Causal-Broadcast Memory*

Amir Karniel[0009-0007-4563-2385] and Ori Lahav[0000-0003-4305-6998]

Tel Aviv University, Tel Aviv, Israel

Abstract. We investigate the precise consistency guarantees provided by a simple and prominent distributed implementation of shared memory (a.k.a. key-value store) based on the causal broadcast abstraction. We formalize these guarantees within a weak memory model, which we call "causal-broadcast memory" (CBM, for short), and relate it to several established weak memory models. In particular, our study reveals that CBM is strictly stronger than "causal memory" consistency, previously proposed to encapsulate the same distributed implementation. Additionally, we address two core verification challenges for CBM: (*i*) deciding whether a single abstract execution graph is consistent, which we show is solvable in polynomial time, and (*ii*) verifying reachability of control states for client programs operating atop causal-broadcast memory, which we prove to be undecidable.

1 Introduction

Consider the following simple reference implementation DistMem of a shared memory in a message-passing distributed system (code shown for process p).

```
1 method write(x,v)          method read(x)      when Wxv is received
2   M_p:=M_p[x ↦ v]            return(M_p(x))    from another process:
3   causal_broadcast(Wxv)                          M_p:=M_p[x ↦ v]
4   return()
```

In words, each process p maintains a local copy of a location-to-value mapping M_p. When a write of value v to location x is requested at a process p, the mapping is updated locally and then the request is sent over the network to all other processes. To read from location x, each process simply uses its local copy of the mapping. Upon receiving a write request from another process, the update is executed on the local copy.

The broadcast operation is non-blocking: it returns without waiting for the reception of the emitted messages. It is also assumed to maintain *causal consistency*, which means that (1) if some message m_1 is a cause of a later message m_2, then all processes will receive m_1 before m_2; and (2) a message m_1 is a *cause* of message m_2 when the send of m_1 is before the send of m_2 in Lamport's

* This research was supported by the Israel Science Foundation (grant 814/22) and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 851811).

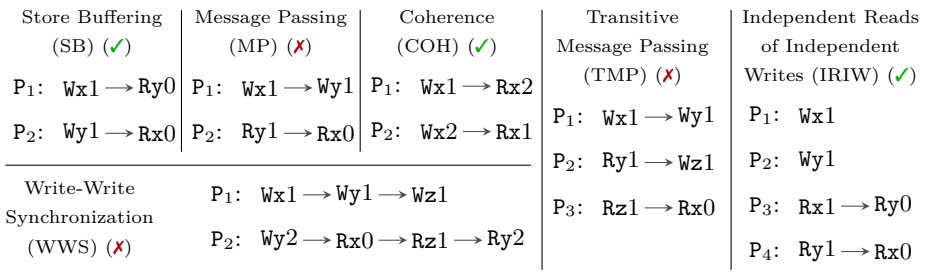


Fig. 1. Examples of behaviors that are possible (✓) or impossible (✗) under DistMem. We assume that the initial value of every location is 0. In each example, every row lists the operations executed by a single process ($\texttt{P}_i$), in the order in which they occur. Operations are either writes of the form $\texttt{Wx}v$, denoting a write of value v to location x, or reads of the form $\texttt{Rx}v$, denoting a read of value v from location x. Message reception is treated as a background activity that is unobservable from the client's point of view and is therefore not depicted in the figure. In SB, Wx1 is received at $\texttt{P}_2$ only after it performs Rx0, so $\texttt{P}_2$ reads the initial value of x. Similarly for Wy1. In MP, Wy1 must be received at $\texttt{P}_2$ before it performs Ry1. Since Wx1 is causally before Wy1, it must be received before it. But then, Wx1 overrides the initial value of x and $\texttt{P}_2$ cannot read 0 from x. In COH, Wx1 is received at $\texttt{P}_2$ only after it performs Wx2, so it overrides it and the value of x is 1. Similarly for Wx2. The explanation for TMP is similar to MP. In IRIW, the two messages of $\texttt{P}_1$ and $\texttt{P}_2$ are received in different orders in each of the other two processes. This is possible since Wx1 and Wy1 are not causally related. In WWS, $\texttt{P}_2$ must receive Wx1 after performing Rx0, and must receive Wz1 before performing Rz1. Since Wx1 is causally before Wy1, which is causally before Wz1, Wy1 must be received between the receive of Wx1 and the receive of Wz1—and thus between Rx0 and Rz1. Hence, this receive overrides Wy2 and $\texttt{P}_2$ cannot read 2 from y.

happens-before [22]—the partial order that (*i*) contains the sequential program order of each process and (*ii*) places each message send before its receive.

Concrete manifestations of DistMem have to implement the causal broadcast abstraction. Implementations of causal broadcast were suggested, e.g., in [28,29]. Ahamad et al. [6], who introduced the "causal memory" (CM, for short) consistency model, implement DistMem on top of a point-to-point send/receive network employing outgoing and incoming message queues. Following this seminal work, other scalable real-world implementations, which tolerate process crashes and long partitions between datacenters, were developed (see, e.g., [24]). Researchers have also studied the verification of DistMem implementations in Coq [16,23]. In particular, the abstract operational specification in [23] is equivalent to DistMem.

Several fundamental questions arise about DistMem:

Q1 What consistency guarantees does DistMem provide? Clearly, it is weaker than sequentially consistent memory (SC) [21] (see Fig. 1 for some examples).
Q2 How do these guarantees relate to existing definitions of causally consistent shared memory? In particular, what is the relation to CM from [6]? Can DistMem consistency be specified in common declarative formalisms for weakly consistent memory?

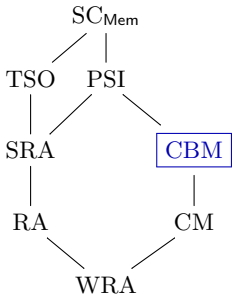
Q3 With the goal of testing implementations of DistMem, how hard is it to check whether a given abstract execution (where message delivery is not exposed) is admitted by DistMem?

Q4 How hard is the verification of (finite-state) shared-memory programs that run on top of DistMem? How does it compare to program verification on top of a centralized (strongly consistent) memory, which is PSPACE-complete [17]?

Our Contribution. This paper addresses these questions. We also identify a subtle gap in previous work on this topic. Concretely, our contributions (in comparison to related work) are as follows:

A1 We propose a novel memory consistency model, called "causal-broadcast memory" (CBM for short). This model precisely captures the observable behaviors of DistMem. Its formulation requires the existence of valid total orders on all operations in the causal past of each process. This definition is obtained as an instance of the definition from [25], which we call CBC for "causal-broadcast consistency". In the same paper, it is related (without proof) to an implementation that lifts an arbitrary sequential transition system to a distributed system by first performing every operation locally and then spreading it over the network by causal broadcast (see Listing 1.1 below). We show that the consistency guarantees provided by this distributed implementation are precisely captured by CBC (see Cor. 1). Then, we prove that queries—operations that do not affect the local state, like reads in a memory system—need not be broadcast (see Cor. 2). As a result, since DistMem is an instance of the more general distributed implementation where queries are not broadcast and CBM is the specialization of CBC for a transition system implementing a memory (a.k.a. key-value store), we obtain the correspondence between CBM and DistMem (see Cor. 3).

A2 We relate CBM to different variants of causal memory consistency that were studied before, all weaker than SC. Our results are summarized in Fig. 2, and we provide examples to demonstrate all "anti-edges" in the figure. In particular, we show that CBM is strictly stronger than CM, which is in contrast to the claim of [25] that argued that DistMem precisely corresponds to CM. Moreover, we establish a data-race-freedom (DRF) guarantee showing that all behaviors allowed by CBM are also admitted by a stronger semantics for programs without write-write races (concurrent writes to the same location). The stronger semantics for this DRF result is "Parallel Snapshot Isolation" (PSI)—a memory model developed in the context of transactional geo-replicated systems, where write-conflicts are all resolved globally [12,30]. Importantly, whether or not a program is write-write-race-free can be checked assuming PSI, which enables client reasoning based solely on the strong semantics. To the best of our knowledge, this is the first result of this kind relating PSI to memory models of causal consistency. Moreover, our DRF guarantee holds for every model between WRA and PSI (see Fig. 2),



SC_{Mem}: Sequential Consistency for Memory [21]
$\propto$ SC: Sequential Consistency [27]
TSO: Total Store Order [26]
PSI: Parallel Snapshot Isolation [12]
SRA: Strong Release-Acquire [19]
$\propto$ CCv: Causal Convergence [27]
CBM: Causal-Broadcast Memory (this paper)
$\propto$ CBC: Causal-Broadcast Consistency ("Causal Consistency" in [25])
RA: the Release-Acquire fragment of the C/C++11 [19]
CM: Causal Memory [6][1]
$\propto$ CC: Causal Consistency [27]
WRA: Weak Release-Acquire [18]
$\propto$ WCC: Weak Causal Consistency (CC in [10])

Fig. 2. Models ordered by strength (higher is stronger): Comparison of models is based on consistency of execution graphs (Def. 19), focusing solely on read and write memory accesses (discarding features that exist in some of these models, such as RMWs and transactions consisting of *multiple* accesses). Models following the $\propto$ symbol are corresponding generalizations of shared-memory consistency to general objects (see §7).

and thus it improves the DRF result from [18] that is against SRA, which is strictly weaker than PSI.

A3 We show that testing CBM-consistency of an abstract execution graph can be performed in polynomial time. The input for this problem is a graph whose vertices consist of the performed memory operations, and edges relate vertices by "program order" (the sequential order within each process) and "reads-from" relation (justifying each read action by a write action). Such execution graphs do not track message broadcasts and receives, which makes consistency testing a challenging task. To show a polynomial time complexity, we devise a definition of CBM that avoids existential quantification over total orders altogether. Instead, this definition proposes a computation that adds ordering constraints at each step and guarantees CBM-consistency if the fixpoint of that computation forms an acyclic relation. This result puts CBM together with multiple other models in terms of complexity of consistency testing [31].

A4 We show that client verification on top of CBM is undecidable. More precisely, control-state reachability is undecidable for finite state concurrent programs accessing shared memory that provides CBM. This is in contrast to decidability results for related models, such as TSO [8,3], and SRA and WRA [18] whose corresponding decision problem is decidable (with non-primitive recursive complexity). To establish undecidability, we show how CBM can simulate perfect FIFO channels, and propose a reduction from reachability in finite transition systems that communicate via two such channels, one in each direction. Notably, the undecidability does not follow from

[1] The formal relations between CM and CC and between CM and WRA require the assumption that values written to every location are unique (see §7.3 and §7.4).

speculation on the future (CBM forbids load-store reordering), which has been used before to establish undecidability of reachability for weak memory model: for the "Promising" model PS 2.0 [4], for Relaxed Memory Ordering (RMO) [8], and for Power [2]. Moreover, CBM does not employ Read-Modify-Writes (RMWs), which were exploited to show a similar undecidability result under the RA memory model [1].[2]

The rest of this paper is organized as follows. In §2, we present basic notations and definitions. In §3, we formalize the generic distributed implementation, of which DistMem is obtained as an instance. In §4 we define causal-broadcast consistency (CBC) and show that it declaratively characterizes histories generated by the distributed implementation. In §5, we define causal-broadcast memory (CBM), and develop an equivalent definition from which we derive a polynomial-time algorithm for testing consistency. In §6, we show the undecidability of the reachability problem for concurrent programs under DistMem. In §7, we compare CBC and CBM to other consistency criteria and weak memory models, and establish the write-write-race-freedom guarantee against PSI. Concluding remarks are in §8. Related work is discussed throughout the paper. The accompanying technical appendix provides full proofs.

2 Preliminaries

Sequences. For an alphabet Σ, we denote by Σ^* (respectively, Σ^+) the set of all finite sequences (non-empty finite sequences) over Σ. We use ϵ to denote the empty sequence. The length of a sequence s is denoted by $|s|$ (in particular $|\epsilon| = 0$). We often identify a sequence s over Σ with its underlying function from $\{1,\ldots,|s|\}$ to Σ, and write $s(k)$ for the symbol at position $1 \leq k \leq |s|$ in s. We write $\sigma \in s$ if the symbol σ appears in s, that is if $s(k) = \sigma$ for some $1 \leq k \leq |s|$. We use "$\cdot$" for the concatenation of sequences. We identify symbols with sequences of length 1, or with their singletons, when needed. Finally, functions in $\Sigma \to \Pi$ are lifted to functions in $\Sigma^* \to \Pi^*$ in the obvious way.

Relations. Given a (binary) relation R, $\mathsf{dom}(R)$ and $\mathsf{codom}(R)$ denote its domain and codomain, and $R^?$, R^+, and R^* denote its reflexive, transitive, and reflexive-transitive closures. The inverse of a relation R is denoted by R^{-1}, and the (left) composition of two relations R_1 and R_2 is denoted by $R_1 ; R_2$ (that is, $\langle x, y\rangle \in R_1;R_2$ iff there exists z such that $\langle x, z\rangle \in R_1$ and $\langle z, y\rangle \in R_2$). The restriction of a relation R on a set A to a subset $B \subseteq A$ is given by $R|_B = R \cap (B \times B)$. We denote by $[A]$ the identity relation on a set A. In particular, $[A] ; R ; [B] = R \cap (A \times B)$. When A is finite, we write $[a_1,\ldots,a_n]$ instead of $[\{a_1,\ldots,a_n\}]$. For an acyclic relation R on a finite set A and a subset $B \subseteq A$, we denote by $\mathsf{lin}(R, B)$ the set of all (strict) total orders on B extending $R|_B$. When R is a total order on a set B, we write $\mathsf{seq}(R)$ for the sequence enumerating B following R. This notation is lifted to sets by defining $\mathsf{seq}(\mathcal{R}) \triangleq \{\mathsf{seq}(R) \mid R \in \mathcal{R}\}$.

