)) \in ((X_0'.\mathsf{so} \cup X_0'.\mathsf{vis})/(X_0'.\sim))^+ \wedge \mathsf{sameobj}(X_0')$$

or $\beta'(e_0) = \beta'(f_0)$ and $(e_0, f_0) \in X.\mathsf{so}$. In the latter case we have $(e_0, f_0) \in X.\mathsf{vis}$ since X is causally consistent. In the former case we get $(\beta'(e_0), \beta'(f_0)) \in X_0'.\mathsf{vis}$ since X_0' is causally consistent. If $\beta'(e_0), \beta'(f_0) \in Y'.E$, then $(e_0, f_0) = (\beta'(e_0), \beta'(f_0))$ and $(e_0, f_0) \in X_0.\mathsf{vis}$. If $\beta'(e_0), \beta'(f_0) \in X'.E$, then $(\beta, \rho) : X \to X'$ implies $(e_0, f_0) \in X.\mathsf{vis} \subseteq X_0.\mathsf{vis}$.

Finally, EVENTUAL for X_0 trivially follows from EVENTUAL for X and X_0'. □

LEMMA 44 *Let*

$$[\![\emptyset \vdash \mathsf{let}\ \{x_j = \mathsf{new}\ T_j\}_{j=1..m}\ \mathsf{in}\ \{o(v_{\mathsf{in}}) : v_{\mathsf{out}} = \mathsf{atomic}\ \{C_o\}\}_{o \in O} : O]\!][\,] = F$$

and consider $X' \in \mathsf{Exec}(\omega)$ *such that* $X' \models_{\mathsf{CC}} [\omega \mapsto F]$. *Further, consider* $obj \in [\{x_j \mid j = 1..m\} \to_{\mathsf{inj}} \mathsf{Obj}]$ *and let* $\mathbb{F} \in [\mathsf{range}(obj) \to \mathsf{Spec}]$ *be such that* $\forall j = 1..m.\, \mathbb{F}(obj(x_j)) = [\![T_j]\!][\,]$. *Then there exists* $X \in \mathsf{Exec}$ *such that* $X \models_{\mathsf{CC}} \mathbb{F}$ *and* $(\beta, [\![\{C_o\}_{o \in O}]\!](obj)) : X \to X'$ *for some* β.

PROOF. We construct the required execution X as a limit of a sequence of executions constructed for fragments of X'.

Consider a finite set of events $E \subseteq X'.E$ that is closed under $X'.\mathsf{so}^{-1}$ and $X'.\mathsf{ar}^{-1}$ and let $Y' = X'|_E$. It is easy to see that $Y' \models_{\mathsf{CC}} [\omega \mapsto F]$. Assume we have constructed $Y \in \mathsf{Exec}$ such that

$$(\beta_0, \rho) : Y \to Y' \quad \wedge \quad Y \models_{\mathsf{CC}} \mathbb{F}$$

for some β_0. If $Y' = X'$, we are done. Otherwise, let us choose an event $e_0 \in X'.E - E$ that is minimal in $X'.\mathsf{so} \cup X'.\mathsf{ar}$, i.e.,

$$\neg\exists e \in X'.E - E.\, e \xrightarrow{X'.\mathsf{so} \cup X'.\mathsf{ar}} e_0.$$

Let $X'_0 = X'|_{E \cup \{e_0\}}$; then $X'_0 \models_{\mathsf{CC}} [\omega \mapsto F]$. Note that e_0 that does not have successors in $X'_0.\mathsf{so} \cup X'_0.\mathsf{ar}$. We now construct $X_0 \in \mathsf{Exec}$ such that

$$(\beta_0, \rho) : X_0 \to X'_0 \quad \wedge \quad X_0 \models_{\mathsf{CC}} \mathbb{F}.$$

Let $\rho = [\![\{C_o\}_{o \in O}]\!](obj)$, $N = \mathsf{ctxt}(X'_0, e_0)$ and $a = X'_0.\mathsf{rval}(e_0)$. Since $X'_0 \models_{\mathsf{CC}} [\omega \mapsto F]$, we have $F(N) = a$. Therefore, by Propositions 17 and 20, we get

$$\exists \beta, Z, Z'.\, (N, e_0) \hookrightarrow Z' \wedge (\beta, \rho) : Z \to Z' \wedge Z \models_{\mathsf{CC}} \mathbb{F} \wedge a = Z'.\mathsf{rval}(e).$$

Let $W' = Z'|_{N.E}$ and $W_1 = Z|_{\beta^{-1}(N.E)}$. From $(\beta, \rho) : Z \to Z'$ and $Z \models_{\mathsf{CC}} \mathbb{F}$ it follows that

$$(\beta|_{\beta^{-1}(N.E)}, \rho) : W_1 \to W' \quad \wedge \quad W_1 \models_{\mathsf{CC}} \mathbb{F}.$$

Let

$$W_2 = ((\beta_0^{-1}(N.E), Y.\mathsf{label}|_{\beta_0^{-1}(N.E)}, Y.\mathsf{so}|_{\{(e,f) | \beta_0(e) = \beta_0(f)\}}, Y.{\sim}|_{\{(e,f) | \beta_0(e) = \beta_0(f)\}}),$$
$$Y.\mathsf{vis}|_{\beta_0^{-1}(N.E)}, Y.\mathsf{ar}|_{\beta_0^{-1}(N.E)}).$$

From $(\beta_0, \rho) : Y \to Y'$ and $Y \models_{\mathsf{CC}} \mathbb{F}$ it follows that

$$(\beta_0|_{\beta_0^{-1}(N.E)}, \rho) : W_2 \to W' \quad \wedge \quad W_2 \models_{\mathsf{CC}} \mathbb{F}.$$

Then by Theorem 19 we get that $W_2 \approx_\pi W_1$ for some $\pi : W_2.E \to_{\mathsf{bij}} W_1.E$ such that

$$\forall f \in W_2.E.\, \beta_0(f) = \beta(\pi(f)).$$

Without loss of generality we can assume that $Y.E \cap Z.E = \emptyset$. Let

$$\begin{array}{l}
X_0 = \\
((Y.E \uplus \beta^{-1}(e_0), Y.\mathsf{label} \uplus Z.\mathsf{label}|_{\beta^{-1}(e_0)}, \\
\quad Y.\mathsf{so} \cup Z.\mathsf{so}|_{\beta^{-1}(e_0)} \cup \{(e,f) \mid e \in Y.E \wedge f \in \beta^{-1}(e_0) \wedge (\beta_0(e), e_0) \in X'_0.\mathsf{so}\} \\
\quad Y.{\sim} \cup Z.{\sim}|_{\beta^{-1}(e_0)} \cup \{(e,f) \mid e \in Y.E \wedge f \in \beta^{-1}(e_0) \wedge (\beta_0(e), e_0) \in X'_0.{\sim}\} \cup \\
\qquad \{(e,f) \mid e \in \beta^{-1}(e_0) \wedge f \in Y.E \wedge (e_0, \beta_0(f)) \in X'_0.{\sim}\}, \\
\quad Y.\mathsf{vis} \cup Z.\mathsf{vis}|_{\beta^{-1}(e_0)} \cup \{(e,f) \mid e \in W_2.E \wedge f \in \beta^{-1}(e_0) \wedge (\pi(e), f) \in Z.\mathsf{vis}\}, \\
\quad R^+),
\end{array}$$

where

$$\begin{array}{rl}
R = Y.\mathsf{ar} \cup Z.\mathsf{ar}|_{\beta^{-1}(e_0)} \cup \{(e,f) \mid e \in W_2.E \wedge f \in \beta^{-1}(e_0) \wedge (\pi(e), f) \in Z.\mathsf{ar}\} \\
\qquad\qquad \cup \{(e,f) \mid e, f \in W_2.E \wedge (\pi(e), \pi(f)) \in W_1.\mathsf{ar}\}.
\end{array}$$

Let $\beta' : X_0.E \to X'_0.E$ be defined as follows:

$$\beta'(e) = (\text{if } e \in Y.E \text{ then } \beta_0(e) \text{ else } e_0).$$

We first show $\beta'(X_0.\mathsf{vis}) - \mathsf{id} \subseteq X_0'.\mathsf{vis}$ and $\beta'(X_0.\mathsf{ar}) - \mathsf{id} \subseteq X_0'.\mathsf{ar}$. Since $(\beta_0, \rho) : Y \to Y'$, we have

$$\beta'(Y.\mathsf{vis}) - \mathsf{id} \subseteq Y'.\mathsf{vis} \subseteq X_0'.\mathsf{vis}.$$

Now consider $e \in W_2.E$ and $f \in \beta^{-1}(e_0)$ such that $(\pi(e), f) \in Z.\mathsf{vis}$ and

$$\beta_0(e) = \beta'(e) \neq \beta'(f) = e_0.$$

We have

$$(\beta(\pi(e)), \beta(f)) = (\beta_0(e), e_0).$$

Then since $(\beta, \rho) : Z \to Z'$, we get

$$(\beta'(e), \beta'(f)) = (\beta_0(e), e_0) \in Z'.\mathsf{vis} \subseteq X_0'.\mathsf{vis}.$$

This shows $\beta'(X_0.\mathsf{vis}) - \mathsf{id} \subseteq X_0'.\mathsf{vis}$. We can similarly show $\beta'(X_0.\mathsf{ar}) - \mathsf{id} \subseteq X_0'.\mathsf{ar}$.

