CRDT Emulation, Simulation, and Representation Independence

NATHAN LIITTSCHWAGER, University of California, Santa Cruz, USA
JONATHAN CASTELLO, University of California, Santa Cruz, USA
STELIOS TSAMPAS, University of Southern Denmark, Denmark
LINDSEY KUPER, University of California, Santa Cruz, USA

Conflict-free replicated data types (CRDTs) are distributed data structures designed for fault tolerance and high availability. CRDTs have historically been taxonomized into *state-based* CRDTs, in which replicas apply updates locally and periodically broadcast their local state to other replicas over the network, and *operation-based* (or *op-based*) CRDTs, in which every state-updating operation is individually broadcast and applied at each replica. In the literature, state-based and op-based CRDTs are considered equivalent due to the existence of algorithms that transform one kind of CRDT into the other, and vice versa. In particular, verification techniques and results for one kind of CRDT are often said to be applicable to the other kind, thanks to this equivalence. However, what it means for state-based and op-based CRDTs to emulate each other has never been made fully precise. In particular, emulation is nontrivial since state-based and op-based CRDTs place different requirements on the behavior of the underlying network with regard to both the causal ordering of message delivery, and the granularity of the messages themselves.

In this paper, we specify and formalize CRDT emulation in terms of *simulation* by modeling CRDTs and their interactions with the network as formal transition systems. We show that emulation can be understood as *weak simulations* between the transition systems of the original and emulating CRDT systems, thus closing a gap in the CRDT literature. We precisely characterize which properties of CRDT systems are preserved by our weak simulations, and therefore which properties can be said to be applicable to state-based CRDTs as long as they are applicable to op-based CRDTs and vice versa. Finally, we leverage our emulation results to obtain a general *representation independence* result for CRDTs: intuitively, clients of a CRDT cannot tell whether they are interacting with a state-based or op-based CRDT in particular.

1 Introduction

In distributed data storage systems, *data replication* is a ubiquitous mechanism for guarding against machine failures and ensuring that data is physically close to far-flung clients. Informally, replication copies a data object over n spatially separated sites, or *replicas*, with each replica acting as an independent copy of the original object. With replication comes the challenge of ensuring that replicas remain consistent with one another in the face of inevitable network partitions and clients who demand "always-on" access to data. The gold standard of consistency for such a replicated system is *linearizability* [Herlihy and Wing 1990], but it can be impractical to implement — indeed, systems that are prone to network partitions and that prioritize *high availability* of data must necessarily do so at the expense of linearizability [Gilbert and Lynch 2002, 2012].

The quest for an optimal point in the availability/consistency trade-off space has led to the development of *conflict-free replicated data types* (CRDTs) [Shapiro et al. 2011; Roh et al. 2011; Preguiça et al. 2018], which are data structures designed for high availability through replication. CRDTs sacrifice linearizability in favor of the weaker safety property *strong convergence* [Shapiro et al. 2011], which says that replicas that have received and applied the same *set* of updates will agree in their observable state, regardless of the order in which those updates are received and applied. When coupled with the liveness guarantee of *eventual delivery* of updates to replicas, a system of CRDT replicas achieves *strong eventual consistency* [Shapiro et al. 2011], which says that

Authors' Contact Information: Nathan Liittschwager, University of California, Santa Cruz, USA, nliittsc@ucsc.edu; Jonathan Castello, University of California, Santa Cruz, USA, jcaste14@ucsc.edu; Stelios Tsampas, University of Southern Denmark, Denmark, stelios@imada.sdu.dk; Lindsey Kuper, University of California, Santa Cruz, USA, lkuper@ucsc.edu.

eventually the observable state of all replicas will agree. CRDTs have been an active area of research, with considerable attention paid to the specification and verification (of various properties, but especially strong convergence) of CRDT designs [Burckhardt et al. 2014; Zeller et al. 2014; Gomes et al. 2017; Gadducci et al. 2018; Liu et al. 2020; Nair et al. 2020; Liang and Feng 2021; Nieto et al. 2022, 2023]; recent work tackles automated verification [Nagar and Jagannathan 2019; De Porre et al. 2023] and even synthesis of correct-by-construction CRDTs [Laddad et al. 2022].

In their pioneering work on CRDTs, Shapiro et al. [2011] taxonomized CRDTs into *state-based* CRDTs, in which replicas apply updates locally and periodically broadcast their local state (which may be the result of multiple local updates) to other replicas over the network, and *operation-based* (or *op-based*) CRDTs, in which every state-updating operation is individually broadcast and applied at each replica.[1] In state-based CRDTs, the replica state space is a join-semilattice, and a replica receiving an update from a remote replica will apply the update locally by taking the least upper bound (join) of its local state and the received update. As a result, the order of received updates is immaterial. Op-based CRDTs, on the other hand, rely on stronger ordering guarantees (in particular, *causal broadcast* [Birman and Joseph 1987; Birman et al. 1991]) from the underlying network transport mechanism, while requiring concurrently applied updates to commute. While causal ordering can be expensive to implement, it is still easier than enforcing a total order on updates. Both the state-based and op-based approaches result in strong convergence, the defining characteristic of CRDTs.

Most work on CRDT specification and verification focuses exclusively on either state-based [Zeller et al. 2014; Gadducci et al. 2018; Nair et al. 2020; Timany et al. 2024; Nieto et al. 2023; Laddad et al. 2022] or op-based [Gomes et al. 2017; Nagar and Jagannathan 2019; Liu et al. 2020; Liang and Feng 2021; Nieto et al. 2022] CRDTs. The justification for this choice is that state-based and op-based CRDTs can *emulate* each other. Shapiro et al. [2011] give general algorithms by which one may construct a state-based CRDT out of a given op-based CRDT, and vice versa. However, Shapiro et al. stop short of formally defining a notion of emulation and proving that their construction satisfies it. In particular, Shapiro et al. do not formalize any relationship between the *observable behaviors* of the original system and the newly constructed system, beyond pointing out that if the original system exhibits strong eventual consistency, then does the new one. This property is insufficient for correctness, since a trivial CRDT that does nothing is also strongly eventually consistent. Yet the notion that state-based and op-based CRDTs can emulate each other is frequently appealed to in the literature. For instance, Nagar and Jagannathan [2019], in their work on verification of op-based CRDTs, write that "our technique naturally extends to state-based CRDTs since they can be emulated by an op-based model," and Laddad et al. [2022], in their work on synthesis of state-based CRDTs, write that state-based CRDTs "can always be translated to op-based CRDTs if necessary." It has not been clear whether the emulation algorithms given by Shapiro et al. actually do preserve the desired safety properties that appear in these works. This makes the notion of emulation in CRDTs "load-bearing", and therefore deserving of being made precise.

In this paper, we seek to close this gap in the CRDT literature and formalize the notion of CRDT emulation. To do so, we model emulation as formal *simulation* of transition systems, where each transition system models a state-based or op-based replicated object, with semantics for client-driven updates, and message passing of updates between individual replicas. Unlike Shapiro et al., we formally model the network behavior as well as the semantics of individual replicas. Our framework allows us to specify emulation as a kind of *weak simulation* between the original system (which we call the *host* CRDT system) and a new system constructed from the original (which we

[1] More recently, Almeida et al. [2015] introduced *delta state* CRDTs, an optimization of traditional state-based CRDTs in which only state *changes*, rather than entire states, must be disseminated over the network.

call the *guest* CRDT system). This model of emulation lets us specify and prove that a guest CRDT exhibits all the observable behaviors of the host CRDT, and vice versa.

A particular challenge of reasoning about CRDT emulation is that state-based and op-based CRDTs place different requirements on the behavior of the underlying network with regard to *causal ordering* of messages. In particular, op-based CRDTs require causal message ordering, but state-based CRDTs do not. Emulating an op-based CRDT with a state-based CRDT requires confronting this mismatch in network behavior, since it can no longer be assumed. Shapiro et al. [2011]'s original construction works by essentially implementing the causal ordering mechanism inside the host CRDT itself. Our simulation results, however, are about an abstract model of CRDTs operating in a network. To ground our results, we show that a simple, yet expressive, stateful programming language can interact with our model of CRDTs by invoking update and query commands, obtaining observable values from the underlying CRDTs. By leveraging our simulation results, we then show that from the programmer's point of view, one can interchange the underlying host CRDT for the constructed guest CRDT with no change in observable behavior.

To summarize, we make the following specific contributions:

- Using our formalism, we give a precise study and characterization of *emulation* between op-based and state-based CRDTs (Section 4). Our main result says that a state-based (resp. op-based) host CRDT is *weakly simulated* by the constructed op-based (resp. state-based) guest CRDT, and vice versa. From a technical point of view, what makes the simulation interesting and challenging is the handling of message delivery. In the state-based-to-op-based direction, merging a state can be weakly simulated by a sequence of message deliveries, and in the op-based-to-state-based direction, delivering a message can be weakly simulated by merging a carefully chosen state. It turns out that asserting the existence of such a state is nontrivial.
- We precisely characterize which properties of CRDT systems are preserved by our weak simulations, and therefore characterize the set of properties that can be said to be applicable to a state-based (resp. op-based) CRDT as long as they are applicable to an op-based (resp. state-based) CRDT (Section 4.3).
- Finally, we leverage our main emulation result to obtain a general *representation independence* result for CRDTs (Section 5). Intuitively, representation independence says that clients of a CRDT cannot tell whether they are interacting with a state-based (op-based) host CRDT or its op-based (state-based) guest CRDT.

Before getting into our contributions, we begin with background and a motivating example in Section 2, and we formalize the semantics of CRDTs in Section 3. We discuss related work in Section 6 and conclude in Section 7.

2 Preliminaries and Motivation

In this section, we give background on CRDTs (Section 2.1) and on simulation (Section 2.2). We then give a motivating example (Section 2.3) to show why the question of whether op-based CRDTs and state-based CRDTs can emulate each other is more subtle than it might appear at first glance.

Notation 2.1. Let A and B be sets. The set of functions $A \to B$ is denoted B^A. We denote the power set of A by $\mathcal{P}(A)$. We write A^* for the *Kleene star* of A and write the empty list as ε. Lists are written as either sequences $\langle a_1, \ldots, a_i, \ldots, a_k \rangle$, or strings $a_1 \cdots a_i \cdots a_k$, where the i in a_i is the *index* (the locations of elements in lists is sometimes relevant). List concatenation is denoted by $- \cdot - : A^* \times A^* \to A^*$ with the understanding that if $a \in A$ and $\omega \in A^*$, then $a \cdot \omega$ makes sense and is shorthand for $\langle a \rangle \cdot \omega$, and likewise for $\omega \cdot a$. We occasionally just use the shorthand $\alpha\omega$ for $\alpha \cdot \omega$.

The symbol $\bot$ appears throughout this paper to denote a kind of "null output" or "silent event", so it will appear in various contexts, but the semantic meaning is generally clear from context.

We also use the following substitution operation: if $f : A \to B$ is a function, and for some $a \in A$, we have $f(a) = b$, then $g = f[a \mapsto c]$ defines a function $g : A \to B$ such that $g(a) = f[a \mapsto c](a) = c$, and $g(a') = f[a \mapsto c](a') = f(a')$ for all $a' \neq a$.

Finally, this paper has some fairly busy notation when discussing simulations. To help keep track of the various states and transitions which occur throughout, we use the following conventions:

- Op-based semantics will be typeset in an orange bold font, e.g., $\rightsquigarrow$, $\longrightarrow$, $\langle \Gamma \mid \Sigma \mid \beta \rangle$;
- State-based semantics will be typeset in a light blue font, e.g., $\rightsquigarrow$, $\longrightarrow$, $\langle \Gamma \mid \Sigma \mid \beta \rangle$;
- Silent transitions (in either op-based or state-based CRDT systems) will be typeset in green: $\overset{\tau}{\rightsquigarrow}$.

2.1 Background on CRDTs

Both op-based and state-based CRDTs enjoy *strong eventual consistency* (SEC for short) [Shapiro et al. 2011], which is a combination of a safety guarantee (*strong convergence*) and a liveness guarantee (*eventual delivery*). Strong convergence says that if any two replicas receive and apply the same set of updates, then they will have the same observable state. Eventual delivery says that eventually all replicas receive and apply all updates. Put together, SEC guarantees that each replica converges to the same state. Op-based and state-based CRDTs accomplish this end by placing different constraints the communication medium of the network, their allowed operations, and their state space.

2.1.1 Op-Based CRDTs. Op-based CRDTs are implemented as a distributed system by having each node (a *replica*) implement the same object (seen as a tuple consisting of: a replica ID `r`, a state s, and a set of methods). Op-based CRDTs achieve strong convergence by requiring each replica `r` to sequentially (and independently of other replicas) apply client-supplied operations as part of a two-phase protocol. First, the replica `r` initiates the *prepare* phase which invokes a `prep` method. `prep` takes a given operation `op`, and generates a message m which contains the effects of the command, along with any needed metadata. Then, that message is propagated to all other replicas in the network via a *broadcast* mechanism. Second, `r` initiates the *effect* phase, which immediately applies the effects of m locally by invoking an `effect` method. It is typical to consider these two phases as a single atomic action called an *update*, expressed in Algorithm 1.

Algorithm 1 The op-based update as described in Shapiro et al. [2011]. The prepare phase is used so that replicas may attach any needed metadata to messages, e.g., a vector clock [Mattern 2002; Fidge 1988; **?**] and the source replica `r` ID to make messages unique, and ordered by potential causality [Lamport 1978].

```
1: procedure OP-BASED UPDATE(r : RID, s : S, op : Op)
2:     m ← prep(r, s, op)                          ▷ prepare phase
3:     broadcast(r, m)
4:     s′ ← effect(s, m)                           ▷ effect phase
5:     putState(s′)
6: end procedure
```

The aforementioned broadcast mechanism must enforce that messages are reliably delivered and are are applied at each replica in an order consistent with the *causal order*, in the specific sense of Lamport's *happens-before* partial order [Lamport 1978], which we denote $\prec_{\text{hb}}$. Two messages m and m' are related by $\prec_{\text{hb}}$ if one is a potential cause for the other, e.g., $m \prec_{\text{hb}} m'$ if the sending of m is a potential cause for the sending of m'. Two messages m, m' are considered *concurrent* (written $m \parallel m'$) precisely when they are unrelated by happens-before, i.e., $\neg(m \prec_{\text{hb}} m') \wedge \neg(m' \prec_{\text{hb}} m)$.

In this case, the effects of m and m' must commute. Figure 1 summarizes the specification of an op-based CRDT replica.

Remark 2.2. If message effects are *always* commutative, i.e.,

$$\forall m, m' \in M \ : \ \texttt{effect}(\texttt{effect}(s, m), m') = \texttt{effect}(\texttt{effect}(s, m'), m), \tag{2.1}$$

then the requirement that the op-based CRDT be implemented on top of a reliable causal broadcast mechanism may be relaxed to simply reliable broadcast — all replicas eventually receive all broadcasted messages, but with no requirements on their order.

Parameters :

S : states, Op : operations, M : messages, Q : queries, V : values,

$s^0 \in S$: initial state

Functions :

$\texttt{prep}(\texttt{r}, s, \texttt{op})$: replica r prepares a message m that contains the effects of op

$\texttt{effect}(s, m)$: applies the effects of m to local state s

$\texttt{query}(q, s)$: q queries the state s returning some value v

Assertions :

Messages m are handled by reliable causal broadcast mechanism (if needed)

m, m' are *concurrent* $\implies \texttt{effect}(\texttt{effect}(s, m), m') = \texttt{effect}(\texttt{effect}(s, m'), m)$

Fig. 1. Specification of an Op-Based CRDT Object

2.1.2 State-based CRDTs. While op-based CRDTs require that updates are applied in an order consistent with the causal order, state-based CRDTs instead require that (i) the local state space S of each replica r forms a *join-semilattice* with join operator $\sqcup$, and (ii) local updates can only make *inflationary* updates to the current state of the respective replica, in the sense that $s \sqcup \texttt{update}(\texttt{r}, s, \texttt{op}) \geq s$ for each operation op. This that means state-based CRDTs do not need a special preparation phase for message passing. Instead, it suffices to have replicas copy their local state s and send that to other replicas, who may invoke a merge method to join their local state and s together, using the given join operator $\sqcup$ that the state space S is equipped with. Since merge is implemented in terms of $\sqcup$, it advances the replica's state toward the least upper bound (lub) of all replicas' states. Figure 2 summarizes the specification of a state-based CRDT replica.

Remark 2.3. Unlike op-based CRDTs, state-based CRDTs are not required to broadcast their states to all replicas as a part of an update cycle, and doing so may lead to performance problems, since sending of an entire local state may lead to a very large message. We assume in this paper that state-based replicas communicate through reliable point-to-point messages on a complete network graph (i.e., each r may send messages to any other r′).

2.1.3 Emulation. Shapiro et al. [2011] argue that it is possible for an op-based CRDT to be *emulated* by a state-based CRDT and vice versa. To that end, they provide two transformations, one that constructs an op-based CRDT given a state-based CRDT, and one that constructs a state-based CRDT given an op-based CRDT. The transformations are given as pair of algorithms, which we denote as a pair of mappings $\mathcal{G}^{\text{st}\to\text{op}}$ and $\mathcal{G}^{\text{op}\to\text{st}}$ for the state-to-op-based transformation and

Parameters :

S : states, $\texttt{Op}$: operations, Q : queries, V : values,

$\sqcup$: a join, i.e., $(S, \sqcup)$ is a join-semilattice

$s^0 \in S$: initial state

Functions :

$\texttt{update}(\texttt{r}, s, \texttt{op})$: inflationary update, i.e., $\forall s, \texttt{op} : s \sqcup \texttt{update}(\texttt{r}, s, \texttt{op}) = s$

$\texttt{merge}(s, s')$: merges states s and s' together, defined in terms of $\sqcup$

$\texttt{query}(q, s)$: q queries the state s returning some value v

Assertions :

$\sqcup : S \times S \to S$ is a join, i.e., $(S, \sqcup)$ is a join-semilattice

$\texttt{merge}(s, s') = s \sqcup s'$

$\texttt{update}$ is inflationary: $\forall s, \texttt{op} : s \sqcup \texttt{update}(\texttt{r}, s, \texttt{op}) \geq s$

Fig. 2. Specification of a State-Based CRDT Object

the op-to-state-based transformation respectively. We defer the precise description of $\mathcal{G}^{\text{st}\to\text{op}}$ and $\mathcal{G}^{\text{op}\to\text{st}}$ to Section 4, but one can think of $\mathcal{G}^{\text{st}\to\text{op}}$ as taking an object satisfying the specificatin in Figure 2 and constructing an object that satisfies the specification in Figure 1. Likewise, $\mathcal{G}^{\text{op}\to\text{st}}$ takes a given op-based object and constructs a state-based one. The original description of the algorithms can be found in Shapiro et al. [2011].

We establish the following terminology with respect to emulation, used throughout the paper.

Definition 2.4 (Host and Guest Objects)**.** Let $\mathcal{O}_\kappa$ and $\mathcal{O}_\lambda$ denote two objects, and suppose there is a translation $\mathcal{G}^{\kappa\to\lambda}$ between them which constructs $\mathcal{O}_\lambda$ given $\mathcal{O}_\kappa$, in the above sense. Then we say $\mathcal{O}_\kappa$ is the *host object*, and $\mathcal{O}_\lambda$ is the *guest object*.

While Shapiro et al. [2011] define their translation algorithms and argue that the resulting objects satisfy SEC, they do not give a formal definition of what is meant by "emulation" other than that the translated CRDT should satisfy SEC. Indeed, any host CRDT object may be mapped to a trivial guest CRDT that returns the identity for every input operation and received message. Such a CRDT is indeed SEC, but is clearly pathological, and not in the spirit that Shapiro et al. [2011] intend. Rather, the idea seems to be that emulation should let user swap out a given op-based (resp. state-based) host CRDT for the corresponding state-based (resp. op-based) guest CRDT, since they will have corresponding interfaces, in the sense that only `update` and `query` are exposed to clients, and the rest of the implementation is hidden "under the hood".

With this apparent understanding, CRDT specification and verification work tends to focus exclusively on either state-based [Zeller et al. 2014; Gadducci et al. 2018; Nair et al. 2020; Timany et al. 2024; Nieto et al. 2023; Laddad et al. 2022] or op-based [Gomes et al. 2017; Nagar and Jagannathan 2019; Liu et al. 2020; Liang and Feng 2021; Nieto et al. 2022] CRDTs, with the justification that since op-based and state-based CRDTs can emulate each other, results that apply to one kind of CRDT should straightforwardly "transfer" or "generalize" to the other. The definition of emulation is thus "load-bearing" in the sense that other works in the literature implicitly or explicitly rely on it. With this in mind, we believe emulation is worth making precise.

2.2 Simulations

The semantic model we use in this paper is a labelled transition system (LTS), or tuples of the form $(S, \Lambda, \longrightarrow)$, where S is the state space, Λ is a label set, and $\longrightarrow$ is a subset of $S \times \Lambda \times S$. called the *transition relation*. We say a state s transitions to a state s' by a $\alpha \in \Lambda$ if $(s, \alpha, s') \in \longrightarrow$, and we write $s \xrightarrow{\alpha} s'$. We use the letter τ (and sometimes $\bot$) to denote transitions that are *silent* or *unobservable* from the point of view of an observer outside the system, e.g., a client of a distributed replicated system does not observe the message-passing behavior that takes place inside the system. We sometimes refer to an LTS purely by its transition relation, if the state space and label set are either clear or irrelevant.