[2] To our knowledge, decidability of reachability under RA *without RMWs* is open.

Listing 1.1. Distributed implementation based on causal broadcast (for process p). We assume a given transducer with set of states Q, initial sate q_0, and transition function $\delta : Q \times O \to D \times Q$, where O is a set of operations and D is a domain of output values. We also assume a set of operations $B \subseteq O$ that is broadcast to other processes. Each process p maintains its local state $q_p \in Q$ (initialized to q_0).

```
1 method do(o)                    when o is received from another process:
2   ⟨d, q_p⟩ := δ(q_p, o)           ⟨_, q_p⟩ := δ(q_p, o)
3   if o ∈ B then
4     causal_broadcast(o)
5   return(d)
```

Labeled Transition Systems. A *labeled transition system* (LTS) is a tuple $A = \langle Q, \Sigma, q_0, T \rangle$, where Q is a set of *states*, Σ is an *alphabet*, $q_0 \in Q$ is the *initial state*, and $T \subseteq Q \times \Sigma \times Q$ is a set of *transitions*. We denote by $A.\mathtt{Q}$, $A.\Sigma$, $A.\mathtt{q}_0$, and $A.\mathtt{T}$ the components of an LTS A. We write $\xrightarrow{\sigma}_A$ for the relation $\{\langle q, q' \rangle \mid \langle q, \sigma, q' \rangle \in A.\mathtt{T}\}$ and $\to_A$ for $\bigcup_{\sigma \in \Sigma} \xrightarrow{\sigma}_A$. A state $q \in A.\mathtt{Q}$ is *reachable* in A if $A.\mathtt{q}_0 \to_A^* q$. A sequence $\sigma_1, \ldots, \sigma_n$ (for $n \geq 0$) is a *trace of A with final state q* if $A.\mathtt{q}_0 \xrightarrow{\sigma_1}_A \cdots \xrightarrow{\sigma_n}_A q$. Our LTSs often employ a distinguished label $\varepsilon \in A.\Sigma$, marking *silent* transitions. A sequence $tr \in (A.\Sigma \setminus \{\varepsilon\})^*$ is an *observable trace of A with final state q* if it is obtained from a trace of A with final state q by removing all ε labels. We denote the set of all observable traces of A with final state q by $\mathsf{otraces}(A, q)$, and let $\mathsf{otraces}(A) \triangleq \bigcup_{q \in A.\mathtt{Q}} \mathsf{otraces}(A, q)$.

3 A Distributed Reference Implementation

Listing 1.1 presents a generic distributed implementation. It generalizes DistMem (from §1), and applies to any given transducer S that determines the local behavior and set B of *broadcast operations*. Intuitively, each process maintains its local state, which is a state of the transducer S. To perform an operation o, the process uses S on the current local state to get the next state and the returned value. Then, if $o \in B$, the process broadcasts o to the other processes using the causal broadcast abstraction. Finally, the process returns the value to its caller. In turn, when receiving o from another process, the process updates its local state using the transducer S on the current local state (and dismiss the return value). The parametric set B allows us to capture implementations like DistMem, where some operations (reads for DistMem) are not broadcast.

This section is devoted to formulate the implementation in Listing 1.1 as an LTS, which we denote by $\mathsf{Dist}(S, B)$ (Dist stands for 'distributed'). We start by defining objects, specifications, and transducers (§3.1), and continue by defining *message sequence charts*, used to give the formal meaning to the causal broadcast abstraction (§3.2). Then, the definition of $\mathsf{Dist}(S, B)$ uses the two parts (§3.3).

3.1 Transducers and Objects

$\mathsf{Dist}(S, B)$ applies to a general transducer, which determines the local semantics of the object.

Definition 1. An *object* $\mathcal{O} = \langle O, D\rangle$ consists of a set of *operations* O and a *domain of (output) values* D. We let $\mathsf{Lab}(\mathcal{O}) \triangleq O \times D$ and refer to its elements as *labels*. For $l = \langle o, d\rangle \in \mathsf{Lab}(\mathcal{O})$, the functions $\mathtt{op}$ and $\mathtt{out}$ retrieve its operation (o) and output value (d). A *specification* for $\mathcal{O}$ is a prefix-closed set $\mathit{Spec} \subseteq \mathsf{Lab}(\mathcal{O})^*$.

Definition 2. A (deterministic) *transducer* for an object $\mathcal{O} = \langle O, D\rangle$ is a tuple $S = \langle Q, q_0, \delta\rangle$, where Q is a set of *states*, $q_0 \in Q$ is the *initial state*, and $\delta : Q \times O \to D \times Q$ is a *transition function*. We denote by $S.\mathtt{Q}$, $S.\mathtt{q}_0$, and $S.\mathtt{T}$ the components of S. The transition function is lifted to $\delta^* : Q \times O^* \to Q$ as expected: $\delta(q, \langle\rangle) = q$ and $\delta(q, o \cdot \overline{o}) = \delta(q', \overline{o})$ where $\delta(q, o) = \langle \_, q'\rangle$. The specification (for $\mathcal{O}$) *derived* from S consists of all sequences $\langle o_1, d_1\rangle, \langle o_2, d_2\rangle, \ldots, \langle o_n, d_n\rangle$ for which there are states $q_1, q_2, \ldots, q_n$ such that $\delta(q_{i-1}, o_i) = \langle d_i, q_i\rangle$ for $1 \le i \le n$.

Specifications derived from transducers have the additional property that all operations are always enabled. That is, for every $s \in \mathit{Spec}$ and $o \in O$, we have $s \cdot \langle o, d\rangle \in \mathit{Spec}$ for some $d \in D$.

The central example used throughout the paper is *memory*:

Example 1. Given a set Loc of *locations*, a set Val of *values*, and a distinguished *initial value* $\mathsf{v}_{\mathsf{init}} \in \mathsf{Val}$, the object $\mathsf{Mem} = \langle O_{\mathsf{Mem}}, D_{\mathsf{Mem}}\rangle$ is defined as follows. The set O_{Mem} consists of *write operations* of the form $\mathtt{W}xv$ and *read operations* of the form $\mathtt{R}x$, where $x \in \mathsf{Loc}$ and $v \in \mathsf{Val}$. We use the functions $\mathtt{loc}$ to retrieve the location (x) of an operation and $\mathtt{val}$ to retrieve the value (v) of a write operation. The domain is $D_{\mathsf{Mem}} = \mathsf{Val} \uplus \{\bot\}$ ($\bot$ is used for the output of write operations). The transducer for Mem is given by $S_{\mathsf{Mem}} = \langle Q, q_0, \delta\rangle$, where $Q = \mathsf{Loc} \to \mathsf{Val}$; $q_0 = \lambda x.\, \mathsf{v}_{\mathsf{init}}$; $\delta(q, \mathtt{W}xv) = \langle \bot, q[x \mapsto v]\rangle$ for all $q \in Q$, $x \in \mathsf{Loc}$, and $v \in \mathsf{Val}$; and $\delta(q, \mathtt{R}x) = \langle q(x), q\rangle$ for all $q \in Q$ and $x \in \mathsf{Loc}$. The derived specification, denoted by $\mathit{Spec}_{\mathsf{Mem}}$, consists of all sequences s satisfying the following:

- If $\langle \mathtt{W}xv, d\rangle \in s$, then $d = \bot$.
- If $s = s_1 \cdot \langle \mathtt{W}xv, \bot\rangle \cdot s_2 \cdot \langle \mathtt{R}x, d\rangle \cdot s_3$ and $\langle \mathtt{W}xv', \bot\rangle \notin s_2$ for every $v' \in \mathsf{Val}$, then $d = v$.
- If $s = s_1 \cdot \langle \mathtt{R}x, d\rangle \cdot s_2$ and $\langle \mathtt{W}xv, \bot\rangle \notin s_1$ for every $v \in \mathsf{Val}$, then $d = \mathsf{v}_{\mathsf{init}}$.

In our examples, we assume $\mathsf{Loc} = \{\mathtt{x}, \mathtt{y}, \ldots\}$, $\mathsf{Val} = \mathbb{N}$, and $\mathsf{v}_{\mathsf{init}} = 0$. For brevity, we write $\mathtt{W}xv$ for $\langle \mathtt{W}xv, \bot\rangle$ and $\mathtt{R}xv$ for $\langle \mathtt{R}x, v\rangle$.

Below, it is useful to distinguish a particular kind of operations, called "queries":

Definition 3. Let *Spec* be a specification for object $\mathcal{O} = \langle O, D\rangle$. An operation $o \in O$ is a *query* w.r.t. *Spec* if the following hold:

- If $s_1 \cdot \langle o, d\rangle \cdot s_2 \in \mathit{Spec}$, then $s_1 \cdot s_2 \in \mathit{Spec}$.
- If $s_1 \cdot s_2 \in \mathit{Spec}$, then $s_1 \cdot \langle o, d\rangle \cdot s_2 \in \mathit{Spec}$ for some $d \in D$.

An operation is an *update* if it is not a query. We denote by Q_{Spec} and U_{Spec} the subsets of O consisting of all queries and updates w.r.t. *Spec*, respectively.

In particular, $\mathsf{Q}_{Spec_{\mathsf{Mem}}}$, the set of queries w.r.t. the memory specification, is the set of read operations, and $\mathsf{U}_{Spec_{\mathsf{Mem}}}$, the set of updates w.r.t. the memory specification, is the set of write operations. Clearly, transitions in a transducer S that do not change the internal state are queries, i.e., if $S.\mathtt{T}(q, o) = \langle \_, q \rangle$ for every $q \in S.\mathtt{Q}$, then o is a query w.r.t. the derived specification.

3.2 Message Sequence Charts

To formalize causal broadcast, we employ *message sequence charts* (which we simply call *charts*). These are directed graphs whose vertices, called *events*, represent actions performed during the run, and edges relate the actions executed in the same process (*"program order"*), as well as the broadcast of each message to its receives (*"propagation relation"*). Charts are used to track the propagation of messages during runs, and make sure that the causal consistency condition holds when messages are received. From here on, we assume a set $\mathsf{Ps} = \{\mathtt{P}_1, \ldots, \mathtt{P}_N\}$ of process identifiers. The formal definitions are given next.

Definition 4. An *event* e for object $\mathcal{O} = \langle O, D \rangle$ is a tuple $\langle p, sn, l \rangle$, where $p \in \mathsf{Ps}$, called the event's *process identifier*, $sn \in \mathbb{N}$, called the event's *serial identifier*, and $l \in \mathsf{Lab}(\mathcal{O}) \cup \{\mathtt{rcv}\}$, called the event's *label*. The functions $\mathtt{proc}$, $\mathtt{sn}$ and $\mathtt{lab}$ return the process identifier (p), serial identifier (sn) and label (l). When $\mathtt{lab}(e) = \mathtt{rcv}$, e is called a *receive event*, and otherwise (i.e., $\mathtt{lab}(e) \in \mathsf{Lab}(\mathcal{O})$), e is called a *do event*. We denote by $\mathsf{Evs}(\mathcal{O})$, Rcv, and $\mathsf{Do}(\mathcal{O})$ the sets of all events for $\mathcal{O}$, all receive events, and all do events for $\mathcal{O}$ (respectively). We lift $\mathtt{op}$ and $\mathtt{out}$ from labels to do events. For $E \subseteq \mathsf{Evs}(\mathcal{O})$, $p \in \mathsf{Ps}$, and $O' \subseteq O$, we let $E|_p \triangleq \{e \in E \mid \mathtt{proc}(e) = p\}$ and $E|_{O'} \triangleq \{e \in E \mid \mathtt{op}(e) \in O'\}$.

Definition 5. A relation po is a *program order* for a set $E \subseteq \mathsf{Evs}(\mathcal{O})$ if $po = \biguplus_{p \in \mathsf{Ps}} po^p$, for some relations po^p, such that each po^p is a strict total order on $E|_p$.

Definition 6. A relation $prop$ is a *propagation relation* for a set $E \subseteq \mathsf{Evs}(\mathcal{O})$ if the following hold:

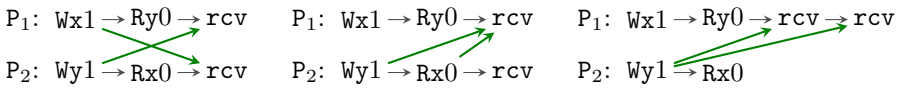
(1) If $\langle e_1, e_2 \rangle \in prop$, then $e_1, e_2 \in E$.
(2) If $\langle e_1, e_2 \rangle \in prop$, then $e_1 \in \mathsf{Do}(\mathcal{O})$, $e_2 \in \mathsf{Rcv}$, and $\mathtt{proc}(e_1) \neq \mathtt{proc}(e_2)$.
(3) $E \cap \mathsf{Rcv} \subseteq \mathsf{codom}(prop)$.
(4) If $\langle e_1, e \rangle, \langle e_2, e \rangle \in prop$, then $e_1 = e_2$.
(5) If $\langle e, e_1 \rangle, \langle e, e_2 \rangle \in prop$ and $\mathtt{proc}(e_1) = \mathtt{proc}(e_2)$, then $e_1 = e_2$.

