Appears as Technical Memo MIT/LCS/TM-590, MIT Laboratory for Computer Science, June 1999

A Correctness Proof for a Practical Byzantine-Fault-Tolerant Replication Algorithm

Miguel Castro and Barbara Liskov
Laboratory for Computer Science,
Massachusetts Institute of Technology,
545 Technology Square, Cambridge, MA 02139
{castro,liskov}@lcs.mit.edu

1 Introduction

We have developed a *practical* algorithm for state-machine replication [7, 11] that tolerates Byzantine faults. The algorithm is described in [4]. It offers a strong safety property — it implements a linearizable [5] object such that all operations invoked on the object execute atomically despite Byzantine failures and concurrency. Unlike previous algorithms [11, 10, 6], ours works correctly in asynchronous systems like the Internet, and it incorporates important optimizations that enable it to outperform previous systems by more than an order of magnitude [4].

Since Byzantine-fault-tolerant algorithms are rather subtle, it is important to reason about them formally. This paper presents a formal specification for the unoptimized version of our algorithm presented in Section 4 of [4] and proves its safety (but not its liveness.) The specification uses the I/O automaton formalism of Tuttle an Lynch [8] and the proof is based on invariant assertions and simulation relations. [1]

The specification and proof presented in this paper have some interesting, novel properties that are independent of our algorithm. First, we use an I/O automaton to formalize the correct behavior of our Byzantine-fault-tolerant object implementation. This technique has been used for benign failures [8] but we believe we are the first to use it for Byzantine faults. The advantage of using an I/O automaton to formalize the correct behavior is that it enables the use of state-based proof techniques like simulation relations. These techniques are more stylized than trace-based proof techniques — they are more convincing and they are amenable to machine verification. Second, our formalization accounts for Byzantine faults of both replicas and clients. A trace-based formalization of linearizability in the presence of Byzantine-faulty clients [9] has been proposed recently. Our formalization has the advantage that it enables the use of simulation relations. And it differs from the one in [9] because it makes authentication and access control explicit in the formalization. Revocable access control is a powerful defense against Byzantine faulty clients.

This research was supported in part by DARPA under contract F30602-98-1-0237 monitored by the Air Force Research Laboratory, and in part by NEC.

[1] The paper assumes the reader is familiar with I/O automata, invariant assertions, and simulation relations. Lynch's book [8] provides a good description of the formalism and the two proof techniques.

Third, we structure our proof such that our assumptions about authenticated communication are isolated in a small number of invariants and in the proofs of a small number of simulation steps. This leads to a simpler proof.

The paper is organized as follows. Section 2 presents the high-level model for the system and our assumptions. Section 3 formalizes the correctness condition for our algorithm using a simple I/O automaton S as a specification of correct behavior. Section 4 defines the automata that compose our replicated system implementation. But it does not attempt to explain the algorithm. The reader is referred to [4] for a natural language description of the algorithm that should be easier to understand. This section also proves the safety of the algorithm by using invariant assertions and simulation relations to show that it implements S. The automata defined in Section 4 implement a simplified version of the algorithm that does not garbage collect information. Section 5 defines a version of the algorithm with garbage collection and proves its safety by using a simulation relation to show that it implements the simplified version of the algorithm in 4.

2 Model

The goal of our algorithm is to provide a Byzantine-fault-tolerant implementation of an *atomic object* [8] for a given variable of some type $\mathcal{T}$. Our atomic object implementation uses replication to enable concurrent sharing of the variable by many *clients* in a distributed system. It ensures linearizability [5] — all operations invoked on the variable execute atomically despite Byzantine failures and concurrency. We start by defining the variable type $\mathcal{T}$ and then describe the architecture of the atomic object implementation.

Variables of type $\mathcal{T}$ have a value in a set $\mathcal{V}$, which is initially equal to v_o. Their behavior is defined by the function:

$$g : \mathcal{C} \times \mathcal{O} \times \mathcal{V} \rightarrow \mathcal{O}' \times \mathcal{V}$$

The arguments to the function are a client identifier in $\mathcal{C}$, an operation in a set $\mathcal{O}$ (which encodes an operation identifier and any arguments to that operation) and an initial value. These arguments are mapped by g to the result of the operation in $\mathcal{O}'$ and a new value for the variable. We require g to be total. This can be achieved in practice by having g map all pairs with an invalid operation to a pair with an error result and the argument value.

The client identifier is included explicitly as an argument to g to make it clear that g can return different results for different clients. In particular, g can perform *access control*; if the client is not allowed to perform the argument operation, g can return a special *no-access* error and leave the variable's value unmodified. Additionally, access control can depend on the state of the service thereby allowing atomic access revocations. Access control with revocable access is an important defense against Byzantine-faulty clients.

We model the atomic object implementation and its clients as a set of I/O automata [8]. Each client has a unique identifier c in a set $\mathcal{C}$ and is modeled by a client automaton C_c. The composition of all clients is denoted by C. The atomic object automaton A is the composition of three types of automata: proxy, multicast channel, and replica. Figure 1 shows the architecture of the system and Figure 2 presents the external interface of A.

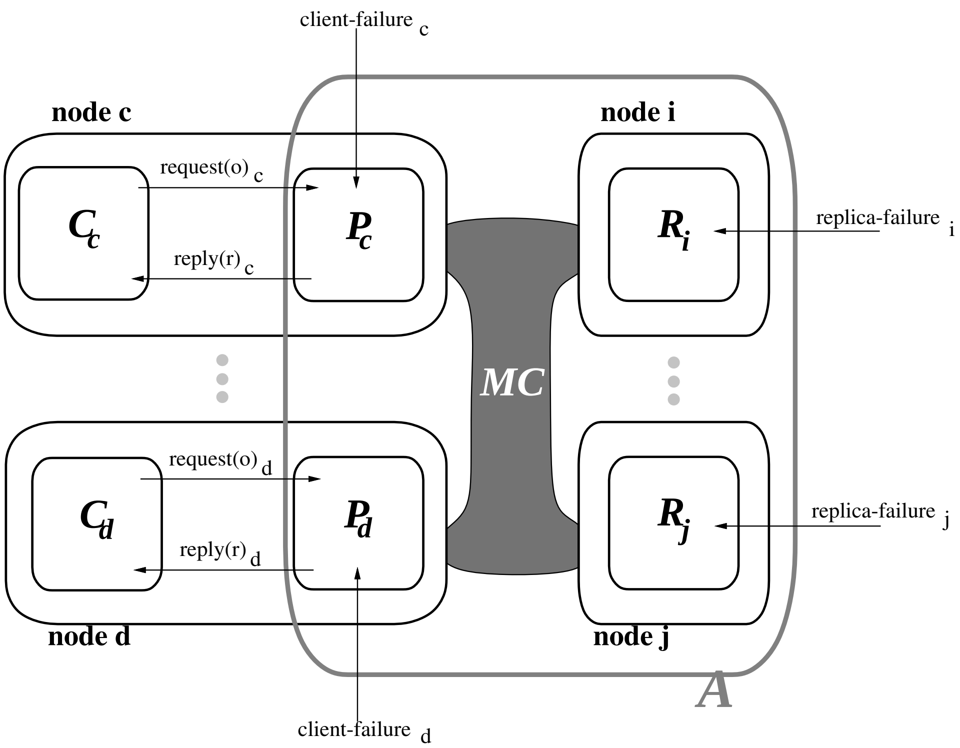

Figure 1: Implementation Architecture

There is a proxy automaton P_c for each client C_c. P_c provides an input action for client c to invoke an operation o on the shared variable, REQUEST$(o)_c$, and an ouput action for c to learn the result r of an operation it requested, REPLY$(r)_c$. The communication between C_c and P_c does not involve any network; they are assumed to execute in the same node in the distributed system. P_c communicates with a set of server replicas to implement the interface it offers to the client.

Each replica has a unique identifier i in a set $\mathcal{R}$ and is modeled by an automaton R_i. We assume $|\mathcal{R}| = 3f + 1$ for some positive integer f. This threshold f is the maximum number of replica faults that can be tolerated by the system. The resiliency of our algorithm is optimal: $3f+1$ is the minimum number of replicas that allow an asynchronous replication system to implement an atomic object when up to f replicas are faulty (see [3] for a proof.)

We assume replicas execute in different nodes in the distributed system. Communication between a proxy and the set of replicas and among replicas is performed using a multicast channel automaton MC. Automata have no access to the state components of automata running on other nodes in the distributed system.

The multicast channel automaton MC may fail to deliver messages, it may delay them, duplicate them, or deliver them out of order. We do not assume synchrony. The nodes are part of an asynchronous distributed system with no known bounds on message delays or on the time for automata to take enabled actions.

We use a Byzantine failure model, i.e., faulty automata may behave arbitrarily (except for the restrictions discussed next.) The CLIENT-FAILURE and REPLICA-FAILURE actions are used to model client and replica failures. Once such a failure action occurs the corresponding automaton is replaced by an arbitrary automaton with the same external interface and it remains faulty for the rest of the execution. We assume however that this arbitrary automaton has a state component called *faulty* that is set to true. It is important to understand that the failure actions and the *faulty* variables are used only to formally model failures for the correctness proof; our algorithm does not know whether a client or replica is faulty or not.

Input:	$\text{REQUEST}(o)_c$, $o \in \mathcal{O}$, $c \in \mathcal{C}$
	CLIENT-FAILURE_c, $c \in \mathcal{C}$
	REPLICA-FAILURE_i, $i \in \mathcal{R}$
Output:	$\text{REPLY}(r)_c$, $r \in \mathcal{O}'$, $c \in \mathcal{C}$

Figure 2: External Signature of A

We allow for a very strong adversary that can coordinate faulty nodes, delay communication, or delay correct nodes in order to cause the most damage to the replicated service. But we assume two restrictions on the adversary and the faulty nodes it controls: automata can use unforgeable digital signatures to authenticate communication; and they can use collision-resistant hash functions. These assumptions are defined in more detail next.

Unforgeable signatures: Any non-faulty client proxy or replica automaton, x, can authenticate messages it sends on the multicast channel by signing them. We denote a message m signed by x as $\langle m \rangle_{\sigma_x}$. And (with high probability) no automaton other than x can send $\langle m \rangle_{\sigma_x}$ (either directly or as part of another message) on the multicast channel for any value of m.

Collision-resistant hash functions Any automaton can compute a digest $D(m)$ of a message m such that (with high probability) it is impossible to find two distinct messages m and m' such that $D(m) \neq D(m')$.

These assumptions are probabilistic but there exist signature schemes (e.g., [2]) and hash functions (e.g, [1]) for which they are believed to hold with very high probability. Therefore, we will assume that they hold with probability one in the rest of the paper.

3 Correctness Condition

We specify the correct behavior for A by using another I/O automaton S with the same external signature as A. We say that A is correct if it implements S. S is a simple abstract atomic object for a variable of type $\mathcal{T}$ that is defined as follows:

Signature:

Input:	$\text{REQUEST}(o)_c$
	CLIENT-FAILURE_c
	REPLICA-FAILURE_i
Internal:	$\text{EXECUTE}(o,t,c)$
	$\text{FAULTY-REQUEST}(o,t,c)$
Output:	$\text{REPLY}(r)_c$

Here, $o \in \mathcal{O}$, $t \in \mathbf{N}$, $c \in \mathcal{C}$, $i \in \mathcal{R}$, and $r \in \mathcal{O}'$

State:

$val \in \mathcal{V}$, initially v_o
$in \subseteq \mathcal{O} \times \mathbf{N} \times \mathcal{C}$, initially $\{\}$
$out \subseteq \mathcal{O}' \times \mathbf{N} \times \mathcal{C}$, initially $\{\}$
$\forall\, c \in \mathcal{C}$, $\textit{last-req}_c \in \mathbf{N}$, initially $\textit{last-req}_c = 0$
$\forall\, c \in \mathcal{C}$, $\textit{last-rep-t}_c \in \mathbf{N}$, initially $\textit{last-rep-t}_c = 0$
$\forall\, c \in \mathcal{C}$, $\textit{faulty-client}_c \in \textit{Bool}$, initially $\textit{faulty-client}_c = \textit{false}$
$\forall\, i \in \mathcal{R}$, $\textit{faulty-replica}_i \in \textit{Bool}$, initially $\textit{faulty-replica}_i = \textit{false}$
$\textit{n-faulty} \equiv |\{\, i \mid \textit{faulty-replica}_i = \textit{true}\,\}|$

Transitions (if $\textit{n-faulty} \leq f$):

$\text{REQUEST}(o)_c$
Eff: $\textit{last-req}_c := \textit{last-req}_c + 1$
$in := in \cup \{\langle o, \textit{last-req}_c, c\rangle\}$

CLIENT-FAILURE_c
Eff: $\textit{faulty-client}_c := \textit{true}$

REPLICA-FAILURE_i
Eff: $\textit{faulty-replica}_i := \textit{true}$

$\text{REPLY}(r)_c$
Pre: $\textit{faulty-client}_c = \textit{true} \vee \exists t : (\langle r,t,c\rangle \in out$
Eff: $out := out - \{\langle r,t,c\rangle\})$

$\text{FAULTY-REQUEST}(o,t,c)$
Pre: $\textit{faulty-client}_c = \textit{true}$
Eff: $in := in \cup \{\langle o,t,c\rangle\}$

$\text{EXECUTE}(o,t,c)$
Pre: $\langle o,t,c\rangle \in in$
Eff: $in := in - \{\langle o,t,c\rangle\}$
if $t > \textit{last-rep-t}_c$ then
$(r, val) := g(c, o, val)$
$out := out \cup \{\langle r,t,c\rangle\}$
$\textit{last-rep-t}_c := t$

Most of the definition of S is self-explanatory but some issues deserve clarification. To model the fact that A does not behave correctly when more than f replicas are Byzantine-faulty, the behavior of S is left unspecified when $\textit{n-faulty} > f$, i.e., S may behave arbitrarily with the restriction that the *faulty-client* and *faulty-replica* variables that have value true cannot be modified. The FAULTY-REQUEST actions model execution of requests by faulty clients that bypass the external signature of A, e.g., by injecting the appropriate messages into the multicast channel. Similarly, the REPLY precondition is weaker for faulty clients to allow arbitrary replies for such clients.

The $\textit{last-req}_c$ component is used to distinguish requests by c to execute the same operation o. And, $\textit{last-rep-t}_c$ remembers the value of $\textit{last-req}_c$ that was associated with the last operation executed for c. This models a well-formedness condition on non-faulty clients: *they are expected to wait for the reply to the last requested operation before they issue the next request.* Otherwise, one of the requests may not even execute and the client may be unable to match the replies with the requests.

4 The System

This section defines the multicast channel, proxy, and replica automata.

4.1 The Multicast Channel Automaton

The multicast channel automaton models the communication network connecting the proxy and replica automata. There is a single multicast automaton in the system with SEND and RECEIVE actions for each proxy and replica. These actions allow automata to send messages in a universal message set $\mathcal{M}$ to any subset of automata with identifiers in $\mathcal{X} = \mathcal{C} \cup \mathcal{R}$. The channel automaton does not provide authenticated communication; the RECEIVE actions do not identify the sender of the message. It is defined as follows.

Signature:
Input: $\text{SEND}(m, X)_x$
Internal: $\text{MISBEHAVE}(m, X, X')$
Output: $\text{RECEIVE}(m)_x$

Here, $m \in \mathcal{M}$, $X, X' \subseteq \mathcal{X}$, and $x \in \mathcal{X}$

State:

wire $\subseteq \mathcal{M} \times 2^{\mathcal{X}}$, initially {}

Transitions:

$\text{SEND}(m, X)_x$
Eff: $wire := wire \cup \{(m, X)\}$

$\text{RECEIVE}(m)_x$
Pre: $\exists (m, X) \in wire : (x \in X$
Eff: $wire := wire - \{(m, X)\} \cup \{(m, X - \{x\})\})$

$\text{MISBEHAVE}(m, X, X')$
Pre: $(m, X) \in wire$
Eff: $wire := wire - \{(m, X)\} \cup \{(m, X')\}$

The MISBEHAVE actions allow the channel to loose messages or duplicate them and the RECEIVE actions are defined such that messages may be reordered. Additionally, the automaton is defined such that every message that was ever sent on the channel is remembered and can be replayed later.

4.2 The Proxy Automaton

Each client C_c Öinteracts with the atomic object through a proxy automaton P_c, which is defined as follows.