We prefer to define transition relations informally using sets of small-step style rules, and omit the precise definition of the label set Λ, instead letting Λ be implicitly defined by the rules. For example, given an LTS $\longrightarrow$, we define its *saturation*, or *weak* transition relation as a starred arrow, e.g. $\longrightarrow^*$, as the least relation closed under the following rules:

$$\frac{}{s \xrightarrow{\tau}{}^* s} \qquad \frac{s \xrightarrow{\tau}{}^* s'' \quad s'' \xrightarrow{\tau} s'}{s \xrightarrow{\tau}{}^* s'} \qquad \frac{s \xrightarrow{\tau}{}^* s_1 \quad s_1 \xrightarrow{\alpha} s_2 \quad s_2 \xrightarrow{\tau}{}^* s'}{s \xrightarrow{\alpha}{}^* s'}$$

Note that $\longrightarrow^*$ is the reflexive and transitive closure of $\longrightarrow$ under silent moves.

Given an LTS $(S, \Lambda, \longrightarrow)$ with a distinguished initial state $s_0 \in s$, we say a sequence of labels $\alpha_1, \ldots, \alpha_n \in \Lambda$ is a *trace* or a *behavior* of the LTS if there exists a sequence of transitions

$$s_0 \xrightarrow{\alpha_1} s_1 \cdots s_{n-1} \xrightarrow{\alpha_n} s_n.$$

We refer to the above alternating sequence of labels and successor states as an *execution*.

From the perspective of an observer, the internal states in LTSes are immaterial, and comparison of two LTSs is done by comparing their traces. *Simulation* is widely used to compare both models and implementations of distributed and concurrent systems [Lamport 1994; Lynch and Tuttle 1988; Milner 1982; Burckhardt et al. 2014; Wilcox et al. 2015].

Definition 2.5 (Simulation)**.** Let X and Y be sets, and Λ a set of labels. Let $\longrightarrow_X \subseteq X \times \Lambda \times X$ and $\longrightarrow_Y \subseteq Y \times \Lambda \times Y$ be a pair of transition systems. We say a relation $R \subseteq X \times Y$ is a *simulation* if for every pair $(x, y) \in R$, and $\alpha \in \Lambda$:

$$x \xrightarrow{\alpha}_X x' \implies \exists y' \in Y : y \xrightarrow{\alpha}_Y y' \wedge (x', y') \in R.$$

In this case, we say y *simulates* x, or x is *simulated by* y.

We can express simulations diagrammatically as below.

$$\begin{array}{ccc} x & \overset{R}{\cdots} & y \\ {\scriptstyle\alpha}\downarrow & & \\ x' & & \end{array} \quad \textit{implies} \quad \begin{array}{ccc} x & \overset{R}{\cdots} & y \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha} \\ x' & \overset{R}{\cdots} & \exists y' \end{array}$$

Simulations specifically require that each single step in system X be matched by an equivalent single step in system Y. Matching two systems in lock-step this way is usually too strict for realistic models of distributed systems, where one may to simulate a single coarse-grained step with a larger number of fine-grained steps, so we prefer *weak simulation*.

Definition 2.6 (Weak simulation)**.** We say a relation $R \subseteq X \times Y$ is a *weak simulation* if for every pair $(x, y) \in R$, and $\alpha \in \Lambda$:

$$x \xrightarrow{\alpha}_X x' \implies \exists y' \in Y : y \xrightarrow{\alpha}{}^*_Y y' \wedge (x', y') \in R.$$

The union of all (weak) simulations is itself a (weak) simulation, therefore it exists and we define the following.

Definition 2.7 ((Weak) similarity)**.** Given x, and y, if there exists a (weak) simulation $\mathcal{R}$ such that $\mathcal{R}(x, y)$, then we say y *(weakly) simulates* x and write $x \lesssim y$ ($x \lessapprox y$). If also $y \lesssim x$, then we say x and y are *similar* and write $x \lesssim\gtrsim y$ ($x \lessapprox\gtrapprox y$).

A stronger form of (weak) simulation is (weak) *bi*simulation.

Definition 2.8 (Bisimulation)**.** If the relation R is a (weak) simulation, and its converse R^T is a (weak) simulation, then R is called a (weak) *bisimulation.*

Like simulation, the union of all (weak) bisimulations is a (weak) bisimulation, so it exists and we denote it by $\sim$ ($\approx$).
Bisimilarity is in some sense "symmetric" in that, if $x \sim y$, and $x \xrightarrow{\alpha} x'$, then we can complete the above simulation diagram, *and* if $y \xrightarrow{\alpha} y'$, then we can still complete the simulation diagram, where we interchange the roles of x and y. This symmetry condition is quite strong, since in general $\sim \neq \lesssim\gtrsim$, e.g.,

$$\begin{array}{ccccc} & & x_1' & & \\ & \overset{a}{\nearrow}_{a} & & & \\ x & \xrightarrow{a} & x_1 & \xrightarrow{b} & x_2 \end{array} \qquad \text{and} \qquad y \xrightarrow{a} y_1 \xrightarrow{b} y_2$$

has $x \lesssim\gtrsim y$ but $x \nsim y$. Indeed, any sequence of actions in one system is simulated by a sequence of actions in the other system, but if $x \xrightarrow{a} x_1'$, then we match by $y \xrightarrow{a} y_1$, but x_1' has no answer if $y_1 \xrightarrow{b} y_2$. This is what we mean by lack of symmetry in the simulation, and this situation turns out to be the case when modeling op-based and state-based CRDTs.

2.3 Motivating Example

If we believe that "emulation" is a synonym for "simulation", we might conclude that the semantics of a distributed system implementing a CRDT can be modeled with an LTS, where labels Λ correspond to a client-facing API (e.g., `update` and `query` commands). Emulation should then mean that given an LTS $\longrightarrow$ modeling a system of op-based (resp. state-based) CRDT replicas, we can use Shapiro et al.'s emulation algorithms to construct a corresponding LTS $\longrightarrow'$ (with the same label set Λ) modeling a system of state-based (resp. op-based) CRDT replicas, such that there is a (weak) bisimulation between them. However, it turns out that that the "obvious" approach quickly leads to a counterexample, which illustrates why the weak simulations we show later on in Section 4 need to be carefully constructed.

Example 2.9 (Non-Bisimulation)**.** The counterexample proceeds as follows: first we define an op-based CRDT that acts as the host, and describe its system semantics in a transition system. We then construct the state-based guest CRDT and define its semantics. We then exhibit an execution in which the host CRDT may take a step, but the guest CRDT may not take a corresponding step without breaking the simulation.

(1) *(Op-based Host CRDT).* We start with the op-based CRDT called a *grow-only multiset* over elements A [Shapiro et al. 2011], and consider a system of two replicas $\mathtt{RID} = \{\mathtt{r}_1, \mathtt{r}_2\}$. Following the specification in Figure 1, the state space is $S = \mathcal{M}(A)$ with initial state $s^0 = \emptyset$, and (for simplicity) we consider only a single type of operation: $\mathtt{Op} \stackrel{\text{def}}{=} \{\langle \mathsf{add}, a\rangle \mid a \in A\}$. Messages are $M = \{\langle \mathtt{r}_i, k, op\rangle \mid \mathtt{r}_i \in \mathtt{RID}, k \in \mathcal{N}, op \in \mathtt{Op}\}$. Values are just the state space, i.e., $V = S$. We define the functions `prep`, `effect`, and `query` as follows:

- $\mathtt{prep} : \mathtt{RID} \times S \times \mathtt{Op} \to M$ defined by $\mathtt{prep}(\mathtt{r}_i, s, op) = \langle \mathtt{r}_i, k, op\rangle$ where $k = |s|$;

- $\texttt{effect} : S \times M \to S$ defined by $\texttt{effect}(s, \langle \mathsf{r}, k, \langle \mathsf{add}, a\rangle\rangle) = s \cup \{a\}$;
- $\texttt{query} : S \to S$ defined by $\texttt{query} = id_S$.

And, as in Algorithm 1, we define an update function `update` that is just the composition of `prep` and `effect`. All operations will commute (Equation (2.1)), so we need not mention any implementation of a causal order.

(2) (*Op-based Host System Semantics*). We can now informally define a simple LTS to give the semantics of the guest CRDT in a network. Since we have two replicas r_1 and r_2, we take the global configuration of the system to be a pair $x_1 \otimes x_2$, and we define $x_i = (s_i, b_i) \in S \times \mathcal{P}(M)$ to be resp. the state and and a set modeling a message buffer of replica r_i. We will use the shorthand $s(x_i) = s_i$ and $b(x_i) = b_i$. There are only three kinds of transitions: update transitions, query transitions, and deliver transitions.

Update transitions are written $x_1 \otimes x_2 \xrightarrow{\mathsf{r}_i : \mathbf{upd}(a)} x_1' \otimes x_2'$ and denote when replica r_i has updates itself with the operation $op = \langle \mathsf{add}, a\rangle$ so that it has the state x_i' where $s(x_i') = \texttt{update}(\mathsf{r}_i, s(x_i), op)$. At the same time, the other replica r_j updates its buffer with the generated message, i.e., $b(x_j') = b(x_j) \cup \{\texttt{prep}(\mathsf{r}_i, s(x_i), op)\}$.

Query transitions are written $x_1 \otimes x_2 \xrightarrow{\mathsf{r}_i : \mathbf{qry}/v} x_1 \otimes x_2$ indicating that the query returned a value $v = \texttt{query}(s(x_i))$ without side effects.

Deliver transitions are written $x_1 \otimes x_2 \xrightarrow{\mathsf{r}_i : \mathbf{dlvr}} x_1' \otimes x_2'$ and denote when replica r_i plucks a message m from its non-empty buffer $b(x_i)$ and updates its state, yielding $s(x_i') = \texttt{effect}(s(x_i), m)$ and $b(x_i') = b(x_i) \setminus \{m\}$.

(3) (*Constructing the State-Based Guest CRDT*). To get our state-based guest CRDT, we can follow Shapiro et al.'s recipe to translate an op-based CRDT into a state-based one. This is accomplished in our case by using a *set of messages* as our state space, and defining $\sqcup$ as the $\cup$ set union operation.[2] We have that $(S', \sqcup) = (\mathcal{P}(M), \cup)$ is a join-semilattice. We take as our initial state $\varnothing$.

In the original algorithm, heavier machinery is needed to define the state-based update and query functions in terms of the op-based update and query.[3] Thankfully, our CRDTs are so simple that we can simply define them as follows:

- $\texttt{update}' : \mathtt{RID} \times \mathcal{P}(M) \times \mathtt{Op} \to \mathcal{P}(M)$ defined by $\texttt{update}'(\mathsf{r}, H, op) = H \cup \{\langle \mathsf{r}, k, op\rangle\}$, where $k = |H|$;
- $\texttt{merge} : \mathcal{P}(M) \times \mathcal{P}(M) \to \mathcal{P}(M)$ defined by $\texttt{merge}(H, H') = H \cup H$;
- $\texttt{query}' : \mathcal{P}(M) \to \mathcal{M}(A)$ defined by $\texttt{query}'(H) = \texttt{query}(s)$ where the set-comprehension $s = \{a \mid \langle \mathsf{r}, k, \mathsf{add}, a\rangle \in H\}$.

(4) (*State-based System Semantics*).

We take our state-based configurations to be pairs $y_1 \otimes y_2$, where $s(y_i) \in \mathcal{P}(M)$ and $b(y_i) \in \mathcal{P}(\mathcal{P}(M))$. The transition semantics here can follow by analogy to the op-based system semantics except now messages are the states, and delivering a message is done with `merge` instead of `effect`. In that sense, state-based update and deliver transitions are, aside from the above differences,

[2] This is slightly different from how Shapiro et al. [2011] define it. They maintain a tuple (s, K, D) where s is the original op-based state, and K is the set of "known" messages, and D is a set of "delivered" messages. Then $\sqcup$ is defined by $(s, K, D) \sqcup (s', K', D') = d(s, K \cup K', D)$, where d recursively applies all yet unapplied messages in $(K \cup K') \setminus D$, updating s and D accordingly. Curiously, this is not actually a join-semilattice, since now $\sqcup$ is not commutative! Our modification, on the other hand, is.

[3] Namely, one needs to be able to define a function $\texttt{interp}_S : \mathcal{P}(M) \to S$ that "interprets" a set of messages H to the corresponding op-based state s. Done completely rigorously in a more general setting, the construction of a well-defined $\texttt{interp}_S$ is non-trivial. While we give the construction in the appendix, it might be worth mentioning that the recursive d function given by [Shapiro et al. 2011] would be formally defined by the same construction.

the same as the op-based ones, though we use $\xrightarrow{\mathsf{r}:\alpha}$ transitions to differentiate them from the op-based ones. Moreover — and this is critical — replicas send their states as messages via a separate transition (i.e., sending of state is not baked into an $\mathsf{r} : \mathsf{upd}(a)$ transition).

This send transition is written as $y_1 \otimes y_2 \xrightarrow{\mathsf{r}_i:\mathsf{send}(\mathsf{r}_j)} y_1' \otimes y_2'$ and denotes replica r_i sending its state $s(y_i)$ to replica r_j, and updating its buffer so that $b(y_j') = b(y_j) \cup \{s(y_i)\}$. The state of the sender is unchanged. We will treat the $\xrightarrow{\mathsf{r}_i:\mathsf{send}(\mathsf{r}_j)}$ transitions as the "silent" transitions $\xrightarrow{\tau}$, to satisfy the definition of a weak bisimulation, but leave the label $\mathsf{r}_i : \mathsf{send}(\mathsf{r}_j)$ exposed in our notation, since it clarifies the proceeding discussion.

Failure of Weak Bisimulation. In an effort to avoid a large number of nested parentheses (...) and brackets {...}, we use a more compressed notation here. We write a set $\{a_1, \ldots, a_k\}$ as a string $a_1 \cdots a_k$, and for messages, we define $a_k^{\mathsf{r}_i}$ as $\langle \mathsf{r}_i, k, \langle \mathsf{add}, a \rangle \rangle$. So, for a configuration $x_1 \otimes x_2$, the state $x_i = (\{a_1, a_2, \ldots, a_k\}, \{m_1, m_2, \ldots, m_l\})$ is, in our compressed notation,

$$x_i = \langle a_1 a_2 \cdots a_k \mid m_1 m_2 \cdots m_k \rangle.$$

Let $\langle | \rangle \otimes \langle | \rangle$ be our starting configurations in the op-based CRDT system and the state-based CRDT system. It is clear that queries on both initial configurations return the same result.

We now sketch the main idea. The op-based host CRDT and the state-based guest CRDT can weakly simulate each other, but there is no weak bisimulation. We show a representative example: suppose we have an execution of the op-based system:

$$\langle | \rangle \otimes_{\mathsf{op}} \langle | \rangle \xrightarrow{\mathsf{r}_1:\mathbf{upd}(a)} \langle a \mid \rangle \otimes_{\mathsf{op}} \langle \mid a_0^{\mathsf{r}_1} \rangle \xrightarrow{\mathsf{r}_1:\mathbf{upd}(a)} \langle aa \mid \rangle \otimes_{\mathsf{op}} \langle \mid a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1} \rangle \xrightarrow{\mathsf{r}_2:\mathbf{dlvr}} \langle aa \mid \rangle \otimes_{\mathsf{op}} \langle a \mid a_1^{\mathsf{r}_1} \rangle$$

It is possible to simulate the above execution in the state-based system as follows. (The state-based system uses a *set* of messages as its state, and therefore replicas send *sets* of messages to other replicas.)

$$\begin{aligned}\langle | \rangle \otimes_{\mathsf{st}} \langle | \rangle &\xrightarrow{\mathsf{r}_1:\mathsf{upd}(a)} \xrightarrow{\mathsf{r}_1:\mathsf{send}(\mathsf{r}_2)} \langle a_0^{\mathsf{r}_1} \mid \rangle \otimes_{\mathsf{st}} \langle \mid \{a_0^{\mathsf{r}_1}\} \rangle \\ &\xrightarrow{\mathsf{r}_1:\mathsf{upd}(a)} \xrightarrow{\mathsf{r}_1:\mathsf{send}(\mathsf{r}_2)} \langle a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1} \mid \rangle \otimes_{\mathsf{st}} \langle \mid \{a_0^{\mathsf{r}_1}\}\{a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1}\} \rangle \\ &\xrightarrow{\mathsf{r}_2:\mathsf{dlvr}} \langle a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1} \mid \rangle \otimes_{\mathsf{st}} \langle a_0^{\mathsf{r}_1} \mid \{a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1}\} \rangle\end{aligned}$$

Note that queries on either replica return the same answers throughout (as required in simulation). For example, unfolding our compressed notation on op-based r_1 and state-based r_1 respectively,

$$\mathsf{query}(aa) = \mathsf{query}(\{a, a\}) = \{a, a\} = \mathsf{query}'(\{a_0^{\mathsf{r}_1}, a_1^{\mathsf{r}_1}\}) = \mathsf{query}'(a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1}).$$

Moreover, the simulation can continue. If r_1 or r_2 in the host system performs an update, there is clearly a matching transition in the guest system. If r_2 in the host system performs another deliver action, r_2 in the guest system can match by performing a merge:

$$\langle aa \mid \rangle \otimes_{\mathsf{op}} \langle a \mid a_1^{\mathsf{r}_1} \rangle \xrightarrow{\mathsf{r}_2:\mathbf{dlvr}} \langle aa \mid \rangle \otimes_{\mathsf{op}} \langle aa \mid \rangle$$

$$\textit{is matched by} \quad \langle a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1} \mid \rangle \otimes_{\mathsf{st}} \langle a_0^{\mathsf{r}_1} \mid \{a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1}\} \rangle \xrightarrow{\mathsf{r}_2:\mathsf{dlvr}} \langle a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1} \mid \rangle \otimes_{\mathsf{st}} \langle a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1} \mid \rangle.$$

And it is clear that both systems are back in a position where the simulation continues.

On the other hand, there is a simulation going the other way from the initial configurations. For example, if we start with the state-based system with the execution:

$$\langle | \rangle \otimes_{\mathsf{st}} \langle | \rangle \xrightarrow{\mathsf{r}_1:\mathsf{upd}(a)} \langle a_0^{\mathsf{r}_1} \mid \rangle \otimes_{\mathsf{st}} \langle | \rangle \xrightarrow{\mathsf{r}_1:\mathsf{upd}(a)} \langle a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1} \mid \rangle \otimes_{\mathsf{st}} \langle | \rangle \xrightarrow{\mathsf{r}_1:\mathsf{send}(\mathsf{r}_2)} \langle a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1} \mid \rangle \otimes_{\mathsf{st}} \langle \mid \{a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1}\} \rangle \tag{2.2}$$

Then, there is a matching execution in the op-based system.

$$\langle |\rangle \otimes_{\mathsf{op}} \langle |\rangle \xrightarrow{\mathsf{r}_1:\mathbf{upd}(a)} \langle a\ |\rangle \otimes_{\mathsf{op}} \langle |\ a_0^{\mathsf{r}_1}\rangle \xrightarrow{\mathsf{r}_1:\mathbf{upd}(a)} \langle aa\ |\rangle \otimes_{\mathsf{op}} \langle |\ a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1}\rangle \tag{2.3}$$

Now, if the state-based r_2 performs a merge, the op-based r_2 can simulate by performing *two* deliver actions. We can obtain a simulation of the state-based guest system by the op-based host system by generalizing appropriately.

But, we cannot have a weak *bi*simulation in this example. It we did, then we could "swap over" to the the op-based host system which has just executed (2.3) and and tell r_2 to deliver just $a_0^{\mathsf{r}_1}$. The state-based guest system in (2.2) *cannot* match this — r_2 can only merge $\{a_0^{\mathsf{r}_1}, a_1^{\mathsf{r}_1}\}$, which then "oversteps", since now r_2 will have state $\{a\}$ in the op-based system, and r_2 will have state $\{a_0^{\mathsf{r}_1}, a_1^{\mathsf{r}_1}\}$ in the state-based system, but

$$\mathsf{query}(a) = \mathsf{query}(\{a\}) = \{a\} \neq \{a, a\} = \mathsf{query}'(\{a_0^{\mathsf{r}_1}, a_1^{\mathsf{r}_1}\}) = \mathsf{query}'(a_0^{\mathsf{r}_1} a_1^{\mathsf{r}_1}).$$

This mismatch means that the simulation has broken.

What this example implies is that the emulation algorithms of Shapiro et al. [2011] are sensitive to the network semantics. While op-based CRDTs rely on a specific network semantics (i.e., reliable causal broadcast), state-based CRDTs are agnostic to it, and this agnosticism means that an external observer could potentially witness a difference in the set of possible executions if one replaces one kind of CRDT implementation with another, as in the above example. This makes the claim that one could easily transfer verification properties from one kind of CRDT to the another a bit dubious in the absence of a formal argument. We tackle this problem from a programmer's perspective, and later, provide some reassuring answers.