For $e_2 \in E \cap \mathsf{Rcv}$, we write $prop^{-1}(e_2)$ to refer to the unique event e_1 with $\langle e_1, e_2 \rangle \in prop$. For a set $E' \subseteq E$, we let $prop^{-1}(E') \triangleq \{\mathtt{prop}^{-1}(e) \mid e \in E' \cap \mathsf{Rcv}\}$.

The conditions above ensure that the propagation relation justifies every receive event by a *single* broadcast of a *different* process, and that every broadcast justifies *at most one* receive event of every other process (since a process cannot receive twice the same message). We now have all ingredients to define charts:

Definition 7. A *message sequence chart* (*chart*, for short) for an object $\mathcal{O}$ is a tuple $C = \langle E, po, prop \rangle$, where $E \subseteq \mathsf{Evs}(\mathcal{O})$ is a finite set of events for $\mathcal{O}$, po is a program order for E, and $prop$ is a propagation relation for E. We denote the components of C by $C.\mathtt{E}$, $C.\mathtt{po}$, and $C.\mathtt{prop}$. For $E' \subseteq \mathsf{Evs}(\mathcal{O})$, we write $C.E'$ for $C.\mathtt{E} \cap E'$ (e.g., $C.\mathsf{Rcv} = C.\mathtt{E} \cap \mathsf{Rcv}$).

Example 2. Consider the following "chart candidates". Events of each process are enumerated from left to right following their po-order and green edges represent the propagation relation:



The left example describes a valid chart for Mem. The middle one is not a chart since conditions (3) and (4) of Def. 6 are invalidated: some receive receives no messages and another receives more than one. The right example is also not a chart since condition (5) of Def. 6 is invalidated: an event is received more than once by the same process.

An important derived relation in charts is *visibility*, which corresponds to Lamport's *happens-before* [22]—the partial order that (*i*) contains the sequential program orders and (*ii*) places each message send before its receive:

Definition 8. The *visibility relation* induced by a chart C (for object $\mathcal{O}$), denoted by $C.\mathtt{vis}$, is given by $C.\mathtt{vis} \triangleq (C.\mathtt{po} \cup C.\mathtt{prop})^+$.

3.3 An Operational Definition of $\mathsf{Dist}(S, B)$

We define $\mathsf{Dist}(S, B)$—the LTS that captures the algorithm in Listing 1.1 using transducer S for object $\mathcal{O} = \langle O, D \rangle$ that only broadcasts operations from a set $B \subseteq O$. DistMem is the instance given by $\mathsf{Dist}(S_{\mathsf{Mem}}, \mathsf{Wr})$, where Wr is the set of write operations. Also, Algorithm 1 from [25] using transducer S and broadcasting all operations is obtained by $\mathsf{Dist}(S, O)$.

The states of $\mathsf{Dist}(S, B)$ are of the form $\langle C, \bar{q} \rangle$, where C is a chart for $\mathcal{O}$ and $\bar{q}$ is a mapping from Ps to $S.\mathtt{Q}$ (the set of states of S). The chart C tracks the generated chart along the run, used to determine what messages can be received at any point, and $\bar{q}$ remembers the local state of each process.

The following notation is useful for the transitions definition:

Notation 1 *The chart obtained from a chart C by adding a do event e, denoted by $\mathsf{Add}(C, e)$, is $\langle E', po', C.\mathtt{prop} \rangle$, where $E' = C.\mathtt{E} \uplus \{e\}$ and $po' = C.\mathtt{po} \cup (C.\mathtt{E}|_{\mathtt{proc}(e)} \times \{e\})$. The chart obtained from a chart C by adding a receive event e receiving from e', denoted by $\mathsf{Add}(C, e, e')$, is $\langle E', po', prop' \rangle$, where E' and po' are as above and $prop' = C.\mathtt{prop} \cup \{\langle e', e \rangle\}$. (We use this notation when $prop'$ satisfies the conditions of Def. 6 for E', so that $\mathsf{Add}(C, e, e')$ is indeed a chart.)*

Definition 9. Let $S = \langle Q, q_0, \delta \rangle$ be a transducer for object $\mathcal{O} = \langle O, D \rangle$, and $B \subseteq O$. $\mathsf{Dist}(S, B)$ is the LTS whose states are tuples $\langle C, \bar{q} \rangle$, where C is a chart

for $\mathcal{O}$ and $\bar{q} \in \mathsf{Ps} \to Q$; its alphabet is $(\mathsf{Ps} \times \mathsf{Lab}(\mathcal{O})) \cup \{\varepsilon\}$; its initial state is $\langle C_0, \bar{q}_0 \rangle$, where $C_0 = \langle \emptyset, \emptyset, \emptyset \rangle$ and $\bar{q}_0 = \lambda p.\, q_0$; and its transitions are given by:

$$\textsc{do}\ \frac{\begin{array}{c}\langle d, q' \rangle = \delta(\bar{q}(p), o) \\ \mathtt{proc}(e) = p \qquad \mathtt{lab}(e) = \langle o, d \rangle \\ C' = \mathsf{Add}(C, e)\end{array}}{\langle C, \bar{q} \rangle \xrightarrow{p, \langle o, d \rangle}_{\mathsf{Dist}(S,B)} \langle C', \bar{q}[p \mapsto q'] \rangle} \qquad \textsc{receive}\ \frac{\begin{array}{c}e' \in C.\mathsf{Do}(\mathcal{O})|_B \qquad e' \notin \mathsf{dom}(C.\mathtt{prop}^? ; [C.\mathtt{E}|_p]) \\ \mathsf{Do}(\mathcal{O})|_B \cap \mathsf{dom}(C.\mathtt{vis} ; [e']) \subseteq \mathsf{dom}(C.\mathtt{prop}^? ; [C.\mathtt{E}|_p]) \\ \langle \_, q' \rangle = \delta(\bar{q}(p), \mathsf{op}(e')) \\ \mathtt{proc}(e) = p \qquad \mathtt{lab}(e) = \mathtt{rcv} \\ C' = \mathsf{Add}(C, e, e')\end{array}}{\langle C, \bar{q} \rangle \xrightarrow{\varepsilon}_{\mathsf{Dist}(S,B)} \langle C', \bar{q}[p \mapsto q'] \rangle}$$

In words, when a process p performs $\mathtt{do}(o)$, the output d and the new state q' are calculated using δ according to the current state $\bar{q}(p)$, and the corresponding event is added to the chart C after all previous events of p. In turn, at any point p may receive o from a different process. Then, the new state q' is calculated using δ according to the current state $\bar{q}(p)$, and the corresponding receive event e is added to C after all previous events of p. Also, $\mathtt{prop}$ is updated by adding an edge from e' to e, where e' is the broadcast event (whose operation is o). The condition $e' \in C.\mathsf{Do}(\mathcal{O})|_B$ expresses that o belongs to the set of operations that are broadcast. The condition $e' \notin \mathsf{dom}(C.\mathtt{prop}^? ; [C.\mathtt{E}|_p])$ means that the broadcast was executed by a different process, and that the message cannot be received twice by p. The condition $\mathsf{Do}(\mathcal{O})|_B \cap \mathsf{dom}(C.\mathtt{vis} ; [e']) \subseteq \mathsf{dom}(C.\mathtt{prop}^? ; [C.\mathtt{E}|_p])$ captures the fact that the broadcast is causal: all operations that were executed $\mathtt{vis}$-before e' (and should be broadcast according to B) must have already been received by p (or performed by it) before it can receive e'. Note that receive transitions are labeled with ε, since we treat them as *silent*—they are not observable to client programs that use $\mathsf{Dist}(S, B)$.

Example 3. Let C be the following chart for Mem:

$$\mathtt{P}_1\colon\ \mathtt{Wx1} \to \mathtt{Wy1}$$

Let $\bar{q} : \{\mathtt{P}_1, \mathtt{P}_2\} \to S_{\mathsf{Mem}}.\mathtt{Q}$ be the mapping given by: $\bar{q}(\mathtt{P}_1)(\mathtt{x}) = \bar{q}(\mathtt{P}_1)(\mathtt{y}) = 1$ and $\bar{q}(\mathtt{P}_2)(\mathtt{x}) = \bar{q}(\mathtt{P}_2)(\mathtt{y}) = 0$. The state $\langle C, \bar{q} \rangle$ is reachable in $\mathsf{DistMem}$, by performing two DO steps (with transition labels $\langle \mathtt{P}_1, \mathtt{Wx1} \rangle$ and $\langle \mathtt{P}_1, \mathtt{Wy1} \rangle$). Now, assume we want $\mathtt{P}_2$ to DO the read operation $\mathtt{Ry1}$ (attempting to get the behavior presented in the MP example in Fig. 1). For the output to be 1, $\bar{q}(\mathtt{P}_2)(\mathtt{y})$ must first be updated using a RECEIVE step in $\mathtt{P}_2$, with the event labeled $\mathtt{Wy1}$ (performed by $\mathtt{P}_1$) as the received event, e'. However, we cannot perform this step from C, since the condition $\mathsf{W} \cap \mathsf{dom}(C.\mathtt{vis} ; [e']) \subseteq \mathsf{dom}(C.\mathtt{prop}^? ; [C.\mathtt{E}|_{\mathtt{P}_2}])$ (where W is the set of write events) does not hold: the event labeled $\mathtt{Wx1}$ is $\mathtt{po}$-before (therefore, $\mathtt{vis}$-before) e', but it has not been received in $\mathtt{P}_2$ yet. $\mathtt{P}_2$ must first RECEIVE the event labeled $\mathtt{Wx1}$, which *is* a valid step. Then it can RECEIVE e', and we can finally perform the desired DO step and obtain the following chart:

$$\begin{array}{ll}\mathtt{P}_1\colon & \mathtt{Wx1} \to \mathtt{Wy1} \\ & \downarrow \quad\ \downarrow \\ \mathtt{P}_2\colon & \mathtt{rcv} \to \mathtt{rcv} \to \mathtt{Ry1}\end{array}$$

Note that because of the first RECEIVE step, now $\bar{q}(\mathtt{P}_2)(\mathtt{x}) = 1$, and the operation $\mathtt{Rx0}$ cannot be added to $\mathtt{P}_2$ (without performing more writes first). This explains why the behavior of MP is impossible for DistMem.

4 A Declarative Presentation of $\mathsf{Dist}(S, B)$

We provide a declarative (a.k.a. axiomatic) presentation of $\mathsf{Dist}(S, B)$. Such presentation focuses on which behaviors are allowed by the system, rather than on how they are obtained operationally. The declarative presentation is obtained in two steps. First, we precisely characterize the charts that are generated in runs of $\mathsf{Dist}(S, B)$ (§4.1). Then, we define consistent *histories*, where receive events are hidden (§4.2). Finally, we use this presentation to show that broadcasting queries has no observable effect (§4.3).

4.1 A Characterization of Charts Generated by $\mathsf{Dist}(S, B)$

We identify conditions on C that ensure that it is generated by $\mathsf{Dist}(S, B)$.

Definition 10. A chart C (for object $\mathcal{O}$) is:

1. *generated by* $\mathsf{Dist}(S, B)$ *with final state* $\bar{q}$ if $\langle C, \bar{q}\rangle$ is reachable in $\mathsf{Dist}(S, B)$.
2. *generated by* $\mathsf{Dist}(S, B)$ if C is generated by $\mathsf{Dist}(S, B)$ with some final state.

In short, charts generated by $\mathsf{Dist}(S, B)$: (i) faithfully follow causal broadcast where only operations from B are broadcast; and (ii) respect the local executions of S. First, we encode what "causal broadcast" means:

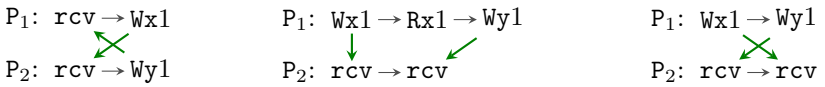
Definition 11. A chart C for object $\mathcal{O} = \langle O, D\rangle$ is *causal for a set* $B \subseteq O$ if the following hold:

(1) $\mathsf{dom}(C.\mathtt{prop}) \subseteq \mathsf{Do}(\mathcal{O})|_B$.
(2) $C.\mathtt{vis}$ is irreflexive.
(3) $\mathsf{Do}(\mathcal{O})|_B \cap \mathsf{dom}(C.\mathtt{vis} ; C.\mathtt{prop} ; [C.\mathtt{E}|_p]) \subseteq \mathsf{dom}(C.\mathtt{prop}^? ; [C.\mathtt{E}|_p])$ for every $p \in \mathsf{Ps}$.
(4) $C.\mathtt{vis} ; C.\mathtt{prop} ; C.\mathrm{po} ; C.\mathtt{prop}^{-1}$ is irreflexive.