We have $(\beta', \rho) : X_0.H \to X_0'.H$ by the construction of X_0. To show $(\beta', \rho) : X_0 \to X_0'$ it remains to establish ${\beta'}^{-1}(X_0'.\mathsf{vis}) \cap \mathsf{sameobj}(X_0) \subseteq X_0.\mathsf{vis}$. Since $(\beta_0, \rho) : Y \to Y'$, we have

$${\beta'}^{-1}(Y'.\mathsf{vis}) \cap \mathsf{sameobj}(Y) \subseteq Y.\mathsf{vis} \subseteq X_0.\mathsf{vis}.$$

Now consider

$$e' \in Y'.E, \qquad e \in \beta_0^{-1}(e') \subseteq Y.E, \qquad f \in \beta^{-1}(e_0) \subseteq Z.E$$

such that $(e', e_0) \in X_0'.\mathsf{vis}$ and $Y.\mathsf{obj}(e) = Z.\mathsf{obj}(f)$. We need to show that $(e, f) \in X_0.\mathsf{vis}$. Since $(e', e_0) \in X_0'.\mathsf{vis}$, we have $e' \in Z'.E$, so that $(e', e_0) \in Z'.\mathsf{vis}$. Then $\pi(e) \in Z.E$. Since $W_1 \approx_\pi W_2$, we have

$$(\beta(\pi(e)), \beta(f)) = (\beta_0(e), e_0) = (e', e_0) \in Z'.\mathsf{vis}$$

and

$$Z.\mathsf{obj}(\pi(e)) = Y.\mathsf{obj}(e) = Z.\mathsf{obj}(f).$$

Taking into account $(\beta, \rho) : Z \to Z'$ we get $(\pi(e), f) \in Z.\mathsf{vis}$, which implies $(e, f) \in X_0.\mathsf{vis}$, as required. Hence, $(\beta', \rho) : X_0.H \to X_0'.H$.

We now show that X_0 satisfies CAUSALAR and CAUSALVIS, similarly how we proceeded in the proof of Lemma 43. This implies that X_0 is an execution.

Since $(\beta', \rho) : X_0 \to X_0'$, for all $e, f \in X_0.E$:

$$\begin{array}{rl} (e, f) \in X_0.\mathsf{ar} & \implies (\beta'(e), \beta'(f)) \in X_0'.\mathsf{ar} \cup \mathsf{id}; \\ (e, f) \in X_0.\mathsf{so} & \implies (\beta'(e), \beta'(f)) \in X_0'.\mathsf{so} \cup \mathsf{id}; \\ (e, f) \in X_0.\sim & \iff (\beta'(e), \beta'(f)) \in X_0'.\sim. \end{array}$$

Hence,

$$\begin{array}{l} \forall e, f.\ (e, f) \in (X_0.\mathsf{so} \cup X_0.\mathsf{ar})/(X_0.\sim) \\ \qquad \implies (\beta'(e), \beta'(f)) \in (X_0'.\mathsf{so} \cup X_0'.\mathsf{ar})/(X_0'.\sim) \cup \mathsf{id}. \end{array}$$

Assume there is a cycle in $(X_0.\mathsf{so} \cup X_0.\mathsf{ar})/(X_0.\sim)$. Then there is also a cycle in $(X_0.\mathsf{so} \cup R)/(X_0.\sim)$. If β' does not map all events on the cycle to the same event in X_0', then the above shows that β' maps the cycle to one in $(X_0'.\mathsf{so} \cup R)/(X_0'.\sim)$, contradicting causal consistency of X_0'.

Consider now the case when β' maps all events on the cycle to the same event $e' \in X_0'.E$. Consider $(e, f) \in (X_0.\mathsf{so} \cup R)/(X_0.\sim)$ such that $\beta'(e) = \beta'(f) = e'$. Then $(e, f) \in X_0.\sim$. Hence, $(e, f) \in X_0.\mathsf{so} \cup R$ and either $(e, f) \in X_0.\mathsf{so}$ or $(f, e) \in X_0.\mathsf{so}$. It is easy to see that we cannot have $(e, f) \in X_0.\mathsf{so}$ and $(f, e) \in X_0.\mathsf{so}$. Assume $(e, f) \in R$ and $(f, e) \in X_0.\mathsf{so}$. If we had $(e, f) \in Y.\mathsf{ar}$ or $(e, f) \in Z.\mathsf{ar}|_{\beta^{-1}(e_0)}$, then this would contradict the causal consistency of Y or Z. Hence, we can only have

$$e, f \in W_2.E \wedge (\pi(e), \pi(f)) \in W_1.\mathsf{ar}$$

and $(f, e) \in Y.\mathsf{so}$. Then $X_0.\mathsf{obj}(e) = X_0.\mathsf{obj}(f)$ and, since Y is causally consistent, we have $(f, e) \in Y.\mathsf{vis}$ and, hence, $(f, e) \in Y.\mathsf{ar}$. But, together with the above, this contradicts $W_2 \approx_\pi W_1$. Thus, we must have $(e, f) \in X_0.\mathsf{so}$. In this way, we get a cycle in $X_0.\mathsf{so}$, which we cannot have by the construction of X_0. We have thus established that X_0 satisfies CAUSALAR. Then CAUSALVIS is shown as in the proof of Lemma 43.

Finally, we prove that $X_0 \models \mathbb{F}$. Consider $e \in X_0.E$ such that $X_0.\mathsf{obj}(e) \in \mathsf{dom}(\mathbb{F})$. First assume that $e \in Y.E$. Since $Y \models \mathbb{F}$ and $Y.\mathsf{ar} \subseteq X_0.\mathsf{ar}$, we have

$$X_0.\mathsf{rval}(e) = Y.\mathsf{rval}(e) = \mathbb{F}(Y.\mathsf{obj}(e))(\mathsf{ctxt}(Y, e)) = \mathbb{F}(X_0.\mathsf{obj}(e))(\mathsf{ctxt}(X_0, e)),$$

as required.

Assume now that $e \in \beta^{-1}(e_0)$. Since $Z \models \mathbb{F}$, we have

$$X_0.\mathsf{rval}(e) = Z.\mathsf{rval}(e) = \mathbb{F}(Z.\mathsf{obj}(e))(\mathsf{ctxt}(Z, e)) = \mathbb{F}(X_0.\mathsf{obj}(e))(\mathsf{ctxt}(Z, e))$$

We now establish a correspondence between $\mathsf{ctxt}(Z, e)$ and $\mathsf{ctxt}(X_0, e)$. First, of all, we have $\mathsf{ctxt}(X_0, e).p = X_0.\mathsf{aop}(e) = Z.\mathsf{aop}(e) = \mathsf{ctxt}(Z, e).p$. We now show that there is a bijection between the events in these contexts.

- Consider $f \in \mathsf{ctxt}(X_0, e).E$. Then $(f, e) \in X_0.\mathsf{vis}$ and, hence, either $f \in W_2.E$ and $(\pi(f), e) \in Z.\mathsf{vis}$ or $f \in \beta^{-1}(e_0)$ and $(f, e) \in Z.\mathsf{vis}$. For $f \in \mathsf{ctxt}(X_0, e).E$ let
$$\pi'(f) = (\text{if } f \in W_2.E \text{ then } \pi(f) \text{ else } f).$$
Then $f \in \mathsf{ctxt}(X_0, e).E$ implies $\pi'(f) \in \mathsf{ctxt}(Z.e).E$, so that $\pi' : \mathsf{ctxt}(X_0, e).E \to \mathsf{ctxt}(Z, e).E$.
- Consider $f' \in \mathsf{ctxt}(Z, e).E$. Then $(f', e) \in Z.\mathsf{vis}$ and, hence, either $f' \in \beta^{-1}(e_0)$ and $(f', e) \in X_0.\mathsf{vis}$ or $f' \in W_1.E$ and $(\pi^{-1}(f'), e) \in X_0.\mathsf{vis}$. Thus, if $(f', e) \in Z.\mathsf{vis}$ then for some $f \in \mathsf{ctxt}(X_0, e).E$ we have $\pi'(f) = f'$.

