Abstract

We show how to upgrade a Reliable Broadcast (RB) primitive to Atomic Reliable Broadcast (ARB) by leveraging a synchronous DenyList (DL) object. In a purely asynchronous message-passing model with crashes, ARB is impossible without additional power. The DL supplies this power by enabling round closing and agreement on a set of "+winners" for each round. We present the algorithm, its safety arguments, and discuss liveness and complexity under the assumed synchrony of DL.

Keywords Atomic broadcast, total order broadcast, reliable broadcast, consensus, synchrony, shared object, linearizability.

1 Introduction

Atomic Reliable Broadcast (ARB)—a.k.a. total order broadcast—ensures that all processes deliver the same sequence of messages. In asynchronous message-passing systems with crashes, implementing ARB is impossible without additional assumptions, as it enables consensus. We assume a synchronous DenyList (DL) object and demonstrate how to combine DL with an asynchronous RB to realize ARB.

2 Model

We consider a static set of n processes with known identities, communicating by reliable point-to-point channels, in a complete graph. Messages are uniquely identifiable.

Synchrony. The network is asynchronous. Processes may crash; at most f crashes occur.

Communication. Processes can exchange through a Reliable Broadcast (RB) primitive (defined below) which's invoked with the functions $\mathsf{RB\text{-}cast}(m)$ and $\mathsf{RB\text{-}received}(m)$. There exists a shared object called DenyList (DL) (defined below) that is interfaced with the functions $\mathsf{APPEND}(x)$, $\mathsf{PROVE}(x)$ and $\mathsf{READ}()$.

Notation. Let Π be the finite set of process identifiers and let $n \triangleq |\Pi|$. Two authorization subsets are $\Pi_M \subseteq \Pi$ (processes allowed to issue APPEND) and $\Pi_V \subseteq \Pi$ (processes allowed to issue PROVE). Indices $i, j \in \Pi$ refer to processes, and p_i denotes the process with identifier i. Let $\mathcal{M}$ denote the universe of uniquely identifiable messages, with $m \in \mathcal{M}$. Let $\mathcal{R} \subseteq \mathbb{N}$ be the set of round identifiers; we write $r \in \mathcal{R}$ for a round. We use the precedence relation $\prec$ for the DL linearization: $x \prec y$ means that operation x appears strictly before y in the linearized history of DL. For any finite set $A \subseteq \mathcal{M}$, $\mathsf{ordered}(A)$ returns a deterministic total order over A (e.g., lexicographic order on $(\mathit{senderId}, \mathit{messageId})$ or on message hashes). For any round $r \in \mathcal{R}$, define $\mathsf{Winners}_r \triangleq \{\, j \in \Pi \mid (j, \mathsf{prove}(r)) \prec \mathsf{APPEND}(r) \,\}$, i.e., the set of processes whose $\mathsf{PROVE}(r)$ appears before the first $\mathsf{APPEND}(r)$ in the DL linearization.

3 Primitives

3.1 Reliable Broadcast (RB)

RB provides the following properties in the model.

- **Integrity**: Every message received was previously sent. $\forall p_i : \ \mathsf{RB\text{-}received}_i(m) \Rightarrow \exists p_j : \mathsf{RB\text{-}cast}_j(m)$.
- **No-duplicates**: No message is received more than once at any process.
- **Validity**: If a correct process broadcasts m, every correct process eventually receives m.

3.2 DenyList (DL)

The DL is a *shared, append-only* object that records attestations about opaque application-level tokens. It exposes the following operations:

- $\mathsf{APPEND}(x)$
- $\mathsf{PROVE}(x)$: issue an attestation for token x; this operation is *valid* (return true) only if no $\mathsf{APPEND}(x)$ occurs earlier in the DL linearization. Otherwise, it is invalid (return false).
- READ(): return a (permutation of the) valid operations observed so far; subsequent reads are monotone (contain supersets of previously observed valid operations).