3 Formal System Semantics of CRDTs

We now properly formalize our operational semantics for CRDTs. The semantics are similar to those of Example 2.9, though the global states are slightly different. We start by describing the semantics in a CRDT-agnostic way, starting with the low-level building blocks of replicas and events, then higher-level components such as the global system state and configurations. For the purposes of this section, we fix a set $\mathtt{RID}$ of replica identifiers, and assume they are totally ordered.

3.1 Replicas and Events

We think of each replica in a system of CRDT replicas as independently implementing the same object specification — Figure 1 in the op-based case and Figure 2 in the state-based case. Let S be the *state space* with initial state $s^0 \in S$, E the set of *events*, and O the set of *outputs*. An object is then simply a state machine $\delta : S \times E \to \{\bot\} + S \times O$ equipped with a current state $s \in S$. All replicas implement this state machine δ, where a replica transitions states s to s' as being incurred by an event e, and producing output o.

Definition 3.1 (Replica)**.** Let S be a state space, and E and O be resp. events and outputs. A *replica* (of a state machine δ) is an identifier $\mathsf{r} \in \mathtt{RID}$ equipped with a current state $s \in S$, and the labeled transition relation $\longrightarrow_{\mathsf{r}}$ defined by the following rule, for all $s, s' \in S$, $e \in E$, and $o \in O$.

$$\frac{\delta(\mathsf{r}, s, e) \neq \bot \quad \delta(\mathsf{r}, s, e) = (s', o)}{s \xrightarrow{e/o}_{\mathsf{r}} s'}\ \textsc{Event}$$

We simply refer to the identifier set $\mathtt{RID}$ as a set of *replicas* if each $\mathsf{r} \in \mathtt{RID}$ is a replica of the same state machine. Implicit in this language is that each replica is initialized at the same starting

state s^0. We note that replicas are deterministic in the sense that for a fixed r and e, we have

$$\forall s \in S : s \xrightarrow{e/o'}_{\mathsf{r}} s' \wedge s \xrightarrow{e/o''}_{\mathsf{r}} s'' \implies s' = s'' \wedge o' = o''.$$

Our event sets E and output sets O have a special syntactic structure, which we now define.

Definition 3.2 (Events)**.** Let A be a set of arguments, Q be a set of queries, V a set of values, and M a set of messages. We use the special symbol $\bot$ to denote "null". We define the set of events E and outputs O according to the following grammar:

$$\begin{aligned} &\text{(Events } E\text{)} && e ::= \bot \mid \mathsf{upd}[\alpha] \mid \mathsf{qry}[q] \mid \mathsf{dlvr}[m] && \text{where } \alpha \in A, q \in Q, m \in M. \\ &\text{(Outputs } O\text{)} && o ::= \bot \mid \mathsf{res}[v] \mid \mathsf{send}[m] && \text{where } v \in V, m \in M. \end{aligned} \tag{3.1}$$

"Silent" or unobservable events are denoted by $\bot$. Update events are denoted $\mathsf{upd}[\alpha]$ and model a request by a client to update a replica with a tuple of arguments α, e.g., $\alpha = \langle \mathsf{add}, a \rangle$ as in Example 2.9. Query events model a request by a client to execute a query q, and are denoted $\mathsf{qry}[q]$. Deliver events $\mathsf{dlvr}[m]$ denote a replica consuming a message m from an external buffer.

The symbol $\bot$ is a "null" output, indicating that an event did not produce an output. A response output $\mathsf{res}[v]$ denotes a replica producing a value v, which is to be forwarded to the client who made the request. The output $\mathsf{send}[m]$ denotes that the message m should be pushed into the network by an external handler (e.g., `broadcast` as in Algorithm 1). We generally think of silent events $\bot$ and update events $\mathsf{upd}[\alpha]$ as producing either an output $\bot$ or $\mathsf{send}[m]$. The output $\mathsf{res}[v]$ is only produced by a query event $\mathsf{qry}[q]$.

3.2 System-level Preliminaries

Next, we model the semantics of a *system* of replicas that interact with each other. Just as individual replicas step following a transition system $\longrightarrow$, a system of replicas will step following a transition system $\rightsquigarrow$. Every system-level transition will correspond to a replica-level transition on some replica; however, the *choice* of which replica may step next is non-deterministic.[4]

We denote the global state of a system (called a *configuration*) with $\langle \Gamma \mid \Sigma \mid \beta \rangle$. It consists of three components: a *global state* denoted by the function $\Sigma : \mathtt{RID} \to S$ from replica ID r to its local state s (with initial state $\Sigma^0 = \lambda \mathsf{r} \in \mathtt{RID}\,.\, s^0$); a *network state* β that is a set of tuples $(\mathsf{r}_i, m_i) \in \mathtt{RID} \times M$, representing sent messages that are currently in transit; and a list Γ of tuples $(\mathsf{r}, e, o) \in \mathtt{RID} \times E \times O$ that we call an *event trace*. The event trace captures the execution of the system so far; if $(\mathsf{r}, e, o)_i \in \Gamma$, then replica r executed event e and produced output o at the ith transition step in the execution. For example, if r (chosen non-deterministically) makes a transition $s \xrightarrow{\bot/\mathsf{send}[m]}_{\mathsf{r}} s'$, the system-level trace Γ takes a step by appending $(\mathsf{r}, \bot, \mathsf{send}[m])$.

Notably, since individual replicas may only indicate a sent message as $\mathsf{send}[m]$, the choice of *recipients* is up to the system-level semantics. This choice will vary between op-based and state-based systems. In a similar vein, the choice of when to *deliver* an in-flight message is also delegated to the system-level semantics. Because clients of a CRDT do not interact with the internal network between replicas, we render delivery transitions as green "silent" transitions $\overset{\tau}{\rightsquigarrow}$.

For the most part, we only care about configurations $\langle \Gamma \mid \Sigma \mid \beta \rangle$ where Γ, Σ, and β "agree" with each other in some sense. That is, we are only interested in configurations $\langle \Gamma \mid \Sigma \mid \beta \rangle$ which are the result of some execution.

Definition 3.3 (Well-formed configurations)**.** Given an initial configuration `init`, we say a configuration $\langle \Gamma \mid \Sigma \mid \beta \rangle$ is *well-formed* if there is an execution from `init` to $\langle \Gamma \mid \Sigma \mid \beta \rangle$, i.e., a

[4] Our model of concurrency inherently has an *interleaving* interpretation.

sequence

$$\texttt{init} \overset{\alpha_1}{\rightsquigarrow}{}^* \langle \Gamma_1 \mid \Sigma_1 \mid \beta_1 \rangle \cdots \overset{\alpha_k}{\rightsquigarrow}{}^* \langle \Gamma \mid \Sigma \mid \beta \rangle.$$

3.3 Op-based CRDT Semantics

Assumption 3.4. In this section, we fix the sets S, $\texttt{Op}$, M, Q, and V of resp. states, operations, messages, queries, and values. We assume a distinguished initial state $s^0 \in S$. We assume functions $\texttt{prep}$, $\texttt{effect}$, and $\texttt{query}$ as in Figure 1, and an ordered set $\texttt{RID}$ of replica identifiers. We further assume an irreflexive, transitive partial order $\prec$ so that $(M, \prec)$ is a partially ordered set. The relevance of $\prec$ is discussed in Section 3.5.

To describe the replicas of an op-based CRDT, we instantiate Definition 3.1, using the op-based CRDT specification in Figure 1 and the algorithm for the op-based update in Algorithm 1 as a guide.

Definition 3.5 (Op-based Replica Semantics)**.** An op-based replica $r \in \texttt{RID}$ is defined by the initial state s^0 and the smallest labeled transition relation $\longrightarrow_{\mathsf{r}}$ closed under the rules in Figure 3.

$$\frac{q \in \texttt{Q} \quad v = \texttt{query}(q, s)}{s \xrightarrow{\texttt{qry}[q]/\texttt{res}[v]}_{\mathsf{r}} s} \qquad \frac{m \in M}{s \xrightarrow{\texttt{dlvr}[m]/\bot}_{\mathsf{r}} \texttt{effect}(s, m)} \qquad \frac{op \in \texttt{Op} \quad m = \texttt{prep}(s, op)}{s \xrightarrow{\texttt{upd}[op]/\texttt{send}[m]}_{\mathsf{r}} \texttt{effect}(s, m)}$$

Fig. 3. Op-based replica Semantics.

Transitions $s \xrightarrow{\texttt{upd}[op]/\texttt{send}[m]}_{\mathsf{r}} s'$ indicate that m is the message containing the effects of op, and should be broadcasted to all other replicas $\mathsf{r}' (\neq \mathsf{r})$. To effect this behavior, we use the following "broadcast" function (compare Algorithm 1):

$$\texttt{bcast}(\mathsf{r}, m)(\beta) = \beta \cup \{(\mathsf{r}', m) \mid \mathsf{r}' \in \texttt{RID} \wedge \mathsf{r}' \neq \mathsf{r}\}.$$

In words, if $\texttt{RID} = \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$, and $\mathsf{r} = \mathsf{r}_i$, then $\texttt{bcast}(\mathsf{r}, m)$ creates a packet

$$\{(\mathsf{r}_1, m), \ldots, (\mathsf{r}_{i-1}, m), (\mathsf{r}_{i+1}, m), \ldots, (\mathsf{r}_n, m)\}$$

of $n - 1$ copies of m tagged with destinations $\mathsf{r}_j (\neq \mathsf{r}_i)$, then pushes that packet into the network state β on behalf of r_i.

$$\frac{\mathsf{r} \in \texttt{RID} \quad op \in \texttt{Op} \quad s = \Sigma(\mathsf{r}) \quad s \xrightarrow{\texttt{upd}[op]/\texttt{send}[m]}_{\mathsf{r}} s'}{\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}:\texttt{upd}[op]/\bot}{\rightsquigarrow} \langle \Gamma \cdot (\mathsf{r}, \texttt{upd}[op], \texttt{send}[m]) \mid \Sigma[\mathsf{r} \mapsto s'] \mid \texttt{bcast}(\mathsf{r}, m)(\beta) \rangle} \text{ OpUpdate}$$

$$\frac{\mathsf{r} \in \texttt{RID} \quad q \in Q \quad s = \Sigma(\mathsf{r}) \quad s \xrightarrow{\texttt{qry}[q]/\texttt{res}[v]}_{\mathsf{r}} s}{\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}:\texttt{qry}[q]/\texttt{res}[v]}{\rightsquigarrow} \langle \Gamma \cdot (\mathsf{r}, \texttt{qry}, \texttt{res}[v]) \mid \Sigma \mid \beta \rangle} \text{ OpQuery}$$

$$\frac{(\mathsf{r}, m) \in \beta \quad (\mathsf{r}, \texttt{dlvr}[m]) \in \texttt{enabled}(\Gamma) \quad s = \Sigma(\mathsf{r}) \quad s \xrightarrow{\texttt{dlvr}[m]/\bot}_{\mathsf{r}} s'}{\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma \cdot (\mathsf{r}, \texttt{dlvr}[m], \bot) \mid \Sigma[\mathsf{r} \mapsto s'] \mid \beta \setminus \{(\mathsf{r}, m)\} \rangle} \text{ OpDeliver}$$

Fig. 4. Op-based system semantics. We assume an initial configuration $\texttt{init} = \langle \varepsilon \mid \Sigma^0 \mid \varnothing \rangle$.

Figure 4 gives the system-level transition system for op-based CRDTs. Note that the $\texttt{enabled}(\Gamma)$ predicate in DlvrEvent ensures that event traces Γ are well-formed in the sense of Definition 3.3. In particular, messages m must be delivered according to the causal order $\prec_{\mathrm{hb}}$. We defer discussion of causality and enabled until Section 3.5.

Definition 3.6 (Op-based System Semantics)**.** Given a set of op-based replicas RID (of the same op-based object), the system semantics of an op-based CRDT are defined by the initial configuration $\texttt{init} = \langle \varepsilon \mid \Sigma^0 \mid \varnothing \rangle$ and the smallest labeled transition relation $\rightsquigarrow$ on configurations closed under the rules in Figure 4.

3.4 State-based CRDT Semantics

Assumption 3.7. In this section, we fix the sets S, Op, Q, and V of resp. states, operations, queries, and values. We assume a distinguished initial state $s^0 \in S$. We assume $(S, \sqcup)$ is a join-semilattice, and we assume functions update, merge, and query as in Figure 2.

The semantics of state-based CRDTs are straightforward, since there are very few constraints on the behavior of the network. Like before, state-based replicas can be thought of as an instantiation of Definition 3.1 with Figure 2.

$$\frac{q \in \texttt{Q} \quad v = \texttt{query}(q, s)}{s \xrightarrow{\text{qry}[q]/\text{res}[v]}_{\texttt{r}} s} \qquad \frac{s' \in S}{s \xrightarrow{\text{dlvr}[s']/\bot}_{\texttt{r}} \texttt{merge}(s, s')} \qquad \frac{}{s \xrightarrow{\bot/\text{send}[s]}_{\texttt{r}} s} \qquad \frac{op \in \texttt{Op}}{s \xrightarrow{\text{upd}[op]/\bot}_{\texttt{r}} \texttt{update}(s, op)}$$

Fig. 5. State-based replica semantics.

Definition 3.8 (State-based Replica Semantics)**.** A replica $\texttt{r} \in \texttt{RID}$ of a state-based CRDT is defined by the initial state s^0 and the smallest labeled transition relation $\longrightarrow_{\texttt{r}}$ closed under the rules in Figure 5.

The semantics of state-based CRDTs are quite similar to op-based CRDTs at a local level, only now replicas transmit their state separately from their local updates, and this is considered a distinct (but silent!) computational event. Sending of state from one replica to another is a point-to-point message, where we assume a source replica r can choose any other replica $\texttt{r}' (\neq \texttt{r})$ as the target. The system-wide semantics are a straightforward lifting of the rules in Definition 3.8 to global configurations $\langle \Gamma \mid \Sigma \mid \beta \rangle$. We omit the enabled predicate, as state-based CRDTs place almost no constraints on the network, thanks to the join-semilattice structure of the state space.

Definition 3.9 (State-based CRDT Systems)**.** Given a set of state-based replicas RID (of the same state-based object), the system semantics of the state-based CRDT are defined by the initial configuration $\texttt{init} = \langle \varepsilon \mid \Sigma^0 \mid \varnothing \rangle$ and the smallest labeled transition relation $\rightsquigarrow$ on configurations closed under the rules in Figure 6.

3.5 Causality and enabled

Recall from Figure 1 that the effects of messages m and m' should commute if m and m' are concurrent, i.e., neither is a cause for the other. We want to encode this assumption in our semantics. Notice that in Assumption 3.7, we assumed the set of messages $(M, \prec)$ is a partially ordered set. Going forward, we will understand $\prec$ to characterize *causality* in the sense of Lamport's happens-before relation [Lamport 1978]:

Assumption 3.10. If a replica r' sent m', and a replica r sent m, then $m' \prec m$ if and only if r' sent m' causally before r sent m.

We then define

$$m' \parallel m \stackrel{\text{def}}{=} \neg(m' \prec m) \wedge \neg(m \prec m'),$$

and make the following assumption with how it interacts with op-based CRDTs:

$$\frac{\mathsf{r} \in \mathtt{RID} \quad op \in \mathtt{Op} \quad s = \Sigma(\mathsf{r}) \quad s \xrightarrow{\mathrm{upd}[op]/\bot}_{\mathsf{r}} s'}{\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}:\mathrm{upd}[op]/\bot}{\rightsquigarrow} \langle \Gamma \cdot (\mathsf{r}, \mathrm{upd}[op], \bot) \mid \Sigma[\mathsf{r} \mapsto s]' \mid \beta \rangle} \text{ StUpdate}$$

$$\frac{\mathsf{r} \in \mathtt{RID} \quad q \in Q \quad s = \Sigma(\mathsf{r}) \quad s \xrightarrow{\mathrm{qry}[q]/\mathrm{res}[v]}_{\mathsf{r}} s}{\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}:\mathrm{qry}[q]/\mathrm{res}[v]}{\rightsquigarrow} \langle \Gamma \cdot (\mathsf{r}, \mathrm{qry}[q], \mathrm{res}[v]) \mid \Sigma \mid \beta \rangle} \text{ StQuery}$$

$$\frac{\mathsf{r}, \mathsf{r}' \in \mathtt{RID} \quad \mathsf{r}' \neq \mathsf{r} \quad s = \Sigma(\mathsf{r}) \quad s \xrightarrow{\bot/\mathrm{send}[s]}_{\mathsf{r}} s}{\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma \cdot (\mathsf{r}, \bot, \mathrm{send}[s]) \mid \Sigma \mid \beta \cup \{(\mathsf{r}', s)\} \rangle} \text{ StSend}$$

$$\frac{(\mathsf{r}, s'') \in \beta \quad (\mathsf{r}, \mathrm{dlvr}[s''], \_) \notin \Gamma \quad s = \Sigma(\mathsf{r}) s \xrightarrow{\mathrm{dlvr}[s'']/\bot}_{\mathsf{r}} s'}{\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma \cdot (\mathsf{r}, \mathrm{dlvr}[s''], \bot) \mid \Sigma[\mathsf{r} \mapsto s'] \mid \beta \setminus \{(\mathsf{r}, s'')\} \rangle} \text{ StDeliver}$$

Fig. 6. State-based system semantics. We assume an initial configuration $\mathtt{init} = \langle \varepsilon \mid \Sigma^0 \mid \varnothing \rangle$.

Assumption 3.11. Let $\langle \Gamma \mid \Sigma \mid \beta \rangle$ a configuration and r a replica. If $m' \parallel m$, and

$$\Sigma(\mathsf{r}) \xrightarrow{\mathrm{dlvr}[m]/\bot}_{\mathsf{r}} s_1' \xrightarrow{\mathrm{dlvr}[m']/\bot}_{\mathsf{r}} s_1'' \text{ and } \Sigma(\mathsf{r}) \xrightarrow{\mathrm{dlvr}[m']/\bot}_{\mathsf{r}} s_2' \xrightarrow{\mathrm{dlvr}[m]/\bot}_{\mathsf{r}} s_2'' \implies s_1'' = s_2''.$$

Next, we must ensure that our semantics enforce an ordering of message delivery that is consistent with causality (i.e., consistent with $\prec$). We define the following safety condition.

Definition 3.12 (Causal message delivery)**.** Let M be a set and $m, m' \in M$ a pair of messages. We say Γ satisfies *causal delivery order* if, $\forall m, m' \in M$, and $\forall \mathsf{r} \in \mathtt{RID}$,

$$(m \prec m') \wedge ((\mathsf{r}, \mathrm{dlvr}[m], \bot)_i \in \Gamma) \wedge ((\mathsf{r}, \mathrm{dlvr}[m'], \bot)_j \in \Gamma) \tag{3.2}$$

$$\implies \neg(\mathrm{dlvr}[m'] \textit{ ordered before } \mathrm{dlvr}[m] \textit{ in } \Gamma|_{\mathsf{r}} \textit{ i.e., } i < j). \tag{3.3}$$

On the other hand, if $m' \parallel m$, then there is no constraint on the order of $\mathrm{dlvr}[m]$, and $\mathrm{dlvr}[m']$.

Having defined the neccesary safety conditions, we define the `enabled` predicate such that, for all well-formed $\langle \Gamma \mid \Sigma \mid \beta \rangle$, event trace Γ satisfies causal delivery delivery.

Definition 3.13. Given an event trace Γ, we say $\mathrm{dlvr}[m]$ is *enabled* at r (wrt Γ) and write $(\mathsf{r}, \mathrm{dlvr}[m]) \in \mathsf{enabled}(\Gamma)$ if and only if both the following hold:

(1) $\forall o \in O$, $(\mathsf{r}, \mathrm{dlvr}[m], o) \notin \Gamma$ (i.e., m has not already been delivered);

(2) $\forall m' \in M$ such that $m' \prec m$, we have $(\mathsf{r}, \mathrm{dlvr}[m'], \bot) \in \Gamma$ (i.e., causally preceding m' have been delivered).

The second condition in Definition 3.13 enforces the causal delivery order. It says that any message m' that is a potential cause for m must be delivered *before* m is allowed to be delivered.

Remark 3.14 (Relaxation of causal delivery order)**.** When all messages commute irrespective of causal order, (e.g., Equation (2.1)), we can drop the second condition without loss of generality. This can simplify some proofs. We will point out explicitly when this occurs.

The next proposition asserts that for all well-formed configurations $\langle \Gamma \mid \Sigma \mid \beta \rangle$, Γ satisifies causal delivery.

Proposition 3.15 (Causal Order Enforcement)**.** *Suppose $\langle \Gamma \mid \Sigma \mid \beta \rangle$ is a well-formed op-based configuration. Then Γ satisfies causal delivery order.*

Remark 3.16. There are known, practical methods of ensuring message ordering characterizes causality, such as including timestamps drawn from a vector clock in each message [Mattern 2002]. The appendices (Appendix A) additionally include a derivation of the happens-before relation itself in terms of traces Γ, and a proof of Proposition 3.15.

4 CRDT Emulation as (Weak) Simulation

In this section, we show that Shapiro et al.'s notion of emulation can be formalized as simulation in the following sense. Let κ and λ denote two distinct classes of CRDTs (e.g., op-based and state-based, with semantics as in resp. Figures 4 and 6). That is, a CRDT object O_κ corresponds to the labeled transition system $(\mathsf{Config}_\kappa, \mathsf{init}_\kappa, \rightsquigarrow_\kappa)$, where $\mathsf{init}_\kappa \in \mathsf{Config}_\kappa$ is the initial configuration.