Hence, π' is a bijection.

Consider $f, g \in \mathsf{ctxt}(X_0, e).E \subseteq W_2.E \cup \beta^{-1}(e_0)$. We have the following cases:

- $f, g \in \beta^{-1}(e_0)$. Then
$$(f, g) \in \mathsf{ctxt}(X_0, e).\mathsf{vis} \iff (\pi'(f), \pi'(g)) = (f, g) \in Z.\mathsf{vis}.$$

– $f \in W_2.E$ and $g \in \beta^{-1}(e_0)$. Then

$$(f,g) \in \mathsf{ctxt}(X_0,e).\mathsf{vis} \iff (\pi'(f),\pi'(g)) = (\pi(f),g) \in Z.\mathsf{vis}.$$

– $f, g \in W_2.E$. Since $W_2 \approx_\pi W_1$, we get

$$(f,g) \in \mathsf{ctxt}(X_0,e).\mathsf{vis} \iff (\pi'(f),\pi'(g)) = (\pi(f),\pi(g)) \in Z.\mathsf{vis}.$$

Hence, $\pi'(\mathsf{ctxt}(X_0,e).\mathsf{vis}) = \mathsf{ctxt}(Z,e).\mathsf{vis}$.

We now prove a similar statement about arbitration. Consider $(f,g) \in \mathsf{ctxt}(Z,e).\mathsf{ar}$; then $f, g \in W_1.E \cup \beta^{-1}(e_0)$. We have the following cases:

– $f, g \in \beta^{-1}(e_0)$. Then

$$(\pi'^{-1}(f),\pi'^{-1}(g)) = (f,g) \in \mathsf{ctxt}(X_0,e).\mathsf{ar}.$$

– $f \in W_1.E$ and $g \in \beta^{-1}(e_0)$. Then

$$(\pi'^{-1}(f),\pi'^{-1}(g)) = (\pi^{-1}(f),g) \in X_0.\mathsf{ar}.$$

– $f, g \in W_1.E$. Then

$$(\pi'^{-1}(f),\pi'^{-1}(g)) = (\pi^{-1}(f),\pi^{-1}(g)) \in X_0.\mathsf{ar}.$$

Hence $\pi^{-1}(\mathsf{ctxt}(Z,e).\mathsf{ar}) \subseteq \mathsf{ctxt}(X_0,e).\mathsf{ar}$.

From the above correspondence between $\mathsf{ctxt}(Z,e)$ and $\mathsf{ctxt}(X_0,e)$ and the properties of data type specifications, we get

$$X_0.\mathsf{rval}(e) = \mathbb{F}(X_0.\mathsf{obj}(e))(\mathsf{ctxt}(Z,e)) = \mathbb{F}(X_0.\mathsf{obj}(e))(\mathsf{ctxt}(X_0,e)),$$

which establishes $X_0 \models \mathbb{F}$. Hence, $X_0 \models_{\mathsf{CC}} \mathbb{F}$.

Applying the above construction of X_0 from Y ad infinitum starting with $E = \emptyset$ and empty Y and Y', in the limit we obtain $X \in \mathsf{Exec}$ such that

$$(\beta,\rho) : X \to X' \quad \wedge \quad X \models_{\mathsf{CCS}} \mathbb{F}$$

for some β (in particular, the prefix-finiteness of $X.\mathsf{ar}$ follows from $(\beta,\rho) : X \to X'$).

We now show that X satisfies EVENTUAL, thereby establishing $X \models_{\mathsf{CC}} \mathbb{F}$. Assume EVENTUAL does not hold, i.e., for some $e \in X.E$ we have

$$\exists \text{ infinitely many } f \in X.E.\, \mathsf{sameobj}(X)(e,f) \wedge \neg(e \xrightarrow{X.\mathsf{vis}} f).$$

Let $\{f_k\}_{k=1}^{\infty}$ be a set of such fs. Since for any $f \in X'.E$, $\beta^{-1}(f)$ is finite, we can assume without loss of generality that $\beta(e) \neq \beta(f_k)$ for any k and $\beta(f_j) \neq \beta(f_k)$ for any j and k such that $j \neq k$. But then, since $(\beta,\rho) : X \to X'$, we have

$$\neg(\beta(e) \xrightarrow{X'.\mathsf{vis}} \beta(f_k)),$$

for k, which contradicts EVENTUAL for X'. □

PROPOSITION 45 *Assume*

$$\Delta \cup \{x : O\} \mid \Sigma \vdash C;$$
$$\Delta \cup \{x_j : O_j \mid j = 1..m\} \mid \Sigma \vdash \mathit{subst}(\{(x, o) \mapsto C_o \mid o \in O\}, C).$$

Then

$$\begin{array}{l}\forall \mathit{obj} : \mathsf{dom}(\Delta \cup \{x : O\} \cup \{x_j : O_j \mid j = 1..m\}) \to_{\mathsf{inj}} \mathsf{Obj}.\\ \forall \sigma : \Sigma \to \mathsf{Val}.\, \forall H, H'.\, \forall \beta : H.E \to H'.E.\\ (((\beta, [\![\{C_o\}_{o \in O}]\!](\mathit{obj}|_{\{x_j \mid j=1..m\}}), \mathit{obj}(x)) : H \Rightarrow H') \wedge\\ \quad H' \in (\!|C|\!)(\mathit{obj}|_{\mathsf{OVar} - \{x_j \mid j=1..m\}}, \sigma))\\ \qquad\qquad \Longrightarrow\ H \in (\!|\mathit{subst}(\{(x, o) \mapsto C_o \mid o \in O\}, C)|\!)(\mathit{obj}|_{\mathsf{OVar} - \{x\}}), \sigma).\end{array}$$

COROLLARY 46 *Assume*

$$\emptyset \mid \Delta \cup \{x : O\} \vdash P_{\mathsf{C}};$$
$$\emptyset \mid \Delta \cup \{x_j : O_j \mid j = 1..m\} \vdash \mathit{subst}(\{(x, o) \mapsto C_o \mid o \in O\}, P_{\mathsf{C}}).$$

Then

$$\begin{array}{l}\forall \mathit{obj} : \mathsf{dom}(\Delta \cup \{x : O\} \cup \{x_j : O_j \mid j = 1..m\}) \to_{\mathsf{inj}} \mathsf{Obj}.\\ \forall H, H'.\, \forall \beta : H.E \to H'.E.\\ (((\beta, [\![\{C_o\}_{o \in O}]\!](\mathit{obj}|_{\{x_j \mid j=1..m\}}), \mathit{obj}(x)) : H \Rightarrow H') \wedge\\ \quad H' \in \langle P_{\mathsf{C}} \rangle(\mathit{obj}|_{\mathsf{OVar} - \{x_j \mid j=1..m\}}))\\ \qquad\quad \Longrightarrow\ H \in \langle \mathit{subst}(\{(x, o) \mapsto C_o \mid o \in O\}, P_{\mathsf{C}}) \rangle(\mathit{obj}|_{\mathsf{OVar} - \{x\}})).\end{array}$$

THEOREM 47 (COMPLETENESS) *Let*

$$\begin{array}{l}D \ = (\mathsf{let}\ \{x_j = \mathsf{new}\ T_j\}_{j=1..m}\ \mathsf{in}\ \{o(v_{\mathsf{in}}) : v_{\mathsf{out}} = \mathsf{atomic}\ \{C_o\}\}_{o \in O});\\ P^2 = (\mathsf{let}\ x = \mathsf{new}\ D\ \mathsf{in}\ P);\\ P^1 = (\mathsf{let}\ \{x_j = \mathsf{new}\ T_j\}_{j=1..m}\ \mathsf{in}\ \mathit{subst}(\{(x, o) \mapsto C_o \mid o \in O\}, P)),\end{array}$$

where P^1 and P^2 are complete. Then $[\![P^2]\!]_{\mathsf{CG}} \subseteq [\![P^1]\!]_{\mathsf{CG}}$.

