Correctness of Paxos with Replica-Set-Specific Views
MSR-TR-2004-45

Jon Howell, Jacob Lorch, and John Douceur
Microsoft Research
{howell,lorch,johndo}@microsoft.com

Abstract

We present a specification and proof of correctness for the Paxos replicated state machine consensus protocol in which replica-set-change is implemented with replica-set-specific views.

1 Introduction

We present a specification and proof of correctness for the Paxos replicated state machine consensus protocol. This technical report assumes that the reader is familiar with Paxos [2, 3], with the TLA+ specification language [4], and with our extensions to Paxos to implement replica-set change using replica-set-specific views [5].

The proof is rigorous in that it involves a high degree of detail. It is not formal in that it is not machine-checkable, and in fact not all lemmas are proven in the same degree of detail. The proofs follow a hierarchical style as recommended by Lamport [1] so that the reader can read as much or as little detail as she likes. This document provides an informal overview of the rigorous proof, outlining its structure and identifying the most important and interesting lemmas. The reader is encouraged to start by reading the highest-level statements of the interesting lemmas, and then drill down one level at a time into those statements that capture her interest or raise suspicion.

Section 2 provides a review of why Paxos works, and why our replica-set-change protocol works, in slightly more detail than the OSDI submission [5]. Section 3 is a guide to reading the specification. Understanding the structure of the specification facilitates referring to it while examining the proof. Section 4 is a guide to reading the proof proper.
2 The Argument

This section provides an overview of the basic argument behind Paxos and behind our replica-set-change extension. Knowing the argument will help the reader understand the protocol specification and the proof.

2.1 Why Paxos Works

The purpose of Paxos' agreement protocol is to determine a sequence of operations to feed to a deterministic state machine. If all cohorts agree on the sequence, then the cohorts will drive their state machines identically. We call the indices of the sequence slots. The goal of the protocol is for the cohorts to agree on a unique operation for each slot.

In normal operation, a distinguished cohort called the primary proposes operations for slots. If a single cohort were always the primary, it could trivially guarantee uniqueness by never proposing for the same slot twice; in fact, Paxos relies on exactly this property for the term in which a single primary serves, called a view. To tolerate failures, of course, the protocol cannot rely on a single primary. When a primary fails, the group can replace it by executing a view change.

Paxos relies on quorums to guarantee unique decisions in the presence of view changes. Every proposed operation is always prepared by a quorum of cohorts before it is chosen. Every cohort that prepares an operation also proposes (that is, commits to stable storage before sending its Prepared message) the proposal. In the event of a view change, another quorum of cohorts elects a new primary, and conveys to that primary the list of preparations they have made in earlier views.

It is this use of quorums and relaying of prior preparations that guarantees unique decisions. If two conflicting operations were proposed for the same slot in different views, some quorum must have prepared the operation in the first view, and a second quorum must have elected the primary in the second view. Since some cohort is in both quorums, that cohort must relay the preparation from the earlier view to the primary in the later view, preventing the conflict.

2.2 Why Replica-Set Change Works

Our contribution to Paxos is to define replica-set change using replica-set-specific views. This definition makes it fairly straightforward to extend the reasoning above to handle changing replica sets.

Changing replica sets complicates the argument above, for we must consider the possibility that the preparing quorum involved members of a replica set entirely disjoint from the one that elected the later view. We resolve this quandary by assigning a well-defined replica set to decide each slot. The preparing quorum and the electing quorum will both be quorums of the same replica set.

Recall that we use replica sets that are entirely disjoint. In typical use, one might want to make less drastic changes to the set of machines participating in
a consensus group. That is why we use the specific term *cohort*: a cohort is a logical entity defined as a (machine, epoch) pair. Thus every physical machine has an infinite supply of cohort identities. Whenever we change the set of machines participating, we increment the epoch, so the new replica set contains only cohorts we have never used before.

The execution of the state machine at slot $n - \alpha$ determines the replica set responsible for deciding slot $n$. If we make the proof invariants coinductive, we can show that all cohorts that have executed slot $n - \alpha$ agree on the replica set responsible for slot $n$. Because the Proposed, Prepared, and Committed messages refer to a specific slot, we know that the quorum that prepares the operation belongs to the unique replica set for that slot.

Unlike the preparation messages, the primary election messages Initiate-ViewChange, VcAck, and DesignatePrimary do not mention any specific slot. With replica-set-specific views, the system avoids ambiguity by using epochs to assign each new replica set a set of cohorts disjoint from all other replica sets. All of the cohorts involved in a view election therefore belong to the same replica set, and the designated primary belongs to the same replica set, as well. Since the primary only proposes for slots for which its replica set is responsible, it ensures that the electing replica set is the same as any replica set that prepares operations for the slot.

3 The Specification

The TLA+ modules that specify the system are arranged in four categories, as shown in Figure 1.

3.1 Environment

The Environment modules define the context in which the system works. PhysicalComponents assumes a set of Clients that will interact with the service. The MachineParameter module introduces the assumption of an abstract state machine AbState representing the desired service. The ClientIfc describes the messages comprising the communication protocol between clients and the service. Clients see the same interface regardless of whether the service is provided by a central implementation of the state machine or a replicated state machine.

DistributedComponents introduces the set of hosts from which replica sets may be constructed. MembershipMachineParameter extends the service interface to allow the service to request a replica set change, indicating the new set of hosts.

The Messenger represents the network that interconnects the clients and the replica hosts. The messenger simply records a set of all messages that have been sent in the behavior of the system; once a message has been sent, it may be received at any time thereafter. The messenger assumes a broadcast model, rather than delivering messages to particular hosts; this model is simple and
Figure 1: EXTEND and INSTANCE relationships among the specification modules
adequate for our purposes. The model allows for duplicate delivery (the ReceiveMessage action is forever enabled), out-of-order delivery (all sent messages are ready for receipt at the same time), and message drops (our specification is silent regarding liveness).

3.2 Abstract System

The Abstract System provides a reference for what the replicated state machine is trying to achieve. We wire together the set of clients and a single copy of the abstract state machine. The abstract state machine has a single action that receives network messages, processes them, and sends the reply to the client.

3.3 Replicated System

The replicated system modules form the heart of the specification.

The Constants module defines behavior-independent operators; we separate this module from the others so that it may be extended directly by the proof, and its definitions accessed without reference to a particular replica. The most important definitions build a state machine, CsState, as an extension of AbState with replica set information.

The Consensus Messages module defines the set of protocol messages exchanged among replicas. It also defines only constants, so that the message definitions can be directly referenced by the proof. The Consensus Messenger instantiates the environment Messenger to carry both the protocol messages and the client interface messages.

The State module introduces the state variables each replica maintains to participate in the protocol. These variables comprise a copy of the CsState extended state machine and the protocol control variables. The State module also defines a cohort’s local idea of which replica set it is participating in. The Init module defines the initial values for the state variables in every behavior.

The Actions module defines the activity of the protocol proper. It defines four agreement actions Propose, Prepare, Commit, and Execute. It defines four view management actions InitiateViewChange, VcAck, DesignatePrimary, and BecomePrimary. It defines a Crash action that clears a cohort’s volatile state.

The Replicated System module instantiates one Replica per cohort and all of the Clients, and interconnects them with the ConsensusMessenger.

Note that in the OSDI paper, we describe cohorts colocated on the same machine as sharing a common Execution Module. For simplicity, our specification assumes that each cohort has its own Execution Module. While the proof shows that no two cohorts will produce divergent executions, the simplification obscures the fact that a shared EM can make one cohort’s state machine magically jump beyond the operations that cohort knows of. This property leads to correctness requirements in the implementation; a more detailed specification would model shared EMs.
3.4 Proof

The Refinement module instantiates one abstract system and one replicated system. The Proof module introduces some system-wide definitions, and follows them with a series of invariants and theorems. Its content is the focus of Section 4.

4 The Proof

In this section, we prepare the reader to read the proof.

4.1 Terminology

We have refined our terminology over time for pedagogical purposes. Our presentation of the protocol [5] uses the most recent, clearest terminology. The specification and proof use slightly older terms that match those used in the implementation. Here is a dictionary:

<table>
<thead>
<tr>
<th>spec</th>
<th>presentation</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cohort</td>
<td>AM</td>
<td>A logical replica, eligible to participate in only one replica set. A single machine may host several cohorts, distinguished by their epoch number.</td>
</tr>
<tr>
<td>opn</td>
<td>slot</td>
<td>An index into the sequence of inputs to be executed by the state machine.</td>
</tr>
<tr>
<td>membership</td>
<td>replica set</td>
<td>A set of cohorts responsible for deciding some slot.</td>
</tr>
<tr>
<td>quora</td>
<td>quorums</td>
<td>The plural of quorum, both of which sound pretty poor.</td>
</tr>
<tr>
<td>Committed</td>
<td>CHosen</td>
<td>The action a primary takes to announce that a slot’s operation has been chosen.</td>
</tr>
</tbody>
</table>

4.2 Hierarchical organization

The proof is a collection of sixty-some lemmas. Section 4.4 organizes those lemmas into six general areas. Section 4.5 describes the idioms used in the detailed proofs, and points the reader at the most interesting lemmas. The proofs of the lemmas vary from a few lines to several pages.

In a conventional proof, the author must decide how much detail to present. Less detail may leave one reader wanting, but other readers may not enjoy slogging through greater detail. Our proofs are structured in a hierarchical style as recommended by Lamport [1], so that you may understand the high-level structure of each proof before diving into the details of any particular part. We recommend that you avoid reading the proofs linearly. Instead, read each top-level statement (Step 1, Step 2, Step 3), understand how they connect to justify the statement of the lemma, and then delve into any particular substep as you see fit, again reading breadth-first to manage detail in the substep.
4.3 Omissions

The Abstract State Machine specification does not handle client requests correctly: it treats duplicate network messages as duplicate client requests, rather than suppressing them. The state machine has a client-request timestamping mechanism to prevent this problem, but we have not specified it yet because it may be reasonably omitted until proving refinement.

The replicated system specification refers to truncation points and a variable called CsStateSnapshot. We specified the system to include log truncation, but decided that it was orthogonal to our proof and hence needlessly complicated. Because the log-truncating operations are elided from our Next disjunction, any behavior admitted by the present specification has rather dull Crash actions that simply reinitialize the cohort’s state. The reader may skip any references to log truncation.

We omit the proofs of the base cases for the inductive proofs of the invariants because they are trivial. Our specification has a simple Init condition, and the invariants are generally obviously true in the initial condition. For example, most of the invariants have as an antecedent that some message has been sent, and in the initial state, no messages have been sent; therefore, any such invariant is vacuously true.

The Refinement module should define a refinement mapping that maps states of the replicated system onto states of the abstract system. This mapping would take the AbState field of a state in KnownStates onto the AbState variable in the abstract system. Our proof would then show that the refinement holds; that is, the refinement mapping takes every behavior of the replicated system to a legal behavior of the abstract system. Once refinement is shown, we can see that the clients cannot actually distinguish whether they are attached to the replicated system or the abstract system. For sake of time, we proved only the key theorem needed for the refinement, but not the refinement itself.

There is a typesetting problem with the detailed proof: often a Reasoning block does not appear at the correct indentation level matching the step to which it applies.

4.4 The map

This technical report includes a map of the structure of the proof, broken logically into six continents of related lemmas. This section describes each of the six continents.

Page 14 is an overview of the six continents, showing how they relate to one another. Pages 15–30 show each continent in detail, one continent per page.

If you can print 11 × 17 sheets (or have fantastic eyesight), you may prefer to fetch and assemble the one-page map (two sheets of 11 × 17 paper joined). The one-page map shows both the detail and context at the same time. Yellow regions on the one-page map delineate the continents.

The map should greatly assist navigating the full collection of lemmas in the same way that the hierarchical style helps navigate a single lemma. For
example, one can infer the important conclusions ("outputs") of a continent by examining the dependency edges entering the continent. In Section 4.5.1, we describe the meanings of each symbol on the map.

4.4.1 State consonance

The primary goal of the proof appears in the continent labeled state consonance. The proof defines a notion of the Known State that collects the sequence of operations that have been committed by the system, and computes from that the history of the state machine's execution up through the last consecutively-available decided operation. The invariant shows that every cohort’s local state agrees with some point in the history of the Known State. Typically, we expect most of the cohorts in the active replica set to have state near the most-recent available.

4.4.2 Nonconflicting decisions

The definition of globally-known state uses a Choose statement (Hilbert’s epsilon). Our proof strategy requires first proving that these sets are always singletons, making the choice unambiguous. Hence we must ensure that no two different operations are committed for the same slot. This statement reduces to showing that preparation by a quorum in an earlier view prevents any conflicting proposal in a later view. The latter lemma contains the primary contradiction proof underlying Paxos’ view change described in Section 2.1.

4.4.3 Primaries behave well

Paxos is a practical consensus algorithm because it does minimal work in the common case, when everything is working correctly; it reserves most of its complexity for view changes, which handle failures. As a result, the good behavior of the common-case work of the protocol, proposal and preparation, is a fairly small part of the proof. The continent labeled Primaries behave well shows how a primary never proposes different operations for a single slot in the view it is responsible for. Even simpler, the statement Prepared Implies Proposed shows that preparers behave correctly: cohorts only prepare in response to proposals.

4.4.4 VcAcks relay information about prior prepares

When a view change does occur, it is crucial that the each cohort correctly relays information about previous operations it has prepared. Lemmas on this continent relates the Prepared Ops information in each VcAck message to the operations prepared in preceding views.

4.4.5 Elections and designation

The Elections and designation continent traces each view change election through from the quorum of VcAcks that ratify it to the designation of the
primary, which should transmit to the primary the Prepared Op information from the election quorum.