C is called *causal* if it is causal for O (the set of all object operations).

Condition (1) ensures that only the events whose operation is in B are broadcast to other processes. Condition (2) ensures that the visibility relation is a partial order. Condition (3) ensures that if some event e propagates to some process p, then all events in $\mathsf{Do}(\mathcal{O})|_B$ that are visible at e also propagate to p. Condition (4) puts further restrictions on the propagation order requiring the events visible at e to propagate to p before e itself. Taken together, the above conditions formalize the semantics of causal broadcast. In particular, they ensure that if $\langle e_1, e_2\rangle \in C.\mathtt{vis}|_{\mathsf{Do}(\mathcal{O})|_B}$ and $\langle e_2, r_2\rangle \in C.\mathtt{prop}$, then either $\langle e_1, r_1\rangle \in C.\mathtt{prop}$ for some r_1 with $\langle r_1, r_2\rangle \in C.\mathrm{po}$, or $\langle e_1, r_2\rangle \in C.\mathrm{po}$. This definition can be seen as an adaptation of *causally-ordered message sequence charts* from [14, Definition 2.4] for broadcasts, instead of point-to-point send, and with the possibility to "block" some operations from being broadcast.

Example 4. Consider the following charts for Mem.



The left chart is not causal since condition (2) of Def. 11 is invalidated: we have a vis cycle. The middle one is not causal since condition (3) is invalidated: the event labeled Rx1 has vis ; prop-path to $\mathtt{P}_2$ but not $\mathtt{prop}^?$-path to $\mathtt{P}_2$. However, this chart *is* causal for the set of write operations. The right chart is not causal since condition (4) is invalidated: there is a vis-path from the event labeled Wx1 to the one labeled Wy1, and they both propagate to $\mathtt{P}_2$ but in the opposite order.

Next, we formalize the consistency of a chart w.r.t. the local computation at each process. This is done by requiring that the sequence of events observed at each process matches the derived specification. However, when justifying the outputs in one process, we may "reevaluate" the outputs of events originated at other processes (see Ex. 5 below). This corresponds to the fact that outputs are not broadcast and ignored when invoking S when operations are received.

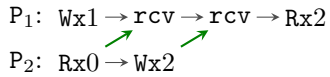
Definition 12. The *event sequence observed* at a process $p \in \mathsf{Ps}$ in a chart C, denoted by $\mathsf{obs}_p(C)$, is the sequence of do events obtained from $\mathsf{seq}(C.\mathtt{po}|_{\mathsf{Evs}|_p})$ by replacing each receive event e with $C.\mathtt{prop}^{-1}(e)$.

Definition 13. A sequence $s' \in \mathsf{Do}(\mathcal{O})^*$ is a *reevaluation* of a sequence $s \in \mathsf{Do}(\mathcal{O})^*$ if they are identical except possibly for events' outputs (i.e., $|s'| = |s|$ and for every $1 \leq k \leq |s'|$, $\mathtt{proc}(s'(k)) = \mathtt{proc}(s(k))$, $\mathtt{sn}(s'(k)) = \mathtt{sn}(s(k))$, and $\mathtt{op}(s'(k)) = \mathtt{op}(s(k))$). A reevaluation s' of s *preserves* a set $E \subseteq \mathsf{Evs}(\mathcal{O})$ if $\mathtt{out}(s'(k)) = \mathtt{out}(s(k))$ whenever $s(k) \in E$. We denote by $\mathsf{reeval}(s, E)$ the set of all reevaluations of s that preserve E.

Definition 14. Let *Spec* be a specification for an object $\mathcal{O}$.

1. A sequence $s \in \mathsf{Do}(\mathcal{O})^*$ *respects Spec* if $\mathtt{lab}(s) \in \mathit{Spec}$.
2. A chart C for $\mathcal{O}$ *respects Spec* if for every $p \in \mathsf{Ps}$, some reevaluation $s \in \mathsf{reeval}(\mathsf{obs}_p(C), \mathsf{Evs}|_p)$ respects *Spec*.

Example 5. Consider the following chart C for Mem:



$\mathtt{lab}(\mathsf{obs}_{\mathtt{P}_1}(C))$ is $\langle \mathtt{Wx1}, \mathtt{Rx0}, \mathtt{Wx2}, \mathtt{Rx2} \rangle$. This sequence is *not* in $\mathit{Spec}_{\mathsf{Mem}}$, so $\mathsf{obs}_{\mathtt{P}_1}(C)$ does not respect $\mathit{Spec}_{\mathsf{Mem}}$. However, if we change the output of the event labeled Rx0 from 0 to 1, we obtain a reevaluation of $\mathsf{obs}_{\mathtt{P}_1}(C)$ that preserves $C.\mathtt{E}|_{\mathtt{P}_1}$ (since the process of this read event is $\mathtt{P}_2$) and respects $\mathit{Spec}_{\mathsf{Mem}}$. Also, $\mathsf{obs}_{\mathtt{P}_2}(C)$ respects $\mathit{Spec}_{\mathsf{Mem}}$. Therefore, C respects $\mathit{Spec}_{\mathsf{Mem}}$.

With the above definitions we characterize the charts generated by $\mathsf{Dist}(S, B)$:

Theorem 1. *Let S be a transducer for object $\mathcal{O} = \langle O, D \rangle$, Spec be the derived specification, and $B \subseteq O$. A chart C for $\mathcal{O}$ is generated by $\mathsf{Dist}(S, B)$ iff C is causal for B and respects Spec.*

4.2 Causal-Broadcast Consistency

The above characterization is rather concrete, as it tracks message propagation in events. In this section, we provide a more abstract declarative presentation, which is decoupled from the underlying message-passing implementation. Our definition adopts the consistency conditions proposed in [25].

Definition 15. A *history* for object $\mathcal{O}$ is a pair $H = \langle E, po \rangle$, where $E \subseteq \mathsf{Do}(\mathcal{O})$ is a finite set of do events for $\mathcal{O}$ and po is a program order for E. The history *induced by a chart* C *(for* $\mathcal{O}$*)*, denoted by $\mathsf{H}(C)$, is given by $\mathsf{H}(C) \triangleq \langle C.\mathsf{Do}(\mathcal{O}), po|_{\mathsf{Do}(\mathcal{O})} \rangle$ (i.e., it is obtained from C by removing all receive events and propagation edges). The notations used for charts (e.g., $C.\mathtt{po}$) are also employed for histories (when applicable). We say that a history H is *generated by* $\mathsf{Dist}(S, B)$ if some chart C with $\mathsf{H}(C) = H$ is generated by $\mathsf{Dist}(S, B)$.

Since the propagation relation is abstracted away, consistency of histories cannot refer to the event sequence observed at each process. Instead, the definition uses existential quantification. It requires the existence of a causal order—corresponding to the chart's visibility relation—and a collection of event sequences, one sequence per process to justify its own output values in a way that aligns with the causal order and adheres to the specification.

Definition 16. A relation co is a *causal order* for a history H if it is a partial order on $H.\mathtt{E}$ extending $H.\mathtt{po}$. The *causal past* w.r.t. co of an event $e \in H.\mathtt{E}$, denoted $co^{-1}(e)$, is given by $co^{-1}(e) \triangleq \mathsf{dom}(co^{?} \,;\, [e])$. For a set $E' \subseteq H.\mathtt{E}$, we let $co^{-1}(E') \triangleq \bigcup_{e \in E'} co^{-1}(e)$.

Definition 17. A history H for $\mathcal{O}$ is *causal-broadcast consistent* (CBC, for short) w.r.t. a specification *Spec* for $\mathcal{O}$ if there exist a causal order co for H and a sequence s_p for each $p \in \mathsf{Ps}$ such that:

(1) $s_p \in \mathsf{seq}(\mathsf{lin}(co, co^{-1}(H.\mathtt{E}|_p)))$.
(2) Some $s'_p \in \mathsf{reeval}(s_p, \mathsf{Evs}|_p)$ respects *Spec*.
(3) If $\mathtt{proc}(s_p(j)) = p$, then $\langle s_p(i), s_p(j) \rangle \in co$ for every $1 \leq i < j$.

Condition (1) requires that each s_p totally orders the causal past of p in a way that agrees with co. Condition (2) requires that s_p indeed matches the local computation in process p (in the same way $\mathsf{obs}_p(C)$ does so in a chart C that respects *Spec*). Condition (3) requires that for every $e \in H.\mathtt{E}|_p$, *only* its causal past comes before it in s_p (corresponds to the notion of a "causal past-constrained serialization" from [25]). Note that two events e, e' may be ordered differently by distinct processes, but if $\mathtt{proc}(e') = p$ and e appears before e' in s_p, then condition (3) forces $\langle e, e' \rangle \in co$, which means that all other processes also have to order e before e' (due to condition (1)).

Example 6. Let C be the chart from Ex. 5. $\mathsf{H}(C)$ is CBC w.r.t. $Spec_{\mathsf{Mem}}$: we can take $co = C.\mathtt{vis}|_{\mathsf{Do}(\mathsf{Mem})}$, $s_{\mathtt{P}_1} = \mathsf{obs}_{\mathtt{P}_1}(C)$, and $s_{\mathtt{P}_2} = \mathsf{obs}_{\mathtt{P}_2}(C)$.

The next example shows the effect of condition (3):

Example 7. Consider the following histories:

$$\begin{array}{ll} \mathtt{P}_1\colon & \mathtt{Wx1} \to \mathtt{Ry1} \to \mathtt{Wz1} \to \mathtt{Rx1} \\ \mathtt{P}_2\colon & \mathtt{Wx2} \to \mathtt{Wy1} \to \mathtt{Rz1} \to \mathtt{Rx2} \end{array} \qquad \begin{array}{ll} \mathtt{P}_1\colon & \mathtt{Wx1} \to \mathtt{Wy2} \\ \mathtt{P}_2\colon & \mathtt{Wx4} \to \mathtt{Ry2} \to \mathtt{Rx4} \to \mathtt{Wz3} \\ \mathtt{P}_3\colon & \mathtt{Rz3} \to \mathtt{Rx1} \end{array}$$

Both histories would be consistent (w.r.t. $Spec_{\mathsf{Mem}}$) without condition (3) in Def. 17. For instance, for the history on the left, we could choose[3] $co = (po \cup \{\langle \mathtt{W}y1, \mathtt{R}y1\rangle, \langle \mathtt{W}z1, \mathtt{R}z1\rangle\})^+$, and the following sequences:

$$s_{\mathtt{P}_1} = \langle \mathtt{Wx2}, \mathtt{Wx1}, \mathtt{Wy1}, \mathtt{Ry1}, \mathtt{Wz1}, \mathtt{Rx1}\rangle \quad s_{\mathtt{P}_2} = \langle \mathtt{Wx1}, \mathtt{Wx2}, \mathtt{Wy1}, \mathtt{Ry1}, \mathtt{Wz1}, \mathtt{Rz1}, \mathtt{Rx2}\rangle$$

Note that condition (3) does not hold, e.g., $\mathtt{Wx2}$ is before $\mathtt{Wx1}$ in $s_{\mathtt{P}_1}$, $\mathtt{Wx1}$ belongs to $\mathtt{P}_1$, but $\langle \mathtt{Wx2}, \mathtt{Wx1}\rangle \notin co$. Thus, this choice cannot work for DistMem: if $\mathtt{P}_1$ received the message about $\mathtt{Wx2}$ before its $\mathtt{Wx1}$, then $\mathtt{P}_2$ cannot receive $\mathtt{Wx1}$ before $\mathtt{Wx2}$. In Ex. 11 below, we use an alternative formulation of CBC to demonstrate that these histories are not CBC. Surprisingly, causal memory (CM) [6], the original formulation of causally consistent memory, allows these histories, despite the fact that they are disallowed by DistMem (see also §7.2).

Remark 1. Our definition is meant to allow "broadcast in progress", where some operations in the history have not yet propagated to all processes. In contrast, the definition in [25] requires that $s_p \in \mathsf{seq}(\mathsf{lin}(co, H.\mathtt{E}))$ instead of our condition (1), which means that the whole history appears in the linearization associated with each process, rather than its causal past only. If operations are always enabled in the object specification (i.e., for every $s \in Spec$ and $o \in O$, we have $s \cdot \langle o, d\rangle \in Spec$ for some $d \in D$), then this difference has no impact on the set of consistent histories. (This condition holds for specifications that are derived from transducers, and assumed in [25].) For specifications that do not enable all operations, some histories may be consistent w.r.t. Def. 17 but not w.r.t. the definition of [25]. For example, for $O = \{\mathtt{o}_1\}$, $D = \{1\}$, and $Spec = \{\epsilon, \langle\langle \mathtt{o}_1, 1\rangle\rangle\}$, the history in which $\mathtt{P}_1$ and $\mathtt{P}_2$ each performs $\langle \mathtt{o}_1, 1\rangle$ is consistent w.r.t. Def. 17 (take $co = \emptyset$), but it is inconsistent w.r.t. the definition of [25], since no linearization of the whole history satisfies condition (2) above.

Next, we establish the correspondence of CBC to charts that are causal and respect a specification *Spec*, and with Thm. 1, we obtain the correspondence of CBC to $\mathsf{Dist}(S, O)$ (which formalizes the claim in [25, Footnote 2]).