Signature:

Input: $\text{REQUEST}(o)_c$

$\text{RECEIVE}(\langle \text{REPLY}, v, t, c, i, r\rangle_{\sigma_i})_c$

CLIENT-FAILURE_c

Output: $\text{REPLY}(r)_c$

$\text{SEND}(m, X)_c$

Here, $o \in \mathcal{O}$, $v, t \in \mathbf{N}$, $c \in \mathcal{C}$, $i \in \mathcal{R}$, $r \in \mathcal{O}'$, $m \in M$, and $X \subseteq \mathcal{X}$

State:

$\mathit{view}_c \in \mathbf{N}$, initially 0
$\mathit{in}_c \subseteq \mathcal{M}$, initially $\{\}$
$\mathit{out}_c \subseteq \mathcal{M}$, initially $\{\}$
$\mathit{last\text{-}req}_c \in \mathbf{N}$, initially 0
$\mathit{retrans}_c \in \mathit{Bool}$, initially *false*
$\mathit{faulty}_c \in \mathit{Bool}$, initially *false*

Transitions:

$\text{REQUEST}(o)_c$
Eff: $\mathit{last\text{-}req}_c := \mathit{last\text{-}req}_c + 1$
$\mathit{out}_c := \{\langle \text{REQUEST}, o, \mathit{last\text{-}req}_c, c\rangle_{\sigma_c}\}$
$\mathit{in}_c := \{\}$
$\mathit{retrans}_c := \mathit{false}$

$\text{RECEIVE}(\langle \text{REPLY}, v, t, c, i, r\rangle_{\sigma_i})_c$
Eff: if $(\mathit{out}_c \neq \{\} \wedge \mathit{last\text{-}req}_c = t)$ then
$\mathit{in}_c := \mathit{in}_c \cup \{\langle \text{REPLY}, v, t, c, i, r\rangle_{\sigma_i}\}$

CLIENT-FAILURE_c
Eff: $\mathit{faulty}_c := \mathit{true}$

$\text{REPLY}(r)_c$
Pre: $\mathit{out}_c \neq \{\} \wedge \exists R : (|R| > f \wedge \forall i \in R : (\exists v : (\langle \text{REPLY}, v, \mathit{last\text{-}req}_c, c, i, r\rangle_{\sigma_i} \in \mathit{in}_c)))$
Eff: $\mathit{view}_c := max(\{v | \langle \text{REPLY}, v, \mathit{last\text{-}req}_c, c, i, r\rangle_{\sigma_i} \in \mathit{in}_c\})$
$\mathit{out}_c := \{\}$

$\text{SEND}(m, \{\mathit{view}_c \bmod |\mathcal{R}|\})_c$
Pre: $m \in \mathit{out}_c \wedge \neg \mathit{retrans}_c$
Eff: $\mathit{retrans}_c := true$

$\text{SEND}(m, \mathcal{R})_c$
Pre: $m \in \mathit{out}_c \wedge \mathit{retrans}_c$
Eff: none

4.3 The Replica Automaton

Each replica automaton R_i is defined as follows.

Signature:

Input: $\text{RECEIVE}(\langle \text{REQUEST}, o, t, c\rangle_{\sigma_c})_i$
$\text{RECEIVE}(\langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_j})_i$
$\text{RECEIVE}(\langle \text{PREPARE}, v, n, d, j\rangle_{\sigma_j})_i$
$\text{RECEIVE}(\langle \text{COMMIT}, v, n, d, j\rangle_{\sigma_j})_i$
$\text{RECEIVE}(\langle \text{VIEW-CHANGE}, v, P, j\rangle_{\sigma_j})_i$
$\text{RECEIVE}(\langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_j})_i$
REPLICA-FAILURE_i

Internal: $\text{SEND-PRE-PREPARE}(m, v, n)_i$
$\text{SEND-COMMIT}(m, v, n)_i$
$\text{EXECUTE}(m, v, n)_i$
$\text{VIEW-CHANGE}(v)_i$
$\text{SEND-NEW-VIEW}(v, V)_i$

Output: $\text{SEND}(m, X)_c$

Here, $t, v, n \in \mathbf{N}$, $c \in \mathcal{C}$, $i, j \in \mathcal{R}$, $m \in \mathcal{M}$, $V, O, N \subseteq \mathcal{M}$, $X \subseteq \mathcal{X}$, and $d \in \mathcal{D} = \{d \mid \exists m \in \mathcal{M} : (d = D(m))\}$

State:

$\mathit{val}_i \in \mathcal{V}$, initially v_o
$\mathit{view}_i \in \mathbf{N}$, initially 0
$\mathit{in}_i \subseteq \mathcal{M}$, initially $\{\}$
$\mathit{out}_i \subseteq \mathcal{M}$, initially $\{\}$
$\mathit{last\text{-}rep}_i : \mathcal{C} \rightarrow \mathcal{O}'$, initially $\forall\, c \in \mathcal{C} : \mathit{last\text{-}rep}_i(c) = \mathit{null\text{-}rep}$
$\mathit{last\text{-}rep\text{-}t}_i : \mathcal{C} \rightarrow \mathbf{N}$, initially $\forall\, c \in \mathcal{C} : \mathit{last\text{-}rep\text{-}t}_i(c) = 0$
$\mathit{seqno}_i \in \mathbf{N}$, initially 0
$\mathit{last\text{-}exec}_i \in \mathbf{N}$, initially 0
$\mathit{faulty}_i \in \mathit{Bool}$, initially *false*

Auxiliary functions:

$\mathit{tag}(m, u) \equiv m = \langle u, ...\rangle$
$\mathit{primary}(v) \equiv v \mod |\mathcal{R}|$
$\mathit{primary}(i) \equiv \mathit{view}_i \mod |\mathcal{R}|$
$\mathit{in\text{-}v}(v, i) \equiv \mathit{view}_i = v$
$\mathit{prepared}(m, v, n, M) \equiv \langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_{\mathit{primary}(v)}} \in M \wedge$
$\quad \exists\, R : (|R| \geq 2f \wedge \mathit{primary}(v) \notin R \wedge \forall\, k \in R : (\langle \text{PREPARE}, v, n, D(m), k\rangle_{\sigma_k} \in M))$
$\mathit{prepared}(m, v, n, i) \equiv \mathit{prepared}(m, v, n, \mathit{in}_i)$
$\mathit{last\text{-}prepared}(m, v, n, M) \equiv \mathit{prepared}(m, v, n, M) \wedge$
$\quad \nexists\, m', v' : ((\mathit{prepared}(m', v', n, M) \wedge v' > v) \vee (\mathit{prepared}(m', v, n, M) \wedge m \neq m'))$
$\mathit{last\text{-}prepared}(m, v, n, i) \equiv \mathit{last\text{-}prepared}(m, v, n, \mathit{in}_i)$
$\mathit{committed}(m, v, n, i) \equiv (\exists\, v' : (\langle \text{PRE-PREPARE}, v', n, m\rangle_{\sigma_{\mathit{primary}(v')}} \in \mathit{in}_i) \vee m \in \mathit{in}_i) \wedge$
$\quad \exists\, R : (|R| \geq 2f + 1 \wedge \forall\, k \in R : (\langle \text{COMMIT}, v, n, D(m), k\rangle_{\sigma_k} \in \mathit{in}_i))$
$\mathit{correct\text{-}view\text{-}change}(m, v, j) \equiv \exists\, P : (m = \langle \text{VIEW-CHANGE}, v, P, j\rangle_{\sigma_j} \wedge$
$\quad \forall\, \langle \text{PRE-PREPARE}, v', n, m'\rangle_{\sigma_{\mathit{primary}(v')}} \in P : (\mathit{last\text{-}prepared}(m', v', n, P) \wedge v' < v)$
$\mathit{merge\text{-}P}(V) \equiv \{\, m \mid \exists\, \langle \text{VIEW-CHANGE}, v, P, k\rangle_{\sigma_k} \in V : m \in P \,\}$
$\mathit{max\text{-}n}(M) \equiv \mathit{max}(\{\, n \mid \langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_i} \in M\})$
$\mathit{correct\text{-}new\text{-}view}(m, v) \equiv$
$\quad \exists\, V, O, N, R : (m = \langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_{\mathit{primary}(v)}} \wedge |V| = |R| = 2f + 1 \wedge$
$\quad \forall\, k \in R : (\exists\, m' \in V : (\mathit{correct\text{-}view\text{-}change}(m', v, k))) \wedge$
$\quad O = \{\, \langle \text{PRE-PREPARE}, v, n, m'\rangle_{\sigma_{\mathit{primary}(v)}} \mid \exists\, v' : \mathit{last\text{-}prepared}(m', v', n, \mathit{merge\text{-}P}(V))\} \wedge$
$\quad N = \{\, \langle \text{PRE-PREPARE}, v, n, \mathit{null}\rangle_{\sigma_{\mathit{primary}(v)}} \mid n < \mathit{max\text{-}n}(O) \wedge$

$$\nexists\, v', m', n \;:\; \textit{last-prepared}(m', v', n, \textit{merge-P}(V)))$$
$$\textit{has-new-view}(v, i) \;\equiv\; v \;=\; 0 \;\vee\; \exists\, m \;:\; (\, m \;\in\; \textit{in}_i \;\wedge\; \textit{correct-new-view}(m, v))$$

Input Transitions:

$\text{RECEIVE}(\langle \text{REQUEST}, o, t, c\rangle_{\sigma_c})_i$
Eff: let $m = \langle \text{REQUEST}, o, t, c\rangle_{\sigma_c}$
if $t = \textit{last-rep-t}_i(c)$ then
$\textit{out}_i := \textit{out}_i \cup \{\langle \text{REPLY}, \textit{view}_i, t, c, i, \textit{last-rep}_i(c)\rangle_{\sigma_i}\}$
else
$\textit{in}_i := \textit{in}_i \cup \{m\}$
if $\textit{primary}(i) \neq i$ then
$\textit{out}_i := \textit{out}_i \cup \{m\}$

$\text{RECEIVE}(\langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: if $j = \textit{primary}(i) \wedge \textit{in-v}(v, i) \wedge \textit{has-new-view}(v, i) \wedge$
$\nexists d : (d \neq D(m) \wedge \langle \text{PREPARE}, v, n, d, i\rangle_{\sigma_i} \in \textit{in}_i)$ then
let $p = \langle \text{PREPARE}, v, n, D(m), i\rangle_{\sigma_i}$
$\textit{in}_i := \textit{in}_i \cup \{\langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_j}, p\}$
$\textit{out}_i := \textit{out}_i \cup \{p\}$
else if $\exists o, t, c : (m = \langle \text{REQUEST}, o, t, c\rangle_{\sigma_c})$ then
$\textit{in}_i := \textit{in}_i \cup \{m\}$

$\text{RECEIVE}(\langle \text{PREPARE}, v, n, d, j\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: if $j \neq \textit{primary}(i) \wedge \textit{in-v}(v, i)$ then
$\textit{in}_i := \textit{in}_i \cup \{\langle \text{PREPARE}, v, n, d, j\rangle_{\sigma_j}\}$

$\text{RECEIVE}(\langle \text{COMMIT}, v, n, d, j\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: if $\textit{view}_i \geq v$ then
$\textit{in}_i := \textit{in}_i \cup \{\langle \text{COMMIT}, v, n, d, j\rangle_{\sigma_j}\}$

$\text{RECEIVE}(\langle \text{VIEW-CHANGE}, v, \mathcal{P}, j\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: let $m = \langle \text{VIEW-CHANGE}, v, \mathcal{P}, j\rangle_{\sigma_j}$
if $v \geq \textit{view}_i \wedge \textit{correct-view-change}(m, v, j)$ then
$\textit{in}_i := \textit{in}_i \cup \{m\}$

$\text{RECEIVE}(\langle \text{NEW-VIEW}, v, X, O, N\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: let $m = \langle \text{NEW-VIEW}, v, X, O, N\rangle_{\sigma_j}$,
$P = \{\langle \text{PREPARE}, v, n', D(m'), i\rangle_{\sigma_i} | \langle \text{PRE-PREPARE}, v, n', m'\rangle_{\sigma_j} \in (O \cup N)\}$
if $v > 0 \wedge v \geq \textit{view}_i \wedge \textit{correct-new-view}(m, v) \wedge \neg\textit{has-new-view}(v, i)$ then
$\textit{view}_i := v$
$\textit{in}_i := \textit{in}_i \cup O \cup N \cup \{m\} \cup P$
$\textit{out}_i := P$

REPLICA-FAILURE_i
Eff: $\textit{faulty}_i := \mathit{true}$

Output Transitions:

$\text{SEND}(m, \mathcal{R} - \{i\})_i$
Pre: $m \in out_i \wedge \neg tag(m, \text{REQUEST}) \wedge \neg tag(m, \text{REPLY})$
Eff: $out_i := out_i - \{m\}$

$\text{SEND}(m, \{primary(i)\})_i$
Pre: $m \in out_i \wedge tag(m, \text{REQUEST})$
Eff: $out_i := out_i - \{m\}$

$\text{SEND}(\langle \text{REPLY}, v, t, c, i, r\rangle_{\sigma_i}, \{c\})_i$
Pre: $\langle \text{REPLY}, v, t, c, i, r\rangle_{\sigma_i} \in out_i$
Eff: $out_i := out_i - \{\langle \text{REPLY}, v, t, c, i, r\rangle_{\sigma_i}\}$

Internal Transitions:

$\text{SEND-PRE-PREPARE}(m, v, n)_i$
Pre: $primary(i) = i \wedge seqno_i = n - 1 \wedge \textit{in-v}(v, i) \wedge \textit{has-new-view}(v, i) \wedge$
$\exists o, t, c : (m = \langle \text{REQUEST}, o, t, c\rangle_{\sigma_c} \wedge m \in in_i) \wedge \nexists \langle \text{PRE-PREPARE}, v, n', m\rangle_{\sigma_i} \in in_i$
Eff: $seqno_i := seqno_i + 1$
let $p = \langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_i}$
$out_i := out_i \cup \{p\}$
$in_i := in_i \cup \{p\}$

$\text{SEND-COMMIT}(m, v, n)_i$
Pre: $prepared(m, v, n, i) \wedge \langle \text{COMMIT}, v, n, D(m), i\rangle_{\sigma_i} \notin in_i$
Eff: let $c = \langle \text{COMMIT}, v, n, D(m), i\rangle_{\sigma_i}$
$out_i := out_i \cup \{c\}$
$in_i := in_i \cup \{c\}$

$\text{EXECUTE}(m, v, n)_i$
Pre: $n = \textit{last-exec}_i + 1 \wedge committed(m, v, n, i)$
Eff: $\textit{last-exec}_i := n$
if $(m \neq null)$ then
let $\langle \text{REQUEST}, o, t, c\rangle_{\sigma_c} = m$
if $t \geq \textit{last-rep-t}_i(c)$ then
if $t > \textit{last-rep-t}_i(c)$ then
$\textit{last-rep-t}_i(c) := t$
$(\textit{last-rep}_i(c), val_i) := g(c, o, val_i)$
$out_i := out_i \cup \{\langle \text{REPLY}, view_i, t, c, i, \textit{last-rep}_i(c)\rangle_{\sigma_i}\}$
$in_i := in_i - \{m\}$

$\text{SEND-VIEW-CHANGE}(v)_i$
Pre: $v = view_i + 1$
Eff: $view_i := v$
let $P' = \{\langle m, v, n\rangle | \textit{last-prepared}(m, v, n, i)\}$,
$P = \bigcup_{\langle m, v, n\rangle \in P'} (\{p = \langle \text{PREPARE}, v, n, D(m), k\rangle_{\sigma_k} | p \in in_i\} \cup \{\langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_{primary(v)}}\})$,
$m = \langle \text{VIEW-CHANGE}, v, P, i\rangle_{\sigma_i}$
$out_i := out_i \cup \{m\}$
$in_i := in_i \cup \{m\}$

SEND-NEW-VIEW$(v, V)_i$

Pre: $primary(v) = i \wedge v \geq view_i \wedge v > 0 \wedge V \subseteq in_i \wedge |V| = 2f+1 \wedge \neg has\text{-}new\text{-}view(v,i) \wedge$
$\exists R : (|R| = 2f+1 \wedge \forall k \in R : (\exists P : (\langle \text{VIEW-CHANGE}, v, P, k\rangle_{\sigma_k} \in V)))$

Eff: $view_i := v$

let $O = \{\langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_i} | \exists v' : last\text{-}prepared(m, v', n, merge\text{-}P(V))\}$,
$N = \{\langle \text{PRE-PREPARE}, v, n, null, k\rangle_{\sigma_i} | n < max\text{-}n(O) \wedge \nexists v', m, n : last\text{-}prepared(m, v', n, merge\text{-}P(V))\}$,
$m = \langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_i}$
$seqno_i := max\text{-}n(O)$
$in_i := in_i \cup O \cup N \cup \{m\}$
$out_i := \{m\}$

4.4 Safety Proof

This section proves the safety of our algorithm, i.e., it proves that A implements S. We start by proving some invariants.

Invariant 4.1 *The following is true of any reachable state in an execution of A,*

$$\forall\, i,j \in \mathcal{R},\ m \in \mathcal{M} : ((\neg faulty_i \wedge \neg faulty_j \wedge \neg tag(m, \text{REPLY})) \Rightarrow$$
$$(((\langle m\rangle_{\sigma_i} \in in_j \vee \exists\, m' = \langle \text{VIEW-CHANGE}, v, P, k\rangle_{\sigma_k} : (m' \in in_j \wedge \langle m\rangle_{\sigma_i} \in P) \vee$$
$$\exists\, m' = \langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_k} : (m' \in in_j \wedge (\langle m\rangle_{\sigma_i} \in V \vee \langle m\rangle_{\sigma_i} \in merge\text{-}P(V))))$$
$$\Rightarrow \langle m\rangle_{\sigma_i} \in in_i))$$

The same is also true if one replaces in_j *by* $\{m \mid \exists X : (m, X) \in \text{wire}\}$ *or by* out_j

Proof: For any reachable state x of A and message value m that is not a reply message, if replica i is not faulty in state x, $\langle m\rangle_{\sigma_i} \in out_i \Rightarrow \langle m\rangle_{\sigma_i} \in in_i$. Additionally, if $\langle m\rangle_{\sigma_i} \in in_i$ is true for some state in an execution, it remains true in all subsequent states in that execution or until i becomes faulty. By inspection of the code for automaton R_i, these two conditions are true because every action of R_i that inserts a message $\langle m\rangle_{\sigma_i}$ in out_i also inserts it in in_i and no action ever removes a message signed by i from in_i.