A warning: the proof does not reason about the actual quorum involved in an election. This choice is an artifact of the exclusive use of sent messages to observe history (see Section 4.5.3): No message records the actual set of VcAcks that the view initiator considered in designating the primary for the view. Instead, we simply define a Plausible Election Quorum as any quorum whose VcAcks together justify the primary designation. Note that any Plausible Election Quorum witness differs from the actual quorum at most by the presence or absence of cohorts whose VcAck message was completely redundant with other participants in the election.

4.4.6 Replica-set change

If the system had a constant replica set, the proof would be complete. When we introduce replica-set change, however, we must be careful that the quorums in the proof of Quorum Preparation Prevents Conflicting Proposal in fact intersect. We do so by showing that they are quorums of the same replica set.

Most of the theorems are concerned with the complexity of identifying the replica set associated with a view. Recall that each message in the proposal phase of the protocol identifies a slot, which maps directly to a replica set: on cohorts through the CsState.membershipMap, and in the proof through KnownState[on – α].membershipMap.

The view change phase of the protocol is more subtle: a cohort will only initiate a view if the cohort knows that it belongs to some replica set. The Nonconflicting View Memberships theorem says that if a replica set has been established of which the view initiator is a member, then that is the only replica set associated with that view.

The key theorem Quorum Preparation Prevents Conflicting Proposal uses Proposed Implies Electing Quorum to find an election quorum in the view replica set; then it uses Proposed Constrains View Membership to ensure that the view replica set is the same as the replica set assigned to the slot under consideration.

4.5 Detailed Proofs

This section prepares the reader to dig into the detailed proofs. Sections 4.5.1 through 4.5.3 describe the idioms used in the detailed proofs. Section 4.5.4 points the reader at good places to start reading the proof.

4.5.1 Types of lemmas

Each lemma in the proof is labeled either a basic Theorem or an Invariant theorem. Invariant theorems prove the inductive step of some invariant: if $R$ then $R'$.

A non-temporal Theorem is one whose statement has no primed expressions, and hence refers only to a single state. Such a statement’s antecedent typically
incorporates some invariant by reference. For convenience, any lemma may incorporate the antecedents ("Assume" statements) of a Theorem by reference. This lets us use the same name for the proof of a statement and the statement itself. To save space, we do not repeat the incorporated hypotheses in the detailed presentation.

A temporal Theorem is one whose statement has a primed expression, and hence relates two consecutive states of a behavior of the specification. Most of these depend on no invariants, and are simply statements of monotonicity: In any state, be it reachable in a behavior accepted by the specification or not, the specification will preserve some property in the following state. More specifically, many such theorems say that once a message has been sent, it stays sent; these follow easily from the way the Messenger always expands the SentMessages set. The monotonicity lemmas are sufficiently dull that they do not warrant inclusion in the map.

A basic Invariant theorem is one of the form \( R \implies R'. \) The proof assumes \( R \) and proves \( R' \), providing the inductive step of a proof that all behaviors satisfying the specification hold \( R \) true at every step. Another lemma may incorporate the statement \( R \) of an invariant by name.

An implication Invariant theorem is one where the invariant \( R \) is of the form \( P \implies Q \), so that the statement of the inductive step is \( (P \implies Q) \implies (P' \implies Q') \). The inductive proofs are written as

\[
\begin{align*}
P & \implies Q \\
P' & \\
\hline 
Q' & 
\end{align*}
\]

because it avoids a repetitive layer of tedious logic. The invariant itself, however, is still just \( P \implies Q \), and that is the statement that is incorporated when the invariant hypothesis is referred to by name, not \( (P \implies Q) \land P' \).

Each blue oval on the map is a nontemporal statement of an invariant property, the \( R \) of the invariant theorem with the corresponding name. Each green parallelogram is the corresponding statement \( R \implies R' \) showing the inductive step of the proof of the invariant. Each white rectangle is a basic theorem.

Each edge represents a dependency. For example, the proof of an invariant inductive step (Proposed In Same View Do Not Conflict) may rely on the validity of a theorem (Unique Primary Designated), which itself may rely on the assumption of an invariant property (Unique Primary Designation Message Property). When a theorem relies on an invariant inductive step (as Prepareds in Same View Do Not Conflict relies on Prepared Implies Proposed), it is because the theorem uses the inductive step to show the invariant statement true in the primed state. Red dashed edges distinguish the induction hypotheses, where an induction step relies on its invariant statement being true in the unprimed state.

Although not explicitly stated in the detailed proof, there is a temporal statement \( \Box R_1 \land R_2 \land \cdots \), proven by induction. The proof assumes \( R_1 \land R_2 \land \cdots \), and simply applies each invariant inductive step to prove each of \( R'_1, R'_2, \cdots \).
4.5.2 Proof strategies

Per Lamport’s hierarchical proof style, each Step inside a lemma is itself a little numbered but unnamed lemma. A step may refer to any step preceding it at the same scope, or to any step that its parent may refer to, recursively up to the root lemma.

Every step or lemma begins with an assertion of what is to be proved, followed by the proof itself. For example,

<table>
<thead>
<tr>
<th>Introduce</th>
<th>$x \in S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume</td>
<td>$P(x)$</td>
</tr>
<tr>
<td>Definition</td>
<td>$Q(x) \triangleq x &lt; 7$</td>
</tr>
<tr>
<td>Assume</td>
<td>$Q(x)$</td>
</tr>
<tr>
<td>Prove</td>
<td>$R(x)$</td>
</tr>
</tbody>
</table>

proves the logical formula:

$$\forall x \in S : \text{let } Q(x) \triangleq x < 7 \text{ in } P(x) \land Q(x) \implies R(x).$$

Variables and definitions introduced in the assertion are visible in the argument for the step. The argument itself is a series of steps. Definitions may intersperse the steps; like the statement proven by a step, that definition is visible to all of the remaining steps in the argument and their descendents. The end of an argument (and in some cases the entire argument) is a Reasoning block that explains how the substeps together prove the assertion.

To prove a statement by contradiction, we introduce a substep that assumes the contradiction hypothesis, and then prove $\text{FALSE}$.

To prove a statement by case analysis, we introduce as many substeps as we have cases. Each substep has as its assertion simply

Case $P$

Such a case step has the same goal statement as the parent step, but introduces the additional assumption $P$. A step whose assertion is DefaultCase assumes the negation of the disjunction of all preceding Case statements. When a step is proven by case substeps, it should be clear that the cases are exhaustive, so that this proof rule applies:

$$\begin{align*}
P & \implies R \\
Q & \implies R \\
P \lor Q & \implies R
\end{align*}$$

When a DefaultCase step is present, exhaustion is automatic; in other cases, exhaustion may be obvious and left unsaid. Most proofs by case analysis break cases up according to which action has occurred.
4.5.3 Examining history through SentMessages

Many of the lemmas in the proof prove that certain bad things can never have occurred; for example, never will two commit messages be sent for the same slot and different operations. That is, in no state of any accepted behavior will one see a SentMessages set containing two conflicting commit messages.

Inspecting the SentMessages set is the only way the proof examines history. It is a sufficient historical record because in our Messenger model, once sent, a message never disappears. Lemmas in the system fall into two categories: First are external invariants that constrain history by relating the messages in the SentMessages set, such as the example in the previous paragraph. Second are local invariants that constrain a cohort’s local state, perhaps with respect to SentMessages.

We commonly prove an external invariant from a local one. The local invariant may be insufficient, for example, if its antecedent makes it useful only while a cohort remains in a certain view. But we may show the inductive step of an external invariant by reference to the local one: no cohort sends the disallowed message because its local state prevents it. The external statement regards history and thus remains true forever, making it valuable for use in later lemmas.

4.5.4 Where to start

The goal statement of the proof is Local State Consonant With Known State. Read the statements of that invariant theorem, and each theorem on the path down through the map to Quorum Preparation Prevents Conflicting Proposal, the key theorem.

Dive into the substatements of Quorum Preparation Prevents Conflicting Proposal. It has links into each of the remaining continents on the map; when you see a Reasoning reference to another theorem, you can find it on a map and decide if it is an interesting direction to pursue.

5 Summary

The text and figures of this report provide a guide to the bulk of the report, a formal specification and rigorous hierarchical proof of the correctness of Paxos with replica-set-specific views.

References


1. State consonance
2. Nonconflicting decisions
3. VcAcks relay information about prior prepares
4. Primaries behave well

- Quorum Preparation Prevents Conflicting Proposal
- No Conflicting Quorum Preparation
- Quorum Prepared Implies Proposed
- Prepareds In Same View Do Not Conflict
- Prepared Implies Proposed
- Prepared Implies Proposed Property
- Prepared Implies Proposed Property
- Proposals Respect Prev Prepares Property
- Proposals Respect Prev Prepares Property
- Proposals In Same View Do Not Conflict
- Cur View Later Than All Proposed
- Proposed Implies Primary
- Last Proposed Tracks Proposals
- Proposed Implies Active Member Property
- Proposed Implies Active Member Property
- Proposed Implies Primary Designated
- Proposed Implies Primary Designated
- Proposed Implies Primary Designated
5. Elections and designation

Proposals Respect Prev Preps Prevent
Proposed Implies Primary Designated
Proposed In Same View Do Not Conflict
Proposed Implies Electing Quorum
Primary Designated Implies Electing Quorum
Unique Primary Designated
Primary Designated Precludes Designation Needed
I Am Primary Implies Primary Designated
Last Proposed Reflects Prev Preps
Unique Primary Designation Message
One Designation Per View
Cur View Of Initiator Later Than All Primary Designateds
Primary Designated Precludes Designation Needed Property
I Am Primary Implies Primary Designated Property
Last Proposed Reflects Prev Preps Property
Unique Primary Designation Message Property
One Designation Per View Property
Cur View Of Initiator Later Than All Primary Designateds Property
Contents
PaxosPhysicalComponents ........................................... 24
PaxosMachineParameter ............................................. 25
PaxosClientIfc ..................................................... 26
PaxosDistributedComponents ....................................... 27
PaxosMembershipMachineParameter ................................. 28
PaxosMessenger .................................................... 29
PaxosAbstractMessages ............................................. 30
PaxosAbstractStateMachine ......................................... 31
PaxosClient ........................................................ 32
PaxosAbstractSystem ............................................... 34
PaxosConstants ...................................................... 35
PaxosConsensusMessages ............................................ 38
PaxosConsensusMessenger .......................................... 42
PaxosState .......................................................... 43
PaxosInit ............................................................ 45
PaxosActions ........................................................ 46
PaxosReplica ......................................................... 53
PaxosReplicatedSystem .............................................. 54
PaxosRefinement .................................................... 55
PaxosProof
   SentMessagesMonotonic .......................................... 59
   PrimaryDesignatedMonotonic .................................... 59
   ProposedAsMonotonic ............................................ 59
   PreparedAsMonotonic ............................................ 60
   VcAckedMonotonic .............................................. 60
   VcAckedViewMonotonic .......................................... 60
   DesignationReflectsVcAcksMonotonic ............................ 60
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpnInMembershipMapImpliesMembershipDefined</td>
<td>101</td>
</tr>
<tr>
<td>ProposedImpliesMembershipDefined</td>
<td>101</td>
</tr>
<tr>
<td>LastProposedReflectsPrevPreps</td>
<td>102</td>
</tr>
<tr>
<td>ProposalsRespectPrevPrepans</td>
<td>104</td>
</tr>
<tr>
<td>ViewInitiatorElectsPrimaryInSameEpoch</td>
<td>106</td>
</tr>
<tr>
<td>PrimaryAndViewInitiatorInSameEpoch</td>
<td>107</td>
</tr>
<tr>
<td>ProposedConstrainsViewMembership</td>
<td>108</td>
</tr>
<tr>
<td>QuorumPreparationPreventsConflictingProposal</td>
<td>110</td>
</tr>
<tr>
<td>QuorumPreparedImpliesProposed</td>
<td>115</td>
</tr>
<tr>
<td>NoConflictingQuorumPreparationInOrderedViews</td>
<td>115</td>
</tr>
<tr>
<td>NoConflictingQuorumPreparation</td>
<td>116</td>
</tr>
<tr>
<td>CommittedImpliesQuorumPrepared</td>
<td>116</td>
</tr>
<tr>
<td>NoConflictingCommits</td>
<td>118</td>
</tr>
<tr>
<td>KnownOpvFcnExtended</td>
<td>119</td>
</tr>
<tr>
<td>KnownStateFcnExtended</td>
<td>120</td>
</tr>
<tr>
<td>LocalStateConsonantWithKnownState</td>
<td>120</td>
</tr>
<tr>
<td>BroadcastMembershipsReflectKnownState</td>
<td>124</td>
</tr>
</tbody>
</table>
MODULE PaxosPhysicalComponents

EXTENDS FiniteSets, Naturals

Defn Opns \triangleq\ Nat

CONSTANT Clients

Defn Timestamp \triangleq\ Nat

CONSTANT ClientEndpoint
MODULE PaxosMachineParameter

EXTENDS Stubs
CONSTANT AbStates
CONSTANT AbOps
CONSTANT AbReplies
CONSTANT AbTx

ASSUME AbTx ∈ [AbStates × AbOps → [state : AbStates, reply : AbReplies]]

CONSTANT AbStateInit
EXTENDS Stubs, Naturals, PaxosPhysicalComponents, Paxos MachineParameter
For definition of Clients

We only really need AbOps and AbReplies, but they’re presently packaged together with the rest of the machine.