Theorem 2. *If C is a causal chart for an object $\mathcal{O}$ that respects a specification Spec for $\mathcal{O}$, then $\mathsf{H}(C)$ is* CBC *w.r.t. Spec.*

Theorem 3. *If H is a history for an object $\mathcal{O}$ that is* CBC *w.r.t. a specification Spec for $\mathcal{O}$, then $H = \mathsf{H}(C)$ for some causal chart C that respects Spec.*

Corollary 1. *Let S be a transducer for an object $\mathcal{O} = \langle O, D\rangle$ and Spec be the derived specification. A history H for $\mathcal{O}$ is generated by $\mathsf{Dist}(S, O)$ iff H is* CBC *w.r.t. Spec.*

[3] For brevity, we identify events with their labels.

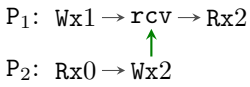
4.3 Update-Driven Distributed Implementations

We use the declarative presentation to prove that queries (see Def. 3) do not need to be propagated in $\mathsf{Dist}(S, O)$, in the sense that the observable behaviors of $\mathsf{Dist}(S, O)$ (the histories generated by it) coincide with the ones of $\mathsf{Dist}(S, \mathsf{U}_{Spec})$.

Lemma 1. *For every chart C' (for $\mathcal{O}$) that is causal for U_{Spec} and respects Spec, there exists a causal chart C (for $\mathcal{O}$) that respects Spec such that $C'.\mathtt{E} = C.\mathtt{E} \setminus \mathit{codom}([\mathsf{Evs}(\mathcal{O})|_{\mathsf{Q}_{Spec}}]\,;C.\mathtt{prop})$, $C'.\mathtt{po} = C.\mathtt{po}|_{C'.\mathtt{E}}$, and $C'.\mathtt{prop} = [\mathsf{Evs}(\mathcal{O})|_{\mathsf{U}_{Spec}}]\,;C.\mathtt{prop}$. In particular, $\mathsf{H}(C) = \mathsf{H}(C')$.*

Lemma 2. *For every causal chart C (for $\mathcal{O}$) that respects Spec, there exists a chart C' (for $\mathcal{O}$) such that: (i) C' is causal for U_{Spec}; (ii) C' respects Spec; and (iii) $C'.\mathtt{E} = C.\mathtt{E} \setminus \mathit{codom}([\mathsf{Evs}(\mathcal{O})|_{\mathsf{Q}_{Spec}}]\,;C.\mathtt{prop})$, $C'.\mathtt{po} = C.\mathtt{po}|_{C'.\mathtt{E}}$, and $C'.\mathtt{prop} = [\mathsf{Evs}(\mathcal{O})|_{\mathsf{U}_{Spec}}]\,;C.\mathtt{prop}$. In particular, $\mathsf{H}(C) = \mathsf{H}(C')$.*

Example 8. The chart C from Ex. 5 is causal and respects $Spec_{\mathsf{Mem}}$. The following chart C', obtained from C by deleting the propagation of the read event, satisfies properties $(i) - (iii)$ from Lemma 2:



Corollary 2. *Let S be a transducer for an object $\mathcal{O}$ and Spec be the derived specification. A history H for $\mathcal{O}$ is generated by $\mathsf{Dist}(S, O)$ iff it is generated by $\mathsf{Dist}(S, \mathsf{U}_{Spec})$.*

In particular, since DistMem is an instance of $\mathsf{Dist}(S, \mathsf{U}_{Spec})$, histories of DistMem coincide with histories that are CBC w.r.t. $Spec_{\mathsf{Mem}}$. As an additional corollary of the above lemmas, we obtain an equivalent definition of CBC consistency that ignores queries of other processes for justifying events of each process (the difference from Def. 17 is highlighted):

Theorem 4. *A history H for $\mathcal{O}$ is* CBC *w.r.t. a specification Spec for $\mathcal{O}$ iff there exist a causal order co for H and a sequence s_p for every $p \in \mathsf{Ps}$ such that:*

- $s_p \in \mathsf{seq}(\mathsf{lin}(co, co^{-1}(H.\mathtt{E}|_p)|_{\mathsf{U}_{Spec}} \cup H.\mathtt{E}|_p))$
- *Some $s'_p \in \mathsf{reeval}(s_p, \mathsf{Evs}|_p)$ respects Spec.*
- *If $\mathtt{proc}(s_p(j)) = p$, then $\langle s_p(i), s_p(j)\rangle \in co$ for every $1 \le i < j$.*

Example 9. In Ex. 6, we showed that the history induced by the chart C from Ex. 5 is CBC w.r.t. $Spec_{\mathsf{Mem}}$. By Thm. 4, the same can be shown by taking $s_{\mathtt{P}_1}$ to be $\mathsf{obs}_{\mathtt{P}_1}(C)$ without the read event labeled Rx0. Note that this is the same as taking $\mathsf{obs}_{\mathtt{P}_1}(C')$, where C' is the chart from Ex. 8.

5 Causal-Broadcast Memory

In this section we use our previous results to derive the declarative consistency guarantees provided by DistMem in the standard formalism of execution graphs used to define weakly consistent memory models (see, e.g., [7,19]).

In the rest of the paper we focus on the memory object. We write Lab, Evs, and Do for Lab(Mem), Evs(Mem), and Do(Mem) (respectively). We lift `loc` and `val` from operations to labels and events in the obvious way. We call an event e a *write (respectively, read) event* if $\mathtt{op}(e)$ is a write (read) operation. We denote the sets of write events and read events by W and R. For a relation R on events, we let $R|_{\mathsf{loc}} \triangleq R \cap \{\langle e, e' \rangle \mid \mathtt{loc}(e) = \mathtt{loc}(e')\}$. To match other existing formulations, we also include explicit *initialization events* in our graphs. These are events of the form $\langle \bot, 0, \mathtt{W}x v_{\mathsf{init}} \rangle$ (with $\mathtt{proc}(e) = \bot$), where $\bot$ signifies that the event is not associated with any particular process. We denote by Init the set of all initialization events.

Unlike histories, execution graphs maintain a "reads-from" relation that justifies every read event by a corresponding write event:

Definition 18. A relation rf is a *reads-from relation* for a set $E \subseteq \mathsf{Do}$ if the following hold:

- If $\langle w, r \rangle \in rf$, then $w \in \mathsf{W}$ and $r \in \mathsf{R}$.
- If $\langle w, r \rangle \in rf$, then $\mathtt{loc}(w) = \mathtt{loc}(r)$ and $\mathtt{val}(w) = \mathtt{out}(r)$.
- If $\langle w_1, r \rangle, \langle w_2, r \rangle \in rf$, then $w_1 = w_2$ (reads have at most one source write).
- $E \cap \mathsf{R} \subseteq \mathsf{codom}(rf)$ (each read reads from some write).

Definition 19. An *execution graph* is a tuple $G = \langle E, po, rf \rangle$, where:

- $E \subseteq \mathsf{Do}$ is a finite set of do events such that $\mathsf{Init} \subseteq E$ and $\mathtt{proc}(e) \neq \bot$ for every $e \in E \setminus \mathsf{Init}$.
- $po = po|_{E \setminus \mathsf{Init}} \uplus (\mathsf{Init} \times (E \setminus \mathsf{Init}))$ and $po|_{E \setminus \mathsf{Init}}$ is a program order for $E \setminus \mathsf{Init}$ (Def. 5).
- rf is a reads-from relation for E.

We denote the components of G by $G.\mathtt{E}$, $G.\mathtt{po}$, and $G.\mathtt{rf}$. We let $G.\mathtt{po}^p \triangleq G.\mathtt{po}\,; [\mathsf{Evs}|_p]$ and $G.\mathtt{rfe} \triangleq G.\mathtt{rf} \setminus G.\mathtt{po}$ ("external" reads-from). For $E' \subseteq \mathsf{Evs}$, we write $G.E'$ for $G.\mathtt{E} \cap E'$ (e.g., $G.\mathsf{W} = G.\mathtt{E} \cap \mathsf{W}$). The *history induced by* G, denoted by $\mathsf{H}(G)$, is given by $\mathsf{H}(G) \triangleq \langle E \setminus \mathsf{Init}, po|_{E \setminus \mathsf{Init}} \rangle$.

Next, we define the CBM memory model—a variant of CBC that applies to execution graphs and ensures that the justification of reads matches the reads-from relation. We start with a definition that utilizes CBC (Def. 21) and later show more direct equivalent formulations (Thms. 5 and 6).

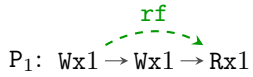
Definition 20. Let rf be a reads-from relation for a set $E \subseteq \mathsf{Do}$. A sequence $s \in E^*$ *respects* rf if the following hold:

- If e_2 occurs in s and $\langle e_1, e_2 \rangle \in [E \setminus \mathsf{Init}]\,; rf$, then e_1 occurs before e_2 in s.

– If $s = s_1 \cdot w \cdot s_2 \cdot r \cdot s_3$, $w \in \mathsf{W}$, $r \in \mathsf{R}$, $\mathtt{loc}(w) = \mathtt{loc}(r)$, and no $w' \in \mathsf{W}$ with $\mathtt{loc}(w') = \mathtt{loc}(w)$ occurs in s_2, then $\langle w, r\rangle \in rf$.

Definition 21. An execution graph G is *causal-broadcast memory consistent* (CBM, for short) if there exist a causal order co for $\mathsf{H}(G)$ and a sequence s_p for every $p \in \mathsf{Ps}$ that satisfies the conditions of Thm. 4 for $\mathsf{H}(G)$ and respects $G.\mathtt{rf}$.

Example 10. The requirement s_p respects $G.\mathtt{rf}$ is essential (and without it, Thms. 5 and 6 below would not hold, as well as the relation to WRA given in §7.4). For instance, consider the following execution graph G (Init is elided):



$\mathsf{H}(G)$ is CBC w.r.t. $Spec_{\mathsf{Mem}}$, since $\langle \mathtt{Wx1}, \mathtt{Wx1}, \mathtt{Rx1}\rangle \in Spec_{\mathsf{Mem}}$. However, the corresponding sequence of events does not respect $\mathtt{rf}$ (invalidates the second condition of Def. 20): $\mathtt{Rx1}$ should read from the second write, not the first one. Therefore, G is not CBM. Note that G is not even WRA (see §7.4).

We can establish the following relation:

Lemma 3. *A history H is* CBC *w.r.t. $Spec_{\mathsf{Mem}}$ iff $\langle H.\mathtt{E} \cup \mathsf{Init}, H.\mathrm{po} \cup (\mathsf{Init} \times H.\mathtt{E}), rf\rangle$ is* CBM *for some reads-from relation rf for $H.\mathtt{E} \cup \mathsf{Init}$.*

We conclude (using Cors. 1 and 2) that $\mathsf{DistMem}$ is captured by CBM: the histories generated by $\mathsf{DistMem}$ are precisely the histories induced by CBM graphs.

Corollary 3. *The following hold:*

– *If H is generated by $\mathsf{DistMem}$, then $H = \mathsf{H}(G)$ for some* CBM *graph G.*
– *If G is* CBM*, then $\mathsf{H}(G)$ is generated by $\mathsf{DistMem}$.*

5.1 Computational Presentation of CBM

We provide an equivalent characterization of CBM that aligns more closely with existing definitions of weak memory models (see §7) and is designed to facilitate efficient consistency checking. The first step is to replace the existential quantification over co and sequences s_p by existential quantification on valid total orders on relevant write events, with one order per process.

Theorem 5. *An execution graph G is* CBM *iff there exist relations $\{wo^p\}_{p\in\mathsf{Ps}}$ such that the following hold for every $p \in \mathsf{Ps}$, where $wo = \bigcup_{q\in\mathsf{Ps}} wo^q ; [G.\mathtt{E}|_q]$:*

(1) wo^p is a total order on $\mathsf{Init} \cup \mathit{dom}([\mathsf{W}] ; wo^* ; G.\mathtt{rfe}^? ; [G.\mathtt{E}|_p])$.
(2) $wo^{p?} ; G.\mathtt{rfe}^? ; G.\mathrm{po}^p$ is irreflexive.
(3) $wo^p|_{\mathsf{loc}} ; wo^{p?} ; G.\mathtt{rfe}^? ; G.\mathrm{po}^p ; G.\mathtt{rf}^{-1}$ is irreflexive.
(4) $wo^p ; wo$ is irreflexive.

The relations wo^p describe the linearizations s_p restricted to W, and wo represents a "global" write order that all processes agree upon. Condition (2) allows us to extend wo^p with the read events of p to obtain s_p, such that it agrees with $\mathtt{po}$, and every read event is preceded by its $\mathtt{rf}$ source. Condition (3) ensures that no write "overrides" the $\mathtt{rf}$ source of a read in s_p. Condition (4) ensures that s_p agrees with the order in which other processes place write events before their own write events, which corresponds to the last condition of Thm. 4.

Next, we give an equivalent definition that does not require finding total orders. Instead, it identifies the minimal constraints on $\{wo^p\}_{p\in\mathsf{Ps}}$ and requires that there are no cycles.