Our assumption on the strength of authentication guarantees that no automaton can impersonate a non-faulty replica R_i by sending $\langle m\rangle_{\sigma_i}$ (for all values of m) on the multicast channel. Therefore, for a signed message $\langle m\rangle_{\sigma_i}$ to be in some state component of a non-faulty automaton other than R_i, it is necessary for SEND$(\langle m\rangle_{\sigma_i}, X)_i$ to have executed for some value of X at some earlier point in that execution. The precondition for the execution of such a send action requires $\langle m\rangle_{\sigma_i} \in out_i$. The latter and the two former conditions prove the invariant. ∎

Invariant 4.2 *The following is true of any reachable state in an execution of A, for any replica i such that faulty$_i$ is* false*:*

1. $\forall \langle \text{PREPARE}, v, n, d, i\rangle_{\sigma_i} \in in_i : (\nexists d' \neq d : (\langle \text{PREPARE}, v, n, d', i\rangle_{\sigma_i} \in in_i))$
2. $\forall v, n, m : ((i = primary(v) \wedge \langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_i} \in in_i) \Rightarrow$
   $\nexists m' : (m' \neq m \wedge \langle \text{PRE-PREPARE}, v, n, m'\rangle_{\sigma_i} \in in_i))$
3. $\forall \langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_i} \in in_i : (i = primary(v) \Rightarrow n \leq seqno_i)$
4. $\forall \langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_{primary(v)}} \in in_i :$
   $(v > 0 \Rightarrow \exists m' = \langle \text{NEW-VIEW}, v, X, O, N\rangle_{\sigma_{primary(v)}} : (m' \in in_i \wedge correct\text{-}new\text{-}view(m', v)))$

5. $\forall m' = \langle \text{NEW-VIEW}, v, X, O, N \rangle_{\sigma_{primary(v)}} \in in_i : \textit{correct-new-view}(m', v)$
6. $\forall m' = \langle \text{VIEW-CHANGE}, v, \mathcal{P}, j \rangle_{\sigma_j} \in in_i : \textit{correct-view-change}(m', v, j)$
7. $\forall \langle \text{PREPARE}, v, n, D(m), i \rangle_{\sigma_i} \in in_i : (\langle \text{PRE-PREPARE}, v, n, m \rangle_{\sigma_{primary(v)}} \in in_i)$
8. $\forall \langle \text{PRE-PREPARE}, v, n, m \rangle_{\sigma_{primary(v)}} \in in_i : (i \neq primary(v) \Rightarrow \langle \text{PREPARE}, v, n, D(m), i \rangle_{\sigma_i} \in in_i)$
9. $\forall \langle \text{PRE-PREPARE}, v, n, m \rangle_{\sigma_{primary(v)}} \in in_i : v \leq view_i$

Proof: The proof is by induction on the length of the execution. The initializations ensure that $in_i = \{\}$ and, therefore, all conditions are vacuously true in the base case. For the inductive step, assume that the invariant holds for every state of any execution α of length at most l. We will show that the invariant also holds for any one step extension α_1 of α.

Condition (1) can be violated in α_1 only if an action that may insert a prepare message signed by i in in_i executes. These are actions of the form:

1. $\text{RECEIVE}(\langle \text{PRE-PREPARE}, v, n, m' \rangle_{\sigma_j})_i$
2. $\text{RECEIVE}(\langle \text{PREPARE}, v, n, d, j \rangle_{\sigma_j})_i$
3. $\text{RECEIVE}(\langle \text{NEW-VIEW}, v, V, O, N \rangle_{\sigma_j})_i$

The first type of action cannot violate condition (1) because the condition in the *if* statement ensures that $\langle \text{PREPARE}, v, n, D(m'), i \rangle_{\sigma_i}$ is not inserted in in_i when there exists a $\langle \text{PREPARE}, v, n, d, i \rangle_{\sigma_i} \in in_i$ such that $D(m') \neq d$. Similarly, the second type of action cannot violate condition (1) because it only inserts the argument prepare message in in_i if it is signed by a replica other than R_i.

For the case $v = 0$, actions of type 3 never have effects on the state of R_i. For the case $v > 0$, we can apply the inductive hypothesis of conditions (7) and (4) to conclude that if there existed a $\langle \text{PREPARE}, v, n, D(m), i \rangle_{\sigma_i} \in in_i$ in the last state in α, there would also exist a new-view message for view v in in_i in that state. Therefore, the precondition of actions of type 3 would prevent them from executing in such a state. Since actions of type 3 may insert multiple prepare messages signed by R_i into in_i, there is still a chance they can violate condition (1). However, this cannot happen because these actions are enabled only if the argument new-view message is correct and the definition of *correct-new-view* ensures that there is at most one pre-prepare message with a given sequence number in $O \cup N$.

Condition (2) can be violated in α_1 only by the execution of an action of one of the following types:

1. $\text{RECEIVE}(\langle \text{PRE-PREPARE}, v, n, m' \rangle_{\sigma_j})_i$,
2. $\text{RECEIVE}(\langle \text{NEW-VIEW}, v, V, O, N \rangle_{\sigma_j})_i$,
3. $\text{SEND-PRE-PREPARE}(m, v, n)_i$, or
4. $\text{SEND-NEW-VIEW}(v, V)_i$

Actions of the first two types cannot violate condition (2) because they only insert pre-prepare messages in in_i that are not signed by R_i. Actions of the third type cannot violate condition (2) because the inductive hypothesis for condition (3) and the precondition for the send-pre-prepare action ensure that the pre-prepare message inserted in in_i has a sequence number that is one higher than the sequence number of any pre-prepare message for the same view signed by R_i in in_i.

Finally, actions of the fourth type cannot violate condition (2). For $v = 0$, they are not enabled. For $v > 0$, the inductive hypothesis of condition (4) and the precondition for the send-new-view action ensure that no pre-prepare for view v can be in in_i when the action executes, and the definition of O and N ensures that there is at most one pre-prepare message with a given sequence number in $O \cup N$.

Condition (3) can potentially be violated by actions that insert pre-prepares in in_i or modify $seqno_i$. These are exactly the actions of the types listed for condition (2). As before, actions of the first two types cannot violate condition (3) because they only insert pre-prepare messages in in_i that are not signed by R_i and they do not modify $seqno_i$. The send-pre-prepare action preserves condition (3) because it increments $seqno_i$ such that it becomes equal to the sequence number of the pre-prepare message it inserts in in_i. The send-new-view actions also preserve condition (3): (as shown before) actions of this type only execute if there is no pre-prepare for view v in in_i and, when they execute, they set $seqno_i := \textit{max-n}(O)$, which is equal to the sequence number of the pre-prepare for view v with the highest sequence number in in_i.

To violate condition (4), an action must either insert a pre-prepare message in in_i or remove a new-view message from in_i. No action ever removes new-view messages from in_i. The actions that may insert pre-prepare messages in in_i are exactly the actions of the types listed for condition (2). The first type of action in this list cannot violate condition (4) because the *if* statement in its body ensures that the argument pre-prepare message is inserted in in_i only when $\textit{has-new-view}(v, i)$ is true. The second type of action only inserts pre-prepare messages for view v in in_i if the argument new-view message is correct and in this case it also inserts the argument new-view message in in_i. Therefore, the second type of action also preserves condition (4). The precondition of send-pre-prepare actions ensures that send-pre-prepare actions preserve condition (4). Finally, the send-new-view actions also preserve condition (4) because their effects and the inductive hypothesis for condition (6) ensure that a correct new-view message for view v is inserted in in_i whenever a pre-prepare for view v is inserted in in_i.

Conditions (5) and (6) are never violated. First, received new-view and view-change messages are always checked for correctness before being inserted in in_i. Second, the effects of send-view-change actions together with the inductive hypothesis of condition (9) and the precondition of send-view-change actions ensure that only correct view-change messages are inserted in in_i. Third, the inductive hypothesis of condition (6) and the effects of send-new-view actions ensure that only correct new-view messages are inserted in in_i.

Condition (7) is never violated because no action ever removes a pre-prepare from in_i and the actions that insert a $\langle \text{PREPARE}, v, n, D(m), i\rangle_{\sigma_i}$ in in_i (namely actions of the form $\text{RECEIVE}(\langle \text{PRE-PREPARE}, v, n, m'\rangle_{\sigma_j})_i$ and $\text{RECEIVE}(\langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_j})_i$) also insert a $\langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_{primary(v)}}$ in in_i.

Condition (8) can only be violated by actions that insert pre-prepare messages in in_i because prepare messages are never removed from in_i. These are exactly the actions listed for condition (2). The first two types of actions preserve condition (8) because whenever they insert a pre-prepare message in in_i they always insert a matching prepare message. The last two types of actions can not violate condition (8) because they never insert pre-prepare messages for views v such that $\textit{primary}(v) \neq i$ in in_i.

The only actions that can violate condition (9) are actions that insert pre-prepare messages in in_i or make $view_i$ smaller. Since no actions ever make $view_i$ smaller, the actions that may violate condition (9) are exactly those listed for condition (2). The *if* statement in the first type of action ensures that it only inserts pre-prepare messages in in_i when their view number is equal to $view_i$. The *if* statement in the second type of action ensures thatv it only inserts pre-prepare messages in in_i when their view number is greater than or equal to $view_i$. Therefore, both types of actions preserve the invariant. The precondition for the third type of action and the effects of the fourth type of action ensure that only pre-prepare messages with view number equal to $view_i$ are inserted in in_i. Thus, these two types of actions also preserve the invariant. $\square$

Definition 4.3 n-faulty $\equiv |\{i \in \mathcal{R} | \text{faulty}_i = \text{true}\}|$

Invariant 4.4 *The following is true of any reachable state in an execution of A,*

$$\forall\, i,j \in \mathcal{R},\ n,v \in \mathbf{N},\ m,m' \in \mathcal{M} : ((\neg \mathit{faulty}_i \wedge \neg \mathit{faulty}_j \wedge \mathit{n\text{-}faulty} \leq f) \Rightarrow$$
$$(\mathit{prepared}(m,v,n,i) \wedge \mathit{prepared}(m',v,n,j) \Rightarrow D(m) = D(m')))$$

Proof: By contradiction, assume the invariant does not hold. Then $\mathit{prepared}(m,v,n,i) = \mathit{true}$ and $\mathit{prepared}(m',v,n,j) = \mathit{true}$ for some values of m, m', v, n, i, j such that $D(m') \neq D(m)$. Since there are $3f + 1$ replicas, this condition and the definition of the *prepared* predicate imply:

(a) $\exists R : (|R| > f \wedge \forall k \in R :$
$(((\langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_k} \in in_i \wedge k = \mathit{primary}(v)) \vee \langle \text{PREPARE}, v, n, D(m), k\rangle_{\sigma_k} \in in_i) \wedge$
$(((\langle \text{PRE-PREPARE}, v, n, m'\rangle_{\sigma_k} \in in_j \wedge k = \mathit{primary}(v)) \vee \langle \text{PREPARE}, v, n, D(m'), k\rangle_{\sigma_k} \in in_j)))$

Since there are at most f faulty replicas and R has size at least $f + 1$, condition (a) implies:

(b) $\exists k \in R : (\mathit{faulty}_k = \mathit{false} \wedge$
$(((\langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_k} \in in_i \wedge k = \mathit{primary}(v)) \vee \langle \text{PREPARE}, v, n, D(m), k\rangle_{\sigma_k} \in in_i) \wedge$
$(((\langle \text{PRE-PREPARE}, v, n, m'\rangle_{\sigma_k} \in in_j \wedge k = \mathit{primary}(v)) \vee \langle \text{PREPARE}, v, n, D(m'), k\rangle_{\sigma_k} \in in_j)))$

Invariant 4.1 and (b) imply:

(c) $\exists k \in R : (\mathit{faulty}_k = \mathit{false} \wedge$
$(((\langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_k} \in in_k \wedge k = \mathit{primary}(v)) \vee \langle \text{PREPARE}, v, n, D(m), k\rangle_{\sigma_k} \in in_k) \wedge$
$(((\langle \text{PRE-PREPARE}, v, n, m'\rangle_{\sigma_k} \in in_k \wedge k = \mathit{primary}(v)) \vee \langle \text{PREPARE}, v, n, D(m'), k\rangle_{\sigma_k} \in in_k)))$

Condition (c) contradicts Invariant 4.2 (conditions 1, 7 and 2.) $\square$

Invariant 4.5 *The following is true of any reachable state in an execution of A,*

$$\forall\, i \in \mathcal{R} : ((\neg \mathit{faulty}_i \wedge \mathit{n\text{-}faulty} \leq f) \Rightarrow$$
$$(\forall\, \langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_k} \in in_i,\ n, v' \in \mathbf{N} :$$
$$(\mathit{prepared}(m, v', n, \mathit{merge\text{-}P}(V)) \wedge \mathit{prepared}(m', v', n, \mathit{merge\text{-}P}(V)) \Rightarrow D(m) = D(m'))))$$

Proof: Since Invariant 4.2 (condition 6) ensures any new-view message in in_i for a non-faulty i satisfies *correct-new-view*, the proof for Invariant 4.4 can also be used here with minor modifications. $\square$

Invariant 4.6 *The following is true of any reachable state in an execution of A,*
$\forall i \in \mathcal{R} : (\neg \mathit{faulty}_i \Rightarrow \forall \langle \text{COMMIT}, v, n, d, i \rangle_{\sigma_i} \in \mathit{in}_i : (\exists m : (D(m) = d \wedge \mathit{prepared}(m, v, n, i) = \mathit{true}))$

Proof: The proof is by induction on the length of the execution. The initializations ensure that $\mathit{in}_i = \{\}$ and, therefore, the condition is vacuously true in the base case. For the inductive step, the only actions that can violate the condition are those that insert commit messages in in_i, i.e., actions of the form $\text{RECEIVE}(\langle \text{COMMIT}, v, n, d, j \rangle_{\sigma_j})_i$ or $\text{SEND-COMMIT}(m, v, n)_i$. Actions of the first type never violate the lemma because they only insert commit messages signed by replicas other than R_i in in_i. The precondition for send-commit actions ensures that they only insert $\langle \text{COMMIT}, v, n, D(m), i \rangle_{\sigma_i}$ in in_i if $\mathit{prepared}(m, v, n, i)$ is true. ∎

Invariant 4.7 *The following is true of any reachable state in an execution of A,*

$$\forall\, i \in \mathcal{R},\ n, v \in \mathbf{N},\ m \in \mathcal{M} : ((\neg \mathit{faulty}_i \wedge \mathit{committed}(m, v, n, i)) \Rightarrow$$
$$(\exists\, R : (|R| > 2f - \mathit{n\text{-}faulty} \wedge \forall\, k \in R : (\mathit{faulty}_k = \mathit{false} \wedge \mathit{prepared}(m, v, n, k)))))$$

Proof: From the definition of the *committed* predicate $\mathit{committed}(m, v, n, i) = \mathit{true}$ implies
(a) $\exists R : (|R| \geq 2f + 1 \wedge \forall k \in R : (\langle \text{COMMIT}, v, n, D(m), k \rangle_{\sigma_k} \in \mathit{in}_i))$.
Invariant 4.1 implies
(b) $\exists R : (|R| > 2f - \mathit{n\text{-}faulty} \wedge \forall k \in R : (\mathit{faulty}_k = false \wedge \langle \text{COMMIT}, v, n, D(m), k \rangle_{\sigma_k} \in \mathit{in}_k))$.
Invariant 4.6 and (b) prove the invariant. ∎

Invariant 4.8 *The following are true of any reachable state in an execution of A, for any replica i such that faulty_i is* false*:*

1. $\forall m, v, n, P : (\langle \text{VIEW-CHANGE}, v, P, i \rangle_{\sigma_i} \in \mathit{in}_i \Rightarrow$
   $\forall v' < v : (\mathit{last\text{-}prepared\text{-}b}(m, v', n, i, v) \Leftrightarrow \mathit{last\text{-}prepared}(m, v', n, P)))$
2. $\forall m = \langle \text{NEW-VIEW}, v, V, O, N \rangle_{\sigma_{\mathit{primary}(v)}} \in \mathit{in}_i : ((O \cup N) \subseteq \mathit{in}_i)$

Where *last-prepared-b* is defined as follows:
$\mathit{last\text{-}prepared\text{-}b}(m, v, n, i, b) \equiv v < b \wedge \mathit{prepared}(m, v, n, \mathit{in}_i) \wedge$
$\nexists m', v' : ((\mathit{prepared}(m', v', n, \mathit{in}_i) \wedge v < v' < b) \vee (\mathit{prepared}(m', v, n, \mathit{in}_i) \wedge m \neq m'))$.

Proof: The proof is by induction on the length of the execution. The initializations ensure that $\mathit{in}_i = \{\}$ and, therefore, the condition is vacuously true in the base case.