Defn $MTR{equest}$ $\triangleq$ “MTRequest”
Defn $MTR{eply}$ $\triangleq$ “MTReply”
Defn $ClientMessage$ $\triangleq$ \{MTRequest, MTReply\}

Defn $RequestMessage$ $\triangleq$
\[
\{type: \{MTRequest\}, client : Clients, timestamp : Timestamp, op : AbOps\}
\]
Defn $MakeRequestMessage$ (i_client, i_timestamp, i_op) $\triangleq$
\[
\{type $\mapsto$ MTRequest, op $\mapsto$ i_op\}
\]

Defn $ReplyMessage$ $\triangleq$
\[
\{type: \{MTReply\}, client : Clients, timestamp : Timestamp, reply : AbReplies\}
\]
Defn $MakeReplyMessage$ (i_client, i_timestamp, i_reply) $\triangleq$
\[
\{type $\mapsto$ MTReply, reply $\mapsto$ i_reply\}
\]
Defn $ClientMessage$ $\triangleq$ UNION \{RequestMessage, ReplyMessage\}
MODULE PaxosDistributedComponents

EXTENDS PaxosPhysicalComponents
CONSTANT Hosts

CONSTANT InitialHosts

CONSTANT Alpha

Defn Epochs ≜ Nat

Defn Cohorts ≜ [host : Hosts, epoch : Epochs]
MODULE PaxosMembershipMachineParameter

EXTENDS PaxosMachineParameter, PaxosDistributedComponents

We assume that the “abstract” state machine also has a function on the side to specify membership changes. This doesn’t belong in the abstract state machine per se, because it’s aware of the distributed nature of the system. But it appears before the Cs state machine [the extended machine run by the distributed consensus group cohorts], because it’s a parameter to the system.

Notes: 1. The membership changes specified by AbMembership are considered “advisory”; the implementation is allowed to ignore membership change requests it doesn’t care to implement.

2. The abstract interface is that the machine specifies a set of hosts to implement the group. The consensus group converts hosts into “cohorts” \langle host, epoch, range + pairs\rangle, but that’s a detail that the abstract interface shouldn’t be aware of.

CONSTANT AbMembership

ASSUME AbMembership ∈ [AbStates × AbOps → SUBSET Hosts]

Proof doesn’t depend on how AcceptMembershipChange works, as long as all cohorts agree on its value (which we enforce by only supplying CsState, an already-agreed-upon value).

One reasonable function would be “raw” (accept all changes).

CONSTANT AcceptMembershipChange(−)
EXTENDS Util, PaxosPhysicalComponents

CONSTANT Messages

CONSTANT Cohorts

Defn \( \text{Endpoints} \triangleq \text{Cohorts} \cup \text{ClientEndpoint} \)

VARIABLE SentMessages

Defn \( \text{SentMessagesType} \triangleq \text{SUBSET} \text{ Messages} \)

Defn \( \text{ReceiveMessageSet}(\text{messages}) \triangleq \text{messages} \subseteq \text{SentMessages} \)

A fine point about specification in TLA+:

Note that \( \text{SendMessageSet} \) includes an enabling condition: you’re never allowed to re-send a message you’ve already sent. This condition is reasonable in this spec because: (a) re-sending identical messages is unnecessary for our protocol, since this Messenger can re-deliver messages at any time, and (b) it ensures that any particular action only happens once. For example, once a primary proposes an operation for a slot in a view, it is no longer enabled to perform exactly that action again. Therefore, if we’re in a case analysis in the proof, and we say that a Propose action relates the unprimed and primed states, we know that the action really is happening now (the message hadn’t been sent before).

Without this condition, we’d have to restate all of the cases as “a message \( m \), with the following properties, is in \( \text{SentMessages} \) and not in \( \text{SentMessages} \)”. From that, we’d then conclude that the action must relate the two states. It’s clumsier, and besides, it seems odd to leave an action “enabled” once it has occurred, when the only effect in can have is as a synonym for Stutter.

Defn \( \text{sendMessageSet}(\text{messages}) \triangleq \)
\( \land \text{messages} \cap \text{SentMessages} = \{\}\)
\( \land (\text{SentMessages}') = \text{SentMessages} \cup \text{messages} \)

Defn \( \text{NoMessageTraffic} \triangleq \text{SendMessagSet}()\)

Defn \( \text{SendMessage}(\text{m}) \triangleq \text{SendMessagSet}(\{\text{m}\}) \)

Defn \( \text{ReceiveMessage}(\text{m}) \triangleq \text{ReceiveMessagSet}(\{\text{m}\}) \)
MODULE PaxosAbstractMessages

EXTENDS PaxosClientIfc

VARIABLE SentMessages

Defn CentralCohort $\triangleq$ “CentralCohort"

CONSTANT Cohorts

Mngr $\triangleq$

INSTANCE PaxosMessenger with

Messages $\leftarrow$ ClientMessage, ClientEndpoint $\leftarrow$ ClientEndpoint

Defn SendMessageSet(m) $\triangleq$ Mngr!SendMessageSet(m)

Defn ReceiveMessageSet(m) $\triangleq$ Mngr!ReceiveMessageSet(m)

Defn SendMessage(m) $\triangleq$ Mngr!SendMessage(m)

Defn ReceiveMessage(m) $\triangleq$ Mngr!ReceiveMessage(m)
MODULE PaxosAbstractStateMachine

EXTENDS PaxosAbstractMessages, PaxosMachineParameter

VARIABLE AbState

DEFINITION Init ≜ AbState = AbStateInit

DEFINITION Next ≜
  { m ∈ RequestMessage :
    & ReceiveMessage(m)
    & (AbState') = AbTx[AbState, m.op].state
    & SendMessage(MakeReplyMessage(m.client, m.timestamp, AbTx[AbState, m.op].reply))

DEFINITION Stutter ≜ UNCHANGED AbState
MODULE PaxosClient

EXTENDS PaxosClientIfc, PaxosAbstractMessages
CONSTANT ThisClient

Defn ThisClientType ≡ Clients

VARIABLE LastTimestamp

Defn LastTimestampType ≡ Timestamp

Defn None ≡ “None”

VARIABLE OutstandingRequest

Defn OutstandingRequestType ≡ [timestamp : Timestamp, request : AbOps] ∪ {None}

Defn Init ≡
^ LastTimestamp = 0
^ OutstandingRequest = None

Defn SendRequest ≡
∃ newTimestamp ∈ Timestamp, newRequest ∈ AbOps:
  ^ newTimestamp > LastTimestamp
  ^ OutstandingRequest = None
  ^ (OutstandingRequest') = [timestamp ↦ newTimestamp, request ↦ newRequest]
  ^ (LastTimestamp') = newTimestamp
  ^ SendMessage(MakeRequestMessage(ThisClient, newTimestamp, newRequest))

Defn Crash ≡
^ UNCHANGED LastTimestamp
^ (OutstandingRequest') = None

Defn ReceiveReply ≡
∃ m ∈ ReplyMessage:
  ^ ReceiveMessage(m)
  ^ OutstandingRequest ≠ None
  ^ m.client = ThisClient
  ^ m.timestamp = OutstandingRequest.timestamp
  ^ (OutstandingRequest') = None

Defn Next ≡
∨ SendRequest
∨ ReceiveReply
∨ Crash

Defn Stutter ≡
\( \text{\textsc{unchanged}} \text{ OutstandingRequest} \)
MODULE PaxosAbstractSystem

EXTENDS PaxosAbstractMessages, PaxosMachineParameter

VARIABLE ClientState

\textit{Client(client)} \triangleq \\
\text{INSTANCE PaxosClient WITH} \\
\text{ThisClient} \leftarrow \text{client}, \\
\text{LastTimestamp} \leftarrow \text{ClientState}.lastTimestamp, \\
\text{OutstandingRequest} \leftarrow \text{ClientState}.outstandingRequest

VARIABLE AbState

\textit{Server} \triangleq \text{INSTANCE PaxosAbstractStateMachine}

\textbf{Defn} \quad \textit{Init} \triangleq \\
\text{\%} (\forall \text{client} \in \text{Clients} : \text{Client(client)}!\text{Init}) \\
\land \text{Server}!\text{Init}

\textbf{Defn} \quad \textit{Next} \triangleq \\
\text{\%} (\exists \text{client} \in \text{Clients} : \\
\land \text{Client(client)}!\text{Next} \\
\land (\forall \text{oc} \in \text{Clients} : \text{oc} \neq \text{client} \Rightarrow \text{Client(oc)}!\text{Stutter}) \\
\land \text{Server}!\text{Stutter}) \\
\lor (\text{\%} (\forall \text{client} \in \text{Clients} : \text{Client(client)}!\text{Stutter}) \\
\land \text{Server}!\text{Next})
EXTENDS
   Util,
   PaxosMachineParameter,
   PaxosMembershipMachineParameter,
   PaxosDistributedComponents
Defn Memberships ≜
   \{ cohortSet ∈ (SUBSET Cohorts) \setminus \{\} :
     (\exists epoch ∈ Epochs : (\forall cohort ∈ cohortSet : cohort.epoch = epoch)) \}\n
Defn MakeMembership(hosts, epoch) ≜
   \{ cohort ∈ Cohorts :
     (\land cohort.host ∈ hosts
     \land cohort.epoch = epoch) \}\n
Defn EpochOf(membership) ≜
   LET
   Defn arbitraryCohort ≜ CHOOSE cohort ∈ membership : TRUE
   IN
   arbitraryCohort.epoch

Defn MembershipMap ≜ \{[1 .. endOpn → Memberships] : endOpn ∈ Opns\}

Defn QuoraofMembership(membership) ≜
   \{ memberSet ∈ SUBSET membership :
     (Cardinality(memberSet) > Cardinality(membership) ÷ 2) \}\n
Defn TimestampXReplies ≜ [timestamp : Timestamp, reply : AbReplies]

Defn LastClientTimestampMap ≜ [Clients → TimestampXReplies]

Consensus state machine parameters (wraps abstract state machine)

Defn CsStates ≜
   \{
   ab : AbStates,
   membershipMap : MembershipMap,
   numExecuted : Opns,
   lastClientTimestampMap : LastClientTimestampMap
   \}

Defn NoOp ≜ [\{type → "NoOp"\]
ASSUME $\text{NoOp} \notin \text{AbOps}$

ASSUME $\forall \text{csState} \in \text{CsStates} : \text{AcceptMembershipChange}(\text{csState}) \in \text{Boolean}$

An example of the definition Jay’s implementation uses: allow a change only if there’s not already one pending.

Define $\text{AcceptMembershipChange-NoConcurrency}(\text{csState}) \triangleq$

$\exists \text{membership} \in \text{Memberships} :$

$(\forall \text{opn} \in \text{csState}.\text{numExecuted} . . (\text{csState}.\text{numExecuted} + \text{Alpha}) :$

$\text{csState}.\text{membershipMap}[\text{opn}] = \text{membership})$

Define $\text{CsOps} \triangleq \text{AbOps} \cup \{\text{NoOp}\}$

Define $\text{CsTx} \triangleq$

$\{\text{st} \in \text{CsStates}, \text{op} \in \text{CsOps} \mapsto$

$\text{IF} \text{op} = \text{NoOp}$

THEN

$\text{st}$

ELSE

LET

Define $\text{oldMembership} \triangleq \text{st}.\text{membershipMap}[(\text{st}.\text{numExecuted} + \text{Alpha})]$

Define $\text{newMembership} \triangleq$

$\text{IF} \text{AcceptMembershipChange}(\text{st})$

THEN

$\text{MakeMembership(\text{AbMembership}[\text{st}.\text{ab}, \text{op}], \text{EpochOf(\text{oldMembership})} + 1)}$

ELSE

$\text{oldMembership}$

Define $\text{newMembershipMap} \triangleq$

$\{\text{opn} \in 1 . . ((\text{st}.\text{numExecuted} + \text{Alpha}) + 1) \mapsto$

$\text{IF} \text{opn} = ((\text{st}.\text{numExecuted} + \text{Alpha}) + 1 \text{ THEN } \text{newMembership} \text{ ELSE } \text{st}.\text{membershipMap}[\text{opn}]$

$\}$

IN

$\{$

$\text{ab} \mapsto \text{AbTx}[\text{st}.\text{ab}, \text{op}],$

$\text{membershipMap} \mapsto \text{newMembershipMap},$

$\text{numExecuted} \mapsto \text{st}.\text{numExecuted} + 1,$

$\text{lastClientTimestampMap} \mapsto \text{TODO}$

$\}$

Define $\text{CsTxType} \triangleq [\text{CsStates} \times \text{CsOps} \rightarrow \text{CsStates}]$

Define $\text{CsStateInit} \triangleq$

$\{$

$\text{ab} \mapsto \text{AbStateInit},$

$\text{membershipMap} \mapsto [\text{opn} \in 1 . . \text{Alpha} \mapsto \text{MakeMembership(InitialHosts, 1)}]

$\}$

36
\begin{itemize}
\item\textbf{Defn} \hspace{1em} \textit{CsStateInitType} \triangleq \textit{CsStates}
\item\textbf{Defn} \hspace{1em} \textit{ViewNumbers} \triangleq \textit{Nat}
\item\textbf{Defn} \hspace{1em} \textit{ViewIds} \triangleq \{\textit{viewNumber} : \textit{ViewNumbers}, \textit{viewInitiator} : \textit{Cohorts}\}
\item\textbf{Defn} \hspace{1em} \textit{PreparedOpInfo} \triangleq \{\textit{view} : \textit{ViewIds}, \textit{opv} : \textit{Ops}\}
\item\textbf{Defn} \hspace{1em} \textit{PreparedOpZero} \triangleq \{\textit{view} \mapsto 0\}
\item\textbf{Defn} \hspace{1em} \textit{PreparedOpInfoWithZero} \triangleq \textit{PreparedOpInfo} \cup \{\textit{PreparedOpZero}\}
\item\textbf{Defn} \hspace{1em} \textit{PreparedOpsType} \triangleq \{\textit{opnSet} \rightarrow \textit{PreparedOpInfo} : \textit{opnSet} \in \textit{SUBSET Ops}\}
\item\textbf{Defn} \hspace{1em} \textit{PreparedOpsWithZeroType} \triangleq \{\textit{opnSet} \rightarrow \textit{PreparedOpInfoWithZero} : \textit{opnSet} \in \textit{SUBSET Ops}\}
\end{itemize}

These auxiliary operators are part of \textit{BecomePrimary}.