Definition 22. The family $\{G.\mathtt{wb}^p\}_{p\in\mathsf{Ps}}$ of *write-before* relations induced by an execution graph G is inductively defined as follows:

$$\frac{\begin{array}{c}w_1, w_2 \in \mathsf{W} \\ \mathtt{proc}(w_2) = p \\ \langle w_1, w_2\rangle \in G.\mathtt{rfe}^? ; G.\mathtt{po}\end{array}}{\langle w_1, w_2\rangle \in G.\mathtt{wb}^p}\ (\mathrm{B}) \qquad \frac{\begin{array}{c}w_1, w_2 \in \mathsf{W} \qquad \mathtt{loc}(w_1) = \mathtt{loc}(w_2) \\ w_1 \neq w_2 \\ \langle w_1, w_2\rangle \in G.\mathtt{wb}^{p?} ; G.\mathtt{rfe}^? ; G.\mathtt{po}^p ; G.\mathtt{rf}^{-1}\end{array}}{\langle w_1, w_2\rangle \in G.\mathtt{wb}^p}\ (\mathrm{C})$$

$$\frac{\begin{array}{c}\mathtt{proc}(w_2) = q \qquad \langle w_1, w_2\rangle \in G.\mathtt{wb}^q \\ w_2 \in \mathsf{dom}((\cup_{q\in\mathsf{Ps}} G.\mathtt{wb}^q ; [G.\mathtt{E}|_q])^* ; G.\mathtt{rfe}^? ; [G.\mathtt{E}|_p])\end{array}}{\langle w_1, w_2\rangle \in G.\mathtt{wb}^p}\ (\mathrm{Q})$$

$$\frac{\langle w_1, w_2\rangle \in G.\mathtt{wb}^p \qquad \langle w_2, w_3\rangle \in G.\mathtt{wb}^p}{\langle w_1, w_3\rangle \in G.\mathtt{wb}^p}\ (\mathrm{T})$$

Definition 22 constructively reflects the requirements of Thm. 5. For example, condition (2) of Thm. 5 is rephrased as "if $\langle w_1, w_2\rangle \in G.\mathtt{rfe}^? ; G.\mathtt{po}^p$ then $\langle w_1, w_2\rangle \in wo^p$", as stated in rule (B). The rules specify the basic restrictions that linearizations must obey, and the existence of wo^p boils down to verifying acyclicity of $G.\mathtt{wb}^p$ for each process.

Theorem 6. *An execution graph G is* CBM *iff $G.\mathtt{wb}^p$ is irreflexive for every p.*

Since reaching a fixpoint following Def. 22 and checking acyclicity require polynomial time, we obtain that testing CBM is in PTIME.

The next example applies Thm. 6 to prove that the execution graphs of the histories from Ex. 7 are not CBM. The histories in that example are *differentiated*—every value is written at most once to each location—so the reads-from relation of the execution graph is uniquely determined from the history.

Example 11. Consider the execution graphs that induce the histories in Ex. 7. For the left history, we have $\langle \mathtt{Wx2}, \mathtt{Wy1}\rangle \in G.\mathtt{po}^{\mathtt{P}_2}$, and so $\langle \mathtt{Wx2}, \mathtt{Wy1}\rangle \in G.\mathtt{wb}^{\mathtt{P}_2}$ by rule (B). Since $\mathtt{proc}(\mathtt{Wy1}) = \mathtt{P}_2$, we have $\langle \mathtt{Wx2}, \mathtt{Wy1}\rangle \in G.\mathtt{wb}^{\mathtt{P}_1}$ by rule (Q). Now, since $\langle \mathtt{Wy1}, \mathtt{Rx1}\rangle \in G.\mathtt{rfe} ; G.\mathtt{po}^{\mathtt{P}_1}$ and $\langle \mathtt{Wx1}, \mathtt{Rx1}\rangle \in G.\mathtt{rf}$, we derive $\langle \mathtt{Wx2}, \mathtt{Wx1}\rangle \in G.\mathtt{wb}^{\mathtt{P}_1}$ by rule (C). Since $\mathtt{proc}(\mathtt{Wx1}) = \mathtt{P}_1$, we have $\langle \mathtt{Wx2}, \mathtt{Wx1}\rangle \in G.\mathtt{wb}^{\mathtt{P}_2}$ by rule

(Q). However, a symmetric argument for $\mathtt{P}_2$ gives us $\langle \mathtt{Wx1}, \mathtt{Wx2} \rangle \in G.\mathtt{wb}^{\mathtt{P}_2}$, which entails a cycle in $G.\mathtt{wb}^{\mathtt{P}_2}$.

For the right history, we explain in less details. Using (B) and (Q), $\langle \mathtt{Wx1}, \mathtt{Wy2} \rangle \in G.\mathtt{wb}^{\mathtt{P}_2}$. Thus, $\langle \mathtt{Wx1}, \mathtt{Rx4} \rangle \in G.\mathtt{wb}^{\mathtt{P}_2}; G.\mathtt{rfe}; G.\mathtt{po}^{\mathtt{P}_2}$, and $\langle \mathtt{Wx1}, \mathtt{Wx4} \rangle \in G.\mathtt{wb}^{\mathtt{P}_2}$ follows by rule (C). By rule (Q), $\langle \mathtt{Wx1}, \mathtt{Wx4} \rangle \in G.\mathtt{wb}^{\mathtt{P}_3}$. However, by similar arguments we obtain $\langle \mathtt{Wx4}, \mathtt{Wx1} \rangle \in G.\mathtt{wb}^{\mathtt{P}_3}$.

Definition 22 and Thm. 6 draw on existing techniques from the study of weak memory models. For example, the *saturation semantics* introduced in [5] for the purpose of efficiently model checking concurrent programs running under the Release-Acquire memory model (RA), uses a similar technique: adding to the execution graph only the edges that must be contained in any modification (a.k.a. coherence) order, and looking for forbidden cycles in the obtained graph. An earlier similar result for RA appears in [20, Appendix B]. A notable difference is that RA uses a single modification order, whereas CBM maintains a separate order per process since processes can disagree on the order of certain writes.

We also note that in [15] it is shown that consistency testing *without a given reads-from relation*, under any model between SLOW and SC, is NP-hard. Since CBM is included in this range and this problem is in NP for CBM (using Thm. 6), we deduce the NP-completeness of testing CBM without a given reads-from relation, which puts it together with a variety of other models [13].

6 Control-State Reachability for DistMem

The control-state reachability for DistMem considers a finite-state concurrent program that uses a shared memory implemented by DistMem for communication between processes and asks whether a particular program state can be reached. In this section we precisely define this problem and sketch the undecidability argument (§6.1).

Example 12. Consider the following concurrent program, given in pseudo-code:

$$\begin{array}{l|l} \mathtt{x} := 1 & \mathtt{y} := 1 \\ \mathtt{a} := \mathtt{y} & \mathtt{b} := \mathtt{x} \end{array}$$

An algorithm solving the reachability problem would have to decide, for example, whether we can have $\mathtt{a} = \mathtt{b} = 0$ after executing this program on top of DistMem. While it cannot happen under sequential consistency (§7.1), it is possible using DistMem. Though it is clear that the reachability problem for SC is decidable and decidability results have been obtained for other memory models [8,3,18], we show this problem is undecidable for DistMem.

The next definitions of a program, linking of a program with a memory system, and control-state reachability are adopted from [18].

Definition 23. A *program* Pr maps each $p \in \mathsf{Ps}$ to a finite LTS $Pr(p)$ over the alphabet $\mathsf{Lab} \cup \{\varepsilon\}$ (ε stands for "silent" actions that do not interact with the

memory) representing a top-level parallel composition of sequential programs. A program Pr induces an LTS, which we also denote by Pr: its states are functions, denoted by $\overline{p}$, assigning a state in $Pr(p).\mathtt{Q}$ to every $p \in \mathsf{Ps}$; its alphabet is $(\mathsf{Ps} \times \mathsf{Lab}) \cup \{\varepsilon\}$; its initial state is $\overline{p}_0$ defined by $\overline{p}_0 = \lambda p.\, Pr(p).\mathtt{q}_0$; and its transitions interleave transitions of Pr's components (i.e., if $\overline{p}(p) \xrightarrow{l}_{Pr(p)} q$ for some $l \in \mathsf{Lab}$, then $\overline{p} \xrightarrow{p,l}_{Pr} \overline{p}[p \mapsto q]$; and if $\overline{p}(p) \xrightarrow{\varepsilon}_{Pr(p)} q$, then $\overline{p} \xrightarrow{\varepsilon}_{Pr} \overline{p}[p \mapsto q]$).

Definition 24. The *linking of a program Pr with* DistMem, denoted by $Pr \bowtie \mathsf{DistMem}$, is the LTS whose set of states is $Pr.\mathtt{Q} \times \mathsf{DistMem}.\mathtt{Q}$; its alphabet is $(\mathsf{Ps} \times \mathsf{Lab}) \cup \{\varepsilon\}$; its initial state is $\langle Pr.\mathtt{q}_0, \mathsf{DistMem}.\mathtt{q}_0\rangle$; and its transitions are:

$$\frac{\overline{p} \xrightarrow{p,l}_{Pr} \overline{p}' \qquad \langle C, \bar{q}\rangle \xrightarrow{p,l}_{\mathsf{DistMem}} \langle C', \bar{q}'\rangle}{\langle \overline{p}, \langle C, \bar{q}\rangle\rangle \xrightarrow{p,l}_{Pr\bowtie\mathsf{DistMem}} \langle \overline{p}', \langle C', \bar{q}'\rangle\rangle}$$

$$\frac{\overline{p} \xrightarrow{\varepsilon}_{Pr} \overline{p}'}{\langle \overline{p}, \langle C, \bar{q}\rangle\rangle \xrightarrow{\varepsilon}_{Pr\bowtie\mathsf{DistMem}} \langle \overline{p}', \langle C, \bar{q}\rangle\rangle} \qquad \frac{\langle C, \bar{q}\rangle \xrightarrow{\varepsilon}_{\mathsf{DistMem}} \langle C', \bar{q}'\rangle}{\langle \overline{p}, \langle C, \bar{q}\rangle\rangle \xrightarrow{\varepsilon}_{Pr\bowtie\mathsf{DistMem}} \langle \overline{p}, \langle C', \bar{q}'\rangle\rangle}$$

The above LTS describes a concurrent client program that uses DistMem as the shared memory (recall that DistMem is formally captured by the LTS $\mathsf{Dist}(S_{\mathsf{Mem}}, \mathsf{Wr})$ with states of the form $\langle C, \bar{q}\rangle$; see §3.3). When both Pr and DistMem step with the same observable label $\langle p, l\rangle$ (p performs a memory operation $\mathtt{op}(l)$ with output $\mathtt{out}(l)$), $Pr \bowtie \mathsf{DistMem}$ steps to the new pair of states. When Pr steps with a silent label (some process performs a silent action), only the Pr component of $Pr \bowtie \mathsf{DistMem}$ steps to the new state. When DistMem steps with a silent label (some process receives a message), only the DistMem component of $Pr \bowtie \mathsf{DistMem}$ steps to the new state.

We can now state the reachability problem we study:

Definition 25. A state $\overline{p} \in Pr.\mathtt{Q}$ of a program Pr is *reachable under* DistMem if $\langle \overline{p}, \langle C, \bar{q}\rangle\rangle$ is reachable in $Pr \bowtie \mathsf{DistMem}$ for some C and $\bar{q}$. The *control-state reachability problem for* DistMem is the following decision problem:

Input: a program Pr and a state $\overline{p} \in Pr.\mathtt{Q}$.
Output: is $\overline{p}$ reachable under DistMem?

In the undecidability proof, we use an equivalent formulation of reachability under DistMem that utilizes CBM execution graphs. The connection between a graph and a program trace is established through histories:

Definition 26. The history *induced by a sequence* $tr = \langle\langle p_1, l_1\rangle, \ldots, \langle p_n, l_n\rangle\rangle \in (\mathsf{Ps} \times \mathsf{Lab})^*$, denoted by $\mathsf{H}(tr)$, is given by:

- $\mathsf{H}(tr).\mathtt{E} = \{\langle p_i, i, l_i\rangle \in \mathsf{Evs} \mid 1 \leq i \leq n\}$
- $\mathsf{H}(tr).\mathtt{po} = \{\langle e, e'\rangle \in \mathsf{H}(tr).\mathtt{E} \times \mathsf{H}(tr).\mathtt{E} \mid \mathtt{proc}(e) = \mathtt{proc}(e') \wedge \mathtt{sn}(e) < \mathtt{sn}(e')\}$

Informally, for every $p \in \mathsf{Ps}$, the program order of $\mathsf{H}(tr)$ orders the events performed by p according to the order in which they appear in tr.

Theorem 7. *A state $\overline{p}$ of a program Pr is reachable under* DistMem *(see Def. 25) iff there exists $tr \in \mathsf{otraces}(Pr, \overline{p})$ such that $\mathsf{H}(tr) = \mathsf{H}(G)$ for some* CBM *execution graph G.*

6.1 Undecidability Proof Sketch

Given the above definitions, our undecidability result is the following:

Theorem 8. *The control-state reachability problem for* DistMem *is undecidable.*

Our proof reduces from control-state reachability for two finite-state machines communicating via perfect FIFO channels, one in each direction [11].