For the inductive step, the only actions that can violate condition (1) are those that insert view-change messages in in_i and those that insert pre-prepare or prepare messages in in_i (no pre-prepare or prepare message is ever removed from in_i.) These actions have one of the following schemas:

1. $\text{RECEIVE}(\langle \text{VIEW-CHANGE}, v, P, j \rangle_{\sigma_j})_i$
2. $\text{VIEW-CHANGE}(v)_i$
3. $\text{RECEIVE}(\langle \text{PRE-PREPARE}, v, n, m' \rangle_{\sigma_j})_i$,
4. $\text{RECEIVE}(\langle \text{PREPARE}, v, n, d, j \rangle_{\sigma_j})_i$,

5. RECEIVE($\langle$NEW-VIEW$, v, V, O, N\rangle_{\sigma_j})_i$,

6. SEND-PRE-PREPARE$(m, v, n)_i$, or

7. SEND-NEW-VIEW$(v, V)_i$

Actions of the first type never violate the lemma because they only insert view-change messages signed by replicas other than R_i in in_i. The effects of actions of the second type ensure that when a view-change message $\langle$VIEW-CHANGE$, v, P, i\rangle_{\sigma_i}$ is inserted in in_i the following condition is true:
(a) $\forall v' < v : (\textit{last-prepared}(m, v', n, i) \Leftrightarrow \textit{last-prepared}(m, v', n, \mathcal{P}))$. Condition (a) and Invariant 4.2 (condition 9) imply condition 1 of the invariant.

For the other types of actions, assume there exists at least a view change message for v signed by R_i in in_i before one of the other types of actions executes (otherwise the lemma would be vacuously true) and pick any $m' = \langle$VIEW-CHANGE$, v, P, i\rangle_{\sigma_i} \in in_i$. The inductive hypothesis ensures that the following condition holds before the actions execute:

$$\forall m, n, v' < v : (\textit{last-prepared-b}(m, v', n, i, v) \Leftrightarrow \textit{last-prepared}(m, v', n, \mathcal{P}))$$

Therefore, it is sufficient to prove that the actions preserve this condition. The logical value of $\textit{last-prepared}(m, v', n, \mathcal{P}))$ does not change (for all m', m, n, v') because the view-change messages in in_i are immutable.

To prove that the value of $\textit{last-prepared-b}(m, v', n, i, v)$ is also preserved (for all m', m, n, v'), we will first prove the following invariant (b): For any reachable state in an execution of A, any non-faulty replica R_i, and any view-change message $m' = \langle$VIEW-CHANGE$, v, P, i\rangle_{\sigma_i}$, $m' \in in_i \Rightarrow view_i \geq v$.

The proof for (b) is by induction on the length of the execution. It is vacuously true in the base case. For the inductive step, the only actions that can violate (b) are actions that insert view-change messages signed by R_i in in_i or actions that make $view_i$ smaller. Since there are no actions that make $view_i$ smaller, these actions have the form VIEW-CHANGE$(v)_i$. The effects of actions of this form ensure the invariant is preserved by setting $view_i$ to the view number in the view-change message.

Given (b) it is easy to see that the other types of actions do not violate condition 1 of the lemma. They only insert pre-prepare or prepare messages in in_i whose view number is equal to $view_i$ after the action executes. Invariant (b) guarantees that $view_i$ is greater than or equal to the view number v of any view-change message in in_i. Therefore, these actions cannot change the value of $\textit{last-prepared-b}(m, v', n, i, v)$ for any m', m, n, v'.

Condition (2) of the lemma can only be violated by actions that insert new-view messages in in_i or remove pre-prepare messages from in_i. Since no action ever removes pre-prepare messages from in_i, the only actions that can violate condition (2) are: RECEIVE($\langle$NEW-VIEW$, v, V, O, N\rangle_{\sigma_j})_i$ and SEND-NEW-VIEW$(v, V)_i$. The first type of action preserves condition (2) because it inserts all the pre-prepares in $O \cup N$ in in_i whenever it inserts the argument new-view message in in_i. The second type of action preserves condition (2) in a similar way. □

Invariant 4.9 *The following is true of any reachable state in an execution of A,*

$$
\begin{array}{l}
\forall\, i \in \mathcal{R},\ m \in \mathcal{M},\ v, n \in \mathbf{N} : ((\neg \textit{faulty}_i \ \wedge\ \textit{n-faulty} \le f \ \wedge \\
\quad \exists\, R : (|R| > f \ \wedge\ \forall\, k \in R : (\neg \textit{faulty}_k \ \wedge\ \textit{prepared}(m, v, n, k))) \Rightarrow \\
\qquad \forall\, v' > v \in \mathbf{N},\ m' \in \mathcal{M} : (\langle \text{PRE-PREPARE}, v', n, m' \rangle_{\sigma_{\textit{primary}(v')}} \in \textit{in}_i \Rightarrow m' = m))
\end{array}
$$

Proof: Rather than proving the invariant directly, we will prove the following condition is true:

$$
\begin{array}{l}
\forall\, i \in \mathcal{R},\ m \in \mathcal{M},\ v, n \in \mathbf{N} : ((\neg \textit{faulty}_i \ \wedge\ \textit{n-faulty} \le f \ \wedge \\
\quad \exists\, R : (|R| > f \ \wedge\ \forall\, k \in R : (\neg \textit{faulty}_k \ \wedge\ \textit{prepared}(m, v, n, k))) \Rightarrow \\
\qquad \forall\, v' > v \in \mathbf{N},\ \langle \text{NEW-VIEW}, v', V, O, N \rangle_{\sigma_{\textit{primary}(v')}} \in \textit{in}_i : \\
\qquad\qquad (\langle \text{PRE-PREPARE}, v', n, m \rangle_{\sigma_{\textit{primary}(v')}} \in O))
\end{array}
$$

Condition (a) implies the invariant. Invariant 4.2 (condition 4) states that there is never a pre-prepare message in $\textit{in}_i$ for a view $v' > 0$ without a correct new-view message in $\textit{in}_i$ for the same view. But if there is a correct new-view message $\langle \text{NEW-VIEW}, v', V, O, N \rangle_{\sigma_{\textit{primary}(v')}} \in \textit{in}_i$ then Invariant 4.8 (condition 2) implies that $(O \cup N) \subseteq \textit{in}_i$. This and condition (a) imply that there is a $\langle \text{PRE-PREPARE}, v', n, m \rangle_{\sigma_{\textit{primary}(v')}} \in \textit{in}_i$ and Invariant 4.2 (conditions 1,2 and 8) implies that no different pre-prepare message for sequence number n and view v' is ever in $\textit{in}_i$.

The proof is by induction on the number of views between v and v'. For the base case, $v = v'$, condition (a) is vacuously true. For the inductive step, assume condition (a) holds for v'' such that $v < v'' < v'$. We will show that it also holds for v'. Assume there exists a new-view message $m_1 = \langle \text{NEW-VIEW}, v', V_1, O_1, N_1 \rangle_{\sigma_{\textit{primary}(v')}}$ in $\textit{in}_i$ (otherwise (a) is vacuously true.) From Invariant 4.2 (condition 5), this message must verify $\textit{correct-new-view}(m_1, v')$. This implies that it must contain $2f+1$ correct view-change messages for view v' from replicas in some set R_1.

Assume that the following condition is true (b) $\exists R : (|R| > f \wedge \forall k \in R : (\textit{faulty}_k = \textit{false} \wedge \textit{prepared}(m, v, n, k) = \textit{true}))$ (otherwise (a) is vacuously true.) Since there are only $3f+1$ replicas, R and R_1 intersect in at least one replica and this replica is not faulty; call this replica k. Let k's view-change message in m_1 be $m_2 = \langle \text{VIEW-CHANGE}, v', P_2, k \rangle_{\sigma_k}$.

Since k is non-faulty and $\textit{prepared}(m, v, n, k) = \textit{true}$, Invariant 4.4 implies that $\textit{last-prepared-b}(m, v, n, k, v+1)$ is true. Therefore, one of the following conditions is true:

1. $\textit{last-prepared-b}(m, v, n, k, v')$
2. $\exists v'', m' : (v < v'' < v' \wedge \textit{last-prepared-b}(m', v'', n, k, v'))$

Since condition (a) implies the invariant, the inductive hypothesis implies that $m = m'$ in the second case. Therefore, Invariants 4.1 and 4.8 imply that (c) $\exists v_2 \ge v : \textit{last-prepared}(m, v_2, n, P_2)$

Condition (c), Invariant 4.5, and the fact that $\textit{correct-new-view}(m_1, v')$ is true imply that one of the following conditions is true:

1. $\textit{last-prepared}(m, v_2, n, \textit{merge-P}(V_1))$
2. $\exists v'', m' : (v_2 < v'' < v' \wedge \textit{last-prepared}(m', v'', n, \textit{merge-P}(V_1)))$

In case (1), (a) is obviously true. If case (2) holds, Invariant 4.1 and Invariant 4.2 (condition 7) imply that there exists at least one non-faulty replica j such that $\langle \text{PRE-PREPARE}, v'', n, m' \rangle_{\sigma_{\textit{primary}(v'')}} \in \textit{in}_j$. Since condition (a) implies the invariant, the inductive hypothesis implies that $m = m'$ in the second case. □

Invariant 4.10 *The following is true of any reachable state in an execution of A,*

$$\forall\, n, v, v' \in \mathbf{N},\ m, m' \in \mathcal{M} : (\textit{n-faulty} \leq f \Rightarrow$$
$$(\exists\, R \subseteq \mathcal{R} : (|R| > f \ \wedge \forall\, k \in R : (\neg \textit{faulty}_k \wedge \textit{prepared}(m, v, n, k))) \wedge$$
$$\exists\, R' \subseteq \mathcal{R} : (|R'| > f \ \wedge \forall\, k \in R' : (\neg \textit{faulty}_k \wedge \textit{prepared}(m', v', n, k)))) \Rightarrow D(m) = D(m'))$$

Proof: Assume without loss of generality that $v \leq v'$. For the case $v = v'$, the negation of this invariant implies that there exist two requests m and m' ($D(m') \neq D(m)$), a sequence number n, and two non-faulty replicas R_i, R_j, such that *prepared*$(m, v, n, i) =$ *true* and *prepared*$(m', v, n, j) =$ *true*; this contradicts Invariant 4.4.

For $v > v'$, assume this invariant is false. The negation of the invariant and the definition of the prepared predicate imply:

$$\exists\, n, v, v' \in \mathbf{N},\ m, m' \in \mathcal{M} : (v > v' \wedge \textit{n-faulty} \leq f \wedge$$
$$(\exists\, R \subseteq \mathcal{R} : (|R| > f \ \wedge \forall\, k \in R : (\neg \textit{faulty}_k \wedge \textit{prepared}(m, v, n, k))) \wedge$$
$$\exists\, i \in \mathcal{R} : (\neg \textit{faulty}_i \wedge \langle \text{PRE-PREPARE}, v', n, m' \rangle_{\sigma_{primary(v')}} \in \textit{in}_i) \wedge D(m) \neq D(m'))$$

But this contradicts Invariant 4.9 as long as the probability that $m \neq m'$ while $D(m) = D(m')$ is negligible. □

Invariant 4.11 *The following is true of any reachable state in an execution of A,*

$$\forall\, i, j \in \mathcal{R},\ n, v, v' \in \mathbf{N},\ m, m' \in \mathcal{M} : ((\neg \textit{faulty}_i \wedge \neg \textit{faulty}_j \wedge \textit{n-faulty} \leq f) \Rightarrow$$
$$(\textit{committed}(m, v, n, i) \wedge \textit{committed}(m', v', n, j) \Rightarrow D(m) = D(m')))$$

Invariant 4.12 *The following is true of any reachable state in an execution of A,*

$$\forall\, i \in \mathcal{R},\ n, v, v' \in \mathbf{N},\ m, m' \in \mathcal{M} : ((\neg \textit{faulty}_i \wedge \textit{n-faulty} \leq f) \Rightarrow (\textit{committed}(m, v, n, i) \wedge$$
$$\exists\, R' \subseteq \mathcal{R} : (|R'| > f \ \wedge \forall\, k \in R' : (\neg \textit{faulty}_k \wedge \textit{prepared}(m', v', n, k)))) \Rightarrow D(m) = D(m'))$$

Proof: Both Invariant 4.11 and 4.12 are implied by Invariants 4.10 and 4.7. □

Rather than proving that A implements S directly, we will prove that A implements S', which implements S and is better suited for the proof. We start by defining a set of auxiliary functions that will be useful for the proof.

Definition 4.13 *We define the following functions inductively:*

$$\textit{val} : (\mathbf{N} \times \mathcal{O}' \times \mathbf{N} \times \mathcal{C})^* \rightarrow \mathcal{V}$$
$$\textit{last-rep} : (\mathbf{N} \times \mathcal{O}' \times \mathbf{N} \times \mathcal{C})^* \rightarrow (\mathcal{C} \rightarrow \mathcal{O}')$$
$$\textit{last-rep-t} : (\mathbf{N} \times \mathcal{O}' \times \mathbf{N} \times \mathcal{C})^* \rightarrow (\mathcal{C} \rightarrow \mathbf{N})$$

$$\textit{val}(\lambda) = v_o$$
$$\forall\, c : (\textit{last-rep}(\lambda)(c) = \textit{null-rep})$$
$$\forall\, c : (\textit{last-rep-t}(\lambda)(c) = 0)$$

$$\textit{val}(\mu.\langle n, o, t, c \rangle) = s$$

$last\text{-}rep(\mu.\langle n, o, t, c\rangle)(c) \ = \ r$
$last\text{-}rep\text{-}t(\mu.\langle n, o, t, c\rangle)(c) \ = \ t$
$\forall\, c' \ \neq \ c \ : \ (last\text{-}rep(\mu.\langle n, o, t, c\rangle)(c') \ = \ last\text{-}rep(\mu)(c'))$
$\forall\, c' \ \neq \ c \ : \ (last\text{-}rep\text{-}t(\mu.\langle n, o, t, c\rangle)(c') \ = \ last\text{-}rep\text{-}t(\mu)(c'))$
where $(r, s) \ = \ g(c, o, val(\mu))$

Automaton S' has the same signature as S except for the addition of an internal action EXECUTE-NULL. It also has the same state components except that the *val* component is replaced by a sequence of operations:

$hist \in (\mathbf{N} \times \mathcal{O}' \times \mathbf{N} \times \mathcal{C})^*$, initially λ;

and there is a new *seqno* component:

$seqno \in \mathbf{N}$, initially 0.

Similarly to S, the transitions for S' are only defined when $n\text{-}faulty \leq f$. Also, the transitions for S' are identical to S's except for those defined bellow.

EXECUTE(o, t, c)
Pre: $\langle o, t, c\rangle \in in$
Eff: $seqno := seqno + 1$
$in := in - \{\langle o, t, c\rangle\}$
if $t > last\text{-}rep\text{-}t(hist)(c)$ then
$hist := hist.\langle seqno, o, t, c\rangle$
$out := out \cup \{\langle last\text{-}rep(c), t, c\rangle\}$

EXECUTE-NULL
Eff: $seqno := seqno + 1$

The EXECUTE-NULL actions allow the *seqno* component to be incremented without removing any tuple from *in*. This is useful to model execution of *null* requests.

Theorem 4.14 *S' implements S*

Proof: The proof uses a *forward simulation* [8] $\mathcal{F}$ from S' to S. $\mathcal{F}$ is defined as follows:

Definition 4.15 *$\mathcal{F}$ is a subset of* states(S') $\times$ states(S)*; (x, y) is an element of $\mathcal{F}$ (also written as $y \in \mathcal{F}[x]$) if and only if all the following conditions are satisfied:*

1. All state components with the same name are equal in x and y.
2. $x.val = val(y.hist)$
3. $x.last\text{-}rep\text{-}t_c = last\text{-}rep(y.hist)(c), \forall c \in \mathcal{C}$

To prove that $\mathcal{F}$ is in fact a forward simulation from S' to S one most prove that both of the following are true [8].

1. For all $x \in start(S')$, $\mathcal{F}[x] \cap start(S) \neq \{\}$
2. For all $(x, \pi, x') \in trans(S')$, where x is a reachable state of S', and for all $y \in \mathcal{F}[x]$, where y is reachable in S, there exists an execution fragment α of S starting with y and ending with some $y' \in \mathcal{F}[x']$ such that $trace(\alpha) = trace(\pi)$.

It is clear that $\mathcal{F}$ verifies the first condition because all variables with the same name in S and S' are initialized to the same values and, since *hist* is initially equal to λ, $x.val = v_o = val(\lambda)$ and $x.last\text{-}rep\text{-}t_c = 0 = last\text{-}rep(\lambda)(c)$.

We use case analysis to show that the second condition holds for each $\pi \in acts(S')$. For all actions π except EXECUTE-NULL, let α consist of a single π step. For π = EXECUTE-NULL, let α be λ. It is clear that this satisfies the second condition for all actions but EXECUTE. For $\pi = \text{EXECUTE}(o,t,c)$, definition 4.13 and the inductive hypothesis (i.e., $x.val = val(y.hist)$ and $x.last\text{-}rep\text{-}t_c = last\text{-}rep(y.hist)(c)$) ensure that $y' \in \mathcal{F}[x']$. □

Definition 4.16 *We define the function prefix :* $(\mathbf{N}\times\mathcal{O}'\times\mathbf{N}\times\mathcal{C})^* \rightarrow (\mathbf{N}\times\mathcal{O}'\times\mathbf{N}\times\mathcal{C})^*$ *as follows: prefix*(μ, n) *is the subsequence obtained from* μ *by removing all tuples whose first component is greater than* n.