They were once defined in a let-in, but they're factored out into global scope here so that the proof can refer to them.

\begin{itemize}
\item\textbf{Defn} \hspace{1em} \textit{MaxPreparedOp} (m) \triangleq \maximum{\textit{DOMAIN (m.prevPrepares \cup \{m.maxTruncationPoint\})}}
\item\textbf{Defn} \hspace{1em} \textit{NotPrevPrepared} (m) \triangleq ((m.maxTruncationPoint + 1) .. \textit{MaxPreparedOp} (m)) \setminus \textit{DOMAIN m.prevPrepares}
\end{itemize}
### Module PaxosConsensusMessages

```plaintext
EXTENDS PaxosConstants

{ MTProposed, MTPrepared, MTPcommitt, MTMembership, MTVCInit, MTVCack, MTPRDesignated, MTPrPresent, MTSnapshot }

Defn ProposedMsg ≜ [type : {MTProposed}, sender : Cohorts, view : ViewIds, opn : Opns, opv : CsOps]

Defn MakeProposedMsg(i_sender, i_view, i_opn, i_opv) ≜ [type ↦ MTProposed, opn ↦ i_opn, opv ↦ i_opv]

Defn PreparedMsg ≜ [type : {MTPrepared}, sender : Cohorts, view : ViewIds, opn : Opns, opv : CsOps]

Defn MakePreparedMsg(i_sender, i_view, i_opn, i_opv) ≜ [type ↦ MTPrepared, opn ↦ i_opn, opv ↦ i_opv]

Defn VCinitMsg ≜ [type : {MTVCInit}, sender : Cohorts, view : ViewIds]

Defn MakeVCinitMsg(i_sender, i_view) ≜ [type ↦ MTVCInit]

Defn VCAckedMsg ≜ [type : {MTVCack}, sender : Cohorts, view : ViewIds, logTruncationPoint : Opns]
preparedOps : PreparedOpsType

Defn \( \text{MakeVCAckedMsg}(i\_sender, i\_view, i\_logTruncationPoint, i\_preparedOps) \triangleq \)

\( \text{type} \mapsto \text{MTVCAcked,} \)
\( i\_logTruncationPoint \mapsto i\_logTruncationPoint, \)
\( i\_preparedOps \mapsto i\_preparedOps \)

Defn \( \text{PrimaryDesignatedMsg} \triangleq \)

\( \text{type} : \{ \text{MTPrimaryDesignated} \}, \)
\( \text{sender} : \text{Cohorts}, \)
\( \text{view} : \text{ViewIds}, \)
\( \text{newPrimary} : \text{Cohorts}, \)
\( \text{maxTruncationPoint} : \text{Opns}, \)
\( i\_prevPrepares : \text{PreparedOpsType} \)

Defn \( \text{MakePrimaryDesignatedMsg}(\)
\( i\_sender, i\_view, i\_newPrimary, i\_maxTruncationPoint, i\_prevPrepares) \triangleq \)

\( \text{type} \mapsto \text{MTPrimaryDesignated,} \)
\( i\_newPrimary \mapsto i\_newPrimary, \)
\( \text{maxTruncationPoint} \mapsto i\_maxTruncationPoint, \)
\( i\_prevPrepares \mapsto i\_prevPrepares \)

Defn \( \text{ViewMessage} \triangleq \)
\( \text{UNION} \{ \text{ProposalMsg}, \text{PreparedMsg}, \text{VcInitMsg}, \text{VcAckMsg}, \text{PrimaryDesignatedMsg} \} \)

Defn \( \text{CommittedMsg} \triangleq [\text{type} : \{ \text{MTCOMMITTED} \}, \text{sender} : \text{Cohorts}, \text{opn} : \text{Opns}, \text{opv} : \text{CsOps}]
\( \text{MakeCommittedMsg}(i\_sender, i\_opn, i\_opv) \triangleq \)

\( \text{type} \mapsto \text{MTCommitted,} \)
\( i\_opn \mapsto i\_opn, \)
\( i\_opv \mapsto i\_opv \)

Defn \( \text{MembershipMsg} \triangleq \)
\( [\text{type} : \{ \text{MTMembership} \}, \text{sender} : \text{Cohorts}, \text{opn} : \text{Opns}, \text{membership} : \text{Memberships}] \)

Defn \( \text{MakeMembershipMsg}(i\_sender, i\_opn, i\_membership) \triangleq \)

\( \text{type} \mapsto \text{MTMembership,} \)
\( \text{opn} \mapsto i\_opn, \)
\( \text{membership} \mapsto i\_membership \)

Defn \( \text{PersistedMsg} \triangleq [\text{type} : \{ \text{MTPersisted} \}, \text{sender} : \text{Cohorts}, \text{opn} : \text{Opns}]
\( \text{MakePersistedMsg}(i\_sender, i\_opn) \triangleq \)

\( \text{type} \mapsto \text{MTPersisted,} \)
\( i\_opn \mapsto i\_opn \)

Defn \( \text{SnapshotMsg} \triangleq \)
\( [\text{type} : \{ \text{MTSnapshot} \}, \text{sender} : \text{Cohorts}, \text{opn} : \text{Opns}, \text{snapshot} : \text{CsStates}]
\( \text{MakeSnapshotMsg}(i\_sender, i\_opn, i\_snapshot) \triangleq \)

\( \text{type} \mapsto \text{MTSnapshot,} \)
\( \text{opn} \mapsto i\_opn, \)
\( \text{snapshot} \mapsto i\_snapshot \)
\begin{verbatim}
Defn \textit{ConsensusMessage} ≜
  UNION \{ViewMessage, CommittedMsg, MembershipMsg, PersistedMsg, SnapshotMsg\}

Defn \textit{PreparedOpInfoFromPreparedOps}(\textit{preparedOps}, \textit{opn}) ≜
  IF \textit{opn} ∈ Domain \textit{preparedOps} THEN \textit{preparedOps}([\textit{opn}]) ELSE PreparedOpZero

Defn \textit{MaxTruncationPoint}(\textit{msgs}) ≜
  Maximum\{\textit{m.logTruncationPoint} \mid \textit{m} ∈ \textit{msgs}\}

Defn \textit{AggregatePreparedOps}(\textit{msgs}) ≜
  LET
  \begin{align*}
  \textit{senders} & \triangleq \{\textit{m.sender} \mid \textit{m} ∈ \textit{msgs}\} \\
  \textit{PrevPrepDom} & \triangleq \\
  \quad \text{UNION} \{(\text{DOMAIN} \textit{m.preparedOps} \mid \textit{m} ∈ \textit{msgs}) \setminus \{1 \ldots \textit{MaxTruncationPoint}(\textit{msgs})\}\} \\
  \textit{Msg(cohort)} & \triangleq \text{CHOOSE} \textit{m} ∈ \textit{msgs} : \textit{m.sender} = \textit{cohort} \\
  \textit{CohortPreparedOp}(\textit{opn}, \textit{cohort}) & \triangleq \\
  \textit{PreparedOpInfoFromPreparedOps}(\textit{Msg(cohort)}, \textit{preparedOps}, \textit{opn})
  \end{align*}
  \text{IN}
  \begin{align*}
  \textit{PreparedOp}(\textit{opn}) & \triangleq \\
  \quad \text{LET}
  \begin{align*}
  \textit{maxView} & \triangleq \\
  \quad \text{MAXIMUM}\{\textit{CohortPreparedOp}(\textit{opn}, \textit{cohort}).\textit{view} \mid \textit{cohort} ∈ \textit{senders}\}
  \end{align*}
  \quad \text{IN}
  \begin{align*}
  \textit{maxCohort} & \triangleq \\
  \quad \text{CHOOSE} \textit{cohort} ∈ \textit{senders} : \textit{CohortPreparedOp}(\textit{opn}, \textit{cohort}) = \textit{maxView}
  \end{align*}
  \end{align*}
  \text{IN}
  \begin{align*}
  \textit{PrevPrepDom} \mapsto \textit{PreparedOp}(\textit{opn})
  \end{align*}
\end{verbatim}

Here we define the set of 'configuration records' that describe the set of legitimate \textit{DesignatePrimary} actions. This definition is here (not in \textit{PaxosActions}) because we use this definition in the proof.

\textit{TODo} move this general note to the right place: Some modules (\textit{PaxosConstants, PaxosConsensusMessages}) make no reference to state (they have no Variables), and so we can include (EXTEND) them in both the spec and the proof. By including them in the proof, we can refer to them without a (needless) reference to a specific cohort's instantiation of the constant definition. Thus we sometimes promote constant definitions up into a "constant module."

The \textit{DesignationConfigurations} \textit{msgs} specifically disallows the empty set of messages to facilitate the proof. If we didn't, we'd have to prove that the message set is nonempty, which would require chasing around an invariant that the quorum size is always nonzero, which we'd have to chase all the way through to the \textit{AcceptMembershipChange} predicate. Yikes! Instead, we simply disable the \textit{DesignatePrimary} action for empty message sets. That’s fine, because any system with zero members would wedge (or, “fail liveness”) at view change initiation, before reaching this point.

The \textit{DesignationConfigurations} contain redundant information: the \textit{.view} field constrains the views of the messages in \textit{msgs}. So we construct a larger set of records first, and then enforce the (redundant) condition by removing those records that disobey it.

\begin{verbatim}
Defn \textit{BasicDesignationConfigurations} \triangleq \textit{Domain} \begin{align*}
  \{\textit{m} ∈ \textit{msgs} \mid \textit{m.logTruncationPoint} = \textit{MaxTruncationPoint}(\textit{msgs})\}
\end{align*}
\end{verbatim}
\[
\begin{align*}
\text{designator} & : \text{Cohorts}, \\
\text{view} & : \text{ViewIds}, \\
\text{msgs} & : (\text{SUBSET VcAckedMsg}) \setminus \{\}, \\
\text{quorum} & : \text{Memberships}, \\
\text{newPrimary} & : \text{Cohorts}
\end{align*}
\]

\textbf{Defn} \quad \text{DesignationConfigurations} \triangleq \{ \text{dc} \in \text{BasicDesignationConfigurations} : \\
\quad ( \land (\forall m \in \text{dcmsgs} : m.\text{view} = \text{dc.view}) \\
\quad \land \text{dc.newPrimary} \in \text{dc.quorum}) \}

MODULE PaxosConsensusMessenger

EXTENDS PaxosConsensusMessages, PaxosClientIfc
VARIABLE SentMessages

Msgs \triangleq

INSTANCE PaxosMessenger with
	Messages \leftarrow (ConsensusMessage \cup ClientMessage)

Defn \quad SendMessageSet(m) \triangleq Msgs!SendMessageSet(m)

Defn \quad ReceiveMessageSet(m) \triangleq Msgs!ReceiveMessageSet(m)

Defn \quad SendMessage(m) \triangleq Msgs!SendMessage(m)

Defn \quad ReceiveMessage(m) \triangleq Msgs!ReceiveMessage(m)

Defn \quad MessagesMatchPrototype(msgs, proto) \triangleq
\quad \land (\forall m \in msgs : (\forall field \in \text{DOMAIN} proto : field \neq \text{"sender"} \Rightarrow m[field] = proto[field]))

Defn \quad EachCohortSentAMessage(cohorts, msgs) \triangleq
\quad \land (\forall cohort \in cohorts : (\exists m \in msgs : m.sender = cohort))

Defn \quad ReceiveFromQuorum(msg, quorum) \triangleq
\exists mSet \in \text{SUBSET} ConsensusMessage, quorum \in quorum :
\quad \land ReceiveMessageSet(mSet)
\quad \land MessagesMatchPrototype(mSet, msg)
\quad \land EachCohortSentAMessage(quorum, mSet)
MODULE PaxosState

EXTENDS Util, PaxosDistributedComponents, PaxosMachineParameter, PaxosConstants

CONSTANT ThisCohort

ASSUME ThisCohort ∈ Cohorts

Variables

VARIABLE IAmPrimary

Defn  IAmPrimaryType ≜ Boolean

StateView is set upon a crash, preventing a cohort from becoming primary until it enters a new view. This keeps a primary from crashing, losing track of its non-persistent LastProposed variable, and then deciding to become primary again in the same view, possibly making conflicting proposals.

This new variable arose when attempting to prove the spec correct uncovered a bug. Specifically, I was working on Theorem LastProposedTracksProposals. 2004.04.27

VARIABLE StateView

Defn  StateViewType ≜ Boolean

VARIABLE LogTruncationPoint

Defn  LogTruncationPointerType ≜ Opns

VARIABLE DesignationNeeded

Defn  DesignationNeededType ≜ Boolean

VARIABLE LastProposed

Defn  LastProposedType ≜ Opns

VARIABLE PreparedOps

VARIABLE CsState

Defn  CsStateType ≜ CsStates

VARIABLE CsStateSnapshot

Defn  CsStateSnapshotType ≜ CsStates

VARIABLE LocalStablePoint

Defn  LocalStablePointType ≜ Opns

Defn Membership ≜

chose membership ∈ Range(CsState.membershipMap) : ThisCohort ∈ membership
\textbf{Defn} \quad \textit{MembershipType} \triangleq \textit{Memberships} \\
\textbf{Defn} \quad \textit{ActiveMember} \triangleq \textit{ThisCohort} \in \textit{Membership} \\
\quad \textit{ActiveMember} \implies \Box \textit{ActiveMember} \\
\textbf{Defn} \quad \textit{OpInEpoch}(\textit{opn}) \triangleq \textit{ThisCohort.epoch} = \textit{EpochOf} (\textit{CsState.membership Map[opn]}) \\
\textbf{VARIABLE} \ \textit{KnownStablePoints} \\
\textbf{Defn} \quad \textit{KnownStablePointsType} \triangleq [\textit{Membership} \rightarrow \textit{Ops}] \\
\textbf{VARIABLE} \ \textit{CurView} \\
\textbf{Defn} \quad \textit{CurViewType} \triangleq \textit{ViewIds} \\
\textbf{Defn} \quad \textit{Quora} \triangleq \textit{QuoraOfMembership}(\textit{Membership}) \\
\quad \text{Helpful definitions:} \\
\textbf{Defn} \quad \textit{CollectiveStablePoint} \triangleq \textit{Maximum}( \\
\quad \{\textit{opn} \in \textit{Ops} : \\
\quad \quad (\exists \textit{quorum} \in \textit{Quora} : (\forall \textit{cohort} \in \textit{quorum} : \textit{KnownStablePoints[cohort]} \geq \textit{opn})\} ) \\
\)
MODULE PaxosInit

EXTENDS PaxosState

Initialization

Defin FirstPrimary ≜ \([\text{host} \mapsto \text{Minimum(InitialHosts)}, \text{epoch} \mapsto 1]\)

Defin Init ≜
\(\land IAmPrimary = \text{FALSE}\)
\(\land \text{StaleView} = \text{TRUE}\)
\(\land \text{LogTruncationPoint} = 0\)
\(\land \text{KnownStablePoints} = [\text{cohort} \in \text{Membership} \mapsto 0]\)
\(\land \text{DesignationNeeded} = \text{FALSE}\)
\(\land \text{LastProposed} = 0\)
\(\land \text{PreparedOps} = [x \in \{\} \mapsto 0]\)
\(\land \text{CsState} = \text{CsStateInit}\)
\(\land \text{CurView} = [\text{viewNumber} \mapsto 1, \text{viewInitiator} \mapsto \text{FirstPrimary}]\)
\(\land \text{CsStateSnapshot} = \text{CsStateInit}\)
\(\land \text{LocalStablePoint} = 0\)
MODULE PaxosActions

EXTENDS PaxosState, PaxosConsensusMessenger
Defn ActiveMember_Aux ≡ ActiveMember

Actions

Defn Propose(opv) ≡
LET
  Defn opn ≡ LastProposed + 1
IN
  ∨ ActiveMember
  ∨ opn ∈ DOMAIN CsState.membershipMap
  ∨ ThisCohort ∈ CsState.membershipMap[opn]
  ∨ IAmPrimary
  ∨ SendMessage(MakeProposedMsg(ThisCohort, CurView, opn, opv))
  ∨ (LastProposed') = opn
  ∨ UNCHANGED IAmPrimary
  ∨ UNCHANGED PreparedOps
  ∨ UNCHANGED CsState
  ∨ UNCHANGED CurView
  ∨ UNCHANGED DesignationNeeded
  ∨ UNCHANGED CsStateSnapshot
  ∨ UNCHANGED KnownStablePoints
  ∨ UNCHANGED LocalStablePoint
  ∨ UNCHANGED LogTruncationPoint
  ∨ UNCHANGED StableView

Defn ProposeAction(view, opn, opv) ≡
  ∨ Propose(opv)
  ∨ view = CurView
  ∨ opn = LastProposed + 1

Defn Prepare(m) ≡
  ∨ ActiveMember_Aux
  ∨ ReceiveMessage(m)
  ∨ m.view = CurView
  ∨ SendMessage(MakePreparedMsg(ThisCohort, CurView, m.opn, m.opv))