Definition 27. Let M be a finite set of *messages*, and L and R be two finite-state LTSs with $\{!m, ?m \mid m \in M\} \subseteq L.\Sigma \cap R.\Sigma$. (The labels $!m$ and $?m$ represent a *send* and a *receive* of message m.) The *(two-point) communicating finite state machine* (CFM, for short) $\mathcal{C}$ induced by L and R is defined by:

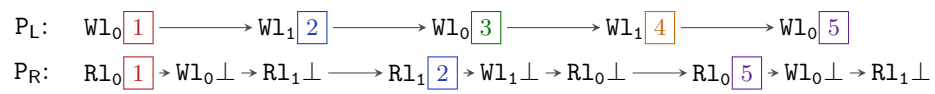
- The states are tuples of the form $\langle\langle q_{\mathsf{L}}, q_{\mathsf{R}}\rangle, w_{\mathsf{LR}}, w_{\mathsf{RL}}\rangle$, where $q_{\mathsf{L}} \in L.\mathtt{Q}$, $q_{\mathsf{R}} \in R.\mathtt{Q}$, and $w_{\mathsf{LR}}, w_{\mathsf{RL}} \in M^*$. Intuitively, w_{LR} maintains the contents of the channel from L to R, that is, the sequence of messages that were sent from L, in the order in which they were sent, but have not been received yet by R (similarly for w_{RL}).
- The alphabet is the disjoint union of $L.\Sigma$ and $R.\Sigma$, given by $(\{\mathsf{L}\} \times L.\Sigma) \cup (\{\mathsf{R}\} \times R.\Sigma)$. For brevity, we write, e.g., $\mathsf{L}!m$ for $\langle \mathsf{L}, !m\rangle$.
- The initial state is given by $\langle\langle L.\mathtt{q}_0, R.\mathtt{q}_0\rangle, \epsilon, \epsilon\rangle$.
- $\langle\langle q_{\mathsf{L}}, q_{\mathsf{R}}\rangle, w_{\mathsf{LR}}, w_{\mathsf{RL}}\rangle \xrightarrow{\sigma}_{\mathcal{C}} \langle\langle q'_{\mathsf{L}}, q'_{\mathsf{R}}\rangle, w'_{\mathsf{LR}}, w'_{\mathsf{RL}}\rangle$ if the following hold:
  - If $\sigma \in L.\Sigma$, then $q_{\mathsf{L}} \xrightarrow{\sigma}_{L} q'_{\mathsf{L}}$ and $q_{\mathsf{R}} = q'_{\mathsf{R}}$ (similarly if $\sigma \in R.\Sigma$).
  - If $\sigma = \mathsf{L}!m$, then $w'_{\mathsf{LR}} = w_{\mathsf{LR}} \cdot m$ and $w'_{\mathsf{RL}} = w_{\mathsf{RL}}$ (similarly if $\sigma = \mathsf{R}!m$).
  - If $\sigma = \mathsf{L}?m$, then $w_{\mathsf{RL}} = m \cdot w'_{\mathsf{RL}}$ and $w'_{\mathsf{LR}} = w_{\mathsf{LR}}$ (similarly if $\sigma = \mathsf{R}?m$).
  - If σ is neither a send nor a receive, then $w'_{\mathsf{LR}} = w_{\mathsf{LR}}$ and $w'_{\mathsf{RL}} = w_{\mathsf{RL}}$.

A *control state* $\langle q_{\mathsf{L}}, q_{\mathsf{R}}\rangle \in L.\mathtt{Q} \times R.\mathtt{Q}$ is *reachable* in $\mathcal{C}$ if $\langle\langle q_{\mathsf{L}}, q_{\mathsf{R}}\rangle, w_{\mathsf{LR}}, w_{\mathsf{RL}}\rangle$ is reachable in $\mathcal{C}$ for some w_{LR} and w_{RL}.

The reduction constructs a concurrent program Pr that simulates runs of $\mathcal{C}$. The program Pr consists of two sequential programs, $Pr(p_{\mathsf{L}})$ and $Pr(p_{\mathsf{R}})$, which mimic the transitions of L and R. To simulate the channels, we use memory locations $\mathsf{Loc} = \{\mathtt{l}_0, \mathtt{l}_1, \mathtt{r}_0, \mathtt{r}_1\}$ with values $\mathsf{Val} = M \uplus \{\bot\}$ consisting of messages and a special marker. The marker $\bot$ also serves as the initial value $\mathsf{v}_{\mathsf{init}}$. The locations $\mathtt{l}_0$ and $\mathtt{l}_1$ are used for the channel from L to R, and $\mathtt{r}_0$ and $\mathtt{r}_1$ are for the channel from R to L. The two variables for each channel are used in alternation, inspired by the reduction from PCP in [1], for making sure that the runs of Pr adhere to the perfect FIFO semantics. Concretely, when L sends m, $Pr(p_{\mathsf{L}})$ alternates between $\mathtt{Wl}_0 m$ and $\mathtt{Wl}_1 m$. (By alternating, we mean using the first for odd-numbered send actions and the second for even-numbered ones.) For receiving a message by R, $Pr(p_{\mathsf{R}})$ alternates between the sequences of operations $\langle \mathtt{Rl}_0, \mathtt{Wl}_0\bot, \mathtt{Rl}_1\bot\rangle$ and $\langle \mathtt{Rl}_1, \mathtt{Wl}_1\bot, \mathtt{Rl}_0\bot\rangle$. (At the program level, a read of $\bot$ from location x is enforced by reading from x, and moving to a "sink" state if the read value is not $\bot$.) The construction is similar for the symmetric actions.

Now, we can show that $\mathcal{C}$ reaches a control state $\langle q_{\mathsf{L}}, q_{\mathsf{R}}\rangle$ iff the constructed program Pr reaches the corresponding state under DistMem. Equivalently, by

Thm. 7, there is an observable trace tr of Pr reaching this state, whose history is induced by a CBM execution graph. To give an example of the main idea in the construction, suppose that L sends messages $\boxed{1},\ldots,\boxed{5}$ to R in this order, and R receives three messages from the channel. Assume, for the sake of contradiction, that R receives $\boxed{1}$ and $\boxed{2}$, but then skips directly to $\boxed{5}$, violating the perfect FIFO semantics (skipping only $\boxed{3}$ or receiving the same message twice are easier to dismiss). Then, the following history is induced by tr:



We show that every execution graph inducing this history is CBM-*inconsistent*. We show it by definition (Def. 21). Indeed, a serialization $s_{\mathsf{P_R}}$ must place $\mathtt{Wl}_0\boxed{5}$ before $\mathtt{Rl}_0\boxed{5}$, and $\mathtt{Wl}_1\boxed{4}$ must be placed before it (to agree with $\mathtt{po}$). To justify the last $\mathtt{Rl}_1\bot$, $\mathtt{Wl}_1\boxed{4}$ needs to go before $\mathtt{Wl}_1\bot$. Similarly, $\mathtt{Wl}_0\boxed{3}$ must now be placed before the first $\mathtt{Wl}_0\bot$. Finally, $\mathtt{Wl}_1\boxed{2}$ has to appear before it, which prevents us from justifying the first $\mathtt{Rl}_1\bot$ event.

7 Relation to Other Criteria and Memory Models

We discuss the relation of CBC (Def. 17) and CBM (Def. 21) to other consistency criteria and weak memory models. We deal with two different notions here: CBC refers to the consistency of histories (Def. 15) w.r.t. a general object specification *Spec*, while CBM concerns the consistency of execution graphs (Def. 19) for the standard memory object. Given two consistency criteria, X for histories and Y for execution graphs, we write $Y \propto X$ if a history H satisfies X w.r.t. $\mathit{Spec}_{\mathsf{Mem}}$ iff $\langle H.\mathtt{E} \uplus \mathsf{Init}, H.\mathtt{po} \uplus (\mathsf{Init} \times H.\mathtt{E}), rf\rangle$ satisfies Y for some reads-from relation rf for $H.\mathtt{E} \uplus \mathsf{Init}$. In particular, Lemma 3 says that CBM $\propto$ CBC. Note that $Y \propto X$ implies that $\mathsf{H}(G)$ satisfies X w.r.t. $\mathit{Spec}_{\mathsf{Mem}}$ whenever G satisfies Y. In addition, for consistency criteria X_1 and X_2, we say that X_1 is stronger than X_2, denoted $X_2 \leq X_1$, if every history satisfying X_1 w.r.t. a specification *Spec* also satisfies X_2 w.r.t. *Spec*. Similarly, for memory models Y_1 and Y_2, we say that Y_1 is stronger than Y_2, denoted $Y_2 \leq Y_1$, if every execution graph satisfying Y_1 also satisfies Y_2. Strict relations (denoted $<$) are also defined as expected.

7.1 Sequential Consistency [21]

A history H is sequentially consistent (SC, for short) w.r.t. an object specification *Spec* if some $s \in \mathsf{seq}(\mathsf{lin}(H.\mathtt{po}, H.\mathtt{E}))$ respects *Spec*.

The corresponding memory model [7], which we denote by $\mathrm{SC}_{\mathsf{Mem}}$, is defined by requiring the existence of a *modification order* mo that totally orders the writes to each location, and requiring that $(G.\mathtt{po} \cup G.\mathtt{rf} \cup mo \cup fr)^+$ is irreflexive, where $fr \triangleq G.\mathtt{rf}^{-1}\,;\, mo$. We have $\mathrm{SC}_{\mathsf{Mem}} \propto \mathrm{SC}$.

It is easy to show that SC is strictly stronger than all other consistency criteria discussed in this paper, and similarly for $\mathrm{SC}_{\mathsf{Mem}}$ and all other memory models. SC disallows all examples in Fig. 1.

7.2 Causal Memory [6]

Following the seminal work [6], an execution graph G satisfies "causal memory" (CM, for short) if for every $p \in \mathsf{Ps}$, some $s_p \in \mathsf{seq}(\mathsf{lin}((G.\mathtt{po} \cup G.\mathtt{rf})^+, G.\mathtt{E}^p \cup G.\mathtt{W}))$ respects $Spec_{\mathsf{Mem}}$. To show that CM is achievable in a distributed settings, the same paper proposed a distributed implementation, which is essentially DistMem with the causal broadcast abstraction implemented from lower level primitives. However, our results show that DistMem actually provides CBM, which is strictly stronger than CM. Indeed, the linearizations required in Def. 21 for CBM can be easily extended to include all write events, and these extensions satisfy the conditions of CM. In addition, the histories in Ex. 7, extended with reads-from relations uniquely determined from the read values, are allowed by CM but not by CBM. (For the left history, the sequences given in Ex. 7 for each process, omitting $\mathtt{Ry1}$ from s_{p_2}, satisfy the conditions of CM.)

We note that the original CM definition stated above has a somewhat strange aspect, since it does not require that reads of process p obtain their values in s_p from the write events specified by $G.\mathtt{rf}$. This has no effect for differentiated execution graphs, where $G.\mathtt{rf}$ is uniquely determined from the induced history. All examples above are differentiated (except for Ex. 10), so we do not exploit this quirk in CM's original formulation. In turn, [27, Figure 3(i)] presents a non-differentiated example, where CM is satisfied without respecting $\mathtt{rf}$, that forms another case of CM but not CBM execution graph. For completeness, we also define a version of CM, which we call $\mathrm{CM}_{\mathtt{rf}}$, that adds the condition that each s_p respects $G.\mathtt{rf}$ (see Def. 20). Then, again, $\mathrm{CM}_{\mathtt{rf}} \leq \mathrm{CBM}$, and since $\mathrm{CM}_{\mathtt{rf}}$ and CM coincide for differentiated executions, Ex. 7 again shows strict inclusion.

7.3 Causal Consistency [27]

In [27], Perrin et al. suggested a generalization of CM from memory to general objects. Their definition is as follows:

Definition 28 ([27]). A history H is *causally consistent* (CC, for short) w.r.t. an object specification *Spec* if there exist a causal order co for H and a sequence s_e for every $e \in H.\mathtt{E}$ such that:

- $s_e \in \mathsf{seq}(\mathsf{lin}(co, co^{-1}(e)))$.
- Some $s'_e \in \mathsf{reeval}(s_e, \mathsf{Evs}|_{\mathtt{proc}(e)})$ respects *Spec*.

We have CC $<$ CBC. Indeed, by taking the per-process linearizations required in CBC and restricting them to the causal past of each event, we obtain the sequences required for CC (and moreover, s_{e_1} is a prefix of s_{e_2} when $\langle e_1, e_2 \rangle \in H.\mathtt{po}$). Example 7 again demonstrates strict inclusion for Mem.

We note that CM $\propto$ CC does not hold in general. First, the example in [27, Figure 3(i)] (where CM is satisfied without respecting $\texttt{rf}$) is an execution graph that satisfies CM, but its induced history does not satisfy CC. Second, even for the $\text{CM}_{\texttt{rf}}$ refinement discussed above, G is $\text{CM}_{\texttt{rf}}$ does imply that $\mathsf{H}(G)$ satisfies CC w.r.t. $\mathit{Spec}_{\mathsf{Mem}}$ but we still do not have $\text{CM}_{\texttt{rf}} \propto \text{CC}$. Nevertheless, if only differentiated histories are considered, then $\text{CM} = \text{CM}_{\texttt{rf}} \propto \text{CC}$ holds.