Validity. $\mathsf{APPEND}(x)$ is valid iff the issuer is authorized (in Π_M) and x belongs to the application-defined domain S. $\mathsf{PROVE}(x)$ is valid iff the issuer is authorized (in Π_V) and there is no earlier $\mathsf{APPEND}(x)$ in the DL linearization.

Progress. If a correct process invokes $\mathsf{APPEND}(x)$, then eventually all correct processes will be unable to issue a valid $\mathsf{PROVE}(x)$, and READ at all correct processes will (eventually) reflect that $\mathsf{APPEND}(x)$ has been recorded.

Termination. Every operation invoked by a correct process eventually returns.

Interface and Semantics. The DL provides a single global linearization of operations consistent with each process's program order. READ is prefix-monotone; concurrent updates become visible to all correct processes within bounded time (by synchrony). Duplicate requests may be idempotently coalesced by the implementation.

4 Target Abstraction: Atomic Reliable Broadcast (ARB)

Processes export $\mathsf{AB\text{-}broadcast}(m)$ and $\mathsf{AB\text{-}deliver}(m)$. ARB requires total order:

$$\forall m_1, m_2,\ \forall p_i, p_j : \ \mathsf{AB\text{-}deliver}_i(m_1) < \mathsf{AB\text{-}deliver}_i(m_2) \Rightarrow \mathsf{AB\text{-}deliver}_j(m_1) < \mathsf{AB\text{-}deliver}_j(m_2),$$

plus Integrity/No-duplicates/Validity (inherited from RB and the construction).

5 Algorithm

Definition 1 (Closed round)**.** Given a DL linearization H, a round $r \in \mathcal{R}$ is *closed* in H iff H contains an operation $\mathsf{APPEND}(r)$. Equivalently, there exists a time after which every $\mathsf{PROVE}(r)$ is invalid in H.

Definition 2 (First APPEND)**.** Given a DL linearization H, for any closed round $r \in \mathcal{R}$, we denote by $\mathsf{APPEND}^\star(r)$ the earliest $\mathsf{APPEND}(r)$ in H.

5.1 Variables

Each process p_i maintains:

received $\leftarrow \emptyset$ ▷ Messages received via RB but not yet delivered
delivered $\leftarrow \emptyset$ ▷ Messages already delivered
prop$[r][j] \leftarrow \bot, \ \forall r, j$ ▷ Proposal from process j for round r

DenyList. The DL is initialized empty. We assume $\Pi_M = \Pi_V = \Pi$ (all processes can invoke APPENDand PROVE).

5.2 Handlers and Procedures

Algorithm A RB handler (at process p_i)

function RBRECEIVED(S, r, j)
  received $\leftarrow$ received $\cup \{S\}$
  prop$[r][j] \leftarrow S$ ▷ Record sender j's proposal S for round r
end function

Algorithm B AB-broadcast(m) (at process p_i)

function ABBROADCAST(m)
  $P \leftarrow$ READ() ▷ Fetch latest DL state to learn recent PROVE operations
  $r_{max} \leftarrow \max(\{r' : \exists j, \ (j, \mathsf{PROVE}(r')) \in P\})$ ▷ Pick current open round
  $S \leftarrow$ (received $\setminus$ delivered) $\cup \{m\}$ ▷ Propose all pending messages plus the new m

  for each $r \in \{r_{max}, r_{max}+1, \cdots\}$ **do**
    RB-cast(S, r, i); PROVE(r); APPEND(r);
    $P \leftarrow$ READ() ▷ Refresh local view of DL
    if $(((i, \mathsf{prove}(r)) \in P \ \vee \ (\exists j, r' : (j, \mathsf{prove}(r')) \in P \wedge \ m \in \mathsf{prop}[r'][j])))$ **then**
      break ▷ Exit loop once m is included in some closed round
    end if
  end for
end function