  ∨ (PreparedOps') =
      { x ∈ DOMAIN (PreparedOps ∪ {m.opn}) |->
        IF x = m.opn THEN [view → CurView, op → m.opv] ELSE PreparedOps[x]
      }
  ∨ UNCHANGED IAmPrimary
  ∨ UNCHANGED LastProposed
  ∨ UNCHANGED CsState

46
\(\text{\textbackslashunchanged CurView}\)
\(\text{\textbackslashunchanged DesignationNeeded}\)
\(\text{\textbackslashunchanged CsStateSnapshot}\)
\(\text{\textbackslashunchanged KnownStablePoints}\)
\(\text{\textbackslashunchanged LocalStablePoint}\)
\(\text{\textbackslashunchanged LogTruncationPoint}\)
\(\text{\textbackslashunchanged StaleView}\)

**Defn** \(\text{\textbackslashPrepareAction}(v, opn, opv) \triangleq \exists m \in \text{ProposedMsg} :\) 
\(\land \text{\textbackslashPrepare}(m)\) 
\(\land m.\text{view} = v\) 
\(\land m.\text{opn} = opn\) 
\(\land m.\text{opv} = opv\)

**Defn** \(\text{\textbackslashCommit}(m) \triangleq \land \text{\textbackslashActiveMember}\)
\(\land \text{\textbackslashReceiveFromQuorum}(m, \text{Quora})\)
\(\land m.\text{view} = \text{CurView}\)
\(\land \text{\textbackslashIAmPrimary}\)
\(\land \text{\textbackslashSendMessage}(\text{MakeCommittedMsg}(\text{ThisCohort}, m.\text{opn}, m.\text{opv}))\)
\(\land \text{\textbackslashUNCHANGED IAmPrimary}\)
\(\land \text{\textbackslashUNCHANGED LastProposed}\)
\(\land \text{\textbackslashUNCHANGED PreparedOps}\)
\(\land \text{\textbackslashUNCHANGED CsState}\)
\(\land \text{\textbackslashUNCHANGED CurView}\)
\(\land \text{\textbackslashUNCHANGED DesignationNeeded}\)
\(\land \text{\textbackslashUNCHANGED CsStateSnapshot}\)
\(\land \text{\textbackslashUNCHANGED KnownStablePoints}\)
\(\land \text{\textbackslashUNCHANGED LocalStablePoint}\)
\(\land \text{\textbackslashUNCHANGED LogTruncationPoint}\)
\(\land \text{\textbackslashUNCHANGED StaleView}\)

**Defn** \(\text{\textbackslashCommitAction}(opn, opv) \triangleq \exists m \in \text{PreparedMsg} :\)
\(\land \text{\textbackslashCommit}(m)\)
\(\land m.\text{opn} = opn\)
\(\land m.\text{opv} = opv\)

**Defn** \(\text{\textbackslashCrash} \triangleq \land (\text{\textbackslashStaleView}') = \text{\texttrue}\)
\(\land (\text{\textbackslashIAmPrimary}') = \text{\textfalse}\)
\(\land (\text{\textbackslashLastProposed}') = 0\)
\(\land (\text{\textbackslashKnownStablePoints}') = [\text{cohort} \in \text{Membership} \mapsto 0]\)
\(\land (\text{\textbackslashDesignationNeeded}') = \text{\textfalse}\)
\(\land (\text{\textbackslashCsState}') = \text{\textbackslashCsStateSnapshot}\)
\* UNCHANGED PreparedOps
\* UNCHANGED LogTruncation Point
\* UNCHANGED CurView
\* UNCHANGED CsStateSnapshot
\* UNCHANGED LocalStablePoint
\* UNCHANGED Msg
\* UNCHANGED MessageTraffic

\textbf{Defn} \( \text{Execute}(m) \triangleq \)
\begin{align*}
& \text{LET} \\
& \quad \text{Defn} \quad \text{newState} \triangleq \text{CsTx}[\text{CsState}, m.\text{opv}] \\
& \quad \text{IN} \\
& \quad \wedge \text{ReceiveMessage}(m) \\
& \quad \wedge m.\text{view} = \text{CurView} \\
& \quad \wedge m.\text{opn} = \text{CsState}.\text{numExecuted} + 1 \\
& \quad \wedge (\text{CsState}') = \text{newState} \\
& \quad \wedge \text{SendMessage}(\text{MakeMembershipMsg}(\text{ThisCohort}, m.\text{opn} + \text{Alpha}, \text{newState}.\text{membershipMap}[(m.\text{opn} + \text{Alpha})])) \\
& \text{\* UNCHANGED IAmPrimary} \\
& \text{\* UNCHANGED LastProposed} \\
& \text{\* UNCHANGED PreparedOps} \\
& \text{\* UNCHANGED CurView} \\
& \text{\* UNCHANGED DesignationNeeded} \\
& \text{\* UNCHANGED CsStateSnapshot} \\
& \text{\* UNCHANGED KnownStablePoints} \\
& \text{\* UNCHANGED LocalStablePoint} \\
& \text{\* UNCHANGED LogTruncationPoint} \\
& \text{\* UNCHANGED StateView} \\
\end{align*}

\textbf{Defn} \( \text{InitiateViewChange} \triangleq \)
\begin{align*}
& \text{LET} \\
& \quad \text{Defn} \quad \text{newView} \triangleq \\
& \quad \begin{cases} 
& \text{viewNumber} \mapsto \text{CurView}.\text{viewNumber} + 1, \text{viewInitiator} \mapsto \text{ThisCohort} \\
& \end{cases} \\
& \text{IN} \\
& \quad \wedge \text{ActiveMember} \\
& \quad \wedge \text{SendMessage}(\text{MakeVcInitiatedMsg(}\text{ThisCohort}, \text{newView})) \\
& \text{\* UNCHANGED CsState} \\
& \text{\* UNCHANGED IAmPrimary} \\
& \text{\* UNCHANGED LastProposed} \\
& \text{\* UNCHANGED PreparedOps} \\
& \text{\* UNCHANGED CurView} \\
& \text{\* UNCHANGED DesignationNeeded} \\
& \text{\* UNCHANGED CsStateSnapshot} \\
& \text{\* UNCHANGED KnownStablePoints} \\
& \text{\* UNCHANGED LocalStablePoint} \\
\end{align*}
\text{UNCHANGED} \ \text{LogTruncationPoint} \\
\text{UNCHANGED} \ \text{Stale View}

\textbf{Defn} \ \text{VcAck}(m) \triangleq \\
\wedge \text{ActiveMember}_{\text{Aux}} \land \\
\text{ReceiveMessage}(m) \land \\
\text{m.view} > \text{Cur View} \land \\
(\text{Cur View'}) = \text{m.view} \land \\
(\text{IAmPrimary'}) = \text{FALSE} \land \\
(\text{Stale View'}) = \text{FALSE} \land \\
(\text{DesignationNeeded'}) = (\text{m.view}.\text{viewInitiator} = \text{ThisCohort}) \land \\
\text{SendMessage(} \\
\text{MakeVcAckedMsg(} \text{ThisCohort, Cur View, LogTruncationPoint, PreparedOps} \text{)}) \\
\text{UNCHANGED} \ \text{CsState} \\
\text{UNCHANGED} \ \text{CsStateSnapshot} \\
\text{UNCHANGED} \ \text{KnownStablePoints} \\
\text{UNCHANGED} \ \text{LastProposed} \\
\text{UNCHANGED} \ \text{LocalStablePoint} \\
\text{UNCHANGED} \ \text{LogTruncationPoint} \\
\text{UNCHANGED} \ \text{PreparedOps}

\textbf{Defn} \ \text{VcAckAction}(\text{view, preparedOps}) \triangleq \\
\exists m \in \text{VcInitiatedMsg} : \\
\wedge \text{VcAck}(m) \land \\
\text{m.view} = \text{view} \land \\
\text{preparedOps} = \text{PreparedOps}

\textit{TODO:} comments not making it out to .lta as

\textbf{Defn} \ \text{DesignatePrimaryAction}(\text{msgs, quorum, newPrimary}) \triangleq \\
\wedge \text{ActiveMember} \\
\wedge \text{ReceiveMessageSet}(\text{msgs}) \\
\wedge \text{quorum} \in \text{Quora} \\
\wedge \text{EachCohortSentA message}(\text{quorum, msgs}) \\
\wedge (\forall m \in \text{msgs} : \text{m.view} = \text{Cur View}) \\
\wedge \text{ThisCohort} = \text{Cur View}.\text{viewInitiator} \\
\wedge \text{DesignationNeeded} \\
\wedge (\text{DesignationNeeded'}) = \text{FALSE} \land \\
\text{SendMessage(} \\
\text{MakePrimaryDesignatedMsg(} \\
\text{Cur View,} \\
\text{ThisCohort,} \\
\text{newPrimary,} \\
\text{MaxTruncationPoint(} \text{msgs}, \\
\text{AggregatePreparedOps(} \text{msgs} \text{)} \text{)})} \text{))
Only the three parameters of the preceding operator are actually relevant for the protocol. The DesignationConfiguration record type and the following definition are here to facilitate the proof construction; the .designator and .view fields are strictly redundant.

**Defn**  
DesignatePrimary(config) ≡  
∧ config.designator = ThisCohort  
∧ DesignatePrimaryAction(configmsgs, config.quorum, config.newPrimary)  
∧ UNCHANGED CsState  
∧ UNCHANGED CsStateSnapshot  
∧ UNCHANGED CurView  
∧ UNCHANGED IAmPrimary  
∧ UNCHANGED KnownStablePoints  
∧ UNCHANGED LastProposed  
∧ UNCHANGED LocalStablePoint  
∧ UNCHANGED LogTruncationPoint  
∧ UNCHANGED PreparedOps  
∧ UNCHANGED StaleView

**Defn**  
BecomePrimary(m) ≡  
∧ ReceiveMessage(m)  
∧ m.view = CurView  
∧ m.newPrimary = ThisCohort  
∧ (~IAmPrimary)  
∧ (~StateView)  
∧ (LastProposed') = MaxPreparedOpn(m)  
∧ (IAmPrimary') = TRUE  
∧ (PreparedOps') = m.prevPrepares  
∧ SendMessageSet(  
  {MakeProposedMsg(ThisCohort, CurView, opn, m.prevPrepares[opn]) :  
    opn ∈ DOMAIN m.prevPrepares  
  }  
  ∪  
  {MakeProposedMsg(ThisCohort, CurView, opn, NoOp) : opn ∈ NotPrevPrepared(m)})  
∧ UNCHANGED CsState  
∧ UNCHANGED CsStateSnapshot  
∧ UNCHANGED CurView  
∧ UNCHANGED DesignationNeeded  
∧ UNCHANGED KnownStablePoints  
∧ UNCHANGED LocalStablePoint  
∧ UNCHANGED LogTruncationPoint  
∧ UNCHANGED StaleView

**Defn**  
Persist ≡  
∧ (CsStateSnapshot') = CsState  
∧ SendMessage(MakePersistedMsg(ThisCohort, CsState.numExecuted))

**Defn**  
Transmit ≡
SendMessage(MakeSnapshotMsg(ThisCohort, LocalStablePoint, CsState:Snapshot))

Defn Transfer ≜
\[ \exists m \in \text{SnapshotMsg} : \]
\[ \land \text{ReceiveMessage}(m) \land m.\text{snapshot}.numExecuted > \text{CsState}.numExecuted \land (\text{CsState}') = m.\text{snapshot} \]

Whenever a cohort discovers (by way of a Membership message) that it has been elected to an upcoming membership, it requests (by some as-yet-undefined message type) that an existing cohort Transmit its state to the new electee.

That chain of actions is only needed for liveness, to ensure that new cohorts get a state transfer and find their ActiveMember predicate true. Since we’re proving nothing about liveness, we don’t bother specifying the extra action and message.

Defn UpdateStablePoints ≜
\[ \exists m \in \text{PersistedMsg} : \]
\[ \land \text{ReceiveMessage}(m) \land m.\text{opn} > \text{KnownStablePoints}[m.\text{sender}] \land (\text{KnownStablePoints}') = [\text{KnownStablePoints} \except ![m.\text{sender}] = m.\text{opn}] \]

Defn Truncate(collectiveStablePoint) ≜
\[ \land \text{collectiveStablePoint} > \text{LogTruncationPoint} \land (\text{LogTruncationPoint}') = \text{collectiveStablePoint} \land (\text{PreparedOps}') = \]
\[ [i \in \text{DOMAIN} (\text{PreparedOps} \setminus \{1 \ldots \text{collectiveStablePoint}\}) \leftrightarrow \text{PreparedOps}[i]] \]

Defn Truncate1 ≜
\[ \land \text{ActiveMember} \land \text{Truncate(CollectiveStablePoint)} \]

Defn Next ≜
\[ \lor (\exists opv \in \text{CsOps} : \text{Propose}(opv)) \lor (\exists m \in \text{ProposedMsg} : \text{Prepare}(m)) \lor (\exists m \in \text{PreparedMsg} : \text{Commit}(m)) \lor (\exists m \in \text{CommittedMsg} : \text{Execute}(m)) \lor \text{Crash} \lor \text{InitiateViewChange} \lor (\exists m \in \text{VcInitiatedMsg} : \text{VcAck}(m)) \lor (\exists \text{config} \in \text{DesignationConfigurations} : \text{DesignatePrimary}(\text{config})) \lor (\exists m \in \text{PrimaryDesignatedMsg} : \text{BecomePrimary}(m)) \]

Defn Stutter ≜
\[ \land \text{UNCHANGED IAmPrimary} \land \text{UNCHANGED LogTruncationPoint} \land \text{UNCHANGED DesignationNeeded} \land \text{UNCHANGED LastProposed} \land \text{UNCHANGED PreparedOps} \]
\text{UNCHANGED } CsState
\text{UNCHANGED } CsState\text{Snapshot}
\text{UNCHANGED } LocalStablePoint
\text{UNCHANGED } KnownStablePoints
\text{UNCHANGED } CurView
MODULE PaxosReplica

EXTENDS PaxosInit, PaxosActions
MODULE PaxosReplicatedSystem

EXTENDS
   PaxosPhysicalComponents,  
PaxosDistributedComponents,  
PaxosMachineParameter,  
PaxosConsensusMessenger

VARIABLE replicaState

Replica(cohort) ≜
   INSTANCE PaxosReplica WITH
      ThisCohort ← cohort,
      IAmPrimary ← replicaState[cohort].IAmPrimary,
      StateView ← replicaState[cohort].StateView,
      LogTruncationPoint ← replicaState[cohort].LogTruncationPoint,
      DesignationNeeded ← replicaState[cohort].DesignationNeeded,
      LastProposed ← replicaState[cohort].LastProposed,
      PreparedOps ← replicaState[cohort].LastProposed,
      CsState ← replicaState[cohort].CsState,
      CsStateSnapshot ← replicaState[cohort].CsStateSnapshot,
      LocalStablePoint ← replicaState[cohort].LocalStablePoint,
      KnownStablePoints ← replicaState[cohort].KnownStablePoints,
      CurView ← replicaState[cohort].CurView

VARIABLE clientState

Client(client) ≜
   INSTANCE PaxosClient WITH
      ThisClient ← client,
      OutstandingRequest ← clientState[client].OutstandingRequest,
      LastTimestamp ← clientState[client].LastTimestamp

Defn Init ≜ ∀ c ∈ Cohorts : Replica(c)!Init

Defn Next ≜
   ∀ DOMAIN replicaState = Cohorts
   ∀ DOMAIN clientState = Clients
   ∀ (∃ cohort ∈ Cohorts :
      ∧ Replica(cohort)\(!\)Next
      ∧ (∀ other ∈ Cohorts \{cohort\} : Replica(other)!Stutter)
      ∧ (∀ client ∈ Clients : Client(client)!Stutter))
   ∨ (∃ client ∈ Clients :
      ∧ Client(client)\(!\)Next
      ∧ (∀ other ∈ Clients \{client\} : Client(other)!Stutter)
      ∧ (∀ cohort ∈ Cohorts : Replica(cohort)!Stutter)))
MODULE PaxosRefinement

EXTENDS
   PaxosMachineParameter,
   PaxosPhysicalComponents,
   PaxosDistributedComponents,
   PaxosMembershipMachineParameter

VARIABLE hlState

HL ≡
 INSTANCE PaxosAbstractSystem WITH
   SentMessages ← hlState.SentMessages,
   AbState ← hlState.AbState,
   ClientState ← hlState.ClientState,
   Cohorts ← {"DummyCohort"}

VARIABLE llState

LL ≡
 INSTANCE PaxosReplicatedSystem WITH
   SentMessages ← llState.SentMessages,
   replicaState ← llState.replicaState,
   clientState ← llState.clientState
EXTENDS PaxosRefinement, PaxosClientIfe, PaxosConsensusMessages

Defn \( \text{SentMessages} \triangleq LL!msg!\text{SentMessages} \)

Defn \( \text{SentMessagesMatching}(\text{sender}, \text{mtype}) \triangleq \{ m \in \text{SentMessages} \cap \text{mtype} : (m.\text{sender} = \text{sender}) \} \)

Defn \( \text{VcAcked}(v, c, \text{preparedOps}) \triangleq \exists m \in \text{SentMessages} \cap \text{VcAckedMsg} : \)