Definition 4.1 (CRDT Emulation)**.** We say a mapping $\mathcal{G}$ is an *emulation* if, given any *host* CRDT object O_κ with system semantics $(\mathsf{Config}_\kappa, \mathsf{init}_\kappa, \rightsquigarrow_\kappa)$, there is a *guest* CRDT object O_λ (with corresponding system semantics) such that:

- $(\mathsf{Config}_\kappa, \mathsf{init}_\kappa, \rightsquigarrow_\kappa) \overset{\mathcal{G}}{\longmapsto} (\mathsf{Config}_\lambda, \mathsf{init}_\lambda, \rightsquigarrow_\lambda)$; and
- the initial configurations weakly simulate each other, e.g., $\mathsf{init}_\kappa \lesssim\gtrsim \mathsf{init}_\lambda$.

In words, the guest system weakly simulates the host system and vice versa. This is not the same as having $\mathcal{G}$ be a homomorphism, because a homomorphism between two state spaces X_κ and X_λ yields a correspondence that is strong enough to imply *strong bi*simulation. Definition 4.1, on the other hand, only requires a pair of weak simulations that contain the *initial states* $\mathsf{init}_\kappa \in X_\kappa, \mathsf{init}_\lambda \in X_\lambda$ (i.e., there is not neccesarily a direct mapping from each state in one system to a state in the other that preserves and reflects transitions).

In this section, we focus on the emulation $\mathcal{G}$ from ***op-based*** CRDTs to ***state-based*** CRDTs. For this single emulation, there are *two* weak simulations, one from the host CRDT to the guest CRDT, and vice versa. We relegate the emulation from state-based CRDTs to op-based CRDTs to Appendix B. The reason for this is because the op-based-to-state-based emulation $\mathcal{G}$ is the more interesting direction and a good exemplar of our techniques, and it corresponds to our motivating example in Example 2.9.

Section 4.1 describes the mapping $\mathcal{G}$ used by the op-based to state-based direction, and Section 4.2 describes the two weak simulations that make $\mathcal{G}$ an emulation. Finally, in Section 4.3 we make good on our earlier assertion that there exist a subset of properties of CRDTs that do transfer via emulation, by describing them as a corollary.

4.1 The mapping $\mathcal{G}$ from op-based to state-based CRDTs

Assumption 4.2. We assume we are given an op-based CRDT object (Figure 1), with its corresponding sets S, Op, M, Q, V, related functions, and semantics as in Figure 4, and the set of messages is an irreflexive partial order $(M, \prec)$ as laid out in Section 3.5. The only sets $h \in \mathcal{P}(M)$ we consider here are *finite* sets, so that $\prec$ restricted to h is well-founded.

Notation 4.3. For this section, the representation of a function $T : X \times A \to X$ as a monoid action is helpful. Let A^* be the set of words on A, and define $T_w : X \to X$ (for $w \in A^*$) recursively by (1) $T_\varepsilon(x) = x$; and (2) if $a \in A$, and $w \in A^*$, then $T_{aw}(x) = T_w(T(x, a))$. We then write $x \cdot w \overset{\text{def}}{=} T_w(x)$.

The mapping $\mathcal{G}$ we consider is inspired by the original algorithm described by Shapiro et al. [2011], and we direct interested readers there to see it in its original presentation.

In simple terms, op-based-to-state-based emulation relies on what has classically been described as the *semantic characterization of state machines*, which says that the current state of a state machine is defined only by the sequence of requests it processes, and by nothing else [Schneider

1990]. Replicas are indeed state machines, so it seems natural to think that, instead of maintaining the current op-based state s, a replica can instead maintain the initial state s^0 and the *the set of messages that led to* s, i.e., an ordered set of messages $\{m_1, m_2, \ldots, m_k\}$ such that

$$s = s^0 \cdot m_1 \cdot m_2 \cdots m_k \stackrel{\text{def}}{=} \texttt{effect}_{m_1 \cdot m_2 \cdots m_k}(s^0).$$

The fact that each set of messages $\{m_1, \ldots, m_k\}$ is an element of the join-semilattice $(\mathcal{P}(M), \cup)$ (where $\cup$ is set-theoretic union) is even better — there is a representation of each op-based state s as a state-based state $\{m_1, \ldots, m_k\}$.[5] We just need each replica to have access to: (1) the initial state s^0, and (2) a function $\texttt{interp}_S : \mathcal{P}(M) \to S$ that can interpret the set of messages $\{m_1, \ldots, m_k\}$ into s. The first point is trivial — we can always assume replicas have access to some additional metadata. For the second point, $\texttt{interp}_S$ is roughly defined by taking a message set $H \in \mathcal{P}(M)$, choosing some linear ordering $\langle m_1, \ldots, m_{|H|} \rangle$ of H that is consistent with $\prec$, then computing $s^0 \cdot m_1 \cdots m_{|H|}$.

A precise definition of $\texttt{interp}_S$ turns out to be interesting, as it is not immediately clear why such a function should even be well-defined: a set H is not equipped with a total order, and so one needs to check that the candidate $\texttt{interp}_S$ gives results independent of the ordering chosen. It turns out that since $\prec$ is an irreflexive, transitive relation that preserves and reflects causal order, we can use the fact that the effects of concurrent messages commute (Section 3.5) to define $\texttt{interp}_S$ recursively. The key idea is to define $\mathsf{Max}(H) \stackrel{\text{def}}{=} \{m \in H \mid \nexists m' \in H \text{ s.t. } m \prec m'\}$. Critically, each pair of messages $m, m' \in \mathsf{Max}(H)$ are concurrent, i.e., $m \parallel m'$. We then define

$$\texttt{interp}_S(H) \stackrel{\text{def}}{=} \begin{cases} s^0 & \text{if } H = \varnothing \\ \texttt{effect}(\texttt{interp}_S(H \setminus \{m\}), m) & \text{if } m \in \mathsf{Max}(H). \end{cases} \tag{4.1}$$

This function is well-defined, and terminates whenever H is finite (which is the case.) It essentially recursively picks an arbitrary order of the messages in H, but in such a way as to be consistent with the order $\prec$. We give a proof of the construction in Appendix C.

With these ingredients in place, the candidate emulation map $\mathcal{G}$ is the one that takes an op-based CRDT object as in Figure 1 and constructs the state-based CRDT object shown in Figure 7. We already know that $(\mathcal{P}(M), \cup)$ is a join-semilattice, so it remains to check that `update` is inflationary — but it clearly is, since it's defined in terms of $\cup$.

Parameters :

$\mathcal{P}(M)$: states, Op : operations, Q : queries, V : values,

$\varnothing \in \mathcal{P}(M)$: initial state

Functions :

$$\texttt{update}(\mathsf{r}, H, \mathsf{op}) \stackrel{\text{def}}{=} H \cup \{\texttt{prep}(\texttt{interp}_S(H), \mathsf{op})\}$$

$$\texttt{merge}(H, H') \stackrel{\text{def}}{=} H \cup H'$$

$$\texttt{query}(q, H) \stackrel{\text{def}}{=} \texttt{query}(q, \texttt{interp}_S(H))$$

Fig. 7. The state-based guest CRDT constructed by $\mathcal{G}$ from a given op-based host CRDT.

[5] We can think of s as the *denotation* of $\{m_1, \ldots, m_k\}$.

4.2 The weak simulations

We now proceed with defining the pair of weak simulations that make $\mathcal{G}$ an emulation in the sense of Definition 4.1.

Notation 4.4. Let $\mathsf{Sent}(\Gamma)$ to denote the set $\{m \mid (\_, \_, \mathsf{send}[m]) \in \Gamma\}$, i.e., the set of all sent messages. Let $\mathsf{Delivered}(\mathsf{r}, \Gamma)$ be the set of "delivered" messages at replica r, i.e., the set of messages $\{m_1, \ldots, m_k\}$ which r either delivered with a $\mathsf{dlvr}[m]$ event, or generated (then consumed via `effect`) with an $\mathsf{upd}[\mathsf{op}]$ event. We use $(m)^{\downarrow\Gamma}$ to denote the set $\{m' \in \mathsf{Sent}(\Gamma) \mid m' \prec m\} \cup \{m\}$. In words, it is the downwards closure of m in the event trace Γ. We write $\rightsquigarrow_{\mathcal{G}}$ to denote the fact that the state-based CRDT system is the *guest* system.

From now on, we assume all configurations $\langle \Gamma \mid \Sigma \mid \beta \rangle$, $\langle \Gamma \mid \Sigma \mid \beta \rangle$ are *well-formed* (Definition 3.3). We further assume a fixed set of replicas $\{\mathsf{r}_1, \ldots, \mathsf{r}_n\} = \mathtt{RID}$. For space reasons, we only sketch proofs by showing archetypical cases. Simulation proofs tend to be quite mechanical, so the other cases follow a similar pattern. Complete proofs can be found in Appendix C.

4.2.1 The guest CRDT weakly simulates the host CRDT. The weak simulation of $\rightsquigarrow$ by $\rightsquigarrow_{\mathcal{G}}$ is rather straightforward, but with one technical hurdle that forms the heart of Example 2.9, and thus of our simulation arguments. The crucial detail is that whenever an op-based replica r executes a $\mathbf{dlvr}[m]$ event, the state-based replica r needs to pick a *set* of messages H whose merge (i.e., $\mathsf{dlvr}[H]$) *simulates* the $\mathbf{dlvr}[m]$ deliver event. This means our weak simulation needs to guarantee that a "small enough" H is always available. The right choice of H is actually the downset $(m)^{\downarrow\Gamma}$, which actually *is* available so long as we always simulate `bcast` with an appropriate sequence of point-to-point sends — forcing the state-based system to broadcast just as often as the op-based system ensures there is always such a "small enough" state.

Theorem 4.5. *$\mathcal{R}_1$ (shown in Figure 8) is a weak simulation of $\rightsquigarrow$ by $\rightsquigarrow_{\mathcal{G}}$ that relates the initial configurations $\langle \varepsilon \mid \Sigma^0 \mid \varnothing \rangle$ and $\langle \varepsilon \mid \Sigma^0 \mid \varnothing \rangle$.*

$\mathcal{R}_1(\langle \varepsilon \mid \Sigma^0 \mid \varnothing \rangle, \langle \varepsilon \mid \Sigma^0 \mid \varnothing \rangle)$ is immediate. We only need to show that $\mathcal{R}_1$ is indeed a weak simulation. We prove the following archetypical case — the other cases follow a similar pattern, and can be found in Appendix C.

Sketch of Theorem 4.5. Assume as our hypothesis: for some given α,

$$\mathcal{R}_1(\langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle) \text{ and we have, } \langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\alpha}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle.$$

The archetypical case is

- (OpDeliver). Then $\alpha = \tau$, and we have $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$, where

  $$\Gamma' = \Gamma \cdot (\mathsf{r}, \mathbf{dlvr}[m], \bot), \quad \Sigma' = \Sigma[\mathsf{r} \to \mathtt{effect}(\Sigma(\mathsf{r}), m)], \quad \beta' = \beta \setminus \{(\mathsf{r}, m)\},$$

  which means r delivered an enabled message $(\mathsf{r}, m) \in \beta$. The hypothesis implies $\exists (\mathsf{r}, (m)^{\downarrow\Gamma}) \in \beta$. Therefore we can invoke StDeliver and match with

  $$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow}_{\mathcal{G}} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle,$$
  $$\Gamma' = \Gamma \cdot (\mathsf{r}, \mathsf{dlvr}[(m)^{\downarrow\Gamma}], \bot), \quad \Sigma' = \Sigma[\mathsf{r}_i \mapsto \Sigma(\mathsf{r}) \cup (m)^{\downarrow\Gamma}], \quad \beta' = \beta \setminus \{(\mathsf{r}, (m)^{\downarrow\Gamma})\}.$$

  To finish, we need to show $\mathcal{R}_1(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle)$ by checking the conditions in Figure 8. Since no messages were sent, the last condition is immediate from the hypothesis. We focus on the changes that happened at replica r (r). Note that since m was enabled at r, all causally preceding messages (i.e., $m' \prec m$) had to have already been delivered to at replica r. But since

$$\mathcal{R}_1 = \{(\langle \Gamma \mid \Sigma \mid \beta\rangle, \langle \Gamma \mid \Sigma \mid \beta\rangle) \mid$$
$$\text{— } \textit{configurations agree on messages, and replicas agree on delivered/merged messages}$$
$$(\mathsf{Sent}(\Gamma) = \bigcup \mathsf{Sent}(\Gamma)) \wedge \forall \mathsf{r} \in \mathtt{RID} : \mathsf{Delivered}(\mathsf{r}, \Gamma) = \Sigma(\mathsf{r})$$
$$\text{— } \textit{replicas agree on their local states}$$
$$\wedge \quad \forall \mathsf{r} \in \mathtt{RID} : \Sigma(\mathsf{r}) = \mathtt{interp}_S(\Sigma(\mathsf{r}))$$
$$\text{— } \textit{every pending message in the op-system has a pending downset in the state-system}$$
$$\wedge \quad \forall (\mathsf{r}, m) \in \beta \implies (\mathsf{r}, (m)^{\downarrow\Gamma}) \in \beta\}.$$

$$\mathcal{R}_2 = \{(\langle \Gamma \mid \Sigma \mid \beta\rangle, \langle \Gamma \mid \Sigma \mid \beta\rangle) \mid$$
$$\text{— } \textit{configurations agree on messages, and replicas agree on delivered/merged messages}$$
$$(\bigcup \mathsf{Sent}(\Gamma) = \mathsf{Sent}(\Gamma)) \wedge \forall \mathsf{r} \in \mathtt{RID} : \Sigma(\mathsf{r}) = \mathsf{Delivered}(\mathsf{r}, \Gamma)$$
$$\text{— } \textit{replicas agree on their local states}$$
$$\wedge \quad \forall \mathsf{r} \in \mathtt{RID} : \mathtt{interp}_S(\Sigma(\mathsf{r})) = \Sigma(\mathsf{r})$$
$$\text{— } \textit{every merge is a merge of a downset, and op-based replica can simulate that merge}$$
$$\wedge \quad \forall (\mathsf{r}, H) \in \beta : \exists U \in \mathsf{deliverable}_\Gamma(\mathsf{r}) : H = (U)^{\downarrow\Gamma}\}.$$

Fig. 8. The relation $\mathcal{R}_1 \subseteq \mathbf{Config} \times \mathsf{Config}$ is a weak simulation of op-based *host* CRDT is simulated by the state-based *guest* CRDT. Dually, the relation $\mathcal{R}_2 \subseteq \mathsf{Config} \times \mathbf{Config}$ is a weak simulation of the state-based *guest* CRDT by the op-based *host*. These relations, while very similar to each other, are not converses of each other — indeed, they cannot be (as Example 2.9 shows).

$\mathsf{Delivered}(\mathsf{r}, \Gamma) = \Sigma(\mathsf{r})$ (by hypothesis), all such $m' \in \Sigma(\mathsf{r})$ as well. Thus,

$$\Sigma'(\mathsf{r}) = \Sigma(\mathsf{r}) \cup (m)^{\downarrow\Gamma} = \Sigma(\mathsf{r}) \cup \{m\} = \mathsf{Delivered}(\mathsf{r}, \Gamma').$$

From the above equality, each condition required in $\mathcal{R}_1$ is immediate. Indeed, since either $m' \parallel m$ or $m' \prec m$ for each $m' \in \Sigma(\mathsf{r})$, we have the key equality

$$\mathtt{interp}_S(\Sigma'(\mathsf{r})) = \mathtt{interp}_S(\Sigma(\mathsf{r}) \cup \{m\}) = \mathtt{effect}(\mathtt{interp}_S(\Sigma(\mathsf{r})), m) = \Sigma'(\mathsf{r}).$$

We immediately obtain $\mathcal{R}_1(\langle \Gamma' \mid \Sigma' \mid \beta'\rangle, \langle \Gamma' \mid \Sigma' \mid \beta'\rangle)$ as desired.

□

4.2.2 The host CRDT weakly simulates the guest CRDT. The weak simulation of $\rightsquigarrow_{\mathcal{G}}$ by $\rightsquigarrow$ proceeds similarly, and is in some sense a bit easier, because the transition semantics of op-based CRDTs are naturally more "fine-grained", as we saw near the end of Example 2.9. The only real technical difficulty is maintaining the invariant: if the state-based replica `r` invokes `merge`, the op-based system can pick a sequence of messages $(\mathsf{r}, m_1), \ldots, (\mathsf{r}, m_k) \in \beta$ whose sequential delivery simulates that merge. As before, we prove an archetypical case. The other cases are mechanically quite similar.

Definition 4.6. In an op-based configuration $\langle \Gamma \mid \Sigma \mid \beta\rangle$, we say a subset of messages

$$U = \{m_1, \ldots, m_k\} \subseteq M$$

is *deliverable at a replica* r (and write $U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$) if for all $j \in 1..k$,

(1) $(\mathsf{r}, m_j) \in \beta$ and

(2) $(\mathsf{r}, \mathsf{dlvr}[m_j]) \in \mathsf{enabled}(\Gamma \cdot (\mathsf{r}, \mathsf{dlvr}[m_1], \bot) \cdots (\mathsf{r}, \mathsf{dlvr}[m_{j-1}], \bot))$.

Theorem 4.7. *$\mathcal{R}_2$ (shown in Figure 8) is a weak simulation of $\rightsquigarrow_{\mathcal{G}}$ by $\rightsquigarrow$ which relates the initial configurations $\langle \varepsilon \mid \Sigma^0 \mid \varnothing \rangle$ and $\langle \varepsilon \mid \Sigma^0 \mid \varnothing \rangle$.*

Sketch of Theorem 4.7. Assume as our hypothesis: for some given α,

$$\mathcal{R}_2(\langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle) \text{ and we have } \langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\alpha}{\rightsquigarrow}_{\mathcal{G}} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle.$$

- (StDeliver). Then $\alpha = \tau$ and we have $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow}_{\mathcal{G}} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$ where

$$\Gamma' = \Gamma \cdot (\mathsf{r}, \mathsf{dlvr}[H], \bot), \quad \Sigma' = \Sigma[\mathsf{r} \mapsto \mathtt{merge}(\Sigma(\mathsf{r}), H)], \quad \beta' = \beta \setminus \{(\mathsf{r}, H)\},$$

which means a replica r merged a state $(\mathsf{r}, H) \in \beta$ that was sent by some other replica. If $\Sigma(\mathsf{r}) \cup H = \Sigma(\mathsf{r})$, we match by the reflexive step $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow}{}^* \langle \Gamma \mid \Sigma \mid \beta \rangle$, and there is nothing to show, so we suppose there is a number $k > 0$, and messages $m_1, \ldots, m_k \notin \Sigma(\mathsf{r})$, so that $\Sigma(\mathsf{r}) \cup H = \Sigma(\mathsf{r}) \cup \{m_1, \ldots, m_k\}$.

(Claim). We can pick as $U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$ the set $\{m_1, \ldots, m_k\}$, therefore deliver the sequence of messages $m_1 \cdots m_k$ at replica r so that

$$\Sigma(\mathsf{r}) \xrightarrow{\mathsf{r} : \mathsf{dlvr}[m_1]/\bot} s_1 \cdots \xrightarrow{\mathsf{r} : \mathsf{dlvr}[m_k]/\bot} s_k$$

and thereby match with k applications of (OpDeliver) with respect to replica r giving:

$$\begin{aligned} &\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow}{}^* \langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \\ &\Gamma' = \Gamma \cdot \langle (\mathsf{r}, \mathsf{dlvr}[m_1], \bot), \ldots, (\mathsf{r}, \mathsf{dlvr}[m_k], \bot) \rangle, \\ &\Sigma' = \Sigma[\mathsf{r} \to s_k], \quad \beta' = \beta \backslash \{(\mathsf{r}, m_1), \ldots, (\mathsf{r}, m_k)\}. \end{aligned} \tag{4.2}$$

(Proof of Claim). From the hypothesis, $\exists U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$ such that $H = (U)^{\downarrow\Gamma}$. Suppose U has $k' \geq 0$ messages so that $\{m'_1, \ldots, m'_{k'}\} = U$. Since $U \subseteq (U)^{\downarrow\Gamma} = H$, and (by the hypothesis) $\mathsf{Delivered}(\mathsf{r}, \Gamma) = \Sigma(\mathsf{r})$, the "already merged" portion of messages at r is $H \setminus U$, hence we obtain:

$$\Sigma(\mathsf{r}) \cup H = \Sigma(\mathsf{r}) \cup U = \mathsf{Delivered}(\mathsf{r}, \Gamma) \cup U \tag{4.3}$$

$$U = \{m'_1, \ldots, m'_{k'}\} = \{m_1, \ldots, m_k\}. \tag{4.4}$$

Since by hypothesis, $U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$, it is clear we can deliver the sequence $m_1 \cdots m_k$ at replica r (reindexing if needed), we obtain (4.2) immediately. □
It remains to show that $\mathcal{R}_2(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle)$. For that, note that (4.3) implies

$$\Sigma'(\mathsf{r}) = \Sigma(\mathsf{r}) \cup U = \mathsf{Delivered}(\mathsf{r}, \Gamma) \cup U = \mathsf{Delivered}(\mathsf{r}, \Gamma'). \tag{4.5}$$

The only other interesting condition is showing that replicas agree in their local states, where it suffices to prove $s_k = \mathtt{interp}_S(\mathtt{merge}(\Sigma(\mathsf{r}), H))$. But this has to be the case: since $\{m_1, \ldots, m_k\} = U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$ means each pair $m_i, m_j \in U$ $(i < j)$ satisfies either $m_i \prec m_j$ or $m_i \parallel m_j$, and (4.5) means r and r have delivered the same set of messages, we obtain

$$s_k = \mathtt{effect}_{m_1 \cdots m_k}(\Sigma(\mathsf{r})) = \mathtt{interp}_S(\Sigma(\mathsf{r}) \cup U) = \mathtt{interp}_S(\mathtt{merge}(\Sigma(\mathsf{r}), H)).$$

The other conditions are immediate, and we obtain $\mathcal{R}_2(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle)$ as needed.