Algorithm C AB-deliver() at process p_i

```
next ← 0                                                   ▷ Next round to deliver
to_deliver ← ∅                                             ▷ Queue of messages ready to be delivered

function ABdeliver
    if to_deliver = ∅ then                                 ▷ If no message is ready to deliver, try to fetch the next round
        P ← READ()                                         ▷ Fetch latest DL state to learn recent PROVE operations
        if ∀j : (j, prove(next)) ∉ P then                  ▷ Check if some process proved round next
            return ⊥                                       ▷ Round next is still open
        end if
        APPEND(next); P ← READ()                           ▷ Close round next if not already closed
        W_next ← {j : (j, prove(next)) ∈ P}                ▷ Compute winners of round next
        if ∃j ∈ W_next, prop[next][j] = ⊥ then             ▷ Check if we have all winners' proposals
            return ⊥                                       ▷ Some winner's proposal for round next is still missing
        end if
        M_next ← ⋃_{j∈W_next} prop[next][j]                ▷ Compute the agreed proposal for round next
        for each m ∈ ordered(M_next) do                    ▷ Enqueue messages in deterministic order
            if m ∉ delivered then
                to_deliver.push(m)                         ▷ Append m to the delivery queue
            end if
        end for
        next ← next + 1                                    ▷ Advance to the next round
    end if
    m ← to_deliver.pop()
    delivered ← delivered ∪ {m}
    return m
end function
```

6 Correctness

Lemma 1 (Stable round closure)**.** *If a round r is closed, then there exists a linearization point t_0 of $\mathsf{APPEND}(r)$ in the DL, and from that point on, no $\mathsf{PROVE}(r)$ can be valid. Once closed, a round never becomes open again.*

Proof. By definition of closed round, some $\mathsf{APPEND}(r)$ occurs in the linearization H.
H is a total order of operations, the set of $\mathsf{APPEND}(r)$ operations is totally ordered, and hence there exists a smallest $\mathsf{APPEND}(r)$ in H. We denote this operation $\mathsf{APPEND}^\star(r)$ and t_0 its linearization point.
By the validity property of DL, a $\mathsf{PROVE}(r)$ is valid iff $\mathsf{PROVE}(r) \prec \mathsf{APPEND}^\star(r)$. Thus, after t_0, no $\mathsf{PROVE}(r)$ can be valid.
H is a immutable grow-only history, and hence once closed, a round never becomes open again.
Hence there exists a linearization point t_0 of $\mathsf{APPEND}(r)$ in the DL, and from that point on, no $\mathsf{PROVE}(r)$ can be valid and the closure is stable. □

Lemma 2 (Across rounds)**.** *If there exists a r such that r is closed, $\forall r'$ such that $r' < r$, r' is also closed.*

Proof. Base. For a closed round $k = 0$, the set $\{k' \in \mathcal{R},\ k' < k\}$ is empty, hence the lemma is true.
Induction. Assume the lemma is true for round $k \geq 0$, we prove it for round $k + 1$.

Assume $k+1$ is closed and let $\mathsf{APPEND}^\star(k+1)$ be the earliest $\mathsf{APPEND}(k+1)$ in the DL linearization H. By Lemma 1, after an $\mathsf{APPEND}(k)$ is in H, any later $\mathsf{PROVE}(k)$ is rejected also, a process's program order is preserved in H.

There are two possibilities for where $\mathsf{APPEND}^\star(k+1)$ is invoked.

Case (B6). Some process $p^\star$ executes the loop (lines B5–B11) and invokes $\mathsf{APPEND}^\star(k+1)$ at line B6. Immediately before line B6, line B5 sets $r \leftarrow r+1$, so the previous loop iteration (if any) targeted k. We consider two sub-cases.