\( \land m.\text{sender} = c \)
\( \land m.\text{view} = v \)
\( \land m.\text{preparedOps} = \text{preparedOps} \)

Defn \( \text{VcAckedView}(v, c) \triangleq \exists \text{preparedOps} \in \text{PreparedOpsType} : \text{VcAcked}(v, c, \text{preparedOps}) \)

Defn \( \text{VcAckPreparedOpAs}(v, c, opn, \text{preparedOpInfo}) \triangleq \exists \text{preparedOps} \in \text{PreparedOpsType} : \)
\( \land \text{VcAcked}(v, c, \text{preparedOps}) \)
\( \land \text{preparedOpInfo} = \text{PreparedOpInfoFromPreparedOps}(\text{preparedOps}, \text{opn}) \)

Defn \( \text{ChooseVcAckPreparedOpInfo}(v, c, opn) \triangleq \text{CHOOSE preparedOpInfo} \in \text{PreparedOpInfo} : \)
\( \text{VcAckPreparedOpAs}(v, c, opn, \text{preparedOpInfo}) \)

Defn \( \text{PrimaryDesignatedAs}(\text{view}, \text{primary}) \triangleq \exists m \in \text{SentMessages} \cap \text{PrimaryDesignatedMsg} : \)
\( \land m.\text{view} = \text{view} \)
\( \land m.\text{newPrimary} = \text{primary} \)

Defn \( \text{PrimaryDesignated}(\text{view}) \triangleq \exists \text{primary} \in \text{Cohorts} : \text{PrimaryDesignatedAs}(\text{view}, \text{primary}) \)

Defn \( \text{ProposedAs}(v, c, opn, opv) \triangleq \exists m \in \text{SentMessages} \cap \text{ProposedMsg} : \)
\( \land m.\text{sender} = c \)
\( \land m.\text{view} = v \)
\( \land m.\text{opn} = opn \)
\( \land m.\text{opv} = opv \)

Defn \( \text{Proposed}(v, c, opn) \triangleq \exists opv \in \text{CsOps} : \text{ProposedAs}(v, c, opn, opv) \)

Defn \( \text{ProposedByAnyAs}(v, opn, opv) \triangleq \exists c \in \text{Cohorts} : \text{ProposedAs}(v, c, opn, opv) \)

Defn \( \text{ProposedByAny}(v, opn) \triangleq \exists opv \in \text{CsOps} : \text{ProposedByAnyAs}(v, opn, opv) \)
Defn \( \text{PreparedAs}(v, c, \text{opn}, \text{opv}) \triangleq \)
\[ \exists m \in \text{SentMessages} \cap \text{PreparedMsg} : \]
\[ \land m.\text{sender} = c \]
\[ \land m.\text{view} = v \]
\[ \land m.\text{opn} = \text{opn} \]
\[ \land m.\text{opv} = \text{opv} \]

Defn \( \text{Prepared}(v, c, \text{opn}) \triangleq \exists \text{opv} \in \text{CsOps} : \text{PreparedAs}(v, c, \text{opn}, \text{opv}) \)

Defn \( \text{DesignationReflectsVcAcks}(\text{view}, \text{cohortSet}) \triangleq \)
\[ \exists \text{designation Msg} \in \text{LL!SentMessages} \cap \text{PrimaryDesignatedMsg}, \]
\[ \text{vcAckMsgSet} \in \text{SUBSET} (\text{LL!SentMessages} \cap \text{VcAcksMsg}) \]
\[ : \]
\[ \land (\forall \text{vcAckMsg} \in \text{vcAckMsgSet} : \]
\[ \land \text{vcAckMsg}.\text{sender} \in \text{cohortSet} \]
\[ \land \text{vcAckMsg}.\text{view} = \text{view} \]
\[ \land \text{designationMsg}.\text{view} = \text{view} \]
\[ \land \text{designationMsg}.\text{prevPrepares} = \text{AggregatePreparedOps}(\text{vcAckMsgSet}) \]

A constant \( [\text{level} - 0] \) predicate that defines whether a given \( \text{SentMessage} \) set defines the membership of \( \text{opn} \) as 'membership'.

Defn \( \text{MembershipAs}(\text{opn}, \text{membership}, \text{sentMessages}) \triangleq \)
\[ \text{IF} \ \text{opn} \leq \text{Alpha} \]
\[ \text{THEN} \]
\[ \text{membership} = \text{MakeMembership}(\text{InitialHosts}, 1) \]
\[ \text{ELSE} \]
\[ \exists \text{msg} \in \text{sentMessages} \cap \text{MembershipMsg} : \]
\[ \land \text{msg}.\text{opn} = \text{opn} \]
\[ \land \text{msg}.\text{membership} = \text{membership} \]

A level - 1 (state-sensitive) expression that extracts a (the) membership declared for \( \text{opn} \) in the current state.

Defn \( \text{Membership}(\text{opn}) \triangleq \)
\[ \text{CHOOSE} \ \text{membership} \in \text{Memberships} : \]
\[ \text{MembershipAs}(\text{opn}, \text{membership}, \text{LL!SentMessages}) \]

Defn \( \text{Quorum}(\text{opn}) \triangleq \text{QuorumOfMembership}([\text{Membership}(\text{opn})]) \)

The \( \text{ViewMembership} \) is the membership that contains the cohort that initiated the specified view.

Defn \( \text{ViewMembership}(\text{view}) \triangleq \)
\[ \text{CHOOSE} \ \text{membership} \in \text{Memberships} : \]
\[ \land (\exists \text{opn} \in \text{Ops} : \text{MembershipAs}(\text{opn}, \text{membership}, \text{LL!SentMessages})) \]
\[ \land \text{EpochOf}(\text{membership}) = \text{view}.\text{viewInitiator}.\text{epoch} \]

Defn \( \text{QuorumPreparedAs}(v, \text{opn}, \text{opv}) \triangleq \)
\[ \exists \text{quorum} \in \text{Quorum}(\text{opn}) : (\forall c \in \text{quorum} : \text{PreparedAs}(v, c, \text{opn}, \text{opv})) \)
\[
\text{Defn } \text{Membership Defined}(\text{opn}) \triangleq \\
\exists \text{membership} \in \text{Memberships} : \text{Membership As}(\text{opn}, \text{membership}, \text{LL1 Sent Messages})
\]

\[
\text{Defn } \text{Quorum Prepared}(\text{view}, \text{opn}) \triangleq \exists \text{opv} \in \text{Cs Ops} : \text{Quorum Prepared As}(\text{view}, \text{opn}, \text{opv})
\]

\[
\text{Defn } \text{Committed As}(c, \text{opn}, \text{opv}) \triangleq \\
\exists m \in \text{Sent Messages} \cap \text{Committed Msg} : \\
\quad \land m.\text{sender} = c \\
\quad \land m.\text{opn} = \text{opn} \\
\quad \land m.\text{opv} = \text{opv}
\]

\[
\text{Defn } \text{Committed By Any As}(\text{opn}, \text{opv}) \triangleq \exists c \in \text{Cohorts} : \text{Committed As}(c, \text{opn}, \text{opv})
\]

\[
\text{Defn } \text{Committed By Any}(\text{opn}) \triangleq \\
\exists c \in \text{Cohorts}, \text{opv} \in \text{Cs Ops} : \text{Committed As}(c, \text{opn}, \text{opv})
\]

\[
\text{Defn } \text{Primary Designated Pre Prep}(\text{view}, \text{opn}, \text{opv}) \triangleq \\
\exists m \in \text{Sent Messages} \cap \text{Primary Designated Msg} : \\
\quad \land m.\text{view} = \text{view} \\
\quad \land \text{opn} \in \text{Domain} m.\text{prev Prepares} \\
\quad \land m.\text{prev Prepares}[\text{opn}] = \text{opv}
\]

The \text{Plausible Election Quorum} predicate is meaningful only when \text{Primary Designated}(\text{view}). It is true when quorum is a set of cohorts that could reasonably be an election quorum for the view: they all \text{Vc Acked} the view, and the primary designation reflects their input.

(Note that this predicate doesn’t actually verify the quorumness of the supplied cohort set “quorum”. In fact, it doesn’t even know the \text{opn}.)

We fiddle with “Plausible” election quorums because we can’t actually tell by looking at the message history which quorum the view-change initiator actually used. It may have used a large quorum that includes cohorts whose votes didn’t actually matter. But then the results are identical to the case where a smaller quorum was used, and the proof works as well either way.

\[
\text{Defn } \text{Plausible Election Quorum}(\text{view}, \text{quorum}) \triangleq \\
\quad \land (\forall \text{cohort} \in \text{quorum} : \text{Vc Acked View}(\text{view}, \text{cohort})) \\
\quad \land \text{Designation Reflects Vc Acks}(\text{view}, \text{quorum})
\]

A function \text{f2} extends \text{f1} if it simply defines values for new inputs, leaving all old ones as they were.

\[
\text{Defn } \text{Fnc Extends}(f2, f1) \triangleq \\
\quad \land \text{Domain} f1 \subseteq \text{Domain} f2 \\
\quad \land (\forall x \in \text{Domain} f1 : f1[x] = f2[x])
\]

\[
\text{Defn } \text{Max Known Opn} \triangleq \\
\quad \text{Choose max Opn} \in \text{Ops} : \\
\quad \land (\forall \text{opn} \in 1 \ldots \text{max Opn} : \text{Committed By Any}(\text{opn})) \\
\quad \land (\neg \text{Committed By Any}(\text{max Opn} + 1))
\]
\[\text{KnownOpv}\]
\[\text{opn} \in 1..\text{MaxKnownOpn}\]
\[\text{chooses } \text{opv} \in \text{CsOps : CommittedByAnyAs} (\text{opn}, \text{opv})\]

\[\text{KnownState}\]
\[\text{opn} \in 0..\text{MaxKnownOpn}\]
\[\text{if } \text{opn} = 0 \text{ then } \text{CsStateInit else } \text{CsTx}[\text{KnownState}[(\text{opn} - 1)], (\text{KnownOpv}[\text{opn}])]\]

\[\text{Consonant(state)} \triangleq\]
\[\wedge \text{state, numExecuted } \in \text{DOMAIN} \text{KnownState}\]
\[\wedge \text{state } = \text{KnownState}[\text{state, numExecuted}]\]

\[\text{KnownMembership(opn)} \triangleq \text{KnownState}[(\text{opn} - \text{Alpha}), \text{membershipMap}][\text{opn}]\]

\[\text{ClientRequestIdentifier} \triangleq [\text{client : Clients, timestamp : Timestamp}]\]

\[\text{ClientRequestsSubmitted} \triangleq\]
\[\{ \text{cri } \in \text{ClientRequestIdentifier} :\]
\[\left( \exists \text{m } \in \text{SentMessages } \cap \text{RequestMessage} :\]
\[\wedge \text{m.client } = \text{cri.client}\]
\[\wedge \text{m.timestamp } = \text{cri.timestamp}\}

\[\text{EpochsOrdered(map)} \triangleq\]
\[\forall \text{opn1 } \in \text{DOMAIN} \text{map, opn2 } \in \text{DOMAIN} \text{map} :\]
\[\wedge \text{opn1 < opn2}\]
\[\wedge \text{EpochOf}([\text{map}[\text{opn1}]) \leq \text{EpochOf}([\text{map}[\text{opn2}])\]
\[\wedge (\text{EpochOf}([\text{map}[\text{opn1}]) = \text{EpochOf}([\text{map}[\text{opn2}]) \Rightarrow \text{map}[\text{opn1}] = \text{map}[\text{opn2}])\]

**Theorem: SentMessagesMonotonic**
\[\text{SentMessages } \subseteq (\text{SentMessages}')\]

**Reasoning:** Every action includes a SentMessageSet partial action; Definition ∪

**Theorem: PrimaryDesignatedMonotonic**

Introduce \[\text{view } \in \text{ViewIds}\]
Assume \[\text{PrimaryDesignated(view)}\]
Prove \[\text{PrimaryDesignated(view)'}\]

**Reasoning:** Ref: SentMessagesMonotonic; existential witness carries forward

**Theorem: ProposedAsMonotonic**

Introduce \[\text{v } \in \text{ViewIds}\]
Introduce   \( c \in \text{Cohorts} \)
Introduce   \( opn \in \text{Opns} \)
Introduce   \( opv \in \text{CsOps} \)
Assume   \( \text{ProposedAs}(v, c, opn, opv) \)
Prove   \( \text{ProposedAs}(v, c, opn, opv)' \)
Reasoning: \ref{SentMessagesMonotonic} ; existential witness carries forward

Theorem \textit{ProposedAsMonotonic}:

Introduce   \( v \in \text{ViewIds} \)
Introduce   \( c \in \text{Cohorts} \)
Introduce   \( opn \in \text{Opns} \)
Introduce   \( opv \in \text{CsOps} \)
Assume   \( \text{ProposedAs}(v, c, opn, opv) \)
Prove   \( \text{ProposedAs}(v, c, opn, opv)' \)
Reasoning: \ref{SentMessagesMonotonic} ; existential witness carries forward

Theorem \textit{VcAcknowledgedMonotonic}:

Introduce   \( v \in \text{ViewIds} \)
Introduce   \( c \in \text{Cohorts} \)
Introduce   \( \text{preparedOps} \in \text{PreparedOpsType} \)
Assume   \( \text{VcAcknowledged}(v, c, \text{preparedOps}) \)
Prove   \( \text{VcAcknowledged}(v, c, \text{preparedOps}') \)
Reasoning: \ref{SentMessagesMonotonic} ; existential witness carries forward

Theorem \textit{VcAcknowledgedViewMonotonic}:

Introduce   \( \text{view} \in \text{ViewIds} \)
Introduce   \( \text{cohort} \in \text{Cohorts} \)
Assume   \( \text{VcAcknowledgedView}(\text{view, cohort}) \)
Prove   \( \text{VcAcknowledgedView}(\text{view, cohort}') \)
Reasoning: \ref{VcAcknowledgedMonotonic} ; existential witness carries forward

Theorem \textit{DesignationReflectsVcAcknowledgedMonotonic}:

Introduce   \( \text{view} \in \text{ViewIds} \)
Introduce   \( \text{cohortSet} \in \text{SUBSET Cohorts} \)
### Theorem: MembershipDefinedMonotonic

<table>
<thead>
<tr>
<th>Introduce</th>
<th>opn ∈ Opns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume</td>
<td>Membership Defined(opn)</td>
</tr>
<tr>
<td>Prove</td>
<td>Membership Defined(opn)'</td>
</tr>
</tbody>
</table>

**Reasoning:** Ref: Sent Messages Monotonic; existential witnesses to Membership As and Membership Defined carry forward

### Invariant: ProposedImpliesActiveMember

<table>
<thead>
<tr>
<th>Introduce</th>
<th>view ∈ ViewIds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduce</td>
<td>cohort ∈ Cohorts</td>
</tr>
<tr>
<td>Introduce</td>
<td>opn ∈ Opns</td>
</tr>
<tr>
<td>Assume</td>
<td>(\land Proposed(view, cohort, opn))</td>
</tr>
<tr>
<td></td>
<td>(\land LL!Replica(cohort)!CurView = view)</td>
</tr>
<tr>
<td></td>
<td>(\Rightarrow)</td>
</tr>
<tr>
<td></td>
<td>(\land LL!Replica(cohort)!ActiveMember)</td>
</tr>
<tr>
<td></td>
<td>(\land[opn ∈ DOMAIN LL!Replica(cohort)!CsState.membership.Map])</td>
</tr>
<tr>
<td>Assume</td>
<td>((\land Proposed(view, cohort, opn)) )</td>
</tr>
<tr>
<td></td>
<td>(\land LL!Replica(cohort)!CurView = view)'</td>
</tr>
<tr>
<td>Prove</td>
<td>((\land LL!Replica(cohort)!ActiveMember) )</td>
</tr>
<tr>
<td></td>
<td>(\land[opn ∈ DOMAIN LL!Replica(cohort)!CsState.membership.Map]')</td>
</tr>
</tbody>
</table>

**Reasoning:**

### Theorem: ProposedImpliesMembershipAs

<table>
<thead>
<tr>
<th>Hypotheses of</th>
<th>ProposedImpliesMembershipDefined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduce</td>
<td>view ∈ ViewIds</td>
</tr>
<tr>
<td>Introduce</td>
<td>cohort ∈ Cohorts</td>
</tr>
<tr>
<td>Introduce</td>
<td>opn ∈ Opns</td>
</tr>
<tr>
<td>Assume</td>
<td>Proposed(view, cohort, opn)</td>
</tr>
<tr>
<td>Prove</td>
<td>Membership As(opn, Membership(opn), LL!SentMessages)</td>
</tr>
</tbody>
</table>