PROOF. Let

$$\emptyset \mid \{x_{\mathsf{io}} : \{o_{\mathsf{io}}\}\} \cup \Delta \cup \{x : O\} \vdash P^2_{\mathsf{C}};$$
$$\emptyset \mid \{x_{\mathsf{io}} : \{o_{\mathsf{io}}\}\} \cup \Delta \cup \{x_j : O_j \mid j = 1..m\} \vdash P^1_{\mathsf{C}}$$

be the client parts of P^1 and P^2, respectively, so that

$$P^1_{\mathsf{C}} = \mathit{subst}(\{(x, o) \mapsto C_o \mid o \in O\}, P^2_{\mathsf{C}}).$$

Consider any $\mathit{obj}'_0 : (\mathsf{dom}(\Delta) \cup \{x_{\mathsf{io}}, x : O\}) \to_{\mathsf{inj}} \mathsf{Obj}$ such that $\mathit{obj}'_0(x_{\mathsf{io}}) = \mathsf{io}$ and $\mathbb{F}'_0 : (\mathsf{range}(\mathit{obj}'_0) - \{\mathsf{io}\}) \to \mathsf{Spec}$ such that $\mathbb{F}'_0(\mathit{obj}'_0(x)) = [\![D]\!][\,]$. Let $\mathit{obj}'_0(x) = \omega$. Further, consider any execution

$$X'_0 \in [\![P^2_{\mathsf{C}}]\!]([\,], \mathit{obj}'_0, \mathbb{F}'_0).$$

Then

$$X_0'.H \in \langle P_{\mathsf{C}}^2 \rangle obj_0' \quad \wedge \quad X_0' \models_{\mathsf{CC}} \mathbb{F}_0'.$$

Let $X' = X_0'|_{\{e|X_0'.\mathsf{obj}(e)=\omega\}}$. Pick any $obj : \{x_j \mid j = 1..m\} \to_{\text{inj}} \mathsf{Obj}$ such that $\mathsf{range}(obj) \cap \mathsf{range}(obj_0') = \emptyset$ and let $\mathbb{F} \in [\mathsf{range}(obj) \to \mathsf{Spec}]$ be such that $\forall j = 1..m.\, \mathbb{F}(obj(x_j)) = [\![T_j]\!][\,]$. It is easy to check that $X_0' \models_{\mathsf{CC}} \mathbb{F}_0'$ implies $X' \models_{\mathsf{CC}} [\omega \mapsto ([\![D]\!][\,])]$. Then by Lemma 44 there exists X and β such that $X \models_{\mathsf{CC}} \mathbb{F}$ and $(\beta, [\![\{C_o\}_{o\in O}]\!](obj)) : X \to X'$ for some β. Without loss of generality we can assume that $X.E \cap X_0'.E = \emptyset$.

Let $Y' = X_0'|_{\{e|X_0'.\mathsf{obj}(e)\neq\omega\}}$ and let X_0 be defined by (57). Then by Lemma 43, X_0 is causally consistent. Define

$$obj_0 : (\mathsf{dom}(\Delta) \cup \{x_{\mathsf{io}}\} \cup \{x_j \mid j = 1..m\}) \to_{\text{inj}} \mathsf{Obj};$$
$$\mathbb{F}_0 : (\mathsf{range}(obj_0) - \{\mathsf{io}\}) \to \mathsf{Spec}$$

as follows: $obj_0(x_j) = obj(x_j)$ and $obj_0(x) = obj_0'(x)$ for all other x; $\mathbb{F}_0(\omega') = \mathbb{F}(\omega')$ for $\omega' \in \mathsf{range}(obj)$ and $\mathbb{F}_0(\omega') = \mathbb{F}_0'(\omega')$ for all other ω'. It is easy to see that $X_0 \models \mathbb{F}$, so that $X_0 \models_{\mathsf{CC}} \mathbb{F}$.

Let $\beta' : X_0.E \to X_0'.E$ be defined as follows:

$$\beta'(e) = (\text{if } e \in X.E \text{ then } \beta(e) \text{ else } e).$$

From $(\beta, [\![\{C_o\}_{o\in O}]\!](obj)) : X \to X'$ it follows that $(\beta', [\![\{C_o\}_{o\in O}]\!](obj), \omega) : X.H \Rightarrow X'.H$. Then, since $X_0'.H \in \langle P_{\mathsf{C}}^2 \rangle obj_0'$, by Corollary 46 we have $X_0.H \in \langle P_{\mathsf{C}}^2 \rangle obj_0$. As we also have $X_0 \models_{\mathsf{CC}} \mathbb{F}$, we get $X_0 \in [\![P_{\mathsf{C}}^1]\!]([\,], obj_0, \mathbb{F}_0)$. Furthermore, $\mathsf{observ}(X_0.H) = \mathsf{observ}(X_0'.H)$, as required. □

E Proof of the social graph data type and additional examples

E.1 Proof of the social graph data type

We now prove that the data type from Figure 3 has the specification $F_{\mathtt{soc}}$ from §5 as its denotation. Take any

$$obj \in \{friends[a], requesters[a] \mid a = 1..N\} \to_{\text{inj}} \mathsf{Obj}$$

and let $\mathbb{F} = \lambda\omega.\, F_{\mathtt{RWset}}$. Let $\rho = [\![\{C_o\}_{o\in O}]\!](obj)$, where O is the data type signature and C_o are the commands from Figure 3. It is easy to check that for any context N over the data type operations we have $\gamma(N, \rho, \mathbb{F}) \neq \emptyset$. We now prove that for any context N we have

$$\forall X, c.\, (X, c) \in \gamma(N, \rho, \mathbb{F}) \implies c = F_{\mathtt{soc}}(N)$$

by induction on the size of $N.E$. Consider a context N such that $|N.E| = n$ and assume the above holds for all contexts with $|N.E| < n$. Let

$$N = (p, M), \quad E' = N.E, \quad \mathsf{aop}' = N.\mathsf{aop}$$

and vis' and ar' are as in Definition 7.

Take $(X, c) \in \gamma(N, \rho, \mathbb{F})$, where $X = (H, \mathsf{vis}, \mathsf{ar}) = ((E, \mathsf{label}, \mathsf{so}, \sim), \mathsf{vis}, \mathsf{ar})$ (we also use derived selectors, such as obj). Then for some $e_0 \notin E'$ and β we have

$$(\beta, \rho) : X \to (N, e_0, c) \wedge X \models_{\mathsf{CC}} \mathbb{F}.$$

Hence, for some $\{c_f \in \mathsf{Val} \mid f \in E'\}$ we have

$$\begin{array}{c}(\forall f \in E'.\, H|_{\beta^{-1}(f)} \in \rho(\mathsf{aop}'(f), c_f)) \ \wedge \\ (H|_{\beta^{-1}(e_0)} \in \rho(p, c)) \ \ \wedge \ \ \beta(H.\mathsf{so}) \subseteq \mathsf{id} \ \ \wedge \ \ \beta(H.{\sim}) \subseteq \mathsf{id}\end{array}$$

and (10)–(12) hold.

From the induction hypothesis and Lemma 35 it is easy to establish

$$\forall f \in E'.\, c_f = F_{\mathsf{soc}}(\mathsf{ctxt}(M, f)). \tag{59}$$

We need to show that $c = F_{\mathsf{soc}}(N)$.

Let us first consider the case when $p = \mathtt{get}(a)$. Then we have:

$$\begin{array}{l}\mathsf{fst}(c) = \\ \{b \mid \exists e \in E.\, \mathsf{obj}(e) = obj(friends[a]) \wedge \mathsf{aop}(e) = \mathtt{add}(b) \wedge \\ \quad \forall f \in E.\, \mathsf{obj}(f) = \mathsf{obj}(e) \wedge \mathsf{aop}(f) = \mathtt{remove}(b) \implies f \xrightarrow{\mathsf{vis}} e\}; \\ \mathsf{snd}(c) = \\ \{b \mid \exists e \in E.\, \mathsf{obj}(e) = obj(requesters[a]) \wedge \mathsf{aop}(e) = \mathtt{add}(b) \wedge \\ \quad \forall f \in E.\, \mathsf{obj}(f) = \mathsf{obj}(e) \wedge \mathsf{aop}(f) = \mathtt{remove}(b) \implies f \xrightarrow{\mathsf{vis}} e\}; \\ \mathsf{fst}(F_{\mathsf{soc}}(N)) = \\ \{b \mid \exists e \in E'.\, (\mathsf{aop}'(e) = \mathtt{accept}((b, a) \mid (a, b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M, e)) \wedge \\ \quad \forall f \in E'.\, (\mathsf{aop}'(f) \in \mathtt{breakup}((b, a) \mid (a, b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M, e)) \\ \qquad \implies f \xrightarrow{\mathsf{vis}'} e\}; \\ \mathsf{snd}(F_{\mathsf{soc}}(N)) = \\ \{b \mid \exists e \in E'.\, (\mathsf{aop}'(e) = \mathtt{request}(b, a)) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M, e)) \wedge \\ \quad \forall f \in E'.\, (\mathsf{aop}'(f) \in (\mathtt{accept} \mid \mathtt{reject})((b, a) \mid (a, b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M, e)) \\ \qquad \implies f \xrightarrow{\mathsf{vis}'} e\}.\end{array}$$

– Take $b \in \mathsf{fst}(c)$. Then for some $e \in E$ we have

$$\begin{array}{c}(\mathsf{obj}(e) = obj(friends[a]) \wedge \mathsf{aop}(e) = \mathtt{add}(b) \wedge \forall f \in E.\, \mathsf{obj}(f) = \mathsf{obj}(e) \\ \wedge \mathsf{aop}(f) = \mathtt{remove}(b)) \implies f \xrightarrow{\mathsf{vis}} e.\end{array} \tag{60}$$

Then $\mathsf{aop}'(\beta(e)) \in \mathtt{accept}((b, a) \mid (a, b)))$, $c_{\beta(e)} = \mathsf{true}$ and by (59), $F_{\mathsf{soc}}(\mathsf{ctxt}(M, \beta(e))) = \mathsf{true}$. Consider $f \in E'$ such that

$$(\mathsf{aop}'(f) \in \mathtt{breakup}((b, a) \mid (a, b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M, e)).$$

Then $f \neq \beta(e)$ and by (59) we have $c_f = \mathsf{true}$. Hence, for some $e' \in E$ such that $\beta(e') = f$ we have

$$\mathsf{obj}(e') = obj(friends[a]) \wedge \mathsf{aop}(e') = \mathtt{remove}(b).$$

Then by (60) we have $e' \xrightarrow{\mathsf{vis}} e$. By (10) this entails $f \xrightarrow{\mathsf{vis}'} \beta(e)$. This shows $b \in \mathsf{fst}(F_{\mathsf{soc}}(N))$.