(i) $p^\star$ is not in its first loop iteration. In the previous iteration, $p^\star$ executed $\mathsf{PROVE}^\star(k)$ at B6, followed (in program order) by $\mathsf{APPEND}^\star(k)$. If round k wasn't closed when $p^\star$ execute $\mathsf{PROVE}^\star(k)$ at B9, then the condition at B8 would be true hence the tuple $(p^\star, \mathsf{prove}(k))$ should be visible in P which implies that $p^\star$ should leave the loop at round k, contradicting the assumption that $p^\star$ is now executing another iteration. Since the tuple is not visible, the $\mathsf{PROVE}^\star(k)$ was rejected by the DL which implies by definition an $\mathsf{APPEND}(k)$ already exists in H. Thus in this case k is closed.

(ii) $p^\star$ is in its first loop iteration. To compute the value r_{max}, $p^\star$ must have observed one or many $(\_, \mathsf{prove}(k+1))$ in P at B2/B3, issued by some processes (possibly different from $p^\star$). Let's call p_1 the issuer of the first $\mathsf{PROVE}(k+1)$ in the linearization H.
When p_1 executed $P \leftarrow \mathsf{READ}()$ at B2 and compute r_{max} at B3, he observed no tuple $(j, \mathsf{prove}(k+1))$ in P because he's the issuer of the first one. So when p_1 executed the loop (B5–B11), he run it for the round k, didn't seen any $(1, \mathsf{prove}(k))$ in P at B8, and then executed the first $\mathsf{PROVE}(k+1)$ at B6 in a second iteration.
If round k wasn't closed when p_1 execute $\mathsf{PROVE}(k)$ at B6, then the condition at B8 should be true which implies that p_1 sould leave the loop at round k, contradicting the assumption that p_1 is now executing $\mathsf{PROVE}(r+1)$. In this case k is closed.

Case (C9). Some process invokes $\mathsf{APPEND}(k+1)$ at C9. Line C9 is guarded by the presence of $\mathsf{PROVE}(\mathit{next})$ in P with $\mathit{next} = k+1$ (C6). Moreover, the local pointer *next* grow by increment of 1 and only advances after finishing the current round (C20), so if a process can reach $\mathit{next} = k+1$ it implies that he has completed round k, which includes closing k at C9 when $\mathsf{PROVE}(k)$ is observed. Hence $\mathsf{APPEND}^\star(k+1)$ implies a prior $\mathsf{APPEND}(k)$ in H, so k is closed.

In all cases, $k+1$ closed implie k closed. By induction on k, if the lemme is true for a closed k then it is true for a closed $k+1$. Therefore, the lemma is true for all closed rounds r. □

Definition 3 (Winner Invariant)**.** Let take a closed round r (which implies by Definition 1 that some $\mathsf{APPEND}(r)$ occurs in the DL linearization H), there exist a unique and defined set of winners such as

$$\mathsf{Winners}_r = \{j : (j, \mathsf{prove}(r)) \prec \mathsf{APPEND}^\star(r)\}$$

with $\mathsf{APPEND}^\star(r)$ being the earliest $\mathsf{APPEND}(r)$ in the DL linearization.

Lemma 3 (Invariant view of closure)**.** *For any closed round r, all correct processes eventually observe the same set of valid tuples $(\_, \mathsf{prove}(r))$ in their DL view.*

Proof. Fix a round r such that some $\mathsf{APPEND}(r)$ occurs; hence r is *closed*. By Lemma 1, there exists a unique earliest $\mathsf{APPEND}(r)$ in the DL linearization, denoted $\mathsf{APPEND}^\star(r)$.

Consider any correct process p that invokes $\mathsf{READ}()$ after $\mathsf{APPEND}^\star(r)$ in the DL linearization. Since $\mathsf{APPEND}^\star(r)$ invalidates all subsequent $\mathsf{PROVE}(r)$, the set of valid tuples $(\_, \mathsf{prove}(r))$ observed by any correct process after $\mathsf{APPEND}^\star(r)$ is fixed and identical across all correct processes.