```plaintext
Assume

DesginationReflects VcAcks(view, cohortSet)

DesginationReflects VcAcks(view, cohortSet)'

Prove

DesginationReflects VcAcks(view, cohortSet)'}

Reasoning: Ref: PrimaryDesginated Monotonic, Ref: VcAcked Monotonic; existential witnesses carry forward

---
Step 1. of 1

∃ membership ∈ Memberships : MembershipAs(opn, membership, LL!SentMessages)

Reasoning: Ref: ProposedImpliesMembershipDefined ; Defn MembershipDefined

Reasoning: Defn Membership; choose axiom

Theorem CommittedMonotonic

Introduce $c ∈ Cohorts$
Introduce $opn ∈ Opns$
Introduce $opv ∈ CsOps$
Assume $CommittedAs(c, opn, opv)$
Prove $CommittedAs(c, opn, opv)'$

Reasoning: Ref: SentMessagesMonotonic; existential witness carries forward

Invariant ProposedImpliesProposed

Introduce $view ∈ Viewslds$
Introduce $c ∈ Cohorts$
Introduce $opn ∈ Opns$
Introduce $opv ∈ CsOps$
Assume $ProposedAs(view, c, opn, opv) ⇒ ProposedByAnyAs(view, opn, opv)$
Assume $ProposedAs(view, c, opn, opv)$
Prove $ProposedByAnyAs(view, opn, opv)'$

Step 1. of 1

Prove $ProposedByAnyAs(view, opn, opv)$

Case 1.1. of 2

$LL!Replica(c)!PrepareAction(view, opn, opv)$
Reasoning: (1.1.): Receive Message[m] provides witness for ProposedByAnyAs

Case 1.2. of 2

$¬LL!Replica(c)!PrepareAction(view, opn, opv)$
Step 1.2.1. of 1

$ProposedAs(view, c, opn, opv)$
Reasoning: (1.2.1.): No prepare sent on this step
Reasoning: (1.2.): induction hypothesis
Reasoning: (1.): Case analysis

Reasoning: Ref: ProposedAsMonotonic

Theorem QuorumProposedAsMonotonic
Introduce $v \in \text{ViewIds}$
Introduce $\text{opn} \in \text{Opns}$
Introduce $\text{opv} \in \text{CsOps}$
Assume $\text{QuorumPreparedAs}(v, \text{opn}, \text{opv})$
Prove $\text{QuorumPreparedAs}(v, \text{opn}, \text{opv})'$

Reasoning: Apply $\text{Ref:PreparedAsMonotonic}$ on each member of the quorum that witnesses to the assumption.

Theorem $\text{CurViewsMonotonic}$
Introduce $\text{cohort} \in \text{Cohorts}$
Prove $\text{llReplica(cohort)}!\text{CurView} \leq (\text{llReplica(cohort)}!)\text{CurView}'$

Reasoning: Case analysis on actions; only $\text{VeAck}$ changes, and its enabling condition is sufficient to prove this theorem.

Invariant $\text{CurViewLaterThanAllPrepares}$
Introduce $\text{view} \in \text{ViewIds}$
Introduce $\text{cohort} \in \text{Cohorts}$
Introduce $\text{opn} \in \text{Opns}$
Assume $\text{Prepared(view, cohort, opn)} \Rightarrow \text{llReplica(cohort)}!\text{CurView} \geq \text{view}$
Assume $\text{Prepared(view, cohort, opn)' \Rightarrow (llReplica(cohort)!CurView' \geq view')}$

Summary: Only the $\text{Prepare}$ action can cause trouble, but its preconditions provide the conclusion.

Case 1. of 2
$\exists m \in \text{ProposedMsg} :$
$\land m.\text{view} = \text{view}$
$\land \text{llReplica(cohort)}!\text{Prepare}(m)$

Defn $m \triangleq$

\text{Choose } m \in \text{ProposedMsg} :
$\land m.\text{view} = \text{view}$
$\land \text{llReplica(cohort)}!\text{Prepare}(m)$

Step 1.1. of 2
$m.\text{view} = \text{llReplica(cohort)}!\text{CurView}$
Reasoning (1.1): $\text{Defn Prepare action}$

Step 1.2. of 2
$(\text{llReplica(cohort)}!\text{CurView'}) = \text{llReplica(cohort)}!\text{CurView}$
Reasoning (1.2): $\text{Defn Prepare action leaves CurView unchanged}$
Reasoning (1.1): last two steps, case conjunct 1

Case 2. of 2
$\forall m \in \text{ProposedMsg} :$
\[
(\neg \\
\land m. \text{view} = \text{view} \\
\land \text{LL!Replica} (\text{cohort})!\text{Prepare} (m))
\]

Step 2.1. of 3

\[(\text{LL!SentMessages'}) \cap \text{PreparedMsg} = \text{LL!SentMessages} \cap \text{PreparedMsg}\]

Reasoning (2.1): Only \text{Prepare} action sends \text{PreparedMsg}

Step 2.2. of 3

\text{Prepared} (\text{view, cohort, opv})

Reasoning (2.2): Definition \text{Prepared} relies only on variable \text{LL!SentMessages}

Step 2.3. of 3

\text{LL!Replica} (\text{cohort})!\text{CurView} \geq \text{view}

Reasoning (2.3): induction hypothesis

Reasoning (2.): Ref: \text{CurViewMonotonic}

Reasoning: Case analysis.