□

4.3 Consequences of CRDT Emulation as Simulation

In this section, we note some direct consequences of our view of CRDT emulation as (weak) simulation.

We first recall the definition of a weak trace set. Let $(X, \Lambda, \longrightarrow)$ be a labeled transition system. The *weak trace set* of $x \in X$ is defined as

$$\mathsf{WTrace}(x) \stackrel{\text{def}}{=} \{\langle \alpha_1, \ldots, \alpha_k \rangle \in (\Lambda \setminus \{\tau\})^* \mid \exists x_1, \ldots, x_k \in X\, (x \xrightarrow{\alpha_1}{}^* x_1 \cdots \xrightarrow{\alpha_k}{}^* x_k)\}.$$

In words, it is the set of all finite (and unbounded) observable behaviors starting from a state x. Critically, weak simulations are *sound* with respect to *weak trace inclusion* in the following sense: if $x \in X$ and $y \in Y$ are states, then

$$x \lessapprox y \implies \mathsf{WTrace}(x) \subseteq \mathsf{WTrace}(y).$$

Further, there is a notion of *weak* trace equivalence:

$$x \lessapprox\gtrapprox y \implies \mathsf{WTrace}(x) = \mathsf{WTrace}(y).$$

Our results in Section 4 thus imply the following.

Corollary 4.8 (Op-based to state-based weak trace equivalence)**.** *Let $\mathcal{O}$ be an op-based host CRDT with corresponding transition semantics $\rightsquigarrow$ and initial configuration* **init***. Let $\mathcal{G}_1$ be an emulation (Definition 4.1) that relates $\mathcal{O}$ to the state-based guest CRDT $\mathcal{O}_{\mathcal{G}_1}$ with transition semantics $\rightsquigarrow_{\mathcal{G}_1}$ and initial configuration $\mathsf{init}_{\mathcal{G}_1}$. Then,* **init** $\lessapprox\gtrapprox \mathsf{init}_{\mathcal{G}_1}$*, and therefore*

$$\mathsf{WTrace}(\textbf{init}) = \mathsf{WTrace}(\mathsf{init}_{\mathcal{G}_1}).$$

The next consequence is derived in a way that is mechanically quite similar to that in Section 4. We sketch it in Appendix B.

Corollary 4.9 (State-based to op-based weak trace equivalence)**.** *Let $\mathcal{O}$ be a state-based host CRDT with corresponding transition semantics $\rightsquigarrow$, and initial configuration* init*. Let $\mathcal{G}_2$ be an emulation (Definition 4.1) that relates to $\mathcal{O}$ the op-based guest CRDT $\mathcal{O}_{\mathcal{G}_2}$ with transition semantics $\rightsquigarrow_{\mathcal{G}_2}$ and initial configuration* $\textbf{init}_{\mathcal{G}_2}$*. Then,* init $\lessapprox\gtrapprox \textbf{init}_{\mathcal{G}_2}$*, and therefore*

$$\mathsf{WTrace}(\mathsf{init}) = \mathsf{WTrace}(\textbf{init}_{\mathcal{G}_2}).$$

What Corollaries 4.8 and 4.9 imply is that for every *given* weak trace σ_w of a given host CRDT system, *there exists* some (non-weak) trace σ' of the guest CRDT system so that $\sigma_w = \sigma'_w$, where σ'_w is σ' with all τ events erased. Moveover, this is only true if the starting configurations in both systems are related by a weak simulation. Weak trace equivalence is therefore the *precise* meaning of CRDT emulation, and the notion that "op-based and state-based CRDTs are equivalent". We now make good on our promise to name a class of properties that are preserved (and reflected) by CRDT emulation.

Corollary 4.10 (Emulation preserves weak trace properties)**.** *Suppose P is a predicate on weak traces (i.e., a $P \subseteq \mathsf{Weak}(\mathcal{T})$, where $\mathcal{T}$ is the set of all traces). If $\mathcal{O}$ is a host CRDT and $\mathcal{O}_{\mathcal{G}}$ a guest CRDT constructed by an emulation $\mathcal{G}$, we have the following, for all resp. configurations C and $\mathsf{C}_{\mathcal{G}}$,*

$$\mathsf{C} \lessapprox\gtrapprox \mathsf{C}_{\mathcal{G}} \implies (\mathsf{WTrace}(\mathsf{C}) \subseteq P \iff \mathsf{WTrace}(\mathsf{C}_{\mathcal{G}}) \subseteq P),$$

and in particular, the following is true: $\mathsf{WTrace}(\mathsf{init}) \subseteq P \iff \mathsf{WTrace}(\mathsf{init}_{\mathcal{G}}) \subseteq P$.

In words, properties on weak traces P are those that can be safely verified in one kind of CRDT (e.g., op-based) and then "transferred" to the other kind of CRDT (e.g., state-based). While this does not seem surprising, we would like to point out that we have shown Corollary 4.10 with very weak assumptions on the communication pattern of CRDTs. Indeed, we only require the op-based CRDTs execute broadcast as they are supposed to, and we let state-based CRDTs execute non-deterministic

point-to-point sends. The assumption on communication has relevance, since one can prove the following theorem. (We sketch the proof in Appendix C.)

Theorem 4.11. *In Definition 3.9, delete the StSend rule, and in place of StUpdate, use the following rule:*

$$\frac{\mathsf{r} \in \mathsf{RID} \quad op \in \mathsf{Op} \quad s = \Sigma(\mathsf{r}) \quad s \xrightarrow{\mathsf{upd}[op]/\mathsf{send}[s']}_{\mathsf{r}} s'}{\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}:\mathsf{upd}[op]/\bot}{\rightsquigarrow} \langle \Gamma \cdot (\mathsf{r}, \mathsf{upd}[op], \mathsf{send}[m]) \mid \Sigma[\mathsf{r} \mapsto s'] \mid \mathbf{bcast}(\mathsf{r}, s')(\beta) \rangle} \; \textit{StUpdBC}$$

where $\mathbf{bcast}(\mathsf{r}, s')(\beta) = \beta \cup \{(\mathsf{r}', s') \mid \mathsf{r}' \in \mathsf{RID} \setminus \{\mathsf{r}\}\}$. *Then if* $\mathcal{O}$ *is an op-based host CRDT, and* $\mathcal{O}_{\mathcal{G}}$ *is a state-based guest CRDT given by the emulation* $\mathcal{G}$ *in Section 4, it follows that*

$$\mathbf{init} \textit{ and } \mathsf{init}_{\mathcal{G}} \textit{ are weakly } \text{bisimilar}, \textit{ i.e., } \mathbf{init} \approx \mathsf{init}_{\mathcal{G}}.$$

There is, of course, an analogous result for when a state-based CRDT is the host and an op-based CRDT is the guest. What Theorem 4.11 implies is that CRDT emulation is somehow "sensitive" to the implementation of communication protocols, and so being able to preserve and reflect properties P on weak traces is just about the best one can do with weak assumptions on the communication protocol.[6]

In fact, we can further upgrade $\approx$ in Theorem 4.11 to $\sim$ (*strong* bisimulation) if we allow the op-based CRDT to deliver as many messages as possible in a single atomic action. In other words, the more "alike" the communication protocols are in two given networks, the more "bisimilar" op-based and state-based CRDTs are.

5 A Representation-Independent Client Interface to CRDTs

In Section 4, we view CRDTs as "open systems", in the sense that the client using a CRDT is an abstract entity living outside the CRDT semantics, interfacing with them by the labels, e.g., $\mathsf{upd}[\mathsf{op}]$, $\mathsf{qry}[q]$. In this section, we "close" our system model by designing a small imperative programming language of client programs, which maintains a store that is modified by interactions with an underlying CRDT. We use our results in Section 4 to show how we can achieve a "contextual approximation"-style result: one can swap out an underlying host CRDT implementation for its corresponding guest CRDT implementation, without seriously affecting the results of the program. We formalized our client language and proved our key theorem correct using Agda (see supplementary material).

Assumptions 5.1. For the purposes of this section, we fix the following data:

(1) A countably infinite set Var of program variables.
(2) A set Expr of arithmetic expressions.
(3) An evaluation function $\mathcal{E}[\![-]\!] : \mathsf{Expr} \to \mathbb{N}^{\mathsf{Var}} \to \mathbb{N}$
(4) The replica IDs RID, along with their operations Oper, and internal state space S.
(5) A query function $q : S \to \mathbb{N}$.

The set of *client programs* Prog is generated by the following grammar for $x \in \mathsf{Var}$, $e \in \mathsf{Expr}$, $\mathsf{o} \in \mathsf{Oper}$:

$$\mathsf{Prog} \ni p, q ::= \mathsf{skip} \mid \mathsf{asn}[x](e) \mid \mathsf{while}(e, p) \mid p \fatsemi q \mid \mathsf{upd}[o] \mid \mathsf{qry}[x] \tag{5.1}$$

[6] Any weaker assumptions on the underlying network are likely to break the assumptions needed by CRDTs, or are somehow not meaningful in practice. For example, if the network graph used by a state-based CRDT is not complete, we don't even have *weak simulation*. But one can always simulate a complete network graph with a multi-hop communication protocol independent of the CRDT implementation!

$$\frac{C \overset{\tau}{\rightsquigarrow}_\kappa C'}{C \triangleright \langle \mu, p\rangle \longrightarrow_\kappa C' \triangleright \langle \mu, p\rangle}\ [\text{CStep}] \qquad \frac{r \in \mathsf{RID} \quad C \overset{\mathsf{r:upd[o]/o}}{\rightsquigarrow}_\kappa C'}{C \triangleright \langle \mu, \mathsf{upd[o]}\rangle \longrightarrow_\kappa C' \triangleright \langle \mu, \mathsf{skip}\rangle}\ [\text{Upd}]$$

$$\frac{r \in \mathsf{RID} \quad C \overset{\mathsf{r:qry[q]/res[v]}}{\rightsquigarrow}_\kappa C \quad \mu' = \mu[x \mapsto n]}{C \triangleright \langle \mu, \mathsf{qry}[x](q)\rangle \longrightarrow_\kappa C \triangleright \langle \mu', \mathsf{skip}\rangle}\ [\text{Qry}]$$

$$\frac{C \triangleright \langle \mu, p\rangle \longrightarrow_\kappa C' \triangleright \langle \mu', p'\rangle}{C \triangleright \langle \mu, p \fatsemi q\rangle \longrightarrow_\kappa C' \triangleright \langle \mu', p' \fatsemi q\rangle}\ [\text{Comp}_1] \qquad \frac{C \triangleright \langle \mu, p\rangle \Downarrow_\kappa}{C \triangleright \langle \mu, p \fatsemi q\rangle \longrightarrow_\kappa C \triangleright \langle \mu, q\rangle}\ [\text{Comp}_2]$$

$$\frac{\mathcal{E}[\![e]\!](\mu) = 0}{C \triangleright \langle \mu, \mathsf{while}(e, p)\rangle \Downarrow_\kappa}\ [\text{WDone}] \qquad \frac{\mathcal{E}[\![e]\!](\mu) \neq 0}{C \triangleright \langle \mu, \mathsf{while}(e, p)\rangle \longrightarrow_\kappa C \triangleright \langle \mu, p \fatsemi \mathsf{while}(e, p)\rangle}\ [\text{WStep}]$$

$$\frac{}{C \triangleright \langle \mu, \mathsf{skip}\rangle \Downarrow_\kappa}\ [\text{Skip}] \qquad \frac{\mathcal{E}[\![e]\!](\mu) = n \quad \mu' = \mu[x \mapsto n]}{C \triangleright \langle \mu, \mathsf{asn}[x](e)\rangle \longrightarrow_\kappa C \triangleright \langle \mu', \mathsf{skip}\rangle}\ [\text{Asn}]$$

Fig. 9. Operational semantics of client programs.

We think of $p \in \mathsf{Prog}$ as a client program running on top of a given CRDT system. In p, the choice of replica that serves client requests (upd and qry) is made by external factors outside the client's control (e.g., latency, availability, phases of the moon, etc.). In that light, the client is served by a replica chosen in a non-deterministic fashion.

Figure 9 gives the operational semantics of client programs. To summarize, each rule in Figure 9 is parameterized by a configuration C of the underlying CRDT system (e.g., $C = \langle \Gamma \mid \Sigma \mid \beta \rangle$, as in Sections 3 and 4), and a variable store $\mu \in \mathbb{N}^{\mathsf{Var}}$. The configuration C represents the *execution environment* of the program state $\langle \mu, p\rangle$. We write $C \overset{\alpha}{\rightsquigarrow}_\kappa C'$ to denote a transition of configurations by action α, under CRDT system κ. The transitions the CRDT system κ may take are those defined in Section 3. On the other hand, client programs have two types of transitions, namely

$$C \triangleright \langle \mu, p\rangle \longrightarrow_\kappa C' \triangleright \langle \mu', p'\rangle \qquad \text{and} \qquad C \triangleright \langle \mu, p\rangle \Downarrow_\kappa .$$

The first arrow is read as *on environment C (and under system κ), client program state $\langle \mu, p\rangle$ progresses to $\langle \mu', p'\rangle$ producing a new environment C'*; the second arrow is read as *on environment C (and under system κ), client program state $\langle \mu, p\rangle$ terminates*. On each computation step, the program produces a new execution environment, reflecting changes in the internal state of some replica r or in the local store μ of the program state.

We also define the arrow $\Downarrow^*_\kappa$ so that

$$C \triangleright \langle \mu, p\rangle \Downarrow^*_\kappa \iff \exists C', \mu', p' \,:\, (C \triangleright \langle \mu, p\rangle \longrightarrow^*_\kappa C' \triangleright \langle \mu', p'\rangle) \wedge (C' \triangleright \langle \mu', p'\rangle \Downarrow_\kappa).$$

We now define a notion of what it means for two (potentially different) CRDT systems to "approximate" each other from the point of view of client programs.

Definition 5.2 (CRDT Approximation)**.** Let κ and λ be two CRDT systems with resp. configurations C_κ and C_λ. Let $\langle \mu_1, p_1\rangle$ and $\langle \mu_2, p_2\rangle$ be program states. We say $C_\lambda \triangleright \langle \mu_1, p_1\rangle$ *approximates* $C_\kappa \triangleright \langle \mu_2, p_2\rangle$ if we have

$$C_\kappa \triangleright \langle \mu_1, p_1\rangle \Downarrow^*_\kappa \implies C_\lambda \triangleright \langle \mu_2, p_2\rangle \Downarrow^*_\lambda,$$

and in that case we write $C_\kappa \triangleright \langle \mu_1, p_1\rangle \sqsubseteq C_\lambda \triangleright \langle \mu_2, p_2\rangle$.

Definition 5.2 is essentially the coarsest form of comparison one would expect from two program states executed in different CRDT environments. Indeed, it is coarser than at least weak simulation.

Theorem 5.3 (Soundness). *Let κ and λ be two arbitrary CRDT systems, and let C_κ, C_λ be configurations of these respective systems. Then, for all program states $\langle \mu, p \rangle$,*

$$C_\kappa \lessapprox C_\lambda \implies C_\kappa \triangleright \langle \mu, p \rangle \sqsubseteq C_\lambda \triangleright \langle \mu, p \rangle.$$

This key theorem tells us that if a configuration of a CRDT system κ weakly simulates a configuration of a CRDT system λ, then a given client program running in the execution environment of the former is an approximation of that client program running in the execution environment of the latter.

We formalized the client program semantics shown in Figure 9 and proved Theorem 5.3 correct in Agda. Since our definition of emulation (Definition 4.1) means that there exist a pair of emulations $\mathcal{G}_1$, $\mathcal{G}_2$ from resp. op-based to state-based and state-based to op-based CRDTs (Section 4 and Appendix B), we know that CRDT emulation is sound with respect to Definition 5.2. We summarize as a corollary to Theorem 5.3.

Corollary 5.4 (CRDT emulation is sound). *Let O_1 and O_2 resp. be an op-based CRDT and state-based CRDT with resp. transition semantics $\rightsquigarrow_1$, $\rightsquigarrow_2$ and resp. initial configurations $\mathbf{init}_1$ and init_2.*

Then there are a pair of emulations $\mathcal{G}_1$ and $\mathcal{G}_2$, and a pair of resp. state-based CRDT O_1' and op-based CRDT O_2' with resp. transition semantics $\rightsquigarrow_{\mathcal{G}_1}$, $\rightsquigarrow_{\mathcal{G}_2}$, and resp. initial configurations $\mathsf{init}_{\mathcal{G}_1}$, $\mathbf{init}_{\mathcal{G}_2}$, such that:

(1) $O_1' = \mathcal{G}_1(O_1)$, *and* $O_2' = \mathcal{G}_2(O_2)$;

(2) $\mathbf{init}_1 \lessapprox \mathsf{init}_{\mathcal{G}_1}$ *and* $\mathsf{init}_2 \lessapprox \mathbf{init}_{\mathcal{G}_2}$,

Therefore,

(1) *For all program states $\langle \mu, p \rangle$, we have $\mathbf{init}_1 \triangleright \langle \mu, p \rangle \sqsubseteq \mathsf{init}_{\mathcal{G}_1} \triangleright \langle \mu, p \rangle$ and $\mathsf{init}_{\mathcal{G}_1} \triangleright \langle \mu, p \rangle \sqsubseteq \mathbf{init}_1 \triangleright \langle \mu, p \rangle$;*

(2) *For all program states $\langle \mu, p \rangle$, we have $\mathsf{init}_2 \triangleright \langle \mu, p \rangle \sqsubseteq \mathbf{init}_{\mathcal{G}_2} \triangleright \langle \mu, p \rangle$ and $\mathbf{init}_{\mathcal{G}_2} \triangleright \langle \mu, p \rangle \sqsubseteq \mathsf{init}_2 \triangleright \langle \mu, p \rangle$.*

Corollary 5.4 concretely shows how our transition semantics of CRDTs in Section 3 interfaces with the language we developed in this section. In words, it says that for all client programs p and all variable stores μ, one can execute the program state $\langle \mu, p \rangle$ in an environment with either an op-based CRDT, or a state-based CRDT, and so long as those CRDTs are related by an emulation $\mathcal{G}$ (and we begin at the initial CRDT configurations), termination behavior of $\langle \mu, p \rangle$ in one execution environment implies termination behavior in the other, and vice versa.

While cotermination is a fairly coarse form of equivalence, it is in some ways, stronger than one would expect. For example, notice that in the [Qry] rule, interacting with the CRDT environment can change the variable store μ. Since programs p are capable of infinite loops in our language, what Corollary 5.4 says is that, if p can terminate with O, then it still can with the $\mathcal{G}(O)$ and vice-versa, even if p contains an arbitrarily large number of interactions with the CRDT via queries, upon which termination might depend. Of course, there is an element of non-determinism here: executing $\langle \mu, p \rangle$ in one CRDT environment and observing some behavior b does not *guarantee* that we will get b in a different CRDT environment, only that b is *possible*, due to the inherent non-determinism of distributed systems.

6 Related Work

As we discussed in Section 1, most existing research on CRDT verification has focused on verifying strong convergence and other safety[7] properties for either state-based [Zeller et al. 2014; Gadducci et al. 2018; Nair et al. 2020; Timany et al. 2024; Nieto et al. 2023; Laddad et al. 2022]

[7]Notably, [Timany et al. 2024] also consider verification of *liveness* properties, such as eventual delivery of messages.

or op-based [Gomes et al. 2017; Nagar and Jagannathan 2019; Liu et al. 2020; Liang and Feng 2021; Nieto et al. 2022] CRDTs. One exception is the work of Burckhardt et al. [2014], who give a framework for axiomatic specification and verification of both op-based and state-based CRDTs, inspired by previous work on axiomatization of weak memory models.

To our knowledge, our work is the first to make precise the sense in which state-based and op-based CRDTs emulate each other. However, the use of (bi)simulation relations in CRDT verification is not new. For instance, Burckhardt et al. [2014]'s framework is based on *replication-aware simulations*, and Nair et al. [2020] use a strong bisimulation argument to justify the use of simpler, easier-to-implement proof rules in an automated verification tool for state-based CRDTs, using the more complicated semantics as a reference implementation.