Invariant 4.17 *The following is true of any reachable state in an execution of* S',

$\forall\ \langle n,o,t,c\rangle \in hist : (t > last\text{-}rep\text{-}t(prefix(hist,\ n-1))(c))$

Proof: The proof is by induction on the length of the execution. The initial states of S' verify the condition vacuously because *hist* is initially λ. For the inductive step, the only actions that can violate the invariant are those that modify *hist*, i.e., EXECUTE(o,t,c). But these actions only modify *hist* if $t > last\text{-}rep\text{-}t(hist)(c)$. □

Invariant 4.18 *The following are true of any reachable state in an execution of* S'*:*
1. $\forall\langle n,o,t,c\rangle \in hist : (\neg faulty_c \Rightarrow t \leq last\text{-}req_c)$
2. $\forall\langle o,t,c\rangle \in in : (\neg faulty_c \Rightarrow t \leq last\text{-}req_c)$

Proof: The proof is by induction on the length of the execution. The initial states of S' verify the condition vacuously because *hist* is initially λ and *in* is empty. For the inductive step, since no action ever decrements $last\text{-}req_c$ or changes $faulty_c$ from true to false, the only actions that can violate the invariant are those that append tuples from a non-faulty client c to *hist*, i.e., EXECUTE(o,t,c) or to *in*, REQUEST(o,c). The EXECUTE actions only append a tuple $\langle n,o,t,c\rangle$ to *hist* if $\langle o,t,c\rangle \in in$; therefore, the inductive hypothesis for condition 2 implies that they preserve the invariant. The REQUEST actions also preserve the invariant because the tuple $\langle o,t,c\rangle$ inserted in *in* has t equal to the value of $last\text{-}req_c$ after the action executes. □

Theorem 4.19 *A implements S*

Proof: We prove that A implements S', which implies that A implements S (Theorem 4.14.) The proof uses a *forward simulation* $\mathcal{G}$ from A' to S' (A' is equal to A but with all output actions not in the external signature of S hidden.) $\mathcal{G}$ is defined as follows.

Definition 4.20 *$\mathcal{G}$ is a subset of* states(A') $\times$ states(S')*; (x, y) is an element of $\mathcal{G}$ if and only if the following are satisfied:*

1. $\forall i \in \mathcal{R} : (x.faulty_i = y.faulty\text{-}replica_i)$
2. $\forall c \in \mathcal{C} : (x.faulty_c = y.faulty\text{-}client_c)$

and the following are satisfied when n-faulty $\leq f$

3. $\forall c \in \mathcal{C} : (\neg x.faulty_c \Rightarrow x.last\text{-}req_c = y.last\text{-}req_c)$
4. $\forall i \in \mathcal{R} : (\neg x.faulty_i \Rightarrow x.last\text{-}exec_i \leq y.seqno)$
5. $\forall i \in \mathcal{R} : (\neg x.faulty_i \Rightarrow x.val_i = val(prefix(y.hist, x.last\text{-}exec_i)))$
6. $\forall i \in \mathcal{R} : (\neg x.faulty_i \Rightarrow \forall c \in \mathcal{C} : (x.last\text{-}rep_i(c) = last\text{-}rep(prefix(y.hist, x.last\text{-}exec_i))(c)))$
7. $\forall i \in \mathcal{R} : (\neg x.faulty_i \Rightarrow \forall c \in \mathcal{C} : (x.last\text{-}rep\text{-}t_i(c) = last\text{-}rep\text{-}t(prefix(x.hist, y.last\text{-}exec_i))(c)))$
8. $\forall 0 < n \leq y.seqno : (\exists \langle n, o, t, c\rangle \in y.hist : (\exists R \subseteq \mathcal{R}, v \in \mathbf{N} : (|R| > 2f - y.n\text{-}faulty \wedge$
   $\forall k \in R : (\neg x.faulty_k \wedge prepared(\langle \text{REQUEST}, o, t, c\rangle_{\sigma_c}, v, n, A'.k))))$
   $\vee \exists R \subseteq \mathcal{R}, v, t \in \mathbf{N}, o \in \mathcal{O}, c \in \mathcal{C} : (|R| > 2f - y.n\text{-}faulty \wedge t \leq last\text{-}rep\text{-}t(prefix(y.hist, n-1))(c)) \wedge$
   $\forall k \in R : (\neg x.faulty_k \wedge prepared(\langle \text{REQUEST}, o, t, c\rangle_{\sigma_c}, v, n, A'.k))))$
   $\vee \exists R \subseteq \mathcal{R}, v \in \mathbf{N} : (|R| > 2f - y.n\text{-}faulty \wedge \forall k \in R : (\neg x.faulty_k \wedge prepared(\langle null, v, n, A'.k))))$
9. $\forall \langle \text{REPLY}, v, t, c, i, r\rangle_{\sigma_i} \in (x.out_i \cup \{m | \exists X : (m, X) \in x.wire\} \cup x.in_c) :$
   $(\neg x.faulty_i \Rightarrow \exists \langle n, o, t, c\rangle \in y.hist : (r = last\text{-}rep(prefix(y.hist, n))(c)))$
10. $\forall \langle n, o, y.last\text{-}req_c, c\rangle \in y.hist :$
    $((\neg x.faulty_c \wedge x.out_c \neq \{\}) \Rightarrow \exists \langle last\text{-}rep(prefix(y.hist, n))(c), y.last\text{-}req_c, c\rangle \in y.out)$
11. Let $M_c = x.out_c \cup \{m | \exists i \in \mathcal{R} : (\neg x.faulty_i \wedge m \in x.in_i \cup x.out_i\} \cup \{m | \exists X : (m, X) \in x.wire\}$,
    and $M_c^1 = merge\text{-}P(\{m = \langle \text{VIEW-CHANGE}, v, P, j\rangle_{\sigma_j} | m \in M_c \vee$
    $\exists \langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_j} \in M_c : (m \in V)\})$,
    $\forall c \in \mathcal{C} : (\neg x.faulty_c \Rightarrow \forall o \in \mathcal{O}, t \in \mathbf{N} : ((m = \langle \text{REQUEST}, o, t, c\rangle_{\sigma_c} \in M_c \vee$
    $\exists \langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_j} \in M_c \cup M_c^1) \Rightarrow (\langle o, t, c\rangle \in y.in \vee \exists n : (\langle n, o, t, c\rangle \in y.hist))))$

Note that most of the conditions in the definition of $\mathcal{G}$ only need to hold when *n-faulty* $\leq f$, for *n-faulty* $> f$ any relation will do because the behavior of S' is unspecified. To prove that $\mathcal{G}$ is in fact a forward simulation from A' to S' one most prove that both of the following are true.

1. For all $x \in start(A')$, $\mathcal{G}[x] \cap start(S') \neq \{\}$
2. For all $(x, \pi, x') \in trans(A')$, where x is a reachable state of A', and for all $y \in \mathcal{G}[x]$, where y is reachable in S', there exists an execution fragment α of S' starting with y and ending with some $y' \in \mathcal{G}[x']$ such that $trace(\alpha) = trace(\pi)$.

It is easy to see that the first condition holds. We use case analysis to show that the second condition 2 holds for each $\pi \in acts(A')$

Non-faulty proxy actions. If $\pi = \text{REQUEST}(o)_c$, $\pi = \text{CLIENT-FAILURE}_c$, or $\pi = \text{REPLY}(r)_c$, let α consist of a single π step. $\mathcal{G}$ is preserved in a trivial way if π is a CLIENT-FAILURE action. If π is a REQUEST action, neither π nor α modify the variables involved in all conditions in the definition of $\mathcal{G}$ except 3, and 10 and 11. Condition 3 is preserved because both π and α increment $y.last\text{-}req_c$. Condition 10 is also preserved because Invariant 4.18 implies that there are no tuples in $y.hist$ with timestamp $y'.last\text{-}req_c$ and α does not add any tuple to $y.hist$. Even though π inserts a new request in $x.out_c$, condition 11 is preserved because α inserts $\langle o, t, c\rangle$ in $y.in$.

If π is a REPLY$(r)_c$ action that is enabled in x, the REPLY$(r)_c$ action in α is also enabled. Since there are less than f faulty replicas, the precondition of π ensures that there is at least one non-faulty replica i and a view v such that $\langle \text{REPLY}, v, x.\textit{last-req}_c, c, i, r\rangle_{\sigma_i} \in x.\textit{in}_c$ and that $x.\textit{out}_c \neq \{\}$. Therefore, the inductive hypothesis (conditions 9 and 10) implies that $\langle r, t, c\rangle \in y.\textit{out}$ and thus REPLY$(r)_c$ is enabled. $\mathcal{G}$ is preserved because π ensures that $x'.\textit{out}_c = \{\}$.

If $\pi =$ RECEIVE$(m)_c$, or $\pi =$ SEND$(m, X)_c$, let α be λ. This preserves $\mathcal{G}$ because $y \in \mathcal{G}[x]$ and the preconditions require that the reply message being received is in some tuple in $x.\textit{wire}$ and the request message being sent is in $x.\textit{out}_c$.

Internal channel actions. If π is a MISBEHAVE(m, X, X') action, let α be λ. $\mathcal{G}$ is preserved because π does not add new messages to $x.\textit{wire}$ and retains a tuple with m on $x'.\textit{wire}$.

Non-faulty replica actions. For all actions π except $\pi =$ REPLICA-FAILURE$_i$ and $\pi =$ EXECUTE$(m, v, n)_i$, let α be λ. It is clear that this could only violate conditions 8, 9 and 11 because these actions do not modify the state components involved in the other conditions. They can not violate condition 8; since no messages are ever removed from $\textit{in}_k$ (where k is any non-faulty replica), if $\textit{prepared}(m, v, n, k) = \textit{true}$, it remains true for the entire execution or until replica k becomes faulty. And these actions do not violate conditions 9 and 11 because any request or reply messages they add to $x.\textit{in}_i$, $x.\textit{out}_i$, or $x.\textit{wire}$ (either directly or as part of other messages) was already in $x.\textit{wire}$, $x.\textit{in}_i$, or $x.\textit{out}_i$.

For $\pi =$ REPLICA-FAILURE$_i$, let α consist of a single π step. This does not violate the conditions in the definition of $\mathcal{G}$. For conditions other than 1 and 8, it either does not change variables involved in these conditions (2 and 3), or makes them vacuously true. Condition 1 is satisfied in a trivial way because α also sets $y.\textit{faulty-replica}_i$ to true. And condition 8 is not violated because the size of the sets R in the condition is allowed to decrease when additional replicas become faulty.

Non-faulty replica execute (non-null request.)

For $\pi =$ EXECUTE$(\langle \text{REQUEST}, o, t, c\rangle_{\sigma_c}, v, n)_i$, there are two cases: if $x.\textit{last-exec}_i < y.\textit{seqno}$, let α be λ; otherwise, let α consist of the execution of a single EXECUTE(o, t, c) action preceeded by FAULTY-REQUEST(o, t, c) in the case where $x.\textit{faulty}_c = \textit{true}$. In any of these cases, it is clear that only conditions 4 to 11 can be violated.

For the case where $\alpha = \lambda$, conditions 4, 8, 10 and 11 are also preserved in a trivial way. For the other conditions we consider two cases (a) $t > \textit{last-rep-t}_i(c)$ and (b) otherwise. The precondition of π ensures that $x.\textit{committed}(\langle \text{REQUEST}, o, t, c\rangle_{\sigma_c}, v, n, i)$ is true. In case (a), this precondition, Invariant 4.12, and the definition of $\mathcal{G}$ (condition 8) imply that there is a tuple in $y.\textit{hist}$ with sequence number n and that it is equal to $\langle n, o, t, c\rangle$. Therefore, conditions 5 to 7 and 9 are preserved. In case (b), the precondition of π, Invariant 4.12, the definition of $\mathcal{G}$ (condition 8), and Invariant 4.17 imply that there is no tuple with sequence number n in $y.\textit{hist}$. Therefore, conditions 5 to 9 are preserved in this case.

For the case where $\alpha \neq \lambda$, when π is enabled in x the actions in α are also enabled in y. In the case where c is faulty, FAULTY-REQUEST(o, t, c) is enabled and its execution enables EXECUTE(o, t, c). Otherwise, since $y \in \mathcal{G}[x]$, condition 11 in Definition 4.20 and the precondition of π imply that EXECUTE(o, t, c) is enabled in y.

It is easy to see that conditions 4 to 7 and 9 to 11 are preserved. For condition 8, we consider two cases (a) $t > \textit{last-rep-t}_i(c)$ and (b) otherwise. In both cases, the precondition of π ensures that $x.\textit{committed}(\langle \text{REQUEST}, o, t, c\rangle_{\sigma_c}, v, n, i)$ is true. This precondition, Invariant 4.7 and the fact that α appends a tuple $\langle y'.\textit{seqno}, o, t, c\rangle$ to $y.\textit{hist}$, ensure that condition 8 is preserved in this case. In case (b), the precondition Invariant 4.7 and the assumption that $t \leq \textit{last-rep-t}_i(c)$, ensure that condition 8 is preserved also in this case.

Non-faulty replica execute (null request.)

For $\pi = \text{EXECUTE}(null, v, n)_i$, if $x.\textit{last-exec}_i < y.\textit{seqno}$, let α be λ; otherwise, let α consist of the execution of a single EXECUTE-NULL action. Execution of a *null* request only increments $x.\textit{last-exec}_i$ and α can at most increment $y.\textit{seqno}$. Therefore, only conditions 4 to 8 can be violated. Condition 4 is not violated because α increments $y.\textit{seqno}$ in the case where $x.\textit{last-exec}_i = y.\textit{seqno}$.

For the case where, $\alpha = \lambda$, conditions 5 to 7 are also not violated because α does not append any new tuple to $y.\textit{hist}$ and all tuples in $y.\textit{hist}$ have sequence number less than $y'.\textit{seqno}$; therefore, $\textit{prefix}(y.hist, x.\textit{last-exec}_i) = \textit{prefix}(y'.hist, x'.\textit{last-exec}_i)$. Since the precondition of π implies that $x.\textit{committed}(null, v, n, i)$ is true, Invariant 4.7 ensures condition 8 is also preserved in this case.

For the case where α consists of a EXECUTE-NULL step, $x.\textit{committed}(null, v, n, i)$, $\textit{n-faulty} \leq f$, Invariant 4.12, and the definition of $\mathcal{G}$ (condition 8) imply that there is no tuple in $y'.\textit{hist}$ with sequence number $x'.\textit{last-exec}_i$; therefore, $\textit{prefix}(y.hist, x.\textit{last-exec}_i) = \textit{prefix}(y'.hist, x'.\textit{last-exec}_i)$.

Faulty replica actions. If π is an action of a faulty replica i (i.e., $x.\textit{faulty}_i = \textit{true}$), let α be λ. Since π can not modify $\textit{faulty}_i$ and a faulty replica cannot forge the signature of a non-faulty automaton this preserves $\mathcal{G}$ in a trivial way.

Faulty proxy actions. If π is an action of a faulty proxy c (i.e., $x.\textit{faulty}_c = \textit{true}$), let α consist of a single π step for REQUEST, REPLY and CLIENT-FAILURE actions and λ for the other actions. Since π can not modify $\textit{faulty}_c$ and faulty clients cannot forge signatures of non-faulty automata this preserves $\mathcal{G}$ in a trivial way. Additionally, if π is a REPLY action enabled in x, π is also enabled in y. □

5 Garbage Collection

This section describes a modified version of our algorithm that garbage collects messages from replica's logs. It also proves that the modified algorithm A_{gc} is safe, i.e., it proves that it implements S.

5.1 The Modified Algorithm

The client proxy and multicast channel automata are identical in A_{gc} and A. The replica automaton R_i is modified as follows. The signature remains the same except for the actions listed below.