7.4 Weak Causal Consistency [27] and Weak Release-Acquire [18]

A weakening of CC, called "weak causal consistency" (WCC, for short), is obtained by requiring the reevaluations to preserve one event at a time, rather than all events of each process (replacing the second condition in Def. 28 by requiring that some $s'_e \in \mathsf{reeval}(s_e, \{e\})$ respects Spec). In [27] it was shown that WCC $<$ CC. Thus, we also have WCC $<$ CBC.

The corresponding memory model is WRA [18], defined by requiring that $G.\texttt{hb} \triangleq (G.\texttt{po} \cup G.\texttt{rf})^+$ and $G.\texttt{hb}|_{\mathsf{loc}} ; [\mathsf{W}] ; G.\texttt{hb} ; G.\texttt{rf}^{-1}$ are irreflexive. We have WRA $\propto$ WCC, WRA $<$ CBM, WRA $<$ $\text{CM}_{\texttt{rf}}$, but *not* WRA $\leq$ CM (the example in [27, Figure 3(i)] is CM but not WRA, not even with a different $\texttt{rf}$).

7.5 Causal Convergence [27] and Strong Release-Acquire [19]

A strengthening of WCC, called "causal convergence" (CCv, for short), ensures eventual consistency—all processes converge to return the same outputs once there are no messages in transit. It is obtained by requiring the sequences justifying each event to agree on a certain "global" order of all events (formally, require that there exists a total "arbitration" order extending the causal order and require that all linearizations s_e agree with this order). CBC is incomparable to CCv (for Mem). Example COH in Fig. 1 is allowed by CBC but not by CCv, whereas WWS is allowed by CCv but not by CBC.

The corresponding memory model is SRA [19], defined by requiring the existence of a *modification order* mo that totally orders the writes to each location, and requiring irreflexivity of $(G.\texttt{po} \cup G.\texttt{rf} \cup \mathit{mo})^+$ and $\mathit{mo} ; G.\texttt{hb} ; G.\texttt{rf}^{-1}$ (where $G.\texttt{hb} \triangleq (G.\texttt{po} \cup G.\texttt{rf})^+$). Then, SRA $\propto$ CCv and CBM is incomparable to SRA.

7.6 Release-Acquire [19]

The Release-Acquire memory model (RA) is a well-studied model, obtained as the fragment of the C/C++11 memory model restricted to include only release and acquire atomic accesses [19]. It weakens SRA by requiring that $\mathit{mo}^? ; (G.\texttt{po} \cup G.\texttt{rf})^+$ is irreflexive, instead of $(G.\texttt{po} \cup G.\texttt{rf} \cup \mathit{mo})^+$. RA is placed strictly between WRA and SRA, and can be shown to be incomparable to CBM. Again, COH is allowed by CBM but not by RA, and WWS is allowed by RA but not by CBM.

7.7 Total Store Order [26]

Total store order (TSO) is a a widely implemented memory model in multicore architectures, in particular in x86. A formal model is given in [26], and [19] provides an alternative definition, and shows that SRA $<$ TSO. TSO is incomparable to CBM: Again, COH is allowed by CBM but not by TSO, whereas WWS is allowed by TSO but not by CBM.

7.8 Parallel Snapshot Isolation [12]

Parallel snapshot isolation (PSI) [12,30] is a consistency criterion for transactions in a geo-replicated key-value store, which weakens the classical snapshot isolation (SI) [9] to improve efficiency. The restriction of PSI to transactions composed of a single access can be defined by requiring the existence of a *modification order* mo that totally orders the writes to each location such that $(G.\mathtt{po} \cup G.\mathtt{rf} \cup mo)^+$ and $mo\,;(G.\mathtt{po} \cup G.\mathtt{rf} \cup mo)^+\,;G.\mathtt{rf}^{-1}$ are irreflexive. With this formulation, and since PSI disallows WWS, it easily follows that SRA $<$ PSI. Moreover, we can also show CBM $<$ PSI (again, COH shows strict inclusion).

Finally, a useful relation between every model stronger than WRA (including CBM) and PSI is stated by the following race-freedom guarantee. To state this guarantee, we define control-state reachability under a general declarative memory model (which generalizes the formulation used in Thm. 7).

Definition 29. A state $\overline{p}$ of Pr is *reachable under a memory model* X if there exists $tr \in \mathsf{otraces}(Pr, \overline{p})$ such that $\mathsf{H}(tr) = \mathsf{H}(G)$ for some X-consistent execution graph G. In this case we also say that G is *generated by* Pr.

Theorem 9. *An execution graph G is* write/write-race free *if for every $w_1, w_2 \in G.\mathsf{W}$ with $\mathtt{loc}(w_1) = \mathtt{loc}(w_2)$, we have $w_1 = w_2$, $\langle w_1, w_2\rangle \in (G.\mathtt{po} \cup G.\mathtt{rf})^+$, or $\langle w_2, w_1\rangle \in (G.\mathtt{po} \cup G.\mathtt{rf})^+$. Suppose that every* PSI*-consistent execution graph generated by a program Pr is write/write-race free. Then, every state of Pr that is reachable under a model stronger than* WRA *is also reachable under* PSI.

This theorem demonstrates that for a broad class of programs the distinctions among WRA, RA, SRA, CM, CBM, and PSI are immaterial. Importantly, users of Thm. 9 can establish the premise of the theorem using PSI, the strongest among these models.

8 Conclusion

DistMem is a classical algorithm that has inspired multiple refinements and verification efforts. In this work, we provide the first rigorous analysis of the guarantees DistMem actually provides. Although DistMem was originally presented to demonstrate that causal memory (CM) can be efficiently implemented in a distributed setting [6], we show that DistMem is captured by our CBM model, which is, in fact, strictly stronger than CM. This observation suggests that CM

may be an artifact of an imprecise analysis, and that research on causal memory should instead focus on CBM. We take the first steps in analyzing CBM by studying the complexity of testing consistency and establishing the undecidability of client program verification under DistMem.

More broadly, our work aims to bridge the gap between message-passing distributed systems and weak memory models. Our main contribution is a declarative characterization of a well-known operational message-passing algorithm, along with a comparison to existing memory models and an analysis leveraging techniques from the study of memory models. We hope this work fosters greater communication between these closely related yet largely distinct subfields.

References

1. Abdulla, P.A., Arora, J., Atig, M.F., Krishna, S.: Verification of programs under the release-acquire semantics. In: PLDI. pp. 1117–1132. ACM, New York, NY, USA (2019). https://doi.org/10.1145/3314221.3314649
2. Abdulla, P.A., Atig, M.F., Bouajjani, A., Derevenetc, E., Leonardsson, C., Meyer, R.: On the state reachability problem for concurrent programs under Power. In: NETYS. pp. 47–59. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-67087-0_4
3. Abdulla, P.A., Atig, M.F., Bouajjani, A., Ngo, T.P.: A load-buffer semantics for total store ordering. Logical Methods in Computer Science **Volume 14, Issue 1** (Jan 2018). https://doi.org/10.23638/LMCS-14(1:9)2018
4. Abdulla, P.A., Atig, M.F., Godbole, A., Krishna, S., Vafeiadis, V.: The decidability of verification under PS 2.0. In: ESOP. pp. 1–29. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-72019-3_1
5. Abdulla, P.A., Atig, M.F., Jonsson, B., Ngo, T.P.: Optimal stateless model checking under the release-acquire semantics. Proc. ACM Program. Lang. **2**(OOPSLA) (Oct 2018). https://doi.org/10.1145/3276505
6. Ahamad, M., Neiger, G., Burns, J.E., Kohli, P., Hutto, P.W.: Causal memory: Definitions, implementation, and programming. Distributed Comput. **9**(1), 37–49 (1995). https://doi.org/10.1007/BF01784241
7. Alglave, J., Maranget, L., Tautschnig, M.: Herding cats: modelling, simulation, testing, and data mining for weak memory. ACM Trans. Program. Lang. Syst. **36**(2), 7:1–7:74 (Jul 2014). https://doi.org/10.1145/2627752
8. Atig, M.F., Bouajjani, A., Burckhardt, S., Musuvathi, M.: On the verification problem for weak memory models. In: POPL. p. 7–18. ACM, New York, NY, USA (2010). https://doi.org/10.1145/1706299.1706303
9. Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., O'Neil, P.: A critique of ansi sql isolation levels. In: SIGMOD. p. 1–10. ACM, New York, NY, USA (1995). https://doi.org/10.1145/223784.223785
10. Bouajjani, A., Enea, C., Guerraoui, R., Hamza, J.: On verifying causal consistency. In: POPL. p. 626–638. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3009837.3009888
11. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2), 323–342 (Apr 1983). https://doi.org/10.1145/322374.322380
12. Cerone, A., Bernardi, G., Gotsman, A.: A framework for transactional consistency models with atomic visibility. In: CONCUR. pp. 58–71. Schloss Dagstuhl

– Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2015). https://doi.org/10.4230/LIPIcs.CONCUR.2015.58
13. Chakraborty, S., Krishna, S.N., Mathur, U., Pavlogiannis, A.: How hard is weak-memory testing? Proc. ACM Program. Lang. **8**(POPL) (Jan 2024). https://doi.org/10.1145/3632908
14. Di Giusto, C., Ferré, D., Laversa, L., Lozes, E.: A partial order view of message-passing communication models. Proc. ACM Program. Lang. **7**(POPL) (Jan 2023). https://doi.org/10.1145/3571248
15. Furbach, F., Meyer, R., Schneider, K., Senftleben, M.: Memory-model-aware testing: A unified complexity analysis. ACM Trans. Embed. Comput. Syst. **14**(4) (Sep 2015). https://doi.org/10.1145/2753761
16. Gondelman, L., Gregersen, S.O., Nieto, A., Timany, A., Birkedal, L.: Distributed causal memory: modular specification and verification in higher-order distributed separation logic. Proc. ACM Program. Lang. **5**(POPL) (Jan 2021). https://doi.org/10.1145/3434323
17. Kozen, D.: Lower bounds for natural proof systems. In: SFCS. p. 254–266. IEEE Computer Society, USA (1977). https://doi.org/10.1109/SFCS.1977.16
18. Lahav, O., Boker, U.: What's decidable about causally consistent shared memory? ACM Trans. Program. Lang. Syst. **44**(2) (Apr 2022). https://doi.org/10.1145/3505273
19. Lahav, O., Giannarakis, N., Vafeiadis, V.: Taming release-acquire consistency. In: POPL. p. 649–662. ACM, New York, NY, USA (2016). https://doi.org/10.1145/2837614.2837643
20. Lahav, O., Vafeiadis, V.: Owicki-gries reasoning for weak memory models. In: ICALP. pp. 311–323. Springer (2015). https://doi.org/10.1007/978-3-662-47666-6_25
21. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Comput. **28**(9), 690–691 (Sep 1979). https://doi.org/10.1109/TC.1979.1675439
22. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (Jul 1978). https://doi.org/10.1145/359545.359563
23. Lesani, M., Bell, C.J., Chlipala, A.: Chapar: certified causally consistent distributed key-value stores. In: POPL. p. 357–370. ACM, New York, NY, USA (2016). https://doi.org/10.1145/2837614.2837622
24. Lloyd, W., Freedman, M.J., Kaminsky, M., Andersen, D.G.: Stronger semantics for low-latency geo-replicated storage. In: NSDI. p. 313–328. USENIX Association, USA (2013), https://dl.acm.org/doi/10.5555/2482626.2482657
25. Mostéfaoui, A., Perrin, M., Raynal, M.: Extending causal consistency to any object defined by a sequential specification. Bull. EATCS **125** (2018), http://eatcs.org/beatcs/index.php/beatcs/article/view/545
26. Owens, S., Sarkar, S., Sewell, P.: A better x86 memory model: x86-TSO. In: TPHOLs. pp. 391–407. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03359-9_27
27. Perrin, M., Mostefaoui, A., Jard, C.: Causal consistency: beyond memory. In: PPoPP. ACM, New York, NY, USA (2016). https://doi.org/10.1145/2851141.2851170
28. Raynal, M., Schiper, A., Toueg, S.: The causal ordering abstraction and a simple way to implement it. Inf. Process. Lett. **39**(6), 343–350 (Oct 1991). https://doi.org/10.1016/0020-0190(91)90008-6
29. Schiper, A., Eggli, J., Sandoz, A.: A new algorithm to implement causal ordering. vol. 392, pp. 219–232 (09 1989). https://doi.org/10.1007/3-540-51687-5_45

30. Sovran, Y., Power, R., Aguilera, M.K., Li, J.: Transactional storage for geo-replicated systems. In: SOSP. p. 385–400. ACM, New York, NY, USA (2011). https://doi.org/10.1145/2043556.2043592
31. Tunç, H.C., Abdulla, P.A., Chakraborty, S., Krishna, S., Mathur, U., Pavlogiannis, A.: Optimal reads-from consistency checking for C11-style memory models. Proc. ACM Program. Lang. **7**(PLDI) (Jun 2023). https://doi.org/10.1145/3591251