Our work takes particular inspiration from Nieto et al. [2023], who observe that whether a CRDT is op-based or state-based is an implementation detail that should be hidden from clients. They show that for a specific CRDT, the *pn-counter*, a particular client program cannot distinguish between handwritten op-based and state-based implementations of the CRDT. By "handwritten" we mean that, unlike in our work, Shapiro et al. [2011]'s emulation recipes are not in use in their work. Nevertheless, their work inspired our representation-independence result. Rather than considering a specific pair of CRDT implementations and specific client program, though, we make precise the exact sense in which *general* op-to-state-based and state-to-op-based emulation algorithms result in the same observable behavior for the original and the emulating object. An interesting observation we have made with respect to their work is that their model of op-based and state-based CRDTs are based on broadcasting network semantics. Our work here (in particular Theorem 4.11) shows that when one assumes broadcast in both types of CRDTs, a weak bisimulation *is* possible, and then one truly has an equivalence of behaviors. In that sense, their results agree with the theory we have laid out here.

Mechanized verification of CRDTs, both interactive [Zeller et al. 2014; Gomes et al. 2017; Timany et al. 2024; Nieto et al. 2022, 2023] and automated [Nair et al. 2020; Nagar and Jagannathan 2019; De Porre et al. 2023; Laddad et al. 2022], is an active area of research with many exciting developments. In our work, we do not aim to provide user-ready verified implementations; our goal instead is to put existing efforts on a firm theoretical foundation, by making precise the sense in which results for op-based CRDTs can be said to transfer to state-based CRDTs and vice versa.

7 Conclusion and Future Work

In this paper, we have made precise the sense in which op-based and state-based CRDT systems emulate each others' behavior, using formal simulation techniques. We have characterized which properties are preserved by CRDT emulation: properties on weak traces. We have shown that CRDT emulation is, in some sense, sensitive to assumptions made about the underlying network semantics. Our results are *sound* in the sense that, if two CRDT systems have starting configurations that are related by a pair of simulations, then a client that interacts with a CRDT or its emulation counterpart through a programming language cannot distinguish between the original system and the emulator. We formalized and proved the soundness result correct in Agda. Therefore we close a long-standing gap in the CRDT literature. Our results give researchers working with CRDTs a rigorous way to think about equivalence of state-based and op-based CRDTs, both abstractly and theoretically in terms of simulation relations, and more concretely, in terms of programming languages that interact with a given CRDT. For future work, it might be interesting to formalize the results of Section 4 in a proof assistant. We expect this would be non-trivial, as research on coinductive techniques in dependently typed languages is ongoing. It also might be interesting to generalize our work to replicated data structures beyond CRDTs.

References

Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. 2015. Efficient State-Based CRDTs by Delta-Mutation. In *Networked Systems*, Ahmed Bouajjani and Hugues Fauconnier (Eds.). Springer International Publishing, Cham, 62–76.

Kenneth Birman, André Schiper, and Pat Stephenson. 1991. Lightweight Causal and Atomic Group Multicast. *ACM Trans. Comput. Syst.* 9, 3 (Aug. 1991), 272–314. https://doi.org/10.1145/128738.128742

Kenneth P. Birman and Thomas A. Joseph. 1987. Reliable Communication in the Presence of Failures. *ACM Trans. Comput. Syst.* 5, 1 (Jan. 1987), 47–76. https://doi.org/10.1145/7351.7478

Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. 2014. Replicated Data Types: Specification, Verification, Optimality. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '14)*. Association for Computing Machinery, New York, NY, USA, 271–284. https://doi.org/10.1145/2535838.2535848

Kevin De Porre, Carla Ferreira, and Elisa Gonzalez Boix. 2023. VeriFx: Correct Replicated Data Types for the Masses. In *37th European Conference on Object-Oriented Programming (ECOOP 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 263)*, Karim Ali and Guido Salvaneschi (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 9:1–9:45. https://doi.org/10.4230/LIPIcs.ECOOP.2023.9

C. J. Fidge. 1988. Timestamps in message-passing systems that preserve the partial ordering. *Proceedings of the 11th Australian Computer Science Conference* 10, 1 (1988), 56–66. http://sky.scitech.qut.edu.au/~fidgec/Publications/fidge88a.pdf

Fabio Gadducci, Hernán Melgratti, and Christian Roldán. 2018. On the semantics and implementation of replicated data types. *Science of Computer Programming* 167 (2018), 91–113. https://doi.org/10.1016/j.scico.2018.06.003

Seth Gilbert and Nancy Lynch. 2002. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *SIGACT News* 33, 2 (jun 2002), 51–59. https://doi.org/10.1145/564585.564601

Seth Gilbert and Nancy Lynch. 2012. Perspectives on the CAP Theorem. *Computer* 45, 2 (2012), 30–36. https://doi.org/10.1109/MC.2011.389

Victor B. F. Gomes, Martin Kleppmann, Dominic P. Mulligan, and Alastair R. Beresford. 2017. Verifying Strong Eventual Consistency in Distributed Systems. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 109 (oct 2017), 28 pages. https://doi.org/10.1145/3133933

Maurice P. Herlihy and Jeannette M. Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (jul 1990), 463–492. https://doi.org/10.1145/78969.78972

Shadaj Laddad, Conor Power, Mae Milano, Alvin Cheung, and Joseph M. Hellerstein. 2022. Katara: Synthesizing CRDTs with Verified Lifting. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 173 (oct 2022), 29 pages. https://doi.org/10.1145/3563336

Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (July 1978), 558–565. https://doi.org/10.1145/359545.359563

Leslie Lamport. 1994. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.* 16, 3 (May 1994), 872–923. https://doi.org/10.1145/177492.177726

Hongjin Liang and Xinyu Feng. 2021. Abstraction for Conflict-Free Replicated Data Types. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 636–650. https://doi.org/10.1145/3453483.3454067

Yiyun Liu, James Parker, Patrick Redmond, Lindsey Kuper, Michael Hicks, and Niki Vazou. 2020. Verifying Replicated Data Types with Typeclass Refinements in Liquid Haskell. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 216 (nov 2020), 30 pages. https://doi.org/10.1145/3428284

Nancy A Lynch and Mark R Tuttle. 1988. *An introduction to input/output automata.*

Friedemann Mattern. 2002. Virtual Time and Global States of Distributed Systems. https://api.semanticscholar.org/CorpusID:7517210

R. Milner. 1982. *A Calculus of Communicating Systems.* Springer-Verlag, Berlin, Heidelberg.

Kartik Nagar and Suresh Jagannathan. 2019. Automated Parameterized Verification of CRDTs. In *Computer Aided Verification*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 459–477.

Sreeja S. Nair, Gustavo Petri, and Marc Shapiro. 2020. Proving the Safety of Highly-Available Distributed Objects. In *Programming Languages and Systems*, Peter Müller (Ed.). Springer International Publishing, Cham, 544–571.

Abel Nieto, Arnaud Daby-Seesaram, Léon Gondelman, Amin Timany, and Lars Birkedal. 2023. Modular Verification of State-Based CRDTs in Separation Logic. In *37th European Conference on Object-Oriented Programming (ECOOP 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 263)*, Karim Ali and Guido Salvaneschi (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 22:1–22:27. https://doi.org/10.4230/LIPIcs.ECOOP.2023.22

Abel Nieto, Léon Gondelman, Alban Reynaud, Amin Timany, and Lars Birkedal. 2022. Modular Verification of Op-Based CRDTs in Separation Logic. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 188 (oct 2022), 29 pages. https://doi.org/10.1145/3563351

Nuno Preguiça, Carlos Baquero, and Marc Shapiro. 2018. *Conflict-Free Replicated Data Types CRDTs.* Springer International Publishing, Cham, 1–10. https://doi.org/10.1007/978-3-319-63962-8_185-1

Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, and Joonwon Lee. 2011. Replicated abstract data types: Building blocks for collaborative applications. *J. Parallel and Distrib. Comput.* 71, 3 (2011), 354–368. https://doi.org/10.1016/j.jpdc.2010.12.006

Fred B. Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.* 22, 4 (dec 1990), 299–319. https://doi.org/10.1145/98163.98167

Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems*, Xavier Défago, Franck Petit, and Vincent Villain (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 386–400.

Amin Timany, Simon Oddershede Gregersen, Léo Stefanesco, Jonas Kastberg Hinrichsen, Léon Gondelman, Abel Nieto, and Lars Birkedal. 2024. Trillium: Higher-Order Concurrent and Distributed Separation Logic for Intensional Refinement. *Proc. ACM Program. Lang.* 8, POPL, Article 9 (jan 2024), 32 pages. https://doi.org/10.1145/3632851

James R Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D Ernst, and Thomas Anderson. 2015. Verdi: A framework for formally verifying distributed system implementations. In *Proceedings of the 2015 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Portland, OR.*

Peter Zeller, Annette Bieniusa, and Arnd Poetzsch-Heffter. 2014. Formal Specification and Verification of CRDTs. In *Formal Techniques for Distributed Objects, Components, and Systems*, Erika Ábrahám and Catuscia Palamidessi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 33–48.

A Construction of Partial Order

To make "concurrent messages" make sense in our semantics, we need to define a "happens-before" style relation (as discussed in Section 2.1.1) that works in our semantics. In Section 3.2, we defined configurations $\langle \Gamma \mid \Sigma \mid \beta \rangle$ where Γ is an event trace of the current execution. We will define a causality relation $\prec$ with respect to Γ, then show how we can use enabled(Γ) as esentially a well-formedness condition to ensure that causality is respected.

Notation A.1. Let Γ be an event trace. Write $\Gamma|_{\mathsf{r}}$ to be the order-preserving restriction of Γ to events concerning replica r. [8] Then write $\Gamma \vdash e_{\mathsf{r},i} \triangleright o$ if (r, e, o) is the i tuple in $\Gamma|_{\mathsf{r}}$.

Notation A.1 gives us a convenient way to discuss the order of events which have happened on a single replica.

Definition A.2 (Causality Relation)**.** Let Γ be an event trace. We inductively define the *causality relation* $\Gamma \vdash (-) \prec (-)$ on pairs $\Gamma \vdash e_{i,\mathsf{r}} \triangleright o$, and $\Gamma \vdash e'_{i',\mathsf{r}'} \triangleright o'$ as follows.

- If $\Gamma \vdash e_{i,\mathsf{r}} \triangleright o$ and $\Gamma \vdash e'_{i',\mathsf{r}} \triangleright o'$, and $i < i'$, write $\Gamma \vdash (e_{i,\mathsf{r}} \triangleright o) \prec (e'_{i',\mathsf{r}} \triangleright o')$;
- If $o = \mathsf{send}[m]$ and $(\mathsf{r}', m) \in m$, and $e_{i',\mathsf{r}'} = \mathsf{dlvr}[m]_{i',\mathsf{r}'}$, then
$$\Gamma \vdash (e_{i,\mathsf{r}} \triangleright \mathsf{send}[m]) \prec (\mathsf{dlvr}[m]_{i',\mathsf{r}'} \triangleright o');$$
- If both $\Gamma \vdash (e_{i,\mathsf{r}} \triangleright o) \prec (e'_{i',\mathsf{r}} \triangleright o')$, and $\Gamma \vdash (e'_{i',\mathsf{r}'} \triangleright o') \prec (e''_{i'',\mathsf{r}''} \triangleright o'')$, then
$$\Gamma \vdash (e_{i,\mathsf{r}} \triangleright o) \prec (e''_{i'',\mathsf{r}''} \triangleright o'')$$

Note that $\Gamma : (-) \prec (-)$ is transitive, and irreflexive, and if the configuration containing Γ takes a transition step, obtainint trace Γ', we can readily extend $\Gamma : (-) \prec (-)$ to $\Gamma' : (-) \prec (-)$ by just adding any newly related pairs.

Notation A.3. Say m *is a potential cause for* m' and write
$$m \prec_\Gamma m' \iff (\Gamma \vdash (e_{i,\mathsf{r}} \triangleright \mathsf{send}[m]) \prec (e'_{i',\mathsf{r}'} \triangleright \mathsf{send}[m'])).$$
We also say m and m' are *concurrent* and write $m \parallel_\Gamma m' \iff \neg(m \prec_\Gamma m') \wedge \neg(m' \prec_\Gamma m)$.

In general, we just work with $\prec_\Gamma$ on messages. The next result is important for later, when we need to do recursion on sets of messages (within the context of an execution).

Proposition A.4 (Well-founded)**.** *Assuming uniqueness of messages,* $\prec_\Gamma$ *is a well-founded strict partial order.*

We can now encode the constraint that the effects of concurrent messages commute in the following assumption.

Assumption A.5. Let $\langle \Gamma \mid \Sigma \mid \beta \rangle$ a configuration and r a replica. If $m \parallel_\Gamma m'$, and both
$$\Sigma(\mathsf{r}) \xrightarrow{\mathsf{dlvr}[m]/\bot}_{\mathsf{r}} s'_1 \xrightarrow{\mathsf{dlvr}[m']/\bot}_{\mathsf{r}} s''_1 \text{ and } \Sigma(\mathsf{r}) \xrightarrow{\mathsf{dlvr}[m']/\bot}_{\mathsf{r}} s'_2 \xrightarrow{\mathsf{dlvr}[m]/\bot}_{\mathsf{r}} s''_2$$
then $s''_1 = s''_2$.

In order to make sure that our semantics enforce an ordering of message delivery that is consistent with causality (i.e., consistent with $\prec_\Gamma$), we define the following safety condition.

Definition A.6 (Causal message delivery)**.** Let M be a set and $m, m' \in M$ a pair of messages. We say Γ satisfies *causal delivery order* if, $\forall m, m' \in M$, and $\forall \mathsf{r} \in \mathtt{RID}$,
$$(m \prec_\Gamma m') \wedge ((\mathsf{r}, \mathsf{dlvr}[m], \bot)_i \in \Gamma) \wedge ((\mathsf{r}, \mathsf{dlvr}[m'], \bot)_j \in \Gamma) \tag{A.1}$$

[8] $\Gamma|_{\mathsf{r}}$ can be defined functionally using a `filter` function. It is easy to check by induction that $\Gamma|_{\mathsf{r}}$ defined this way really does preserve the ordering of Γ.

$$\implies \neg(\mathsf{dlvr}[m'] \textit{ ordered before } \mathsf{dlvr}[m] \textit{ in } \Gamma|_{\mathsf{r}} \textit{ i.e., } i < j). \tag{A.2}$$

On the other hand, if $m \parallel_\Gamma m'$, then there is no constraint on the order of $\mathsf{dlvr}[m]$, and $\mathsf{dlvr}[m']$.

Proof of Proposition 3.15. An invariant style argument. Initially causal delivery order holds on ε in `init`. So, suppose it holds on some reachable $\langle \Gamma' \mid \Sigma' \mid \beta' \rangle$ and $\langle \Gamma' \mid \Sigma' \mid \beta' \rangle \overset{\sigma}{\rightsquigarrow} \langle \Gamma \mid \Sigma \mid \beta \rangle$. Doing structural induction on the transition, the only case of interest is the OpDeliver case. In this case, we have $\Gamma = \Gamma' \cdot (\mathsf{r}, \mathsf{dlvr}[m], \bot)$, so it suffices check that any $m' \prec_\Gamma m$ where $m' \neq m$ was already delivered. Suppose not, then we can find a m' satisfying $m' \prec_\Gamma m$ and $(\mathsf{r}, \mathsf{dlvr}[m'], \bot) \notin \Gamma'$. But by rule inversion on OpDeliver, we have $(\mathsf{r}, \mathsf{dlvr}[m]) \in \mathsf{enabled}(\Gamma')$, which says that any $m'' \prec_{\Gamma'} m$ which had r as a target was already delivered, i.e., $(\mathsf{r}, \mathsf{dlvr}[m''], \bot) \in \Gamma'$. Setting $m'' = m'$ yields the desired contradiction. □

B State-based to Op-based Emulation

Here we sketch how to do state-to-op CRDT emulation in our model. We omit some details, because they are quite analagous to the situation for op-to-state CRDT emulation. There are, thankfully, no complicated constructions involved, no recursion on partial orders. The reason for this is some elements of doing the simulation proofs for state-to-op emulation are trivialized by the fact that we can consider the sending of replica state as a *message* in the op-based guest system.

Assumption B.1. Assume we are given a state-based CRDT object Figure 2, with its corresponding sets S, Oper, Q and V, related functions, and semantics Figure 6. That means $S = (S, \sqcup)$ is a join-semilattice with some initial state s^0, and `update` is inflationary wrt $\sqcup$: $\forall \mathsf{o} \in \mathrm{Oper}$, we have

$$s \sqcup \mathsf{update}(\mathsf{r}, s, \mathsf{o}) = \mathsf{update}(\mathsf{r}, s, \mathsf{o}).$$

For brevity, we just write $\sqcup$ for `merge`.

The candidate emulation we consider is a mapping $\mathcal{G}$ which takes our hosting state-based CRDT object O and constructs an op-based guest CRDT object $O' = \mathcal{G}(O)$ as specified in Figure 10.

Parameters :

S : states, Oper : operations, S : messages, Q : queries, V : values,

s^0 : initial state

Functions :

$\mathsf{prep}(\mathsf{r}, s, \mathsf{o}) = \mathsf{update}(\mathsf{r}, s, \mathsf{o})$

$\mathsf{effect}(s, m) = s \sqcup m$ (since m is a message, $m \in S$)

$\mathsf{query}(q, s) = \mathsf{query}(q, s)$.

Fig. 10. The state-based guest CRDT constructed by $\mathcal{G}$ from a given op-based host CRDT.

In op-based CRDTs, update is essentially a composition of `prep` and `effect`. Note that this still makes sense wrt Figure 10 since by the fact that $(S, \sqcup)$ is a join-semilattice one can show,

$$\mathsf{update}(\mathsf{r}, s, \mathsf{o}) \overset{\text{def}}{=} \mathsf{effect}(s, \mathsf{prep}(\mathsf{r}, s, \mathsf{o})) = s \sqcup \mathsf{update}(\mathsf{r}, s, \mathsf{o}) = \mathsf{update}(\mathsf{r}, s, \mathsf{o}).$$

In other words, the op-based guest CRDT is really just a delegation of `update`, `effect`, and `query` to resp. `update`, `merge`(= $\sqcup$), and `query`. This follows the recipe given in Shapiro et al. [2011] almost identically.

However, we are not quite done — state-based CRDTs and op-based CRDTs have different semantics in our model, and we still need to show a simulation. Thankfully, since $\sqcup$ is associative and commutative by assumption, we have for all $s \in S$, and messages $m, m' \in S$,

$$s \cdot m \cdot m' = (s \sqcup m) \sqcup m' = s \sqcup (m \sqcup m') = s \sqcup (m' \sqcup m) = s \cdot m' \cdot m.$$

So we may simply drop the assumption that messages are delivered in any particular order (Equation (2.1)). We now assume O and $O' = \mathcal{G}(O)$ have transition semantics as in Section 3, and identical starting configurations $\langle \varepsilon \mid \lambda \mathsf{r} \,.\, s^0 \mid \varnothing \rangle$ and $\langle \varepsilon \mid \boldsymbol{\lambda \mathsf{r} \,.\, s^0} \mid \boldsymbol{\varnothing} \rangle$, respectively, and sketch the two weak simulation proofs.

B.1 The Guest CRDT simulates the Host CRDT

Note that the roles of op-based and state-based CRDTs are reversed now, so the candidate weak simulation $\mathcal{Q}_1$ we are about to introduce is actually analagous to $\mathcal{R}_2$ in Figure 8. Along those lines, the main thing we deal with is asserting that a "larger" merge can be simulated by a series of "smaller" merges.

Definition B.2. In a set $\beta \in \mathcal{P}(\mathtt{RID} \times S)$, we say a set

$$C = \{s_1, \ldots, s_k\} \subseteq S$$

is a *mergeable set at replica* r and write $C \in \mathsf{Mergeable}_\beta(\mathsf{r})$ if for all $j \in 1..k$, we have $(\mathsf{r}, s_j) \in \beta$.

We take as our candiate simulation $\mathcal{Q}_1$ the relation, which clearly includes the initial configurations $\langle \varepsilon \mid \lambda \mathsf{r} \,.\, s^0 \mid \varnothing \rangle$ and $\langle \varepsilon \mid \boldsymbol{\lambda \mathsf{r} \,.\, s^0} \mid \boldsymbol{\varnothing} \rangle$.

$$\begin{aligned}
\mathcal{Q}_1 = \{(&\langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma \mid \Sigma \mid \boldsymbol{\beta} \rangle) \mid \\
&\text{— } \textit{configurations need to be reachable} \\
&\quad \langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma \mid \Sigma \mid \boldsymbol{\beta} \rangle \text{ are well-formed} \\
&\text{— Agreement on global states} \qquad \wedge \quad \Sigma = \Sigma \\
&\text{— } \textit{the op-based replica can simulate a "large" merge by the state-based replica} \\
&\wedge \quad \forall (\mathsf{r}, s') \in \beta, \exists C \in \mathsf{Mergeable}_\beta(\mathsf{r}) \,:\, \Sigma(\mathsf{r}) \sqcup s' = \bigsqcup (C \cup \{\Sigma(\mathsf{r})\})\}
\end{aligned}$$

Theorem B.3. *$\mathcal{Q}_1$ is a weak simulation of $\rightsquigarrow$ by $\rightsquigarrow$, which contains the initial configurations.*

Proof of Theorem B.3. Assume as our hypothesis: for some given α, we have

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\alpha}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle.$$

By case analysis on the transition, we have the following cases.