Input: $\text{RECEIVE}(\langle \text{CHECKPOINT}, v, n, d, j\rangle_{\sigma_j})_i$

$\text{RECEIVE}(\langle \text{VIEW-CHANGE}, v, n, s, C, P, j\rangle_{\sigma_j})$

Internal: COLLECT-GARBAGE_i

Here, $v, n \in \mathbf{N}$, $i, j \in \mathcal{R}$, $s \in \mathcal{V}'$, $C, P \subseteq \mathcal{M}$, $d \in \mathcal{D}'$
where $\mathcal{V}' = \mathcal{V} \times (\mathcal{C} \rightarrow \mathcal{O}) \times (\mathcal{C} \rightarrow \mathbf{N})$ and $\mathcal{D}' = \{d \mid \exists s \in \mathcal{V}' : (d = D(m))\}$

The state components also remain the same except for the addition of a new variable chkpts_i and a new initial value for in_i:

$\mathit{in}_i \subseteq \mathcal{M}$, initially $\{\langle \text{CHECKPOINT}, 0, D(\langle v_0, \mathit{null\text{-}rep}, 0\rangle), k\rangle_{\sigma_k} \mid \forall k \in \mathcal{R}\}$
$\mathit{chkpts}_i \subseteq \mathbf{N} \times \mathcal{V}'$, initially $\{\langle 0, \langle v_0, \mathit{null\text{-}rep}, 0\rangle\rangle\}$
$\mathit{stable\text{-}n}_i \equiv \mathit{min}(\{n \mid \langle n, \nu\rangle \in \mathit{chkpts}_i\})$
$\mathit{stable\text{-}chkpt}_i \equiv \nu \mid \langle \mathit{stable\text{-}n}_i, \nu\rangle \in \mathit{chkpts}_i$

The auxiliary functions used in the description of a replica's automaton also remain the same except for those that are defined below:

$\mathit{in\text{-}w}(n, i) \equiv 0 < n - \mathit{stable\text{-}n}_i \leq \mathit{max\text{-}out}$, where $\mathit{max\text{-}out} \in \mathbf{N}$
$\mathit{in\text{-}wv}(v, n, i) \equiv \mathit{in\text{-}w}(n, i) \wedge \mathit{in\text{-}v}(v, i)$
$\mathit{correct\text{-}view\text{-}change}(m, v, j) \equiv \exists\, n, s, C, P : (m = \langle \text{VIEW-CHANGE}, v, n, s, C, P, j\rangle_{\sigma_j} \wedge$
$\quad \exists\, R : (|R| > f \wedge \forall\, k \in R : (\exists\, v'' < v : (\langle \text{CHECKPOINT}, v'', n, D(s), k\rangle_{\sigma_k} \in C)) \wedge$
$\quad \forall\, \langle \text{PRE-PREPARE}, v', n', m'\rangle_{\sigma_{\mathit{primary}(v')}} \in P :$
$\qquad (\mathit{last\text{-}prepared}(m', v', n', P) \wedge v' < v \wedge 0 < n' - n \leq \mathit{max\text{-}out})$
$\mathit{merge\text{-}P}(V) \equiv \{\, m \mid \exists\, \langle \text{VIEW-CHANGE}, v, n, s, C, P, k\rangle_{\sigma_k} \in V : (m \in P)\,\}$
$\mathit{max\text{-}n}(M) \equiv \mathit{max}(\{\, n \mid \langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_i} \in M \vee \langle \text{VIEW-CHANGE}, v, n, s, C, P, i\rangle_{\sigma_i} \in M\})$
$\mathit{correct\text{-}new\text{-}view}(m, v) \equiv$
$\quad \exists\, V, O, N, R : (m = \langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_{\mathit{primary}(v)}} \wedge |V| = |R| = 2f + 1 \wedge$
$\quad \forall\, k \in R : (\exists\, m' \in V : (\mathit{correct\text{-}view\text{-}change}(m', v, k))) \wedge$
$\quad O = \{\, \langle \text{PRE-PREPARE}, v, n, m'\rangle_{\sigma_{\mathit{primary}(v)}} \mid n > \mathit{max\text{-}n}(V) \wedge \exists\, v' : \mathit{last\text{-}prepared}(m', v', n, \mathit{merge\text{-}P}(V))\} \wedge$
$\quad N = \{\, \langle \text{PRE-PREPARE}, v, n, \mathit{null}\rangle_{\sigma_{\mathit{primary}(v)}} \mid \mathit{max\text{-}n}(V) < n < \mathit{max\text{-}n}(O) \wedge$
$\qquad \nexists\, v', m', n : \mathit{last\text{-}prepared}(m', v', n, \mathit{merge\text{-}P}(V)))$
$\mathit{take\text{-}chkpt}(n) \equiv (n \bmod \mathit{chkpt\text{-}int}) = 0$, where $\mathit{chkpt\text{-}int} \in \mathbf{N} \wedge \mathit{chkpt\text{-}int} < \mathit{max\text{-}out}$
$\mathit{update\text{-}state\text{-}nv}(i, v, V, m) \equiv$
 if $\mathit{max\text{-}n}(V) > \mathit{stable\text{-}n}_i$ then
  $\mathit{in}_i := \mathit{in}_i \cup (\text{pick } C : \exists\, \langle \text{VIEW-CHANGE}, v, \mathit{max\text{-}n}(V), s, C, P, k\rangle_{\sigma_k} \in V)$
  if $\langle \text{CHECKPOINT}, v, \mathit{max\text{-}n}(V), D(s), i\rangle_{\sigma_i} \notin \mathit{in}_i$ then
   $\mathit{in}_i = \mathit{in}_i \cup \{\langle \text{CHECKPOINT}, v, \mathit{max\text{-}n}(V), D(s), i\rangle_{\sigma_i}\}$
   $\mathit{out}_i = \mathit{out}_i \cup \{\langle \text{CHECKPOINT}, v, \mathit{max\text{-}n}(V), D(s), i\rangle_{\sigma_i}\}$
  $\mathit{chkpts}_i := \mathit{chkpts}_i - \{p = \langle n', s'\rangle \mid p \in \mathit{chkpts}_i \wedge n' < \mathit{max\text{-}n}(V)\}$
  if $\mathit{max\text{-}n}(V) > \mathit{last\text{-}exec}_i$ then
   $\mathit{chkpts}_i := \mathit{chkpts}_i \cup \{\langle \mathit{max\text{-}n}(V), s\rangle \mid \exists\, \langle \text{VIEW-CHANGE}, v, \mathit{max\text{-}n}(V), s, C, P, k\rangle_{\sigma_k} \in V\}$
   $(\mathit{val}_i, \mathit{last\text{-}rep}_i, \mathit{last\text{-}rep\text{-}t}_i) := \mathit{stable\text{-}chkpt}_i$
   $\mathit{last\text{-}exec}_i := \mathit{max\text{-}n}(V)$

Many of the actions for automaton R_i are modified to use the new functions but otherwise remain identical. The exceptions are listed below:

Input Transitions

RECEIVE$\langle$PRE-PREPARE$, v, n, m\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: if $j = \mathit{primary}(i) \wedge \mathit{in\text{-}wv}(v, n, i) \wedge \mathit{has\text{-}new\text{-}view}(v, i) \wedge$
$\nexists d : (d \neq D(m) \wedge \langle\text{PREPARE}, v, n, d, i\rangle_{\sigma_i} \in \mathit{in}_i)$ then
let $p = \langle\text{PREPARE}, v, n, D(m), i\rangle_{\sigma_i}$
$\mathit{in}_i := \mathit{in}_i \cup \{\langle\text{PRE-PREPARE}, v, n, m\rangle_{\sigma_j}, p\}$
$\mathit{out}_i := \mathit{out}_i \cup \{p\}$
else if $\exists o, t, c : (m = \langle\text{REQUEST}, o, t, c\rangle_{\sigma_c})$ then
$\mathit{in}_i := \mathit{in}_i \cup \{m\}$

RECEIVE$(\langle$PREPARE$, v, n, d, j\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: if $j \neq \mathit{primary}(i) \wedge \mathit{in\text{-}wv}(v, n, i)$ then
$\mathit{in}_i := \mathit{in}_i \cup \{\langle\text{PREPARE}, v, n, d, j\rangle_{\sigma_j}\}$

RECEIVE$(\langle$COMMIT$, v, n, d, j\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: if $\mathit{view}_i \geq v \wedge \mathit{in\text{-}w}(n, i)$ then
$\mathit{in}_i := \mathit{in}_i \cup \{\langle\text{COMMIT}, v, n, d, j\rangle_{\sigma_j}\}$

RECEIVE$(\langle$CHECKPOINT$, v, n, d, j\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: if $\mathit{view}_i \geq v \wedge \mathit{in\text{-}w}(n, i)$ then
$\mathit{in}_i := \mathit{in}_i \cup \{\langle\text{CHECKPOINT}, v, n, d, j\rangle_{\sigma_j}\}$

RECEIVE$(\langle$VIEW-CHANGE$, v, n, s, C, P, j\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: let $m = \langle\text{VIEW-CHANGE}, v, n, s, C, P, j\rangle_{\sigma_j}$
if $v \geq \mathit{view}_i \wedge \mathit{correct\text{-}view\text{-}change}(m, v, j)$ then
$\mathit{in}_i := \mathit{in}_i \cup \{m\}$

RECEIVE$(\langle$NEW-VIEW$, v, V, O, N\rangle_{\sigma_j})_i \ (j \neq i)$
Eff: let $m = \langle\text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_j}$
if $v > 0 \wedge v \geq \mathit{view}_i \wedge \mathit{correct\text{-}new\text{-}view}(m, v) \wedge \neg\mathit{has\text{-}new\text{-}view}(v, i)$ then
$\mathit{view}_i := v$
$\mathit{out}_i := \{\}$
$\mathit{in}_i := \mathit{in}_i \cup O \cup N \cup \{m\}$
for all $\langle\text{PRE-PREPARE}, v, n', m'\rangle_{\sigma_j} \in (O \cup N)$ do
$\mathit{out}_i := \mathit{out}_i \cup \{\langle\text{PREPARE}, v, n', D(m'), i\rangle_{\sigma_i}\}$
if $n' > \mathit{stable\text{-}n}_i$ then
$\mathit{in}_i := \mathit{in}_i \cup \{\langle\text{PREPARE}, v, n', D(m'), i\rangle_{\sigma_i}\}$
$\mathit{update\text{-}state\text{-}nv}(i, v, V, m)$
$\mathit{in}_i := \mathit{in}_i - \{\langle\text{REQUEST}, o, t, c\rangle_{\sigma_c} \in \mathit{in}_i | t \leq \mathit{last\text{-}rep\text{-}t}_i(c)\}$

Internal Transitions

SEND-PRE-PREPARE$(m, v, n)_i$
Pre: $\mathit{primary}(i) = i \wedge \mathit{seqno}_i = n - 1 \wedge \mathit{in\text{-}wv}(v, n, i) \wedge \mathit{has\text{-}new\text{-}view}(v, i) \wedge$
$\exists o, t, c : (m = \langle\text{REQUEST}, o, t, c\rangle_{\sigma_c} \wedge m \in \mathit{in}_i) \wedge \nexists\langle\text{PRE-PREPARE}, v, n', m\rangle_{\sigma_i} \in \mathit{in}_i$
Eff: $\mathit{seqno}_i := \mathit{seqno}_i + 1$
let $p = \langle\text{PRE-PREPARE}, v, n, m\rangle_{\sigma_i}$
$\mathit{out}_i := \mathit{out}_i \cup \{p\}$
$\mathit{in}_i := \mathit{in}_i \cup \{p\}$

$\text{EXECUTE}(m, v, n)_i$
Pre: $n = \textit{last-exec}_i + 1 \land \textit{committed}(m, v, n, i)$
Eff: $\textit{last-exec}_i := n$
if $(m \neq \textit{null})$ then
let $\langle \text{REQUEST}, o, t, c \rangle_{\sigma_c} = m$
if $t \geq \textit{last-rep-t}_i(c)$ then
if $t > \textit{last-rep-t}_i(c)$ then
$\textit{last-rep-t}_i(c) := t$
$(\textit{last-rep}_i(c), \textit{val}_i) := g(c, o, \textit{val}_i)$
$\textit{out}_i := \textit{out}_i \cup \{\langle \text{REPLY}, \textit{view}_i, t, c, i, \textit{last-rep}_i(c) \rangle_{\sigma_i}\}$
$\textit{in}_i := \textit{in}_i - \{m\}$
if $\textit{take-chkpt}(n)$ then
let $m' = \langle \text{CHECKPOINT}, \textit{view}_i, n, D(\langle \textit{val}_i, \textit{last-rep}_i, \textit{last-rep-t}_i \rangle), i \rangle_{\sigma_i}$
$\textit{out}_i := \textit{out}_i \cup \{m'\}$
$\textit{in}_i := \textit{in}_i \cup \{m'\}$
$\textit{chkpts}_i := \textit{chkpts}_i \cup \{\langle n, \langle \textit{val}_i, \textit{last-rep}_i, \textit{last-rep-t}_i \rangle \rangle\}$

$\text{SEND-VIEW-CHANGE}(v)_i$
Pre: $v = \textit{view}_i + 1$
Eff: $\textit{view}_i := v$
let $P' = \{\langle m, v, n \rangle | \textit{last-prepared}(m, v, n, i)\}$,
$P = \bigcup_{\langle m, v, n \rangle \in P'} (\{p = \langle \text{PREPARE}, v, n, D(m), k \rangle_{\sigma_k} | p \in \textit{in}_i\} \cup \{\langle \text{PRE-PREPARE}, v, n, m \rangle_{\sigma_{\textit{primary}(v)}}\})$,
$C = \{m' = \langle \text{CHECKPOINT}, v'', \textit{stable-n}_i, D(\textit{stable-chkpt}_i), k \rangle_{\sigma_k} | m' \in \textit{in}_i\}$,
$m = \langle \text{VIEW-CHANGE}, v, \textit{stable-n}_i, \textit{stable-chkpt}_i, C, P, i \rangle_{\sigma_i}$
$\textit{out}_i := \textit{out}_i \cup \{m\}$
$\textit{in}_i := \textit{in}_i \cup \{m\}$

$\text{SEND-NEW-VIEW}(v, V)_i$
Pre: $\textit{primary}(v) = i \land v \geq \textit{view}_i \land v > 0 \land V \subseteq \textit{in}_i \land |V| = 2f + 1 \land \neg \textit{has-new-view}(v, i) \land$
$\exists R : (|R| = 2f + 1 \land \forall k \in R : (\exists n, s, C, P : (\langle \text{VIEW-CHANGE}, v, n, s, C, P, k \rangle_{\sigma_k} \in V)))$
Eff: $\textit{view}_i := v$
let $O = \{\langle \text{PRE-PREPARE}, v, n, m \rangle_{\sigma_i} | n > \textit{max-n}(V) \land \exists v' : \textit{last-prepared}(m, v', n, \textit{merge-P}(V))\}$,
$N = \{\langle \text{PRE-PREPARE}, v, n, null, k \rangle_{\sigma_i} | \textit{max-n}(V) < n < \textit{max-n}(O) \land$
$\nexists v', m, n : \textit{last-prepared}(m, v', n, \textit{merge-P}(V))\}$,
$m = \langle \text{NEW-VIEW}, v, V, O, N \rangle_{\sigma_i}$
$\textit{seqno}_i := \textit{max-n}(O)$
$\textit{in}_i := \textit{in}_i \cup O \cup N \cup \{m\}$
$\textit{out}_i := \{m\}$
$\textit{update-state-nv}(i, v, V, m)$
$\textit{in}_i := \textit{in}_i - \{\langle \text{REQUEST}, o, t, c \rangle_{\sigma_c} \in \textit{in}_i | t \leq \textit{last-rep-t}_i(c)\}$

COLLECT-GARBAGE_i
Pre: $\exists R, n, d : (|R| > f \land i \in R \ \land \forall k \in R : (\exists v : (\langle \text{CHECKPOINT}, v, n, d, k \rangle_{\sigma_k} \textit{in}_i)$
Eff: $\textit{in}_i := \textit{in}_i - \{m = \langle \text{PRE-PREPARE}, v', n', m' \rangle_{\sigma_j} | m \in \textit{in}_i \land n' \leq n\}$
$\textit{in}_i := \textit{in}_i - \{m = \langle \text{PREPARE}, v', n', d', j \rangle_{\sigma_j} | m \in \textit{in}_i \land n' \leq n\}$
$\textit{in}_i := \textit{in}_i - \{m = \langle \text{COMMIT}, v', n', d', j \rangle_{\sigma_j} | m \in \textit{in}_i \land n' \leq n\}$
$\textit{in}_i := \textit{in}_i - \{m = \langle \text{CHECKPOINT}, v', n', d', j \rangle_{\sigma_j} | m \in \textit{in}_i \land n' < n\}$
$\textit{chkpts}_i := \textit{chkpts}_i - \{p = \langle n', s \rangle | p \in \textit{chkpts}_i \land n' < n\})$

5.2 Safety Proof

This section proves that A_{gc} implements S. We start by introducing some definitions and proving an invariant.