– Take $b \in \mathsf{fst}(F_{\mathsf{soc}}(N))$. Then for some $e \in E'$ we have

$$\begin{aligned}&(\mathsf{aop}'(e) = \mathtt{accept}((b,a) \mid (a,b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M,e)) \wedge \\ &\forall f \in E'.\, (\mathsf{aop}'(f) \in \mathtt{breakup}((b,a) \mid (a,b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M,e)) \\ &\qquad\qquad \implies f \xrightarrow{\mathsf{vis}'} e.\end{aligned} \tag{61}$$

By (59) we get $c_e = \mathsf{true}$. Hence, for some $e' \in E$ such that $\beta(e') = e$ we have

$$\mathsf{obj}(e') = obj(friends[a]) \wedge \mathsf{aop}(e') = \mathtt{add}(b).$$

Consider $f \in E$ such that

$$\mathsf{obj}(f) = \mathsf{obj}(e') \wedge \mathsf{aop}(f) = \mathtt{remove}(b).$$

Then $\mathsf{aop}'(\beta(f)) \in \mathtt{breakup}((b,a) \mid (a,b))$, $c_{\beta(f)} = \mathsf{true}$ and by (59) we have $F_{\mathsf{soc}}(\mathsf{ctxt}(M, \beta(f))) = \mathsf{true}$. Hence, by (61) we have $\beta(f) \xrightarrow{\mathsf{vis}'} \beta(e')$. Then by (11) we have $f \xrightarrow{\mathsf{vis}} e'$, which shows that $b \in \mathsf{fst}(c)$.

– Take $b \in \mathsf{snd}(c)$. Then for some $e \in E$ we have

$$\begin{aligned}&(\mathsf{obj}(e) = obj(requesters[a]) \wedge \mathsf{aop}(e) = \mathtt{add}(b) \wedge \forall f \in E.\, \mathsf{obj}(f) = \mathsf{obj}(e) \\ &\qquad \wedge \mathsf{aop}(f) = \mathtt{remove}(b)) \implies f \xrightarrow{\mathsf{vis}} e.\end{aligned} \tag{62}$$

Then $\mathsf{aop}'(\beta(e)) = \mathtt{request}(b,a)$ and $c_{\beta(e)} = \mathsf{true}$. Also, by (59),

$$F_{\mathsf{soc}}(\mathsf{ctxt}(M, \beta(e))) = \mathsf{true}.$$

Consider $f \in E'$ such that

$$(\mathsf{aop}'(f) \in (\mathtt{accept} \mid \mathtt{reject})((b,a) \mid (a,b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M,e)).$$

Then $f \neq \beta(e)$ and by (59) we have $c_f = \mathsf{true}$. Hence, for some $e' \in E$ such that $\beta(e') = f$ we have

$$\mathsf{obj}(e') = obj(requesters[a]) \wedge \mathsf{aop}(e') = \mathtt{remove}(b).$$

Then by (62) we have $e' \xrightarrow{\mathsf{vis}} e$. By (10) this entails $f \xrightarrow{\mathsf{vis}'} \beta(e)$. This shows $b \in \mathsf{snd}(F_{\mathsf{soc}}(N))$.

– Take $b \in \mathsf{snd}(F_{\mathsf{soc}}(N))$. Then for some $e \in E'$ we have

$$\begin{aligned}&(\mathsf{aop}'(e) = \mathtt{request}(b,a)) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M,e)) \wedge \\ &\forall f \in E'.\, (\mathsf{aop}'(f) \in (\mathtt{accept} \mid \mathtt{reject})((b,a) \mid (a,b))) \wedge F_{\mathsf{soc}}(\mathsf{ctxt}(M,e)) \\ &\qquad\qquad \implies f \xrightarrow{\mathsf{vis}'} e.\end{aligned} \tag{63}$$

Fig. 8. A shopping-cart data type. We use some syntactic sugar and a generalisation `removeAll` of the `remove` operation of `AWset`.

$$
\begin{array}{l}
D_{\texttt{cart}} = \mathsf{let}\ \{items = \mathsf{new}\ \texttt{AWset}; \\
\qquad\qquad \mathsf{inline}\ \texttt{resolve}(book) = \{ \\
\qquad\qquad\ \mathsf{var}\ entries = \{(book, n) \mid (book, n) \in items.\texttt{get}\}; \\
\qquad\qquad\ items.\texttt{removeAll}(entries);\ \mathsf{return}\ \max(\{n \mid (\_, n) \in entries\} \cup \{0\})\}\}\ \mathsf{in}\ \{ \\
\texttt{inc}(book, n) = \mathsf{atomic}\ \{\ \mathsf{var}\ m = \texttt{resolve}(book);\ items.\texttt{add}(book, m + n)\ \}; \\
\texttt{dec}(book, n) = \mathsf{atomic}\ \{\ \mathsf{var}\ m = \texttt{resolve}(book); \\
\qquad\qquad\qquad\qquad \mathsf{if}\ (m - n > 0)\ \mathsf{then}\ items.\texttt{add}(book, m - n)\ \}; \\
\texttt{count}(book) = \mathsf{atomic}\ \{\ \mathsf{var}\ m = \texttt{resolve}(book);\ items.\texttt{add}(book, m);\ v_{\mathsf{out}} = m\ \}\ \}
\end{array}
$$

By (59) we get $c_e = \mathsf{true}$. Hence, for some $e' \in E$ such that $\beta(e') = e$ we have

$$\mathsf{obj}(e') = obj(requesters[a]) \wedge \mathsf{aop}(e') = \texttt{add}(b).$$

Consider $f \in E$ such that

$$\mathsf{obj}(f) = \mathsf{obj}(e') \wedge \mathsf{aop}(f) = \texttt{remove}(b).$$

Then $\mathsf{aop}'(\beta(f)) \in (\texttt{accept} \mid \texttt{reject})((b, a) \mid (a, b))$, $c_{\beta(f)} = \mathsf{true}$ and by (59) we have $F_{\mathsf{soc}}(\mathsf{ctxt}(M, \beta(f))) = \mathsf{true}$. Hence, by (63) we have $\beta(f) \xrightarrow{\mathsf{vis}'} \beta(e')$. Then by (11) we have $f \xrightarrow{\mathsf{vis}} e'$, which shows that $b \in \mathsf{snd}(c)$.

The above establishes $c = F_{\mathsf{soc}}(N)$.