(1) (StUpdate). Then $\alpha = \mathsf{r} : \mathtt{upd}[\mathsf{o}]/\bot$, and we have

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}:\mathtt{upd}[\mathsf{o}]/\bot}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$$

$$\Gamma' = \Gamma \cdot (\mathsf{r}, \mathtt{upd}[\mathsf{o}], \bot), \qquad \Sigma' = \Sigma[\mathsf{r} \mapsto \mathtt{update}(\mathsf{r}, \Sigma(\mathsf{r}), \mathsf{o})], \qquad \beta' = \beta.$$

Let $s' = \mathtt{update}(\mathsf{r}, \Sigma(\mathsf{r}), \mathsf{o})$. We invoke the (OpUpdate) rule and match with

$$\langle \Gamma \mid \Sigma \mid \boldsymbol{\beta} \rangle \xrightarrow{\mathsf{r}:\mathtt{upd}[\mathsf{o}]/\bot} \langle \Gamma' \mid \Sigma' \mid \boldsymbol{\beta'} \rangle$$

$$\Gamma' = \Gamma \cdot \boldsymbol{(\mathsf{r}, \mathtt{upd}[\mathsf{o}], \mathtt{send}[s'])}, \qquad \Sigma' = \Sigma[\mathsf{r} \to s'], \qquad \boldsymbol{\beta' = \mathtt{bcast}(\mathsf{r}, s')(\beta)}.$$

It is immediate that $\Sigma' = \Sigma'$. The second condition is also immediate, as $\boldsymbol{\beta} \subseteq \boldsymbol{\beta'}$. Thus, $\mathcal{Q}_1(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \boldsymbol{\beta'} \rangle)$. ◀

(2) (StSend). Then, $\alpha = \tau$, and there is a pair of replicas $\mathsf{r}, \mathsf{r}^\star$ with $\mathsf{r} \neq \mathsf{r}^\star$, and a state $s^\star = \Sigma(\mathsf{r}^\star)$ so that $\mathsf{r}^\star$ sent its state $s^\star$ to replica r, whence

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$$
$$\Gamma' = \Gamma \cdot (\mathsf{r}^\star, \bot, \mathsf{send}[s^\star]), \qquad \Sigma' = \Sigma, \qquad \beta' = \beta \cup \{(\mathsf{r}, s^\star)\}.$$

We simulate with the reflexive step, i.e.,

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow}{}^* \langle \Gamma \mid \Sigma \mid \beta \rangle.$$

We show that $Q_1(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle)$. For this, it suffices to just check that the most recently sent $(\mathsf{r}, s^\star)$ has a $C \in \mathsf{Mergeable}_\beta(\mathsf{r})$, i.e., the op-based r can simulate the state-based r merging $s^\star$. If $\Sigma(\mathsf{r}) \sqcup s^\star = \Sigma(\mathsf{r})$, take the empty set $C = \varnothing$. If instead $\Sigma(\mathsf{r}) \sqcup s^\star = s^\star$, we use the fact that configurations are well-formed to pick a mergeable set C.

Since $\Sigma = \Sigma$ by hypothesis, and configurations are well-formed, replica $\mathsf{r}^\star$ must have executed a sequence $f_1, f_2, \ldots, f_N$ of updates and delivers which advanced its state from s^0 to

$$s^\star = \Sigma(\mathsf{r}^\star) = \Sigma(\mathsf{r}^\star).$$

We construct a sequence of states $s_1, s_2, \ldots, s_N$ so that

$$\bigsqcup(\{s^0\} \cup \{s_1, \ldots, s_N\}) = s^\star = \Sigma(\mathsf{r}^\star).$$

For each $j \in 1..N$,

- if f_i is an update, then $\mathsf{r}^\star$ updated its state to (say) t, then broadcast t to each other replica. Set $s_i = t$.
- if f_i was a deliver, then replica $\mathsf{r}^\star$ merged some *other* state t' (broadcasted by some replica) into its current state. Set $s_i = t'$.

We obtain the sequence $s_1, s_2, \ldots, s_N$ from the above algorithm by induction. Critically, each s_i in the sequence $s_1, s_2, \ldots, s_N$ was, at some point during the execution, broadcasted to r. Therefore, there is a subsequence $s_{N_1}, \ldots, s_{N_k}$ so that $\{s_{N_1}, \ldots, s_{N_k}\} \in \mathsf{Mergeable}_\beta(\mathsf{r})$. Since $\Sigma(\mathsf{r}) = \Sigma(\mathsf{r}) \leq s^\star$ in the partial order (induced by $\sqcup$) we have that,

$$\Sigma(\mathsf{r}) \sqcup s^\star = s^\star = \bigsqcup(\{s^0\} \cup \{s_1, \ldots, s_N\}) = \bigsqcup(\{\Sigma(\mathsf{r})\} \cup \{s_{N_1}, \ldots, s_{N_k}\}) \geq \Sigma(\mathsf{r}).$$

We thus take as our set $C = \{s_{N_1}, \ldots, s_{N_k}\}$. It follows that $Q_1(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle)$. ◀

(3) (StDeliver). Then, $\alpha = \tau$, and a replica r delivered some state s from β. We have,

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$$
$$\Gamma' = \Gamma \cdot (\mathsf{r}, \mathsf{dlvr}[s], \bot), \qquad \Sigma' = \Sigma[\mathsf{r} \mapsto \Sigma(\mathsf{r}) \sqcup s], \qquad \beta' = \beta \setminus \{(\mathsf{r}, s)\}.$$

From the hypothesis, there is a set $\{s_1, \ldots, s_k\} = C \in \mathsf{Mergeable}_\beta(\mathsf{r})$. We match by delivering this set in any order at the op-based replica r. The details are an easier version of the (StDeliver) case in the proof of Theorem 4.7, so we omit them. ◀

(4) (StQuery). Since $\Sigma = \Sigma$, and queries don't change the configuration, this case is immediate. ◀

□

B.2 The Host CRDT simulates the Guest CRDT

Because the underlying replica states are both elements in S, simulating broadcast with the state-based host CRDT means that at all times, the global states and network states are identical at every step of the simulation. This makes this direction rather straightforward.

We take as our candiate simulation Q_2 the relation clearly includes the initial configurations $\langle \varepsilon \mid \lambda \mathsf{r} \,.\, s^0 \mid \varnothing \rangle$ and $\langle \varepsilon \mid \lambda \mathsf{r} \,.\, s^0 \mid \varnothing \rangle$. And our statement of the theorem.

$$\begin{aligned}\mathcal{Q}_2 = \{(&\langle \Gamma \mid \Sigma \mid \beta\rangle, \langle \Gamma \mid \Sigma \mid \beta\rangle) \mid \\ &\text{— } \textit{agreement on global states} \\ &\quad \langle \Gamma \mid \Sigma \mid \beta\rangle, \langle \Gamma \mid \Sigma \mid \beta\rangle \text{ are well-formed} \\ &\text{— } \textit{agreement in global states, and network states} \\ &\wedge \quad \Sigma = \Sigma \wedge \beta = \beta\}.\end{aligned}$$

Theorem B.4. *$\mathcal{Q}_2$ is a weak simulation of $\rightsquigarrow$ by $\rightsquigarrow$, which contains the initial configurations.*

Since op-based and state-based CRDTs have swapped roles, the simulation $\mathcal{Q}_2$ here is analagous to $\mathcal{R}_1$, and so the mechanical details of the simulation proof follow that of Theorem 4.5 almost identically, though a bit easier since now the network buffers are identical. We thus omit the proof.

C Proofs

C.1 Construction of (4.1)

Recall the definition of $\texttt{interp}_S$:

$$\texttt{interp}_S(H) \stackrel{\text{def}}{=} \begin{cases} s^0 & \text{if } H = \varnothing \\ \texttt{effect}(\texttt{interp}_S(H \setminus \{m\}), m) & \text{if } m \in \mathsf{Max}(H). \end{cases} \tag{C.1}$$

We show this function is well-defined. We drop our colors here for the moment.

First, note that since we only consider well-formed configurations, sets H are finite. Thus, the set

$$\mathsf{Max}(H) = \{m \in H \mid \nexists m' \,.\, m \prec m'\}$$

is also finite.

Lemma C.1. *For each pair $m, m' \in \mathsf{Max}(H)$, we have $m \parallel m'$.*

Proof. It is a well known fact that in a partial order $(A, \leq)$, the set of maximal elements is an anti-chain, i.e., consists of only the incomparable elements wrt $\leq$. □

Let $N = |\mathsf{Max}(H)|$. The above lemma then implies that any linearization $\ell = m_1 m_2 \cdots m_k$ of $\mathsf{Max}(H)$ is a valid representative for $\mathsf{Max}(H)$. That is, we have an equivalence class $[\ell]$ with respect to $\texttt{effect}$. Observe:

Let $\ell = m_1 m_2 \cdots m_k$ be any linearization of the set $\mathsf{Max}(H)$, and let t be any adjacent transposition (of say, $j, j+1 \leq k$). We have for all $s \in S$,

$$\begin{aligned}\texttt{effect}_\ell(s) &= s \cdot m_1 \cdots m_j m_{j+1} \cdots m_k \\ &= s \cdot m_1 \cdots m_{j+1} m_j \cdots m_k \quad (\text{since } \texttt{effect}_{mm'}(s) = \texttt{effect}_{m'm}(s) \text{ whenever } m \parallel m') \\ &= s \cdot m_{t(1)} \cdots m_{t(j)} m_{t(j+1)} \cdots m_{t(k)}.\end{aligned}$$

Since any linearization ℓ can be expressed as a permutation σ and permutations can be built by a sequence of adjacent transpositions, by an induction argument, the choice of ℓ was immaterial. We can thus represent $\mathsf{Max}(H)$ as an equivalence class of linearizations $[\ell]$ by simply ordering the elements of $\mathsf{Max}(H)$ in some way.

Let $A_1 = \mathsf{Max}(H)$, and define the recursion

$$A_{i+1} \stackrel{\text{def}}{=} \mathsf{Max}(H \setminus (\cup_{j \leq i} A_j)).$$

Since H was finite, this recursion terminates in (say) N steps, giving sets $A_1, \ldots, A_N$.

It now follows that $\texttt{interp}_S$ is well-defined — by definition, the recursion must first process all the elements $m_1, \ldots m_{k_N} \in A_N$ by choosing them in some way. The choices made here induce a linearization ℓ_N. But since $\texttt{interp}_S$ is defined in terms of `effect`, it does not depend on this linearization. It then proceeds "down" to the set A_{N-1} and orders the elements into a linearization ℓ_{N-1} and so on. At the end, we have,

$$\texttt{interp}_S(H) = s^0 \cdot \ell_N \cdot \ell_{N-1} \cdots \ell_1.$$

Each pair m, m' in each ℓ_i may be swapped, but the ℓ_i's themselves cannot be rearranged (if they could, we contradict the definition of $\texttt{interp}_S$). It follows that $\texttt{interp}_S$ does not depend on the ordering for each ℓ_i, so we could have just as easily written

$$\texttt{interp}_S(H) = s^0 \cdot [\ell_N] \cdot [\ell_{N-1}] \cdots [\ell_1] = s^0 \cdot A_N \cdot A_{N-1} \cdots A_1,$$

and the meaning of this is unambiguous.

Proof of Theorem 4.5.

Hypothesis: $(\langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle) \in \mathcal{R}_1$ *and we have,* $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\alpha}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$, *for some* α.

By case analysis on the transition:

(1) (OpUpdate). $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}_i : \mathbf{upd[op]}/\bot}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$, where

$$\Gamma' = \Gamma \cdot (\mathsf{r}_i, \mathbf{upd[op]}, \mathbf{send[}m\mathbf{]}), \quad \Sigma' = \Sigma[\mathsf{r}_i \to s'], \quad \beta' = \mathbf{bcast}(\mathsf{r}_i, m)(\beta).$$

We match with

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}_i : \mathbf{upd[op]}/\bot}{\rightsquigarrow}_{\mathcal{G}} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle \overset{\tau}{\rightsquigarrow}^*_{\mathcal{G}} \langle \Gamma'' \mid \Sigma'' \mid \beta'' \rangle,$$

where the silent transitions $\langle \Gamma' \mid \Sigma' \mid \beta' \rangle \overset{\tau}{\rightsquigarrow}^*_{\mathcal{G}} \langle \Gamma'' \mid \Sigma'' \mid \beta'' \rangle$ are chosen to simulate **bcast** by $n-1$ applications of the StSend , choosing replica r_i to send its local state $\Sigma'(\mathsf{r}_i) = \Sigma(\mathsf{r}_i) \cup \{\mathbf{prep}(\texttt{interp}_S(\Sigma(\mathsf{r}_i)), \text{op})\}$ to each replica $\mathsf{r}_j (\neq \mathsf{r}_i)$. Buffers and event traces are updated accordingly.

We need to show that $(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma'' \mid \Sigma'' \mid \beta'' \rangle) \in \mathcal{R}_1$ by checking the conditions in **??**. It suffices to consider just the r_i which performed the update. The only two conditions of interest are:

(a) $\Sigma'(\mathsf{r}_i) = \Sigma''(\mathsf{r}_i)$.

(b) $(\forall \mathsf{r} \in \mathtt{RID}, (\mathsf{r}, m) \in \beta' \implies (\mathsf{r}, (m)^{\downarrow \Gamma''}) \in \beta'')$.

Since StSend does not change the global state (only the event trace, and network state), it suffices to check $\Sigma'(\mathsf{r}_i) = \Sigma'(\mathsf{r})$.

We know $\Gamma' = \Gamma \cdot (\mathsf{r}_i, \mathbf{upd[op]}, \mathbf{send[}m\mathbf{]})$, so by OpUpdate , we know $m = \mathbf{prep}(s, \text{op})$ was the most recently sent message in the op-based system. Using the hypothesis, we let

$$s = \Sigma(\mathsf{r}_i) = \texttt{interp}_S(\Sigma(\mathsf{r}_i)),$$

and therefore obtain by StUpdate that the corresponding message sent in the state based system is

$$\Sigma'(\mathsf{r}_i) = \Sigma(\mathsf{r}_i) \cup \{\mathbf{prep}(\texttt{interp}_S(\Sigma(\mathsf{r}_i)), \text{op})\} = \Sigma(\mathsf{r}_i) \cup \{\mathbf{prep}(s, \text{op})\} = \Sigma(\mathsf{r}_i) \cup \{m\}.$$

Note that m is (by definition), the most recently sent message by replica r_i, hence,

$$\Sigma'(\mathsf{r}_i) = \Sigma(\mathsf{r}) \cup \{m\} = \text{Delivered}(\mathsf{r}_i, \Gamma') = (m)^{\downarrow \Gamma'}.$$

and since $\Sigma(\mathsf{r}_i) \cup \{m\}$ was sent to all other r_j during the last set of $\overset{\tau}{\rightsquigarrow}^*$ transitions, we have $(\mathsf{r}_j, (m)^{\downarrow \Gamma'}) \in \beta''$, which proves the (b) condition.

For the (a) condition, we now just need to check that the following equality holds:

$$\Sigma'(\mathsf{r}_i) = \mathbf{effect}(s, m) = \texttt{interp}_S(\Sigma(\mathsf{r}_i) \cup \{m\}) = \texttt{interp}_S(\Sigma'(\mathsf{r}_i)).$$

Indeed, since messages are unique (by assumption), and m was the last sent message in the op-based system, there is no $m' \in \mathsf{Sent}(\Gamma') = \mathsf{Sent}(\Gamma'')$ such that $m \prec m'$, and therefore,

$$\mathtt{interp}_S(\Sigma(\mathsf{r}_i) \cup \{m\}) = \mathtt{effect}(\mathtt{interp}_S(\Sigma(\mathsf{r}_i)), m),$$

which implies the desired equality. ◀

(2) (OpQuery). We match $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}_i : \mathtt{qry}[q]/\mathtt{res}(v)}{\rightsquigarrow} \langle \Gamma \mid \Sigma \mid \beta \rangle$ with

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}_i : \mathtt{qry}[q]/\mathtt{res}(v')\langle \Gamma \mid \Sigma \mid \beta \rangle}{\rightsquigarrow} \langle \Gamma \mid \Sigma \mid \beta \rangle,$$

where the hypothesis combined with StQuery immediately implies

$$v = \mathtt{query}(q, \Sigma(\mathsf{r}_i)) = \mathtt{query}(q, \mathtt{interp}_S(\Sigma(\mathsf{r}_i))) = \mathtt{query}(q, \Sigma(\mathsf{r}_i)) = v',$$

and we are done. ◀

(3) (OpDeliver). Then $\alpha = \tau$, and we have $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$, where

$$\Gamma' = \Gamma \cdot (\mathsf{r}, \mathtt{dlvr}[m], \bot), \quad \Sigma' = \Sigma[\mathsf{r} \to \mathtt{effect}(\Sigma(\mathsf{r}), m)], \quad \beta' = \beta \setminus \{(\mathsf{r}, m)\},$$

which means r delivered an enabled message $(\mathsf{r}, m) \in \beta$. The hypothesis implies $\exists (\mathsf{r}, (m)^{\downarrow \Gamma}) \in \beta$. Therefore we can invoke StDeliver and match with

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle,$$
$$\Gamma' = \Gamma \cdot (\mathsf{r}, \mathtt{dlvr}[(m)^{\downarrow \Gamma}], \bot), \quad \Sigma' = \Sigma[\mathsf{r}_i \mapsto \Sigma(\mathsf{r}) \cup (m)^{\downarrow \Gamma}], \quad \beta' = \beta \setminus \{(\mathsf{r}, (m)^{\downarrow \Gamma})\}.$$

To finish, we need to show $(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle) \in \mathcal{R}_1$ by checking the conditions in **??**. Since no messages were sent, the last condition is immediate from the hypothesis. We focus on the changes which happened at replica r (r). Note that since m was enabled at r, all causally preceding messages (i.e., $m' \prec m$) had to have already been delivered to at replica r. But since $\mathsf{Delivered}(\mathsf{r}, \Gamma) = \Sigma(\mathsf{r})$ (by hypothesis), all such $m' \in \Sigma(\mathsf{r})$ as well. Thus,

$$\Sigma'(\mathsf{r}) = \Sigma(\mathsf{r}) \cup (m)^{\downarrow \Gamma} = \Sigma(\mathsf{r}) \cup \{m\} = \mathsf{Delivered}(\mathsf{r}, \Gamma').$$

From the above equality, each condition in **??** is immediate. Indeed, since either $m' \parallel m$ or $m' \prec m$ for each $m' \in \Sigma(\mathsf{r})$, we have the key equality

$$\mathtt{interp}_S(\Sigma'(\mathsf{r})) = \mathtt{interp}_S(\Sigma(\mathsf{r}) \cup \{m\}) = \mathtt{effect}(\mathtt{interp}_S(\Sigma(\mathsf{r})), m) = \mathtt{effect}(\Sigma(\mathsf{r}), m) = \Sigma'(\mathsf{r}).$$

$(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle) \in \mathcal{R}_1$ as desired. ◀

□

Proof of Theorem 4.7.

Hypothesis: $\mathcal{R}_2(\langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle)$ and we have $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\alpha}{\rightsquigarrow}_{\mathcal{G}} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$, some α.

- (StDeliver). Then $\alpha = \tau$ and we have $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow}_{\mathcal{G}} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$ where

  $$\Gamma' = \Gamma \cdot (\mathsf{r}, \mathtt{dlvr}[H], \bot), \quad \Sigma' = \Sigma[\mathsf{r} \mapsto \mathtt{merge}(\Sigma(\mathsf{r}), H)], \quad \beta' = \beta \setminus \{(\mathsf{r}, H)\},$$

  which means a replica r merged a state $(\mathsf{r}, H) \in \beta$ that was sent by some other replica. If $\Sigma(\mathsf{r}) \cup H = \Sigma(\mathsf{r})$, we match by the reflexive step $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow}^* \langle \Gamma \mid \Sigma \mid \beta \rangle$, and there is nothing to show, so we suppose there is a number $k > 0$, and messages $m_1, \ldots, m_k \notin \Sigma(\mathsf{r})$, so that $\Sigma(\mathsf{r}) \cup H = \Sigma(\mathsf{r}) \cup \{m_1, \ldots, m_k\}$.