Definition 5.1 *We define the following functions inductively:*

Let $\mathcal{RM} = \{\langle\text{REQUEST}, o, t, c\rangle_{\sigma_c} \mid o \in \mathcal{O} \wedge t \in \mathbf{N} \wedge c \in \mathcal{C}\} \cup \{\mathit{null}\}$,
$\mathit{r\text{-}val} : \mathcal{RM}^* \rightarrow \mathcal{V}$
$\mathit{r\text{-}last\text{-}rep} : \mathcal{RM}^* \rightarrow (\mathcal{C} \rightarrow \mathcal{O}')$
$\mathit{r\text{-}last\text{-}rep\text{-}t} : \mathcal{RM}^* \rightarrow (\mathcal{C} \rightarrow \mathbf{N})$

$\mathit{r\text{-}val}(\lambda) = v_o$
$\forall\, c \in \mathcal{C} : (\mathit{r\text{-}last\text{-}rep}(\lambda)(c) = \mathit{null\text{-}rep})$
$\forall\, c \in \mathcal{C} : (\mathit{r\text{-}last\text{-}rep\text{-}t}(\lambda)(c) = 0)$

$\forall\, \mu \in \mathcal{RM}^+$,
$\mathit{r\text{-}val}(\mu.\mathit{null}) = \mathit{r\text{-}val}(\mu)$
$\mathit{r\text{-}last\text{-}rep}(\mu.\mathit{null}) = \mathit{r\text{-}last\text{-}rep}(\mu)$
$\mathit{r\text{-}last\text{-}rep\text{-}t}(\mu.\mathit{null}) = \mathit{r\text{-}last\text{-}rep\text{-}t}(\mu)$

$\forall\, \langle\text{REQUEST}, o, t, c\rangle_{\sigma_c} \in \mathcal{RM},\ \mu \in \mathcal{RM}^+$,
$\forall\, c' \neq c : (\mathit{r\text{-}last\text{-}rep}(\mu.\langle\text{REQUEST}, o, t, c\rangle_{\sigma_c})(c') = \mathit{r\text{-}last\text{-}rep}(\mu)(c'))$
$\forall\, c' \neq c : (\mathit{r\text{-}last\text{-}rep\text{-}t}(\mu.\langle\text{REQUEST}, o, t, c\rangle_{\sigma_c})(c') = \mathit{r\text{-}last\text{-}rep\text{-}t}(\mu)(c'))$
if $t > \mathit{r\text{-}last\text{-}rep\text{-}t}(\mu)(c)$ then
let $(r, s) = g(c, o, \mathit{r\text{-}val}(\mu))$
$\mathit{r\text{-}val}(\mu.\langle\text{REQUEST}, o, t, c\rangle_{\sigma_c}) = s$
$\mathit{r\text{-}last\text{-}rep}(\mu.\langle\text{REQUEST}, o, t, c\rangle_{\sigma_c})(c) = r$
$\mathit{r\text{-}last\text{-}rep\text{-}t}(\mu.\langle\text{REQUEST}, o, t, c\rangle_{\sigma_c})(c) = t$
else
$\mathit{r\text{-}val}(\mu.\langle\text{REQUEST}, o, t, c\rangle_{\sigma_c}) = \mathit{r\text{-}val}(\mu)$
$\mathit{r\text{-}last\text{-}rep}(\mu.\langle\text{REQUEST}, o, t, c\rangle_{\sigma_c})(c) = \mathit{r\text{-}last\text{-}rep}(\mu)(c)$
$\mathit{r\text{-}last\text{-}rep\text{-}t}(\mu.\langle\text{REQUEST}, o, t, c\rangle_{\sigma_c})(c) = \mathit{r\text{-}last\text{-}rep\text{-}t}(\mu)(c)$

Definition 5.2 *We define the following subsets of* $\mathcal{M}$ *and predicate:*

$\mathit{Wire} \equiv \{\, m \mid \exists\, X : ((m, X) \in \mathit{wire})\,\}$
$\mathit{Wire{+}o} \equiv \mathit{Wire} \cup \{\, m \mid \exists\, j \in \mathcal{R} : (\neg\mathit{faulty}_j \wedge m \in \mathit{out}_j)\,\}$
$\mathit{Wire{+}io} \equiv \mathit{Wire{+}o} \cup \{\, m \mid \exists\, j \in \mathcal{R} : (\neg\mathit{faulty}_j \wedge m \in \mathit{in}_j)\,\}$

$$
\begin{aligned}
&\text{committed-Wire}(s, l, t, n, v, \mu) \equiv \\
&\quad \exists\, m_1 ... m_n = \mu \in \mathcal{RM}^* : (s = \text{r-val}(\mu) \wedge l = \text{r-last-rep}(\mu) \wedge t = \text{r-last-rep-t}(\mu) \wedge \\
&\quad\quad \forall\, 0 < k \leq n : (\exists\, v' \leq v,\ R : (|R| > 2f \wedge \\
&\quad\quad\quad\quad \forall\, q \in R : (\langle\text{COMMIT}, v', k, D(m_k), q\rangle_{\sigma_q} \in \text{Wire+o})) \\
&\quad\quad\quad \wedge (\exists\, v' \leq v : (\langle\text{PRE-PREPARE}, v', k, m_k\rangle_{\sigma_{\text{primary}(v')}} \in \text{Wire+o}) \\
&\quad\quad\quad\quad \vee\, m_k \in \text{Wire+o})))
\end{aligned}
$$

Invariant 5.3 *The following is true of any reachable state in an execution of* A_{gc}*:*

1. $\forall\, i \in \mathcal{R} : ((\neg\mathit{faulty}_i \wedge \mathit{n\text{-}faulty} \leq f) \Rightarrow$
$\exists\, \mu \in \mathcal{RM}^* : \mathit{committed\text{-}Wire}(\mathit{val}_i, \mathit{last\text{-}rep}_i, \mathit{last\text{-}rep\text{-}t}_i, \mathit{last\text{-}exec}_i, \mathit{view}_i, \mu))$

2. $\forall\, i \in \mathcal{R} : (\neg\mathit{faulty}_i \wedge \mathit{n\text{-}faulty} \leq f) \Rightarrow$
$\forall\, \langle\text{CHECKPOINT}, v, n, D(\langle s, l, t\rangle), i\rangle_{\sigma_i} \in N : (\exists\, \mu \in \mathcal{RM}^* : \mathit{committed\text{-}Wire}(s, l, t, n, v, \mu))$

where:
$N = \{\, m \mid m \in \mathit{Wire{+}io} \vee \exists\, \langle\text{VIEW-CHANGE}, v, n, s, C, P, j\rangle_{\sigma_j} \in \mathit{Wire{+}io} : (m \in C) \vee$
$\exists\, \langle\text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_j} \in \mathit{Wire{+}io} : (\exists\, \langle\text{VIEW-CHANGE}, v, n, s, C, P, q\rangle_{\sigma_q} \in V : (m \in C))\,\}$,

Proof: The proof is by induction on the length of the execution. For the base case, the initializations ensure that $val_i = r\text{-}val(\lambda)$, $last\text{-}rep_i = r\text{-}last\text{-}rep(\lambda)$, and $last\text{-}rep\text{-}t_i = r\text{-}last\text{-}rep\text{-}t(\lambda)$. Therefore, 1 is obviously true in the base case and 2 is also true because all the checkpoint messages $\langle \text{CHECKPOINT}, v, n, D(\langle s, l, t\rangle), i\rangle_{\sigma_i} \in N$ have $s = val_i, l = last\text{-}rep_i, t = last\text{-}rep\text{-}t_i$.

For the inductive step, assume that the invariant holds for every state of any execution α of length at most l. We will show that the lemma also holds for any one step extension α_1 of α. The only actions that can violate 1 are actions that change $val_i, last\text{-}rep_i, last\text{-}rep\text{-}t_i, last\text{-}exec_i$, decrement $view_i$, or remove messages from $Wire{+}o$. But no actions ever decrement $view_i$. Similarly, no actions ever remove messages from $Wire{+}o$ because $wire$ remembers all messages that were ever sent over the multicast channel and messages are only removed from out_j (for any non-faulty replica j) when they are sent over the multicast channel. Therefore, the only actions that can violate 1 are:

1. RECEIVE$(\langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_j})_i$
2. EXECUTE$(m, v, n)_i$
3. SEND-NEW-VIEW$(v, V)_i$

The inductive hypothesis of condition 2 ensures that actions of the first and third type do not violate condition 1 because they set $val_i, last\text{-}rep_i, last\text{-}rep\text{-}t_i$ and $last\text{-}exec_i$ to the corresponding values in a checkpoint message from a non-faulty replica.

Actions of the second type also do not violate 1 because of the inductive hypothesis, and because the executed request, m_n, verifies $committed(m_n, v, n, i)$ for $v \leq view_i$ and $n = last\text{-}exec_i + 1$. Since $committed(m_n, v, n, i)$ is true, the $2f + 1$ commits and the pre-prepare (or m_n) necessary for $committed\text{-}Wire$ to hold are in in_i. These messages were either received by i over the multicast channel or they are messages from i, in which case they are in out_i or have already been sent over the multicast channel.

The only actions that can violate condition 2 are those that insert checkpoint messages in N:

1. RECEIVE$(\langle \text{CHECKPOINT}, v, n, d, i\rangle_{\sigma_i})_j$
2. RECEIVE$(\langle \text{VIEW-CHANGE}, v, n, s, C, P, q\rangle_{\sigma_q})_j$
3. RECEIVE$(\langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_q})_j$
4. SEND$(m, R)_i$
5. EXECUTE$(m, v, n)_j$
6. SEND-VIEW-CHANGE$(v)_j$
7. SEND-NEW-VIEW$(v, V)_j$

where j is any non-faulty replica. Actions of types 1, 2, 4, and 6 preserve 2 because the checkpoints they insert into N are already in N before the action executes and because of the inductive hypothesis. Actions of types 3 and 7 may insert a new checkpoint message from j into N; but they also preserve condition 2 because this message has the same sequence number and checkpoint digest as some checkpoint message from a non-faulty replica that is already in N before the action executes and because of the inductive hypothesis. Finally, the argument to show that actions of the fifth type preserve 1 also shows that they preserve condition 2. □

Invariant 5.4 *The following is true of any reachable state in an execution of A:*

$n\text{-}faulty \leq f \Rightarrow \forall \mu, \mu' \in \mathcal{RM}^* : ((\exists s, l, t, v, s', l', t', v' : (committed\text{-}Wire(s, l, t, n, v, \mu) \wedge committed\text{-}Wire(s', l', t', n', v', \mu')) \wedge \mu.length \leq \mu'.length) \Rightarrow \exists \mu'' \in \mathcal{RM}^* : (\mu' = \mu.\mu''))$

Proof: (By contradiction) Suppose that the invariant is false. Then, there may exist some sequence number k ($0 < k \leq \mu.length$) and two different requests m_{k_1} and m_{k_2} such that:

$\exists v_1, R_1 : (|R_1| > 2f \wedge \forall q \in R_1 : (\langle \text{COMMIT}, v_1, k, D(m_{k_1}), q\rangle_{\sigma_q} \in Wire{+}o))$ and
$\exists v_2, R_2 : (|R_2| > 2f \wedge \forall q \in R_2 : (\langle \text{COMMIT}, v_2, k, D(m_{k_2}), q\rangle_{\sigma_q} \in Wire{+}o))$

This, Invariant 4.1 and Invariant 4.6 contradict Invariant 4.10. $\square$

Theorem 5.5 *A_{gc} implements S*

Proof: We prove that A_{gc} implements A, which implies that it implements S (Theorems 4.19 and 4.14.) The proof uses a forward simulation $\mathcal{H}$ from A'_{gc} to A' (A'_{gc} is equal to A_{gc} but with all output actions not in the external signature of S hidden.)

Definition 5.6 *$\mathcal{H}$ is a subset of* states$(A'_{gc}) \times$ states(A')*; (x, y) is an element of $\mathcal{H}$ if and only if all the following conditions are satisfied for any replica i such that $x.\text{faulty}_i = \text{false}$, and for any replica j:*

1. The values of the state variables in y are equal to the corresponding values in x except for $y.wire$, $y.in_i$ and $y.out_i$.
2. $y.in_i - \{m = \langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_j} \vee m = \langle \text{PREPARE}, v, n, d, j\rangle_{\sigma_j} \vee m = \langle \text{COMMIT}, v, n, d, j\rangle_{\sigma_j} \mid m \in y.in_i \wedge n \leq x.stable\text{-}n_i\}$
$- \{m \mid m \in y.in_i \wedge (tag(m, \text{VIEW-CHANGE}) \vee tag(m, \text{NEW-VIEW}))\}$
$= x.in_i - \{m = \langle \text{PRE-PREPARE}, v, n, m\rangle_{\sigma_j} \vee m = \langle \text{PREPARE}, v, n, d, j\rangle_{\sigma_j} \vee m = \langle \text{COMMIT}, v, n, d, j\rangle_{\sigma_j} \mid m \in x.in_i \wedge n \leq x.stable\text{-}n_i\}$
$- \{m \mid m \in x.in_i \wedge (tag(m, \text{CHECKPOINT}) \vee tag(m, \text{VIEW-CHANGE}) \vee tag(m, \text{NEW-VIEW}))\}$
3. Let $consistent\text{-}vc(m^1, m^2) \equiv$
$\exists v, n, s, l, t, C, P, P', j : (m^1 = \langle \text{VIEW-CHANGE}, v, n, \langle s, l, t\rangle, C, P, j\rangle_{\sigma_j} \wedge$
$m^2 = \langle \text{VIEW-CHANGE}, v, P', j\rangle_{\sigma_j} \wedge$
$A'_{gc}.correct\text{-}view\text{-}change(m^1, v, j) \Leftrightarrow (A'.correct\text{-}view\text{-}change(m^2, v, j) \wedge$
$P = P' - \{m = \langle \text{PRE-PREPARE}, v', n', m'\rangle_{\sigma_k} \vee m = \langle \text{PREPARE}, v', n', d', k\rangle_{\sigma_k} \mid m \in P' \wedge n' \leq n\})))$
$consistent\text{-}vc\text{-}set(M^1, M^2) \equiv$
$\forall m^1 \in M^1 : (\exists m^2 \in M^2 : consistent\text{-}vc(m^1, m^2)) \wedge$
$\forall m^2 \in M^2 : (\exists m^1 \in M^1 : consistent\text{-}vc(m^1, m^2))$,
and let $y.vc_i = \{\langle \text{VIEW-CHANGE}, v, P, j\rangle_{\sigma_j} \in y.in_i\}$,
$x.vc_i = \{\langle \text{VIEW-CHANGE}, v, n, \langle s, l, t\rangle, C, P, j\rangle_{\sigma_j} \in x.in_i\}$
then $consistent\text{-}vc\text{-}set(x.vc_i, y.vc_i)$ is true
4. Let $consistent\text{-}nv\text{-}set(M_1, M_2) \equiv$
$M_2 = \{m^2 = \langle \text{NEW-VIEW}, v, V', O', N'\rangle_{\sigma_j} \mid$
$\exists m^1 = \langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_j} \in M_1 : (consistent\text{-}vc\text{-}set(V, V') \wedge$
$A'_{gc}.correct\text{-}new\text{-}view(m^1, v) \Leftrightarrow (A'.correct\text{-}new\text{-}view(m^2, v) \wedge$
$O = O' - \{m = \langle \text{PRE-PREPARE}, v, n, m'\rangle_{\sigma_j} \mid m \in O' \wedge n \leq max\text{-}n(V)\} \wedge$
$N = N' - \{m = \langle \text{PRE-PREPARE}, v, n, m'\rangle_{\sigma_j} \mid m \in N' \wedge n \leq max\text{-}n(V)\}))\}$,
and let $y.nv_i = \{\langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_j} \in y.in_i\}$,
$x.nv_i = \{\langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_j} \in x.in_i\}$
then $consistent\text{-}nv\text{-}set(x.nv_i, y.nv_i)$ is true.

5. Let $\textit{consistent-all}(M^1, M^2) \equiv$
   $\forall\, m \in M^1 : (\exists\, m' \in M^2 : (\textit{tag}(m, \text{VIEW-CHANGE}) \wedge \textit{consistent-vc}(m, m')) \vee$
   $(\textit{tag}(m, \text{NEW-VIEW}) \wedge \textit{consistent-nv-set}(\{m\}, \{m'\})) \vee$
   $(\neg \textit{tag}(m, \text{VIEW-CHANGE}) \wedge \neg \textit{tag}(m, \text{NEW-VIEW}) \wedge m = m')),$
   $X_i = x.\textit{out}_i \cup \{\langle m \rangle_{\sigma_i} \mid \langle m \rangle_{\sigma_i} \in x.\textit{Wire}\} - \{m \mid \textit{tag}(m, \text{CHECKPOINT})\},$
   and $Y_i = y.\textit{out}_i \cup \{\langle m \rangle_{\sigma_i} \mid \langle m \rangle_{\sigma_i} \in y.\textit{Wire}\},$
   then $\textit{consistent-all}(X_i.\, Y_i)$

6. Let $X_{\textit{faulty}} = \{\, \langle m \rangle_{\sigma_j} \mid x.\textit{faulty}_j \wedge \langle m \rangle_{\sigma_j} \in x.\textit{Wire}\},$
   $Y_{\textit{faulty}} = \{\, \langle m \rangle_{\sigma_j} \mid y.\textit{faulty}_j \wedge \langle m \rangle_{\sigma_j} \in y.\textit{Wire}\},$
   $\textit{consistent-all}(X_{\textit{faulty}}, Y_{\textit{faulty}})$

7. $\forall\, \langle r \rangle_{\sigma_c} \in x.\textit{Wire} : (\exists\, \langle r \rangle_{\sigma_c} \in y.\textit{Wire})$

Additionally, we assume faulty automata in x are also faulty and identical in $\mathcal{H}[x]$ (i.e., they have the same actions and the same state.) Note that the conditions in the definition of $\mathcal{H}$ only need to hold when n-faulty $\leq f$*, for* n-faulty $> f$ *the behavior of S is unspecified.*

To prove that $\mathcal{H}$ is in fact a forward simulation from A'_{gc} to A' one most prove that both of the following are true:

1. For all $x \in \textit{start}(A'_{gc})$, $\mathcal{H}[x] \cap \textit{start}(A') \neq \{\}$
2. For all $(x, \pi, x') \in \textit{trans}(A'_{gc})$, where x is a reachable state of A'_{gc}, and for all $y \in \mathcal{H}[x]$, where y is reachable in A', there exists an execution fragment α of A' starting with y and ending with some $y' \in \mathcal{H}[x']$ such that $\textit{trace}(\alpha) = \textit{trace}(\pi)$.