Therefore, for any closed round r, all correct processes eventually observe the same set of valid tuples $(\_, \mathsf{prove}(r))$ in their DL view. □

Lemma 4 (Well-defined winners)**.** *In any execution, when a process computes W_r, r is closed.*

Proof. Fix a process p that reaches line C10 (computing W_{next}). Immediately before C10, line C9 executes

$$\mathsf{APPEND}(next);\ P \leftarrow \mathsf{READ}()$$

Line C9 is guarded by the condition at line C6, which ☐

Lemma 5 (View-Invariant Winners)**.** *For any closed round r, there exists a unique set*

$$\mathsf{Winners}_r = \{j : (j, \mathsf{prove}(r)) \prec \mathsf{APPEND}^\star(r)\}$$

with $\mathsf{APPEND}^\star(r)$ being the earliest $\mathsf{APPEND}(r)$ in the DL linearization.
Such that for any correct process p that computes W_r, $W_r = \mathsf{Winners}_r$.

Proof. Fix a round r such that some $\mathsf{APPEND}(r)$ occurs; hence r is *closed*. By Lemma 1, there exists a unique earliest $\mathsf{APPEND}(r)$ in the DL linearization, denoted $\mathsf{APPEND}^\star(r)$.

Consider any correct process p that computes W_r at line C10. By Lemma 4, r is closed when p computes W_r. By Lemma 1, the linearization point of $\mathsf{APPEND}^\star(r)$ precedes that of the subsequent $\mathsf{READ}()$ at C9, therefore the P returned by this $\mathsf{READ}()$ is posterior to $\mathsf{APPEND}^\star(r)$ in the DL linearization.

Hence, at the moment C10 computes

$$W_r \ \leftarrow\ \{\, j : (j, \mathsf{prove}(r)) \in P \,\},$$

the view P is posterior to $\mathsf{APPEND}^\star(r)$, so

$$W_r = \{\, j : (j, \mathsf{prove}(r)) \prec \mathsf{APPEND}^\star(r) \,\} = \mathsf{Winners}_r.$$

Since this argument holds for any correct process computing W_r, all correct processes compute the same winner set $\mathsf{Winners}_r$ for closed round r. ☐

Lemma 6 (No APPEND without PROVE)**.** *If some correct process invokes $\mathsf{APPEND}(r)$, then some correct process must have previously invoked $\mathsf{PROVE}(r)$.*

Proof. Consider the round r such that some correct process invokes $\mathsf{APPEND}(r)$. There are two possible cases

Case (B6). There exists a process $p^\star$ who's the issuer of the earliest $\mathsf{APPEND}^\star(r)$ in the DL linearization H. By program order, $p^\star$ must have previously invoked $\mathsf{PROVE}(r)$ before invoking $\mathsf{APPEND}^\star(r)$ at B6. In this case, there is at least one $\mathsf{PROVE}(r)$ valid in H issued by a correct process before $\mathsf{APPEND}^\star(r)$.

Case (C9). There exist a process $p^\star$ invokes $\mathsf{APPEND}^\star(r)$ at C9. Line C9 is guarded by the condition at C6, which ensures that p observed some $(\_, \mathsf{prove}(r))$ in P. In this case, there is at least one $\mathsf{PROVE}(r)$ valid in H issued by some process before $\mathsf{APPEND}^\star(r)$.

In both cases, if some correct process invokes $\mathsf{APPEND}(r)$, then some correct process must have previously invoked $\mathsf{PROVE}(r)$. ☐

Lemma 7 (No empty winners)**.** *For any correct process that computes W_r, $W_r \neq \emptyset$.*

Proof. By Lemma 4, any correct process that computes W_r implies that round r is closed. By Lemma 1, there exists a unique earliest $\mathsf{APPEND}(r)$ in the DL linearization, denoted $\mathsf{APPEND}^\star(r)$.