(Claim). We can pick as $U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$ the set $\{m_1, \ldots, m_k\}$, therefore deliver the sequence of messages $m_1 \cdots m_k$ at replica r so that

$$\Sigma(\mathsf{r}) \xrightarrow{\mathsf{r}:\mathbf{dlvr}[m_1]/\bot} s_1 \cdots \xrightarrow{\mathsf{r}:\mathbf{dlvr}[m_k]/\bot} s_k$$

and thereby match with k applications of (OpDeliver) wrt replica r giving:

$$\begin{aligned} &\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow}{}^* \langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \\ &\Gamma' = \Gamma \cdot \langle (\mathsf{r}, \mathbf{dlvr}[m_1], \bot), \ldots, (\mathsf{r}, \mathbf{dlvr}[m_k], \bot) \rangle, \\ &\Sigma' = \Sigma[\mathsf{r} \to s_k], \quad \beta' = \beta \backslash \{(\mathsf{r}, m_1), \ldots, (\mathsf{r}, m_k)\}. \end{aligned} \tag{C.2}$$

(Proof of Claim). From the hypothesis, $\exists U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$ such that $H = (U)^{\downarrow \Gamma}$. Suppose U has $k' \geq 0$ messages so that $\{m'_1, \ldots, m'_{k'}\} = U$. Since $U \subseteq (U)^{\downarrow \Gamma} = H$, and (by the hypothesis) $\mathsf{Delivered}(\mathsf{r}, \Gamma) = \Sigma(\mathsf{r})$, the "already merged" portion of messages at r is $H \setminus U$, hence we obtain:

$$\Sigma(\mathsf{r}) \cup H = \Sigma(\mathsf{r}) \cup U = \mathsf{Delivered}(\mathsf{r}, \Gamma) \cup U \tag{C.3}$$

$$U = \{m'_1, \ldots, m'_{k'}\} = \{m_1, \ldots, m_k\}. \tag{C.4}$$

Since by hypothesis, $U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$, it is clear we can deliver the sequence $m_1 \cdots m_k$ at replica r (reindexing if needed), (C.2) is immediate. □

It remains to show that $\mathcal{R}_2(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle)$. The only interesting condition is showing that replicas agree in their local states. For that, suffices to prove $s_k = \mathsf{interp}_S(\mathsf{merge}(\Sigma(\mathsf{r}), H))$

But this has to be the case: (C.3) implies

$$\mathsf{Delivered}(\mathsf{r}, \Gamma') = \mathsf{Delivered}(\mathsf{r}, \Gamma) \cup U = \Sigma(\mathsf{r}) \cup U = \Sigma'(\mathsf{r})$$

and since $\{m_1, \ldots, m_k\} = U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$ means each pair $m_i, m_j \in U$ $(i < j)$ satisfies either $m_i \prec m_j$ or $m_i \parallel m_j$, we obtain

$$s_k = \mathbf{effect}_{m_1 \cdots m_k}(\Sigma(\mathsf{r})) = \mathsf{interp}_S(\Sigma(\mathsf{r}) \cup U) = \mathsf{interp}_S(\mathsf{merge}(\Sigma(\mathsf{r}), H)).$$

The other conditions are immediate, and we obtain $\mathcal{R}_2(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle)$ as needed. ◀

- (StUpdate). Then $\alpha = \mathsf{r} : \mathsf{upd}[\mathsf{o}]/\bot$. Let $H = \Sigma(\mathsf{r})$. Then we have

$$\begin{aligned} &\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\mathsf{r}:\mathsf{upd}[\mathsf{o}]/\bot}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle \\ &\Gamma' = \Gamma \cdot (\mathsf{r}, \mathsf{upd}[op], \bot), \quad \Sigma' = \Sigma[\mathsf{r} \mapsto H'], \quad \beta' = \beta, \end{aligned}$$

where $H' \cup \{\mathbf{prep}(\mathsf{r}, \mathsf{interp}_S(H), \mathsf{o})\}$.

Let $s' = \mathbf{update}(\mathsf{r}, \Sigma(\mathsf{r}), \mathsf{o})$. We match by invoking the (OpUpdate) rule at replica r, obtaining

$$\begin{aligned} &\langle \Gamma \mid \Sigma \mid \beta \rangle \xrightarrow{\mathsf{r}:\mathbf{upd}[\mathsf{o}]/\bot} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle \\ &\Gamma' = \Gamma \cdot (\mathsf{r}, \mathbf{upd}[\mathsf{o}], \mathbf{send}[m]), \qquad \Sigma' = \Sigma[\mathsf{r} \to s'], \qquad \beta' = \mathbf{bcast}(\mathsf{r}, m)(\beta). \end{aligned}$$

We show $\mathcal{R}_2(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle)$.

Since $(\Sigma(\mathsf{r})) = \mathsf{interp}_S(H) = \mathsf{interp}_S((\Sigma(\mathsf{r})))$, it is immediate that

$$\mathbf{prep}(\mathsf{r}, \mathsf{interp}_S(H), \mathsf{o}) = m = \mathbf{prep}(\mathsf{r}, \Sigma(\mathsf{r}), \mathsf{o})$$

and therefore $\mathsf{Delivered}(\mathsf{r}, \Gamma') = \Sigma(\mathsf{r}) \cup \{m\} = \Sigma'(\mathsf{r})$. Moreover, by construction, m the set of causal predecessors

$$(m)^{\downarrow \Gamma'} \setminus \{m\} = \mathsf{Delivered}(\mathsf{r}, \Gamma)$$

which means $\forall m' \in \Sigma(\mathsf{r})$, we have $m' \prec m$, and therefore

$$\Sigma'(\mathsf{r}) = \mathbf{effect}(\Sigma(\mathsf{r}), m) = \mathbf{effect}(\mathsf{interp}_S(\Sigma(\mathsf{r})), m) = \mathsf{interp}_S(\Sigma'(\mathsf{r})).$$

The other conditions are immediate, hence we obtain $\mathcal{R}_2(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle)$.

- (StSend) Then, $\alpha = \tau$, and there is a pair of replicas $\mathsf{r}, \mathsf{r}^\star$ with $\mathsf{r} \neq \mathsf{r}^\star$, and a state $H^\star = \Sigma(\mathsf{r}^\star)$ so that $\mathsf{r}^\star$ sent its state $H^\star$ to replica r, whence

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$$
$$\Gamma' = \Gamma \cdot (\mathsf{r}^\star, \bot, \mathsf{send}[H^\star]), \qquad \Sigma' = \Sigma, \qquad \beta' = \beta \cup \{(\mathsf{r}, H^\star)\}.$$

We simulate with the reflexive step, i.e.,

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow}{}^* \langle \Gamma \mid \Sigma \mid \beta \rangle.$$

We show that $\mathcal{R}_2(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle)$.
For this, it suffices to just check that the most recently sent $(\mathsf{r}, H^\star)$ has a $U \in \mathsf{deliverable}_\beta(\mathsf{r})$, i.e., the op-based r can simulate the state-based r merging $H^\star$. There are two sub-cases.
(1) If $\Sigma(\mathsf{r}) \cup H^\star = \Sigma(\mathsf{r})$, then we take the empty set $U = \varnothing$.
(2) If instead $\Sigma(\mathsf{r}) \sqcup H^\star = H^\star$, we use our assumption that configurations are well-formed to construct a deliverable set U.
Let $N = |\mathsf{Delivered}(\mathsf{r}^\star, \Gamma)|$, and rewrite

$$\{m_1, m_2, \ldots, m_N\} = \mathsf{Delivered}(\mathsf{r}^\star, \Gamma).$$

Since configurations are well-formed, it is a fact then, replica $\mathsf{r}^\star$ executed N transitions $f_1, f_2, \ldots, f_N$ consisting of either updates, or deliver events, where we can associate each $m_i \in \{m_1, m_2, \ldots, m_N\}$ with the transition f_i. As follows, for each $i \in 1..N$,

- if f_i was a deliver event, then some $\bullet \xrightarrow{\mathsf{dlvr}[m']} \bullet$ transition occured. We set $m_i = m'$.
- If f_i was an update event, then some $\bullet \xrightarrow{\mathsf{upd}[o']/\mathsf{send}[m']} \bullet$ transition occured, then $\mathsf{r}^\star$ updated its state by executing $\mathtt{effect}(\mathsf{r}^\star, s, m')$ for some predecessor state s. We set $m_i = m'$

We thus obtain the exact sequence of messages $m_1, \ldots, m_N$ which advanced $\mathsf{r}^\star$ to its current state $\Sigma(\mathsf{r}^\star)$ from s^0. In particular, m_N was the message which transitioned $\mathsf{r}^\star$ to have state $\Sigma(\mathsf{r}^\star)$. Critically, in each case, each message m_i was *broadcast* (by well-formedness), hence made available to replica r at some point during the execution. That means, there exists a subsequence $m_{N_1}, \ldots, m_{N_k}$ of undelivered messages, and by construction, $i < j$ implies $\neg(m_{N_j} \prec m_{N_i})$. Additionally, there exists a minimal set U' of messages so that the set $U = \{m_{N_1}, \ldots, m_{N_j}\} \cup U'$ satisfies

$$H^\star = \Sigma(\mathsf{r}) \cup U \wedge U \in \mathsf{deliverable}_\Gamma(\mathsf{r}) \tag{C.5}$$

and this has to be the case since

$$\mathsf{Delivered}(\mathsf{r}^\star, \Gamma) = \Sigma(\mathsf{r}^\star) = H^\star = \Sigma(\mathsf{r}) \cup H^\star \geq \Sigma(\mathsf{r}),$$

combined with the hypothesis that

$$\bigcup \mathsf{Sent}(\Gamma) = \mathsf{Sent}(\Gamma).$$

This U is either empty, or the set we need closing the subcase.
In either subcase, $\mathcal{R}_2(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle)$ follows.

◀

- (StQuery). Immediate from the hypothesis.

◀

□

Proof of Theorem 4.11. We will give a candidate relation, and prove it is a weak bisimulation using the transition semantics of Section 4, but modified under the premise of Theorem 4.11.

To that end, let $\mathbf{EvtTr}$ denote the type of event traces Γ. Define $f : \mathbf{EvtTr} \to M \to \mathcal{P}(M)$ as the function which takes a message m, then constructs the downset wrt Γ, i.e.,

$$f_\Gamma(m) = (m)^{\downarrow\Gamma}.$$

We assume only well-formed configurations here. Let us say say (Γ, Σ) *agrees* with (Γ, Σ) iff

$$(\mathsf{Sent}(\Gamma) = \bigcup \mathsf{Sent}(\Gamma)) \wedge \forall \mathsf{r} \in \mathtt{RID} : (\mathsf{Delivered}(\mathsf{r}, \Gamma) = \Sigma(\mathsf{r})) \wedge (\Sigma(\mathsf{r}) = \mathtt{interp}_S(\Sigma(\mathsf{r}))),$$

and in that case, we write $(\Gamma, \Sigma) \bowtie_{\text{agree}} (\Gamma, \Sigma)$.

Let $\mathcal{X}$ be the candidate relation,

$$\mathcal{X} = \{(\langle \Gamma \mid \Sigma \mid \beta\rangle, \langle \Gamma \mid \Sigma \mid \beta\rangle) \mid (\Gamma, \Sigma) \bowtie_{\text{agree}} (\Gamma, \Sigma) \wedge \mathsf{map}(f_\Gamma)(\beta) = \beta\}.$$

We claim $\mathcal{X}$ is a weak bisimulation and write the proof. To witness, assume we are given $\mathcal{X}(\langle \Gamma \mid \Sigma \mid \beta\rangle, \langle \Gamma \mid \Sigma \mid \beta\rangle)$. We show the that for all α,

- $\langle \Gamma \mid \Sigma \mid \beta\rangle \overset{\alpha}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta'\rangle \implies \exists \langle \Gamma' \mid \Sigma' \mid \beta'\rangle : \langle \Gamma \mid \Sigma \mid \beta\rangle \overset{\alpha}{\rightsquigarrow} \overset{\tau}{\rightsquigarrow}{}^* \langle \Gamma' \mid \Sigma' \mid \beta'\rangle$
- $\langle \Gamma \mid \Sigma \mid \beta\rangle \overset{\alpha}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta'\rangle \implies \exists \langle \Gamma' \mid \Sigma' \mid \beta'\rangle : \langle \Gamma \mid \Sigma \mid \beta\rangle \overset{\alpha}{\rightsquigarrow} \overset{\tau}{\rightsquigarrow}{}^* \langle \Gamma' \mid \Sigma' \mid \beta'\rangle$.

By case analysis on α, we have the following cases.

(1) $(\alpha = \mathsf{r}_i : \mathsf{upd}[\mathsf{o}]/\bot)$. We have the subcases,

(a) $\langle \Gamma \mid \Sigma \mid \beta\rangle \overset{\mathsf{r}_i : \mathsf{upd}[\mathsf{o}]/\bot}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta'\rangle$.

Let $s = \Sigma(\mathsf{r}_i)$, and let $s' = \mathtt{update}(\mathsf{r}_i, s, \mathsf{o})$, $m = \mathtt{prep}(\mathsf{r}_i, s, \mathsf{o})$. Then,

$$\Gamma' = \Gamma \cdot (\mathsf{r}_i, \mathbf{upd[op]}, \mathbf{send}[m]), \quad \Sigma' = \Sigma[\mathsf{r}_i \to s'], \quad \beta' = \mathtt{bcast}(\mathsf{r}_i, m)(\beta).$$

Let $H' = \Sigma(\mathsf{r}_i) \cup \{\mathtt{prep}(\mathsf{r}_i, \mathtt{interp}_S(\Sigma(\mathsf{r}_i)), \mathsf{o})\}$. We match with

$$\langle \Gamma \mid \Sigma \mid \beta\rangle \overset{\mathsf{r} : \mathsf{upd}[\mathsf{o}]/\bot}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta'\rangle$$

$$\Gamma' = \Gamma' \cdot (\mathsf{r}_i, \mathsf{upd}[\mathsf{o}], \mathsf{send}[H']), \qquad \Sigma' = \Sigma[\mathsf{r}_i \mapsto H'], \qquad \beta' = \mathtt{bcast}(\mathsf{r}_i, H')(\beta).$$

From the hypothesis $\mathcal{X}(\langle \Gamma \mid \Sigma \mid \beta\rangle, \langle \Gamma \mid \Sigma \mid \beta\rangle)$, we know that $s = \mathtt{interp}_S(\Sigma(\mathsf{r}_i))$, and therefore $\mathtt{prep}(\mathsf{r}_i, s, \mathsf{o}) = m = \mathtt{prep}(\mathsf{r}_i, \mathtt{interp}_S(\Sigma(\mathsf{r}_i)), \mathsf{o})$. Following the details of the (OpUpdate) case in the proof of Theorem 4.5, one can show that $(\Gamma', \Sigma') \bowtie_{\text{agree}} (\Gamma', \Sigma')$, so we just need to show that

$$\mathsf{map}(f_{\Gamma'})(\beta') = \beta'.$$

Since initially $\mathsf{map}(f_\Gamma)(\beta) = \beta$, it suffices to consider just the most recently broadcasted H' and message m and prove that $f_{\Gamma'}(m) = H'$.

To that end, note that $(\Gamma', \Sigma') \bowtie_{\text{agree}} (\Gamma', \Sigma')$ implies in particular $\mathsf{Delivered}(\mathsf{r}_i, \Gamma') = \Sigma'(\mathsf{r}_i)$. Since $\forall m' \in \mathsf{Sent}(\Gamma')$ with $m' \prec m$ implies (by definition) $m' \in \mathsf{Delivered}(\mathsf{r}_i, \Gamma')$, it follows that

$$(m)^{\downarrow\Gamma'} = \mathsf{Delivered}(\mathsf{r}_i, \Gamma') = \Sigma'(\mathsf{r}_i) = H',$$

and therefore $f_{\Gamma'}(m) = (m)^{\downarrow\Gamma'} = H'$ as claimed. Thus, $\mathcal{X}(\langle \Gamma' \mid \Sigma' \mid \beta'\rangle, \langle \Gamma' \mid \Sigma' \mid \beta'\rangle)$.

(b) $\langle \Gamma \mid \Sigma \mid \beta\rangle \overset{\mathsf{r}_i : \mathsf{upd}[\mathsf{o}]/\bot}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta'\rangle$. In this case, match with

$$\langle \Gamma \mid \Sigma \mid \beta\rangle \overset{\mathsf{r}_i : \mathsf{upd}[\mathsf{o}]/\bot}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta'\rangle$$

given in the above sub-case and proceed with the same argument. ◀

(2) $(\alpha = \mathsf{r} : \mathsf{qry}[q]/\mathsf{res}[v])$. Directly follows from the hypothesis $\mathcal{X}(\langle \Gamma \mid \Sigma \mid \beta\rangle, \langle \Gamma \mid \Sigma \mid \beta\rangle)$. ◀

(3) ($\alpha = \tau$). Some replica r (r) must have delivered a message m (H). We handle both transition sub-cases at once by arguing that $\mathcal{X}$ essentially implies $\mathcal{R}_1$ and $\mathcal{R}_2$. More precisely, if $\mathcal{X}(\langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle)$ for well-formed $\langle \Gamma \mid \Sigma \mid \beta \rangle$, $\langle \Gamma \mid \Sigma \mid \beta \rangle$, we have

$$\bullet\ \forall (\mathsf{r}, m) \in \beta \implies (\mathsf{r}, (m)^{\downarrow \Gamma}) \in \beta, \tag{C.6}$$

$$\bullet\ \forall (\mathsf{r}, H) \in \beta\ :\ \exists U \in \mathsf{deliverable}_\Gamma(\mathsf{r})\ :\ H = (U)^{\downarrow \Gamma}, \tag{C.7}$$

which are the last conditions in resp. $\mathcal{R}_1$ and $\mathcal{R}_2$ (cf. Figure 8). Note that by definition of f, (C.6) is immediate.

For (C.7), let $(\mathsf{r}, H) \in \beta$. We need to show there is a set $U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$ so that $H = (U)^{\downarrow \Gamma}$. By construction, $H = (m)^{\downarrow \Gamma}$ for some m such that $(\mathsf{r}, m) \in \beta$. Since the configurations are well formed, we know that each $m' \in (m)^{\downarrow \Gamma}$ has a corresponding send event $\mathsf{send}[m']$ in Γ, and due to broadcast, either $m' \in \mathsf{Delivered}(\mathsf{r}', \Gamma)$ or $(\mathsf{r}', m') \in \beta$, for all r'. This means there is a (possibly empty) sequence of message deliveries which make m' deliverable at replica r. Since this can be done for each $m' \in (m)^{\downarrow \Gamma}$ we obtain a larger sequence of message deliveries which enable replica r to deliver all of $(m)^{\downarrow \Gamma}$. But this is precisely a deliverable set, and by construction, it cannot be larger than $(m)^{\downarrow \Gamma}$. We take as U the set given this way, hence we obtain (C.7), QED.

What we have just argued is that if $\mathcal{X}(\langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle)$, then $\langle \Gamma \mid \Sigma \mid \beta \rangle$ and $\langle \Gamma \mid \Sigma \mid \beta \rangle$ can mutually simulate each other's deliver transitions. More precisely,

(a) if $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$, by (say) $\Sigma(\mathsf{r}) \xrightarrow{\mathsf{dlvr}[m]} s'$, then the hypothesis $\mathcal{X}(\langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle)$ implies we can execute the local transition

$$\Sigma(\mathsf{r}) \xrightarrow{\mathsf{dlvr}[(m)^{\downarrow \Gamma}]} \Sigma(\mathsf{r}) \cup (m)^{\downarrow \Gamma},$$

which simulates the $\mathbf{dlvr}[m]$ event, and thereby obtain the matching transition

$$\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle.$$

Following the details of the (OpDeliver) case in the proof of Theorem 4.5, we can obtain

$$(\Gamma', \Sigma') \bowtie_{\text{agree}} (\Gamma', \Sigma').$$

Moreover, since $\beta' = \beta \setminus \{(\mathsf{r}, m)\}$ and $\beta' = \beta \setminus \{(\mathsf{r}, (m)^{\downarrow \Gamma})\}$, the equality

$$\mathsf{map}(f_{\Gamma'})(\beta') = \beta'$$

is immediate, and therefore $\mathcal{X}(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle)$.

(b) if $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$, by (say) $\Sigma(\mathsf{r}) \xrightarrow{\mathsf{dlvr}[H]} \Sigma(\mathsf{r}) \cup H$, we know from the hypothesis $\mathcal{X}(\langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle)$ that $H = (m)^{\downarrow \Gamma}$, for some $(\mathsf{r}, m) \in \beta$.
Since $\mathcal{X}(\langle \Gamma \mid \Sigma \mid \beta \rangle, \langle \Gamma \mid \Sigma \mid \beta \rangle)$ implies there is a deliverable set $U \in \mathsf{deliverable}_\Gamma(\mathsf{r})$ which simulates the $\mathsf{dlvr}[H]$ event, we follow the details of the (StDeliver) case in the proof of Theorem 4.7, and obtain a sequence of matching transitions

$$\Sigma(\mathsf{r}) \xrightarrow{\mathsf{r}:\mathsf{dlvr}[m_1]/\bot} s_1 \cdots \xrightarrow{\mathsf{r}:\mathsf{dlvr}[m_k]/\bot} s_k$$

that imply $\langle \Gamma \mid \Sigma \mid \beta \rangle \overset{\tau}{\rightsquigarrow}{}^{*} \langle \Gamma' \mid \Sigma' \mid \beta' \rangle$ with $(\Gamma', \Sigma') \bowtie_{\text{agree}} (\Gamma', \Sigma')$. The equality

$$\mathsf{map}(f_{\Gamma'})(\beta') = \beta'$$

ends up being immediate, and hence $\mathcal{X}(\langle \Gamma' \mid \Sigma' \mid \beta' \rangle, \langle \Gamma' \mid \Sigma' \mid \beta' \rangle)$. ◀

All together, it follows that $\mathcal{X}$ is a weak bisimulation. □