We now consider the case when $p = \texttt{accept}(b, a)$; the others are analogous. We have

$$F_{\mathsf{soc}}(N) = (b \in \mathsf{snd}(F_{\mathsf{soc}}(\texttt{get}(a), M)))$$

and

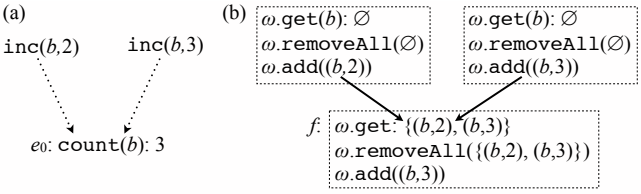
$$
\begin{array}{rl}
c = \mathsf{true} \iff & (\exists e \in E.\, \mathsf{obj}(e) = obj(requesters[a]) \wedge \mathsf{aop}(e) = \texttt{add}(b) \wedge \\
& \forall f \in E.\, \mathsf{obj}(f) = \mathsf{obj}(e) \wedge \mathsf{aop}(f) = \texttt{remove}(b) \implies f \xrightarrow{\mathsf{vis}} e).
\end{array}
$$

But we have already shown that the above is equivalent to $b \in \mathsf{snd}(F_{\mathsf{soc}}(\texttt{get}(a), M))$, as required. □

E.2 A shopping-cart data type

The example appearing in Figure 8 implements a shopping cart for an online book-store. Conceptually, a shopping cart is a map from book titles to non-negative integers, representing the number of copies requested. The data type provides three operations: $\texttt{inc}(book, n)$ and $\texttt{dec}(book, n)$ for increasing or decreasing the count associated with a $book$ by a positive integer n, and $\texttt{count}(book)$ for querying the count (an operation giving the full cart contents can be implemented similarly). The cart can be accessed concurrently in different sessions, e.g., by spouses. This can lead to conflicts when the

Fig. 9. (a) A context of the data type in Figure 8; and (b) an execution belonging to its concretisation. We use the same conventions as in Figure 5.



count for the same book is updated concurrently, as in the context shown in Figure 9(a). Our implementation takes one possible choice of resolving such conflicts so that increases win against concurrent decreases or smaller increases; this results in customers buying more. Thus, its denotation returns 3 on the context in Figure 9(a).

The implementation represents the cart using an add-wins set that stores pairs of a book ordered and the corresponding positive count. Multiple concurrent updates to the count of the same $book$ result in separate entries $(book, n_1), \ldots, (book, n_k)$; see the event f in the execution in Figure 9(b), which concretises the context in Figure 9(a). To resolve such conflicts, an operation involving the $book$ replaces the above entries by their maximum $\max\{n_1, \ldots, n_k\}$ (implemented by the macro `resolve`). Using an add-wins set to represent $items$ is crucial for the correctness of our implementation: it ensures that `removeAll` in `resolve` removes only the entries it computes the maximum of; in a remove-wins set it would also cancel all concurrent operations.

Below we prove that the shopping cart data type has the following specification $F_{\mathtt{cart}}$ as its denotation in our semantics. Informally, $F_{\mathtt{cart}}$ applies the sequential semantics of shopping cart (formalised by the function G below) to all maximal paths of events in visibility in the context (where maximality is given with respect to the subsequence order) and takes the maximum of the results:

$$\begin{aligned} &F_{\mathtt{cart}}(\mathtt{inc}(book, n), M) = F_{\mathtt{cart}}(\mathtt{dec}(book, n), M) = \bot;\\ &F_{\mathtt{cart}}(\mathtt{count}(book), M) = \max(\{0\} \cup \{G(\pi, 0) \mid \pi \text{ is a}\\ &\qquad\qquad \text{maximal vis-path of events in } M.E \text{ that operate on } book\}), \end{aligned}$$

where $G(\pi, m)$ are defined as follows:

$$\begin{aligned} G(\varepsilon, m) &= m\\ G(e\pi, m) &= G(\pi, \max\{0, m-n\}) && \text{if } M.\mathsf{aop}(e) = \mathtt{dec}(book, n)\\ G(e\pi, m) &= G(\pi, m+n) && \text{if } M.\mathsf{aop}(e) = \mathtt{inc}(book, n)\\ G(e\pi, m) &= G(\pi, m) && \text{otherwise} \end{aligned}$$

Thus, $F_{\mathtt{cart}}$ explains the outcome of operations in the distributed setting M in terms of their sequential executions consistent with visibility: the number of copies of $book$ is the maximum possible value that can result from such executions.

Proof of the shopping-cart data type. Let $\{C_o\}_{o\in O}$ be the implementations of operations of the shopping-cart data type. We will prove that

$$\begin{array}{l}\forall N.\ \forall e.\ \forall obj \in [\{items\} \to_{\text{inj}} \mathsf{Obj}].\ \forall \mathbb{F}.\ \forall X.\ \forall \beta.\\ \quad (e \notin (N.E) \wedge \mathbb{F}(obj(items)) = F_{\texttt{AWset}} \wedge (\beta, [\![\{C_o\}_o]\!] obj) : X \to (N, e, \_) \wedge X \models_{\mathsf{CC}} \mathbb{F})\\ \quad \Longrightarrow\\ \quad \exists e_0 \in \beta^{-1}(e).\ X.\mathsf{op}(e_0) = \texttt{get} \ \wedge\\ \quad \forall book.\ \max\left(\{0\} \cup \{n \mid (book, n) \in X.\mathsf{rval}(e_0)\}\right) = F_{\texttt{cart}}(\texttt{contains}(book), N.M),\end{array} \tag{64}$$

where $N.M$ is the partial operation context of M (i.e., M except the first component of the applied operation).

Before proving this implication, we show that it entails the correctness of our specification $F_{\texttt{cart}}$ for the shopping-cart data type. Consider

$$N, \quad e \notin (N.E), \quad obj \in [\{items\} \to_{\text{inj}} \mathsf{Obj}], \quad \mathbb{F}, \quad X, \quad \beta, \quad c \in \mathsf{Val}$$

such that

$$\mathbb{F}(obj(items)) = F_{\texttt{AWset}} \ \wedge \ (\beta, [\![\{C_o\}_{o\in O}]\!] obj) : X \to (N, e, c) \ \wedge \ X \models_{\mathsf{CC}} \mathbb{F}.$$

We need to show that $F_{\texttt{cart}}(N) = c$. By the definition of $(\beta, [\![\{C_o\}_{o\in O}]\!] obj) : X \to (N, e, c)$, we have

$$(X.H)|_{\beta^{-1}(e)} \in [\![\{C_o\}_{o\in O}]\!] obj(N.p, c). \tag{65}$$

Recall that the shopping-cart data type implements only three operations, that is, those in $O = \{\texttt{inc}, \texttt{dec}, \texttt{contains}\}$. Hence, by the semantics of $\{C_o\}_{o\in O}$, the set membership in (65) implies that for some nonnegative integers $book'$ and n',

$$\begin{array}{lll}(N.p = \texttt{inc}(book', n') \wedge c = \bot) & \vee & (N.p = \texttt{dec}(book', n') \wedge c = \bot)\\ & \vee & (N.p = \texttt{contains}(book')).\end{array}$$

It is easy to see that the desired $F_{\texttt{cart}}(N) = c$ follows in the first two cases. In the third case, we use (64) and deduce that

$$\begin{array}{l}\exists e_0 \in \beta^{-1}(e).\ X.\mathsf{op}(e_0) = \texttt{get} \ \wedge\\ \Big(\max\left(\{0\} \cup \{n \mid (book', n) \in X.\mathsf{rval}(e_0)\}\right) = F_{\texttt{cart}}(\texttt{contains}(book'), N.M)\Big).\end{array}$$

By (65) and the definition of $[\![\{C_o\}_{o\in O}]\!]$ (in particular, the part for $C_{\texttt{contains}}$), the LHS of the above equation is equal to c. But the RHS of the equation is precisely $F_{\texttt{cart}}(N)$. Hence, the above equation implies that $F_{\texttt{cart}}(N) = c$, as desired.

We now go back to the proof of (64). Consider

$$N, \quad e \notin (N.E), \quad obj \in [\{items\} \to_{\text{inj}} \mathsf{Obj}], \quad \mathbb{F}, \quad X, \quad \beta$$

such that

$$\begin{array}{lll}\mathbb{F}(obj(items)) = F_{\texttt{AWset}} & \wedge & (\beta, [\![\{C_o\}_{o\in O}]\!] obj) : X \to (N, e, \_)\\ & \wedge & X \models_{\mathsf{CC}} \mathbb{F}.\end{array}$$

We should show that

$$\exists e_0 \in \beta^{-1}(e).\ X.\mathsf{op}(e_0) = \texttt{get}\ \wedge$$
$$\Big(\forall book.\ \max\left(\{0\} \cup \{n \mid (book, n) \in X.\mathsf{rval}(e_0)\}\right) = F_{\texttt{cart}}(\texttt{contains}(book), N.M)\Big),$$

where $N.M$ is the partial operation context of N. Note that in the implementation of all the operations of the shopping-cart data type, the macro $\texttt{resolve}(book)$ for some nonnegative integer $book$ is run first, and this macro starts by calling the `get` operation of the *items* object. Furthermore, $(X.H)|_{\beta^{-1}(e)}$ should describe the computation of one of such shopping-cart operations, because $(X.H)|_{\beta^{-1}(e)} \in [\![\{C_o\}_{o \in O}]\!] obj(N.p, \_)$. Hence,

$$\exists e_0 \in \beta^{-1}(e).\ X.\mathsf{op}(e_0) = \texttt{get}.$$

It is sufficient to show that this e_0 satisfies the following property:

$$\forall book.\ \max\left(\{0\} \cup \{n \mid (book, n) \in X.\mathsf{rval}(e_0)\}\right) = F_{\texttt{cart}}(\texttt{contains}(book), N.M). \tag{66}$$