By Lemma 5, any correct process computing W_r obtains

$$W_r = \mathsf{Winners}_r = \{\, j : (j, \mathsf{prove}(r)) \prec \mathsf{APPEND}^\star(r) \,\}.$$

So

$$W_r \neq \emptyset \implies \exists j : (j, \mathsf{prove}(r)) \prec \mathsf{APPEND}^\star(r).$$

Consider any correct process p that computes W_k at line C10. We show that k is closed when p executed the READ operation at C9 by Lemma 4. This implies that some process must have invoked $\mathsf{APPEND}(k)$, which by Lemma 6 implies that some correct process must have previously invoked $\mathsf{PROVE}(k)$. Hence there exists at least one j such that $(j, \mathsf{prove}(k)) \prec \mathsf{APPEND}^\star(k)$, so $W_k \neq \emptyset$. ☐

Lemma 8 (Eventual proposal closure)**.** *For any correct process p_i that computes M_r for round r, there exist no j such that $j \in \mathsf{Winners}_r$ and $\mathsf{prop}^{(i)}[r][j] = \bot$.*

Proof. Fix a correct process p_i that computes M_r at line C14. By Lemma 4, r is closed when p_i executed the READ operation at C9. By Lemma 5, p_i computes the unique winner set $\mathsf{Winners}_r$.

By Lemma 7, $\mathsf{Winners}_r \neq \emptyset$. Consider any $j \in \mathsf{Winners}_r$. Since j is a winner, by definition of $\mathsf{Winners}_r$, j must have invoked a valid $\mathsf{PROVE}(r)$ before the earliest $\mathsf{APPEND}(r)$ in the DL linearization. Since j is correct and by program order, if he invoked a valid $\mathsf{PROVE}(r)$ he must have invoked $\mathsf{RB\text{-}cast}(S^{(j)}, r, j)$ directly before. By RB *Validity*, every correct process eventually receives j's RB message for round r, which sets $\mathsf{prop}[r][j]$ to a non-$\bot$ value (A3). The instruction C14 where p_i computes M_r is guarded by the condition at C11, which ensures that p_i has received all RB messages from every winner $j \in \mathsf{Winners}_r$. Hence, when p_i computes $M_r = \bigcup_{j \in \mathsf{Winners}_r} \mathsf{prop}[r][j]$, we have $\mathsf{prop}[r][j] \neq \bot$ for all $j \in \mathsf{Winners}_r$. ☐

Lemma 9 (Unique proposal per sender per round)**.** *For any round r and any process j, j invokes at most one $\mathsf{RB\text{-}cast}(S, r, j)$.*

Proof. By program order, any process j invokes $\mathsf{RB\text{-}cast}(S, r, j)$ at line B6 must be in the loop (B5–B11). No matter the number of iterations of the loop, line B5 always uses the current value of r which is incremented by 1 at line B5. Hence, in any execution, any process j invokes $\mathsf{RB\text{-}cast}(S, r, j)$ at most once for any round r. ☐

Lemma 10 (Proposal convergence)**.** *For any closed round r, after all RB messages from $\mathsf{Winners}_r$ are received, every correct process computes the same $M_r = \bigcup_{j \in \mathsf{Winners}_r} \mathsf{prop}[r][j]$.*

Proof. Fix a closed round r. By Lemma 5, any correct process computing W_r obtains the same winner set $\mathsf{Winners}_r$. By Lemma 8, every correct process that computes M_r has received all RB messages from every winner $j \in \mathsf{Winners}_r$, so $\mathsf{prop}[r][j]$ is non-$\bot$ for all $j \in \mathsf{Winners}_r$. By Lemma 9, each winner j invokes at most one $\mathsf{RB\text{-}cast}(S^{(j)}, r, j)$, so $\mathsf{prop}[r][j] = S^{(j)}$ is uniquely defined. Hence, every correct process computes the same

$$M_r = \bigcup_{j \in \mathsf{Winners}_r} \mathsf{prop}[r][j] = \bigcup_{j \in \mathsf{Winners}_r} S^{(j)}.$$

☐

Lemma 11 (Broadcast Termination)**.** *If a correct process p invokes $\mathsf{AB\text{-}broadcast}(m)$, then p eventually quit the function and returns.*

Proof. Fix a correct process p that invokes $\mathsf{AB\text{-}broadcast}(m)$. The lemma is true iff $(i, \mathsf{prove}(r)) \in P$ or $\exists j, r' : (j, \mathsf{prove}(r')) \in P \wedge m \in \mathsf{prop}[r'][j]$ (like guarded at B8).