Condition 1 holds because $(x, y) \in \mathcal{H}$ for any initial state x of A'_{gc} and y of A'. It is clear that x and y satisfy the first clause in the definition of $\mathcal{H}$ because the initial value of the variables mentioned in this clause is the same in A'_{gc} and A'. Clauses 2 to 7 are satisfied because $x.\textit{in}_i$ only contains checkpoint messages, and $y.\textit{in}_i$, $x.\textit{out}_i$, $y.\textit{out}_i$, $x.\textit{wire}$, and $y.\textit{wire}$ are empty.

We prove condition 2 by showing it holds for every action of A'_{gc}. We start by defining an auxiliary function $\beta(y, m, a)$ to compute a sequence of actions of A' starting from state y to simulate a receive of message m by an automaton a (where a is either a client or replica identifier):

$\beta(y, m, a) =$
  if $\exists\, X : ((m, X) \in y.\textit{wire})$ then
    if $\exists\, X : ((m, X) \in y.\textit{wire} \wedge a \in X)$ then
      $\text{RECEIVE}(m)_a$
    else
      $\text{MISBEHAVE}(m, X, X \cup \{a\}).\ \text{RECEIVE}(m)_a \mid (m, X) \in y.\textit{wire}$
  else
    if $\exists\, i : (y.\textit{faulty}_i = \textit{false} \wedge m \in y.\textit{out}_i)$ then
      $\text{SEND}(m, \{a\})_i.\ \text{RECEIVE}(m)_a$
    else
      $\perp$

If $\text{RECEIVE}(m)_a$ is enabled in a state x, there is an m' such that $\beta(y, m', a)$ is defined and the actions in $\beta(y, m', a)$ are enabled for all $y \in \mathcal{H}[x]$, and:

- $m = m'$, if m is not a checkpoint, view-change, or new-view message
- *consistent-vc*(m, m'), if m is a view-change message
- *consistent-nv-set*$(\{m\}, \{m'\})$, if m is a new-view message

This is guaranteed by clauses 5, 6, and 7 in the definition of $\mathcal{H}$.

Now, we proceed by cases proving condition 2 holds for each $\pi \in acts(A'_{gc})$

Non-faulty proxy actions. If π is an action of a non-faulty proxy automaton P_c other than RECEIVE$(m = \langle \text{REPLY}, v, t, c, i, r\rangle_{\sigma_i})_c$, let α consist of a single π step. For the receive actions, let $\alpha = \beta(y, m, c)$. In either case, when π is enabled in x all the actions in α are also enabled starting from y and an inspection of the code shows that the state relation defined by $\mathcal{H}$ is preserved in all these cases.

Internal channel actions. If π is a MISBEHAVE(m, X, X') action, there are two cases: if π is not enabled in y, let α be λ; otherwise, let α contain a single π step. In either case, $\mathcal{H}$ is preserved. because π does not add new messages to $x.Wire$.

Receive of request, pre-prepare, prepare, or commit. For actions π = RECEIVE$(m)_i$ where m is a syntactically valid request, pre-prepare, prepare, or commit message, let $\alpha = \beta(y, m, i)$; α transforms y into $y' \in \mathcal{H}[x']$:

- π and α modify *wire* in a way that preserves clauses 5, 6, and 7.
- For receives of request messages, α and π add the same messages to out_i and in_i thereby preserving the state correspondence defined by $\mathcal{H}$.
- For the other message types, the definition of $\mathcal{H}$ and the definition of *in-wv* ensure that when the first if condition is true in x, it is also true in y (because the condition is more restrictive in A'_{gc}, and $x.in_i$ and $y.in_i$ have the same prepare and commit messages with sequence numbers higher than $x.stable\text{-}n_i$.) Thus, in this case, the state correspondence defined by $\mathcal{H}$ is preserved. But it is possible for the if condition to be true in y and false in x; this will cause a message to be added to $y.in_i$ and (possibly) $y.out_i$ that is not added to $x.in_i$ or $x.out_i$. Since this happens only if the sequence number of the message received is lower than or equal to $x.stable\text{-}n_i$, the state correspondence is also preserved in this case.

Garbage collection. If π = RECEIVE$(\langle \text{CHECKPOINT}, v, n, d, j\rangle_{\sigma_j})_i$, or π = COLLECT-GARBAGE$_i$, the condition holds when α is λ. It is clear that the condition holds for the first type of action. For the second type, the condition is satisfied because all the messages removed from $x.in_i$ have sequence number lower than or equal to n and the action sets $x.stable\text{-}n_i$ to n. The action sets $x.stable\text{-}n_i$ to n because it removes all triples with sequence number lower than n from $x.chkpts_i$ and there is a triple with sequence number n in $x.chkpts_i$. The existence of this triple is guaranteed because the precondition for the collect-garbage$_i$ action requires that there is a checkpoint message from i with sequence number n in $x.in_i$ and i only inserts checkpoint messages in in_i when it inserts a corresponding checkpoint in $chkpts_i$.

Receive view-change. If π = RECEIVE$(m = \langle \text{VIEW-CHANGE}, v, n, s, C, P, j\rangle_{\sigma_j})_i$, let $\alpha = \beta(y, m', i)$ such that *consistent-vc*(m, m'). The definition of *consistent-vc* ensures that either both messages are incorrect or both are correct. In the first case, π and α only modify the destination set of the messages in *wire*; otherwise, they both insert the view change message in in_i. In either case, the state correspondence defined by $\mathcal{H}$ is preserved.

Receive new-view. When $\pi = \text{RECEIVE}(m = \langle \text{NEW-VIEW}, v, V, O, N\rangle_{\sigma_j})_i$, we consider two cases. Firstly, if the condition in the outer if is not satisfied, let $\alpha = \beta(y, m', i)$, where *consistent-nv-set*$(\{m\}, \{m'\})$. It is clear that this ensures $y' \in \mathcal{H}[x']$ under the assumption that $y \in \mathcal{H}[x]$. Secondly, if the condition in the outer if is satisfied when π executes in x, let α be the execution of the following sequence of actions of A':

1. The actions in $\beta(y, m' = \langle \text{NEW-VIEW}, v, V', O', N'\rangle_{\sigma_j}, i)$, where *consistent-nv-set*$(\{m\}, \{m'\})$

2. Let C be a sequence of tuples (v_n, R_n, m_n) from $\mathbf{N} \times 2^{\mathcal{R}} \times \mathcal{RM}$ such that the following conditions are true:
   i) $\forall\, n \;:\; (x.\mathit{last\text{-}exec}_i \;<\; n \;\leq\; \mathit{max\text{-}n}(V))$

   ii) $\forall\, (v_n, R_n, m_n) \;:\; (v_n \;<\; v \;\wedge\; |R_n| \;>\; 2f \;\wedge\; \forall\, k \;\in\; R_n \;:\; (\langle \text{COMMIT}, v_n, n, D(m_n), k\rangle_{\sigma_k} \;\in\; x.\mathit{Wire{+}o})$
   $\wedge\; (\exists\, v' \;:\; (\langle \text{PRE-PREPARE}, v', n, m_n\rangle_{\sigma_{\mathit{primary}(v')}} \;\in\; x.\mathit{Wire{+}o}) \;\vee\; m_n \;\in\; x.\mathit{Wire{+}o})$

   for each $(v_n, R_n, m_n) \;\in\; C$ in order of increasing n execute:

   a) $\beta(y, c_{n_k} = \langle \text{COMMIT}, v_n, n, D(m_n), k\rangle_{\sigma_k}, i)$, for each $k \;\in\; R_n$

   b) if enabled $\beta(y, p_n = \langle \text{PRE-PREPARE}, v', n, m_n\rangle_{\sigma_{\mathit{primary}(v')}}, i)$ else $\beta(y, m_n, i)$

   c) $\text{EXECUTE}(m_n, v_n, n)_i$

The definition of $\mathcal{H}$ (clauses 1, 4, 5 and 6) ensures that, when the receive of the new-view message executes in y, the condition in the outer if is true exactly when it is satisfied in x. Let y_1 be the state after $\beta(y, m', i)$ executes; we show that when C is empty (i.e., $\mathit{max\text{-}n}(V) \leq \mathit{last\text{-}exec}_i$), $y' = y_1 \in \mathcal{H}[x']$. This is true because:

- Both π and $\beta(y, m', i)$ set $view_i$ to v, add all the pre-prepares in $O \cup N$ to in_i, and add consistent new-view messages to in_i.
- $\beta(y, m', i)$ also adds the pre-prepares in $(O' \cup N') - (O \cup N)$ to in_i but this does not violate $\mathcal{H}$ because π ensures that $x'.\mathit{stable\text{-}n}_i$ is greater than or equal to the sequence numbers in these pre-prepares.
- Both π and $\beta(y, m', i)$ add prepares to in_i and out_i; $\beta(y, m', i)$ adds all the prepares added by π and some extra prepares whose sequence numbers are less than or equal to $x'.\mathit{stable\text{-}n}_i$.

When C is not empty (i.e., $\mathit{max\text{-}n}(V) > \mathit{last\text{-}exec}_i$), it is possible that $y_1 \notin \mathcal{H}[x']$ because some of the requests whose execution is reflected in the last checkpoint in x' may not have executed in y_1. The extra actions in α ensure that $y' \in \mathcal{H}[x']$.

We will first show that C is well-defined, i.e., there exists a sequence with one tuple for each n between $x.\mathit{last\text{-}exec}_i$ and $\mathit{max\text{-}n}(V)$ that satisfies conditions i) and ii). Let $m'' = \langle \text{VIEW-CHANGE}, v, \mathit{max\text{-}n}(V), \langle s, l, t\rangle, C', P, k\rangle_{\sigma_k}$ be the view-change message in V whose checkpoint value, $\langle s, l, t\rangle$, is assigned to $(val_i, \mathit{last\text{-}rep}_i, \mathit{last\text{-}rep\text{-}t}_i)$. Since m'' is correct, C' contains at least $f + 1$ checkpoint messages with sequence number $\mathit{max\text{-}n}(V)$ and the digest of

$\langle s, l, t \rangle$. Therefore, the bound on the number of faulty replicas, and Invariant 5.3 (condition 2) imply there is a sequence of requests μ_1 such that *committed-Wire*$(s, l, t, \textit{max-n}(V), v, \mu_1)$.

Since by the inductive hypothesis $y \in \mathcal{H}[x]$, all the the commit, pre-prepare and request messages corresponding to μ_1 are also in $y.\textit{Wire+o}$. Therefore, all the actions in a) and at least one of the actions in b) are enabled starting from y_1 for each n and each $k \in R_n$. Since $v_n < v$ for all the tuples in C, each receive in $\beta(y, c_{n_k}, i)$ will insert c_{n_k} in $\textit{in}_i$. Similarly, the receive of the pre-prepare or request will insert a matching pre-prepare or request in $\textit{in}_i$. This enables execute$(m_n, v_n, n)_i$.

Invariant 5.3 (condition 1) also asserts that there exists a sequence of requests μ_2 such that *committed-Wire*$(x.\textit{val}_i, x.\textit{last-rep}_i, x.\textit{last-rep-t}_i, x.\textit{last-exec}_i, x.\textit{view}_i, \mu_2)$. Since by the inductive hypothesis $y \in \mathcal{H}[x]$, all the the commit, pre-prepare and request messages corresponding to μ_1 and μ_2 are also in $y.\textit{Wire+o}$. This and Invariant 5.4 imply that μ_2 is a prefix of μ_1. Therefore, after the execution of α, $\textit{val}_i, \textit{last-rep}_i, \textit{last-rep-t}_i, \textit{last-exec}_i$ have the same value in x' and y' as required by $\mathcal{H}$.

Send. If $\pi = \text{SEND}(m, X)_i$, let α be:

- A single send$(m, X)_i$ step, if m does not have the CHECKPOINT, VIEW-CHANGE, or NEW-VIEW tag and this action is enabled in y.
- λ, if m has the CHECKPOINT tag or the action is not enabled in y (because the message is already in the channel.)
- A single send$(m', X)_i$ step, if m has the VIEW-CHANGE tag and this action is enabled in y (where *consistent-vc*(m, m').)
- A single send$(m', X)_i$ step, if m has the NEW-VIEW tag and this action is enabled in y (where *consistent-nv-set*$(\{m\}, \{m'\})$.)

Send-pre-prepare and send-commit. If $\pi = \text{SEND-PRE-PREPARE}(m, v, n)_i$ or $\pi = \text{SEND-COMMIT}(m, v, n)_i$, let α contain a single π step. This ensures $y' \in \mathcal{H}[x']$ because these actions are only enabled in x when they are enabled in y, and they insert and remove the same messages from $\textit{in}_i$ and $\textit{out}_i$.

Execute. When $\pi = \text{EXECUTE}(m, v, n)_i$, let α contain a single π step. The action is enabled in y when it is enabled in x because it is only enabled in x for $n > x.\textit{stable-n}_i$ and $x.\textit{in}_i$ and $y.\textit{in}_i$ have the same pre-prepare and commit messages with sequence numbers greater than $x.\textit{stable-n}_i$ and the same requests. It is easy to see that the state correspondence defined by $\mathcal{H}$ is preserved by inspecting the code.

View-change. If $\pi = \text{VIEW-CHANGE}(v)_i$, let α contain a single π step. The action is enabled in y when it is enabled in x because $\textit{view}_i$ has the same value in x and y. Both π and α insert view-change messages m and m' (respectively) in $\textit{in}_i$ and $\textit{out}_i$; it is clear that this ensures $y' \in \mathcal{H}[x']$ provided *consistent-vc*(m', m') is true. Clause 2 in the definition of $\mathcal{H}$ ensures that m and m' contain the same messages in the P component for sequence numbers greater than $x.\textit{stable-n}_i$; therefore, *consistent-vc*(m', m') is true.

Send-new-view. If $\pi = \text{SEND-NEW-VIEW}(v, V)_i$, let α be the execution of the following sequence of actions of A':

1. send-new-view$(v, V')_i$ step, where *consistent-vc-set*(V, V').

2. Let C be a sequence of tuples (v_n, R_n, m_n) from $\mathbf{N} \times 2^{\mathcal{R}} \times \mathcal{RM}$ such that the following conditions are true:
i) $\forall\, n \;:\; (x.\textit{last-exec}_i \;<\; n \;\leq\; \textit{max-n}(V))$

ii) $\forall\, (v_n, R_n, m_n) \;:\; (v_n \;<\; v \;\wedge\; |R_n| \;>\; 2f \;\wedge\; \forall\, k \;\in\; R_n \;:\; (\langle \text{COMMIT}, v_n, n, D(m_n), k\rangle_{\sigma_k} \;\in\; x.\textit{Wire+o})$
$\wedge\, (\exists\, v' \;:\; (\langle \text{PRE-PREPARE}, v', n, m_n\rangle_{\sigma_{\textit{primary}(v')}} \;\in\; x.\textit{Wire+o}) \;\vee\; m_n \;\in\; x.\textit{Wire+o})$
for each $(v_n, R_n, m_n) \;\in\; C$ in order of increasing n execute:

a) $\beta(y, c_{n_k} = \langle \text{COMMIT}, v_n, n, D(m_n), k\rangle_{\sigma_k}, i)$, for each $k \;\in\; R_n$

b) if enabled $\beta(y, p_n = \langle \text{PRE-PREPARE}, v', n, m_n\rangle_{\sigma_{\textit{primary}(v')}}, i)$ else $\beta(y, m_n, i)$

c) EXECUTE$(m_n, v_n, n)_i$

This simulation and the argument why it preserves $\mathcal{H}$ is very similar to the one presented for receives of new-view messages.

Failure. If $\pi =$ REPLICA-FAILURE$_i$ or $\pi =$ CLIENT-FAILURE$_i$, let α contain a single π step. It is easy to see that $y' \in \mathcal{H}[\S']$.

Actions by faulty nodes. If π is an action of a faulty automaton, let α contain a single π step. The definition of $\mathcal{H}$ ensures that α is enabled in y whenever π is enabled in x. Modifications to the internal state of the faulty automaton cannot violate $\mathcal{H}$. The only actions that could potentially violate $\mathcal{H}$ are sends. But this is not possible because a faulty automaton cannot forge the signature of a non-faulty one. □

References

[1] FIPS 180-1. Secure hash standard. NIST US Dept. of Commerce. 1995.

[2] M. Bellare and P. Rogaway. The exact security of digital signatures- How to sign with RSA and Rabin. In *Advance in Cryptology - EUROCRYPT '96, Lecture Notes in Computer Science, Vol. 1070, U. Maurer, ed., Springer-Verlag*, 1996.

[3] G. Bracha and S. Toueg. Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM*, 32(4), 1995.

[4] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, LA, February 1999.

[5] M. Herlihy and J. Wing. Axioms for Concurrent Objects. In *ACM Symposium on Principles of Programming Languages*, 1987.

[6] K. Kihlstrom, L. Moser, and P. Melliar-Smith. The SecureRing Protocols for Securing Group Communication. In *Hawaii International Conference on System Sciences*, 1998.

[7] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7), 1978.

[8] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[9] D. Malkhi, M. Reiter, and N. Lynch. A Correctness Condition for Memory Shared by Byzantine Processes. Submitted for publication, September 1998.

[10] M. Reiter. The Rampart Toolkit for Building High-Integrity Services. *Theory and Practice in Distributed Systems (LNCS 938)*, 1995.

[11] F. Schneider. Implementing Fault-Tolerant Services Using The State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4), 1990.