We will prove the following slightly more general fact. For every $E' \subseteq (N.E)$ closed under $N.\mathsf{vis}^{-1}$ (i.e., $N.\mathsf{vis}^{-1}(E') \subseteq E'$), if we let $E'_0 = \beta^{-1}(E')$ and $N' = N|_{E'}$, we have that

$$\begin{aligned}&\forall book.\ F_{\texttt{cart}}(\texttt{contains}(book), N'.M) = \\ &\max\left(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\texttt{get}, E'_0, X.\mathsf{aop}|_{E'_0}, X.\mathsf{vis}|_{E'_0}, X.\mathsf{ar}|_{E'_0})\}\right).\end{aligned} \tag{67}$$

where $N'.M$ is the partial context of N'. Note that this implies (66) because

$$E'_0 = \beta^{-1}(N.E) \implies \mathsf{ctxt}(X, e_0) = (\texttt{get}, E'_0, X.\mathsf{aop}|_{E'_0}, X.\mathsf{vis}|_{E'_0}, X.\mathsf{ar}|_{E'_0}).$$

Our proof of (67) is based on the induction on the size of E'. The base case is that $E' = \emptyset$. In this case, the RHS of (67) is 0. Also, since E' is empty, so is $E'_0 = \beta^{-1}(E')$. Thus, $F_{\texttt{AWset}}(\texttt{get}, \ldots)$ on the LHS of (67) returns the empty set, from which follows that the LHS is 0.

Next we handle the inductive case. Let E' be a nonempty $N.\mathsf{vis}^{-1}$-closed subset of $N.E$. Let

$$E'_0 = \beta^{-1}(E') \quad \text{and} \quad N' = N|_{E'}.$$

Pick a nonnegative integer $book$. Let π be a maximal vis-path of events in E' that operate on $book$ and $G(\pi) = F_{\texttt{cart}}(\texttt{contains}(book), N'.M)$, if such π exists. Otherwise, we set $\pi = \epsilon$, so that $G(\pi) = F_{\texttt{cart}}(\texttt{contains}(book), N'.M)$ even in this degenerate case. If $G(\pi) = 0$,

$$\begin{aligned}&F_{\texttt{cart}}(\texttt{contains}(book), N'.M)\ =\ 0\ \leq \\ &\qquad \max\left(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\texttt{get}, E'_0, X.\mathsf{aop}|_{E'_0}, X.\mathsf{vis}|_{E'_0}, X.\mathsf{ar}|_{E'_0})\}\right).\end{aligned}$$

Otherwise, π should be of the form $\pi' f$. Let

$$\begin{aligned}&E'' = (N.\mathsf{vis})^{-1}(f), \quad E''_0 = \beta^{-1}(E''), \quad N'' = N|_{E''}, \quad E''' = E'' \cup \{f\}, \\ &E'''_0 = \beta^{-1}(E'''), \quad \text{and} \quad N''' = N|_{E'''}.\end{aligned}$$

By the induction hypothesis,

$$F_{\mathtt{cart}}(\mathtt{contains}(book), N''.M) \;=\; \max\left(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathsf{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\}\right).$$

By the claim that we will show at the end of this proof, the above equality implies that

$$F_{\mathtt{cart}}(\mathtt{contains}(book), N'''.M) \;=\; \max\left(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathsf{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}\right).$$

Hence,

$$\begin{aligned}
&F_{\mathtt{cart}}(\mathtt{contains}(book), N'.M) \\
&= G(\pi) \\
&= F_{\mathtt{cart}}(\mathtt{contains}(book), N'''.M) \\
&= \max\left(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathsf{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}\right). \\
&\leq \max\left(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathsf{get}, E_0', X.\mathsf{aop}|_{E_0'}, X.\mathsf{vis}|_{E_0'}, X.\mathsf{ar}|_{E_0'})\}\right).
\end{aligned}$$

The last inequality holds because for every nonnegative integer n,

$$\begin{aligned}
&(book, n) \in F_{\mathtt{AWset}}(\mathsf{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''}) \\
&\quad\Longrightarrow\; (book, n) \in F_{\mathtt{AWset}}(\mathsf{get}, E_0', X.\mathsf{aop}|_{E_0'}, X.\mathsf{vis}|_{E_0'}, X.\mathsf{ar}|_{E_0'}).
\end{aligned} \tag{68}$$

This implication itself holds for the following reason. $\pi' f$ is a maximal vis-path of events that operate on $book$, so f is not $N.\mathsf{vis}$-related to any $\mathtt{inc}(book, \ldots)$ or $\mathtt{dec}(book, \ldots)$ events in E_0'. This and the fact that $(\beta, [\![\{C_o\}_{o \in O}]\!] obj) : X \to (N, e, \_)$ together ensure that all the uncancelled $\mathtt{add}(book, \ldots)$ events in $(E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})$ remain uncancelled in the bigger operation context $(E_0', X.\mathsf{aop}|_{E_0'}, X.\mathsf{vis}|_{E_0'}, X.\mathsf{ar}|_{E_0'})$. From this follows the implication in (68).

Now let n be the maximum of

$$\{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathsf{get}, E_0', X.\mathsf{aop}|_{E_0'}, X.\mathsf{vis}|_{E_0'}, X.\mathsf{ar}|_{E_0'})\},$$

if this set is not empty. Otherwise, we set $n = 0$. If $n = 0$,

$$\begin{aligned}
&\max\left(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathsf{get}, E_0', X.\mathsf{aop}|_{E_0'}, X.\mathsf{vis}|_{E_0'}, X.\mathsf{ar}|_{E_0'})\}\right) \\
&= 0 \\
&\leq F_{\mathtt{cart}}(\mathtt{contains}(book), N'.M).
\end{aligned}$$

Assume now that $n \neq 0$. Then, there exists $f_0 \in E_0'$ such that $X.\mathsf{aop}(f_0) = \mathtt{add}(book, n)$ and f_0 is not $(X.\mathsf{vis})$-related to any event in E_0' that removes $(book, n)$. Let

$$\begin{aligned}
&f = \beta(f_0), \quad E'' = (N.\mathsf{vis})^{-1}(f), \quad E_0'' = \beta^{-1}(E''), \quad N'' = N|_{E''}, \\
&E''' = E'' \cup \{f\}, \quad E_0''' = \beta^{-1}(E'''), \quad \text{and} \quad N''' = N|_{E'''}.
\end{aligned}$$

By the induction hypothesis,

$$F_{\texttt{cart}}(\texttt{contains}(book), N''.M) \;=\; \max\left(\{0\} \cup \{m \mid (book, m) \in F_{\texttt{AWset}}(\mathsf{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\}\right).$$

By the claim that we will show at the end of this proof, the above equality implies that

$$F_{\texttt{cart}}(\texttt{contains}(book), N'''.M) \;=\; \max\left(\{0\} \cup \{m \mid (book, m) \in F_{\texttt{AWset}}(\mathsf{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}\right).$$

All the shopping-cart operations on $book$ remove every entry involving $book$ from the $items$ object before adding any entry on $book$ to the same object. Also, the adding operation happens at the end of these operations, if it ever happens. Thus,

$$\{m \mid (book, m) \in F_{\texttt{AWset}}(\mathsf{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\} = \{n\}.$$

From the same reason and the fact that f_0 is not $(X.\mathsf{vis})$-related to any event in E_0' that removes $(book, n)$, it follows that f is not $(N.\mathsf{vis})$-related to any event in E' that operates on $book$. Because of these two observations,

$$n \;=\; F_{\texttt{cart}}(\texttt{contains}(book), N'''.M) \;\leq\; F_{\texttt{cart}}(\texttt{contains}(book), N'.M).$$

The only remaining part of our proof is to discharge the following claim: for all $f \in N.E$ and nonnegative integers $book$, if we let

$$E'' = (N.\mathsf{vis})^{-1}(f), \quad E_0'' = \beta^{-1}(E''), \quad N'' = N|_{E''},$$
$$E''' = E'' \cup \{f\}, \quad E_0''' = \beta^{-1}(E'''), \quad \text{and} \quad N''' = N|_{E'''},$$

and

$$F_{\texttt{cart}}(\texttt{contains}(book), N''.M) \;=\; \max\left(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\mathsf{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\}\right), \tag{69}$$

then

$$F_{\texttt{cart}}(\texttt{contains}(book), N'''.M) \;=\; \max\left(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\mathsf{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}\right). \tag{70}$$

Pick $f \in N.E$ and $book$. Define $E'', E''', E_0'', E_0''', N'', N'''$ as described in our claim above. Also, assume that these data satisfy the equality in (69). We will prove the required equality in (70) by the case analysis on $N.\mathsf{aop}(f)$. Since

$$\forall f' \in (N.E).\ (X.H)|_{\beta^{-1}(f')} \in [\![\{C_o\}_{o \in O}]\!] obj(N.\mathsf{aop}(f'), \_)$$

and $[\![\{C_o\}_{o \in O}]\!] obj$ is defined only for the `inc`, `dec` and `contains` operations with nonnegative integer arguments, we only need to consider the following three cases of $N.\mathsf{aop}(f)$:

$$N.\mathsf{aop}(f) = \texttt{inc}(book', n') \quad \vee \quad N.\mathsf{aop}(f) = \texttt{dec}(book', n') \quad \vee \quad N.\mathsf{aop}(f) = \texttt{contains}(book')$$

for some nonnegative integers $book'$ and n'.