- **Case 1:** there exists a round r such that p invokes $\mathsf{PROVE}(r)$ and this $\mathsf{PROVE}(r)$ is valid. Hence eventually $(i, \mathsf{prove}(r)) \in P$ and p exits the loop at B8.
- **Case 2:** there exists no round r such that p invokes $\mathsf{PROVE}(r)$ and this $\mathsf{PROVE}(r)$ is valid. In this case p invokes infinitely many $\mathsf{RB\text{-}cast}(S, r, i)$ at B6 with $m \in S$ (line B4).
  The assumption that no $\mathsf{PROVE}(r)$ invoked by p is valid implies by Lemma 7 that for every possible round r there at least one winner. Because there is an infinite number of rounds, and a finite number of processes, there exists at least one correct process j that invokes infinitely many valid $\mathsf{PROVE}(r)$ and by extension infinitely many $\mathsf{AB\text{-}broadcast}(\_)$. By RB *Validity*, j eventually receives p 's RB messages. Let call t the time when j receives p 's RB message
  For the first invocation of $\mathsf{AB\text{-}broadcast}(m)$ by j after t, j must include m in his proposal S at B4 for round r' (because m is pending in j 's $\mathsf{received} \setminus \mathsf{delivered}$). Because $j \in \mathsf{Winners}_{r'}$ and by RB *Validity*, every correct process eventually receives j 's RB message for round r', including p. When p receives j 's RB message, $\mathsf{prop}[r'][j]$ is set to a non-$\perp$ value (A3) which includes m. Hence eventually $\exists j, r' : (j, \mathsf{prove}(r')) \in P \wedge m \in \mathsf{prop}[r'][j]$ and p exits the loop at B8.

The first case explicit the case where p is a winner and also covers the condition $(i, \mathsf{prove}(r)) \in P$. And in the second case, we show that if the first condition is never satisfied, the second one will eventually be satisfied. Hence either the first or the second condition will eventually be satisfied, and p eventually exits the loop and returns from $\mathsf{AB\text{-}broadcast}(m)$. ☐

Lemma 12 (Inclusion)**.** *If some correct process invokes $\mathsf{AB\text{-}broadcast}(m)$, then there exist a (closed) round r and a corrcet process $j \in \mathsf{Winners}_r$ such that j invokes*

$$\mathsf{RB\text{-}cast}(S, r, j) \quad \textit{with} \quad m \in S.$$

Proof. Fix a correct process p that invokes $\mathsf{AB\text{-}broadcast}(m)$. By Lemma 11, p eventually exits the loop (B5–B11) at some round r. There are two possible cases.

Case 1: p exits the loop because $(i, \mathsf{prove}(r)) \in P$. In this case, p is a winner in round r by definition of $\mathsf{Winners}_r$. By program order, p must have invoked $\mathsf{RB\text{-}cast}(S, r, i)$ at B6 before invoking $\mathsf{PROVE}(r)$ at B7. By line B4, $m \in S$. Hence there exist a closed round r and a correct process $j = i \in \mathsf{Winners}_r$ such that j invokes $\mathsf{RB\text{-}cast}(S, r, j)$ with $m \in S$.