– Assume that $N.\mathsf{aop}(f) = \mathtt{inc}(book', n')$ for nonnegative integers $book'$ and n'. If $book \neq book'$,

$$
\begin{aligned}
&F_{\mathtt{cart}}(\mathtt{contains}(book), N'''.M)\\
&= F_{\mathtt{cart}}(\mathtt{contains}(book), N''.M)\\
&= \max(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathtt{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\})\\
&= \max(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathtt{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}).
\end{aligned}
$$

The first equality comes from the definition of $F_{\mathtt{cart}}$ and the fact that $book' \neq book$, and the second holds because of the assumption (69). The third holds because none of the events in $E_0''' - E_0''$ is an `inc` or `dec` operation involving $book$. Assume now that $book = book'$.

$$
\begin{aligned}
&F_{\mathtt{cart}}(\mathtt{contains}(book), N'''.M)\\
&= F_{\mathtt{cart}}(\mathtt{contains}(book), N''.M) + n'\\
&= \max(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathtt{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\}) + n'\\
&= \max\{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathtt{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}\\
&= \max(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathtt{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}).
\end{aligned}
$$

The first equality holds because N''' is the extension of N'' with the $N'''.\mathsf{vis}$-maximum event f and the operation performed by f is $\mathtt{inc}(book', n') = \mathtt{inc}(book, n')$. The second comes from the assumption (69). The third holds because the implementation of the $\mathtt{inc}(book', n')$ operation removes all the elements involving $book'$ from the $items$ set and adds the following tuple to the set:

$$
\begin{aligned}
&(book',\\
&\max(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathtt{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\}) + n').
\end{aligned}
$$

Furthermore, by the same reason, for every nonnegative integer n,

$$
\begin{aligned}
&(book, n) \in F_{\mathtt{AWset}}(\mathtt{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\\
&\qquad\qquad \Longleftrightarrow\\
&n =\\
&\max(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathtt{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\}) + n'.
\end{aligned}
$$

From this and the fact that $n' \geq 0$ follows the last equality in our derivation above.

– Assume that $N.\mathsf{aop}(f) = \mathtt{dec}(book', n')$ for nonnegative integers $book'$ and n'. The overall structure of the proof of this case is similar to that of the previous one. If $book \neq book'$,

$$
\begin{aligned}
&F_{\mathtt{cart}}(\mathtt{contains}(book), N'''.M)\\
&= F_{\mathtt{cart}}(\mathtt{contains}(book), N''.M)\\
&= \max\left(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathtt{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\}\right)\\
&= \max\left(\{0\} \cup \{n \mid (book, n) \in F_{\mathtt{AWset}}(\mathtt{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}\right).
\end{aligned}
$$

The reasons for all the equalities are identical to those used in the previous case. The first equality comes from the definition of $F_{\texttt{cart}}$ and the fact that $book' \neq book$, and the second from the assumption (69). The third holds because none of the events in $E_0''' - E_0''$ is an `inc` or `dec` operation involving $book$. Assume now that $book = book'$.

$$
\begin{aligned}
&F_{\texttt{cart}}(\texttt{contains}(book), N'''.M)\\
&= \max\{0,\ F_{\texttt{cart}}(\texttt{contains}(book), N''.M) - n'\}\\
&= \max\{0,\ \max\ (\{0\} \cup\\
&\qquad\qquad \{n \mid (book, n) \in F_{\texttt{AWset}}(\mathsf{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\})\\
&\qquad - n'\}\\
&= \max(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\mathsf{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}).
\end{aligned}
$$

The first equality holds because N''' is the extension of N'' with f, this f event is a maximum according to the $N'''.\mathsf{vis}$ relation, and the operation performed by f is $\texttt{dec}(book', n') = \texttt{dec}(book, n')$. The second comes from the assumption (69). Finally, observe that the implementation of the $\texttt{inc}(book', n')$ operation removes all the elements involving $book = book'$ from the $items$ object and adds the following tuple to the object:

$$
\begin{aligned}
&(book,\\
&\max(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\mathsf{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\}) - n')
\end{aligned}
$$

if the second component of the tuple is positive; otherwise, it does not add any tuple. The third equality follows from this observation.

– Assume that $N.\mathsf{aop}(f) = \texttt{contains}(book')$ for a nonnegative integer $book'$. If $book \neq book'$,

$$
\begin{aligned}
&F_{\texttt{cart}}(\texttt{contains}(book), N'''.M)\\
&= F_{\texttt{cart}}(\texttt{contains}(book), N''.M)\\
&= \max(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\mathsf{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\})\\
&= \max(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\mathsf{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}).
\end{aligned}
$$

The reasons for all the equalities are identical to those used in the previous two cases. Assume now that $book = book'$.

$$
\begin{aligned}
&F_{\texttt{cart}}(\texttt{contains}(book), N'''.M)\\
&= F_{\texttt{cart}}(\texttt{contains}(book), N''.M)\\
&= \max(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\mathsf{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\})\\
&= \max(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\mathsf{get}, E_0''', X.\mathsf{aop}|_{E_0'''}, X.\mathsf{vis}|_{E_0'''}, X.\mathsf{ar}|_{E_0'''})\}).
\end{aligned}
$$

The first equality holds because N''' is the extension of N'' with f, this f event is maximum in $N'''.E$ according to the $N'''.\mathsf{vis}$ relation, and the operation performed by f is $\texttt{contains}(book') = \texttt{contains}(book)$. The second comes from the assumption (69). The third equality holds because the implementation of the

$\texttt{contains}(book')$ operation removes all the elements involving $book = book'$ from the $items$ object and adds the following tuple to the object:

$$(book, \\ \max(\{0\} \cup \{n \mid (book, n) \in F_{\texttt{AWset}}(\mathsf{get}, E_0'', X.\mathsf{aop}|_{E_0''}, X.\mathsf{vis}|_{E_0''}, X.\mathsf{ar}|_{E_0''})\})).$$

□

E.3 An example of composing last-writer-wins objects

We now illustrate that composing objects with last-writer-wins conflict-resolution policies yields an object with the same policy. To this end, we use the primitive data type LWWreg of a ***last-writer-wins register***. The data type has the signature $\{\texttt{write}, \texttt{read}\}$ and its specification $F_{\texttt{LWWreg}}$ is defined as follows. We let $F_{\texttt{LWWreg}}((\texttt{write}, a), M) = \bot$ and let $F_{\texttt{LWWreg}}((\texttt{read}, \bot), M)$ be defined if and only if $M.\mathsf{ar}$ is a total order on the set $\{e \in M.E \mid M.\mathsf{aop}(e) = (\texttt{write}, \_)\}$; in this case $F_{\texttt{LWWreg}}$ returns the parameter of the last write event in this order, or $\bot$ if there are no such events.

The following composite data type combines operations on a pair of last-writer-wins registers:

$$\begin{aligned} D_{\texttt{Pair}} = \mathsf{let}\ \{x_1 = \mathsf{new}\ \texttt{LWWreg}; x_2 = \mathsf{new}\ \texttt{LWWreg}\}\ \mathsf{in}\ \{ \\ \texttt{write}(v_1, v_2) = \mathsf{atomic}\ \{\ x_1.\texttt{write}(v_1); x_2.\texttt{write}(v_2)\ \}; \\ \texttt{read} = \mathsf{atomic}\ \{\ v_{\mathsf{out}} = (x_1.\texttt{read}, x_2.\texttt{read})\ \}\ \} \end{aligned}$$

This data type satisfies the specification $F_{\texttt{Pair}}$ that is analogous to $F_{\texttt{LWWreg}}$, but operates on pairs of values: $F_{\texttt{Pair}}((\mathsf{get}, \bot), M)$ returns the pair of values supplied as the parameter to the last write operation in the context according to $M.\mathsf{ar}$, or $\bot$ if there are none; it is undefined if $M.\mathsf{ar}$ does not totally order write operations. We omit the trivial proof that $F_{\texttt{Pair}}$ is indeed the denotation of $D_{\texttt{Pair}}$.