Case 2: p exits the loop because $\exists j, r' : (j, \mathsf{prove}(r')) \in P \wedge m \in \mathsf{prop}[r'][j]$. In this case, by Lemma 5, any correct process computing $W_{r'}$ obtains the unique winner set $\mathsf{Winners}_{r'}$, so $j \in \mathsf{Winners}_{r'}$. By program order, j must have invoked $\mathsf{RB\text{-}cast}(S, r', j)$ at B6 before invoking $\mathsf{PROVE}(r')$ at B7. By line B4, $m \in S$. Hence there exist a closed round $r' = r$ and a correct process $j \in \mathsf{Winners}_{r'}$ such that j invokes $\mathsf{RB\text{-}cast}(S, r', j)$ with $m \in S$.

In both cases, if some correct process invokes $\mathsf{AB\text{-}broadcast}(m)$, then there exist a (closed) round r and a correct process $j \in \mathsf{Winners}_r$ such that j invokes

$$\mathsf{RB\text{-}cast}(S, r, j) \quad \text{with} \quad m \in S.$$

☐

Lemma 13 (Validity)**.** *If a correct process p invokes $\textsf{AB-broadcast}(m)$, then every correct process who eventually invokes $\textsf{AB-deliver}()$ delivers m.*

Proof. Fix a correct process p that invokes $\textsf{AB-broadcast}(m)$. By Lemma 12, there exist a closed round r and a correct process $j \in \textsf{Winners}_r$ such that j invokes

$$\textsf{RB-cast}(S, r, j) \quad \text{with} \quad m \in S.$$

Consider any correct process q that eventually invokes $\textsf{AB-deliver}()$. By Lemma 8, when q computes M_r at line C14, $\textsf{prop}[r][j]$ is non-$\perp$ because $j \in \textsf{Winners}_r$. By Lemma 9, j invokes at most one $\textsf{RB-cast}(S, r, j)$, so $\textsf{prop}[r][j] = S$ is uniquely defined. Hence, when q computes

$$M_r = \bigcup_{k \in \textsf{Winners}_r} \textsf{prop}[r][k],$$

we have $m \in \textsf{prop}[r][j] = S$, so $m \in M_r$. By line C16–C18, m is enqueued into *to_deliver* unless it has already been delivered. Therefore, when q eventually invokes $\textsf{AB-deliver}()$, it eventually delivers m. $\square$

Lemma 14 (Total Order)**.** *For any two messages m_1 and m_2 broadcast by correct processes such that m_1 is broadcast before m_2, any correct process p_j that delivers both m_1 and m_2 delivers m_1 before m_2.*

Proof. Fix two messages m_1 and m_2 broadcast by correct processes. Consider any correct process p_i that delivers m_1 before m_2. By program order, p_i must have invoked $\textsf{AB-broadcast}(m_1)$ before $\textsf{AB-broadcast}(m_2)$. Let r_1 and r_2 be the rounds at which p_i exits the loop (B5–B11) when invoking $\textsf{AB-broadcast}(m_1)$ and $\textsf{AB-broadcast}(m_2)$ respectively. By program order, $r_1 < r_2$.

Consider any correct process p_j that delivers both m_1 and m_2. By Lemma 13, there exist closed rounds r'_1 and r'_2 and correct processes $k_1 \in \textsf{Winners}_{r'_1}$ and $k_2 \in \textsf{Winners}_{r'_2}$ such that

$$\textsf{RB-cast}(S_1, r'_1, k_1) \quad \text{with} \quad m_1 \in S_1,$$

$$\textsf{RB-cast}(S_2, r'_2, k_2) \quad \text{with} \quad m_2 \in S_2.$$

By program order, p_i must have waited for the loop (B5–B11) to exit at round r_1 before invoking $\textsf{AB-broadcast}(m_2)$. Hence, $r_1 \le r'_1 < r_2 \le r'_2$, so $r'_1 < r'_2$. Because p_j delivers messages in round order (C5–C20), p_j delivers all messages from round r'_1 (including m_1) before delivering any message from round r'_2 (including m_2). Therefore, p_j delivers m_1 before m_2. $\square$

Theorem 15 (ARB)**.** *Under the assumed DL synchrony and RB reliability, the algorithm implements Atomic Reliable Broadcast.*

References