\textbf{Invariant:} \text{CurViewLaterThanAllProposed}

\textbf{Introduce:}
- \text{view} \in \text{ViewIds}
- \text{cohort} \in \text{Cohorts}
- \text{opn} \in \text{Opns}
- \text{opv} \in \text{CsOpns}

\textbf{Assume:}

\text{ProposedAs} (\text{view, cohort, opn, opv}) \Rightarrow \text{LL!Replica} (\text{cohort})!\text{CurView} \geq \text{view}

\textbf{Assume:}

\text{ProposedAs} (\text{view, cohort, opn, opv})'

\textbf{Prove:}

\text{LL!Replica} (\text{cohort})!\text{CurView} \geq \text{view}'

\textbf{Summary:} Only the \text{Propose} and \text{BecomePrimary} actions can cause trouble, but their preconditions provide the conclusion.

\textbf{Case 1.} of 3

\text{LL!Replica} (\text{cohort})!\text{ProposeAction} (\text{view, opn, opv})

\textbf{Step 1.1.} of 1

\text{LL!Replica} (\text{cohort})!\text{CurView} = \text{view}

Reasoning (1.1): Defn \text{Propose action}

Reasoning (1.): algebra

\textbf{Case 2.} of 3

\land (3 \text{ m} \in \text{PrimaryDesignatedMsg} : \text{LL!Replica} (\text{cohort})!\text{BecomePrimary} (\text{m}))

\land (\neg \text{ProposedAs} (\text{view, cohort, opn, opv}))

\textbf{Step 2.1.} of 1

\text{LL!Replica} (\text{cohort})!\text{CurView}' = \text{view}

Reasoning (2.1.): If \text{ProposedAs} became true on this action, it’s because \text{BecomePrimary}

added a new message to \text{SentMessages}; Defn \text{BecomePrimary} says that all new messages

have \text{m.view} \equiv \text{CurView}.
Reasoning (2): algebra
DefaultCase 3. of 3
Step 3.1. of 2
UNCHANGED ProposedAs\(\text{view, cohort, opn, opv}\)
Reasoning (3.1): Inspection of remaining actions: none add an appropriate Proposed\(\text{Msg}\)
to Sent\(\text{Messages}\).
Step 3.2. of 2
\(LL!\text{Replica(cohort)}!\text{CurView} \geq \text{view}\)
Reasoning (3.2): induction hypothesis
Reasoning (3): Ref: Cur\(\text{ViewsMonotonic}\)
Reasoning: Case analysis.

Invarian\(\text{t CurView} \text{LaterThan AllVcAcked}\)
Introduce \(\text{view} \in \text{View\(\text{Ids}\)}\)
Introduce \(\text{cohort} \in \text{Cohorts}\)
Assume \(\text{VcAckedView}(\text{view, cohort}) \Rightarrow LL!\text{Replica(cohort)}!\text{CurView} \geq \text{view}\)
Assume \(\text{VcAckedView}(\text{view, cohort})^{'}\)
Prove \(\text{v}_{\text{LL!Replica(cohort)!CurView} \geq \text{view}}^{'}\)
Summary: Only the VcAck action can cause trouble, but its assignment of Cur\(\text{View}\) provides the conclusion.
Case 1. of 2
\(\exists \text{preparedOps} \in \text{PreparedOpsType} :\)
\(\text{LL!Replica(cohort)}!\text{VcAckAction}(\text{view, preparedOps})\)
Step 1.1. of 1
\((\text{LL!Replica(cohort)}!\text{CurView})^{'} = \text{view}\)
Reasoning (1.1): Def\(\text{VcAck action}\)
Reasoning (1): algebra
DefaultCase 2. of 2
Step 2.1. of 2
UNCHANGED Vc\(\text{AckedView}(\text{view, cohort})\)
Reasoning (2.1): Inspection of remaining actions: none add an appropriate Proposed\(\text{Msg}\)
to Sent\(\text{Messages}\).
Step 2.2. of 2
\(\text{LL!Replica(cohort)}!\text{CurView} \geq \text{view}\)
Reasoning (2.2): induction hypothesis
Reasoning (2): Ref: Cur\(\text{ViewsMonotonic}\)
Reasoning: Case analysis.
\begin{itemize}
\item **Invariant** \textit{PrimaryDesignatedSentByInitiator}
\item \textbf{Introduce} \hspace{1cm} \( m \in \text{PrimaryDesignatedMsg} \cap \text{SentMessages} \)
\item \textbf{Assume} \hspace{1cm} \( m.\text{sender} = m.\text{view}.\text{viewInitiator} \)
\item \textbf{Prove} \hspace{1cm} \text{(m.}\text{sender} = m.\text{view}.\text{viewInitiator})'
\item \textbf{Case 1. of 2}
\item \( \exists \text{config} \in \text{Designation Configurations} : \)
\item \( \land \text{LL!Replica(m.}\text{sender})!\text{DesignatePrimary(config)} \)
\item \( \land m \notin \text{SentMessages} \)
\item \textbf{Step 1.1. of 2}
\item \( m.\text{view} = \text{LL!Replica(m.}\text{sender})!\text{Cur View} \)
\item \textbf{Reasoning (1.1.):} \text{Defn DesignatePrimary; MakePrimaryDesignatedMsg}
\item \textbf{Step 1.2. of 2}
\item \( \text{LL!Replica(m.}\text{sender})!\text{Cur View. viewInitiator} = m.\text{sender} \)
\item \textbf{Reasoning (1.2.):} \text{Defn DesignatePrimary}
\item \textbf{Reasoning (1.): substitution}
\item \textbf{DefaultCase 2. of 2}
\item \textbf{Reasoning (2.):} No other action could send \( m \)
\item \textbf{Reasoning:} Proof by case analysis
\end{itemize}

---

\begin{itemize}
\item **Invariant** \textit{CurViewOfInitiatorLaterThanAllPrimaryDesignateds}
\item \textbf{Introduce} \hspace{1cm} \( \text{view} \in \text{ViewIds} \)
\item \textbf{Assume} \hspace{1cm} \text{PrimaryDesignated(view)} \Rightarrow \text{LL!Replica(view. viewInitiator)! Cur View \geq \text{view} }
\item \textbf{Assume} \hspace{1cm} \text{PrimaryDesignated(view)}'
\item \textbf{Prove} \hspace{1cm} (LL!Replica(view. viewInitiator)! Cur View \geq \text{view})'
\item \textbf{Summary:} Only the DesignatePrimary action can cause trouble, but its preconditions provide the conclusion.
\item \textbf{Case 1. of 2}
\item \( \exists \text{config} \in \text{Designation Configurations} : \)
\item \( \land \text{LL!Replica(config.designator)! DesignatePrimary(config)} \)
\item \( \land \text{config. view} = \text{view} \)
\item \textbf{Defn config \( \triangleq \)}
\item \textbf{CHOOSE config \( \in \text{Designation Configurations} : \)}
\item \( \land \text{LL!Replica(config.designator)! DesignatePrimary(config)} \)
\item \( \land \text{config. view} = \text{view} \)
\item \textbf{Step 1.1. of 1}
\item config.designator = view. viewInitiator
\item \textbf{Reasoning (1.1.):} \text{Defn DesignatePrimary action}
\item \item \textbf{Reasoning (1.):} \text{Defn DesignatePrimary action}
\item \textbf{DefaultCase 2. of 2}
\item \textbf{Reasoning (2.):} No other actions send PrimaryDesignatedMsg; apply induction hypothesis; apply Ref: CurViewMonotonic.
\end{itemize}
Reasoning: Case analysis.

\[
\text{Invariant ProposedImpliesPrimary}
\]
\[
\text{Hypotheses of CurViewLaterThanAllProposeds}
\]
\[
\text{Introduce view } \in \text{ ViewIds}
\]
\[
\text{Introduce cohort } \in \text{ Cohorts}
\]
\[
\text{Introduce opn } \in \text{ Opns}
\]
\[
\text{Introduce opv } \in \text{ CsOps}
\]
\[
\text{Assume}
\]
\[
\wedge \text{LL!Replica(cohort)}!\text{CurView} = \text{view}
\]
\[
\wedge \text{ProposedAs} (\text{view, cohort, opn, opv})
\]
\[
\Rightarrow
\]
\[
\vee \text{LL!Replica(cohort)}!\text{IAmPrimary}
\]
\[
\vee \text{LL!Replica(cohort)}!\text{StaleView}
\]
\[
\text{Assume}
\]
\[
(\wedge \text{LL!Replica(cohort)}!\text{CurView} = \text{view}
\wedge \text{ProposedAs} (\text{view, cohort, opn, opv}))'
\]
\[
\text{Prove}
\]
\[
(\vee \text{LL!Replica(cohort)}!\text{IAmPrimary}
\vee \text{LL!Replica(cohort)}!\text{StaleView})'
\]
\[
\text{Case 1. of 5}
\]
\[
\exists m \in \text{PrimaryDesignatedMsg} : \text{LL!Replica(cohort)}!\text{BecomePrimary}(m)
\]
\[
\text{Step 1.1. of 1}
\]
\[
\text{LL!Replica(cohort)}!\text{IAmPrimary}
\]
\[
\text{Reasoning (1.1.): Definition BecomePrimary}
\]
\[
\text{Reasoning (1.): algebra}
\]
\[
\text{Case 2. of 5}
\]
\[
\text{LL!Replica(cohort)}!\text{ProposeAction} (\text{view, opn, opv})
\]
\[
\text{Reasoning (2.): Definition Propose}
\]
\[
\text{Case 3. of 5}
\]
\[
\exists m \in \text{VcInitiatedMsg} : \text{LL!Replica(cohort)}!\text{VcAck}(m)
\]
\[
\text{Step 3.1. of 3}
\]
\[
\text{LL!Replica(cohort)}!\text{CurView} < (\text{LL!Replica(cohort)}!\text{CurView}')
\]
\[
\text{Reasoning (3.1.): Definition VcAck}
\]
\[
\text{Step 3.2. of 3}
\]
\[
(\text{LL!Replica(cohort)}!\text{CurView}') = \text{view}
\]
\[
\text{Reasoning (3.2.): Antecedent}
\]
\[
\text{Case 4. of 5}
\]

67
\[ LL!\text{Replica}(\text{cohort})!\text{Crash} \]

```
Step 4.1. of 1

LL!\text{Replica}(\text{cohort})!\text{Stale View}'

Reasoning (4.1.): Definition \text{Crash}

Reasoning (4.): algebra

Default Case 5. of 5

Step 5.1. of 1

\( \land \ \text{UNCHANGED} \ LL!\text{Replica}(\text{cohort})!\text{IAmPrimary} \)

\( \land \ \text{UNCHANGED} \ LL!\text{Replica}(\text{cohort})!\text{Stale View} \)

Reasoning (5.1.): inspection of remaining actions

Reasoning (5.): induction hypothesis

Reasoning: Case analysis.
```

**Invariant** \( \text{LastProposedTracksProposals} \)

**Hypotheses of** \( \text{ProposedImpliesPrimary} \)

- Introduce \( \text{view} \in \text{ViewIds} \)
- Introduce \( \text{cohort} \in \text{Cohorts} \)
- Introduce \( \text{opn} \in \text{Opns} \)
- Introduce \( \text{opv} \in \text{CsOps} \)

**Assume**

\( \land \ LL!\text{Replica}(\text{cohort})!\text{CurView} = \text{view} \)

\( \land \ LL!\text{Replica}(\text{cohort})!\text{IAmPrimary} \)

\( \land \ \text{ProposedAs}(\text{view}, \text{cohort}, \text{opn}, \text{opv}) \)

\( \Rightarrow \)

\( \text{opn} \leq LL!\text{Replica}(\text{cohort})!\text{LastProposed} \)

**Assume**

\( (\land LL!\text{Replica}(\text{cohort})!\text{CurView} = \text{view} \land LL!\text{Replica}(\text{cohort})!\text{IAmPrimary} \land \text{ProposedAs}(\text{view}, \text{cohort}, \text{opn}, \text{opv}))' \)

**Prove** \( (\text{opn} \leq LL!\text{Replica}(\text{cohort})!\text{LastProposed})' \)

**Case 1. of 5**

\( \exists \text{m} \in \text{PrimaryDesignatedMsg} : \)

\( \land LL!\text{Replica}(\text{cohort})!\text{BecomePrimary}(\text{m}) \land \text{m.view} = \text{view} \land \text{m.opn} = \text{opn} \land \text{m.opv} = \text{opv} \)

**Step 1.1. of 2**

\( \neg LL!\text{Replica}(\text{cohort})!\text{Stale View} \)

Reasoning (1.1.): Definition \text{BecomePrimary}

**Step 1.2. of 2**

\( \neg \text{ProposedAs}(\text{view}, \text{cohort}, \text{opn}, \text{opv}) \)

Reasoning (1.2.): \text{Ref hypothesis:ProposedImpliesPrimary}
Reasoning (1): Case eliminated by contradiction

Case 2. of 5

\[ \text{LL!Replica(\text{cohort})!ProposeAction(view, opn, opv)} \]

Step 2.1. of 1

\[ (\text{LL!Replica(\text{cohort})!LastProposed'}) = \text{opn} \]

Reasoning (2.1): Definition Propose

Case 3. of 5

\[ \exists m \in \text{VcInitiatedMsg} : \text{LL!Replica(\text{cohort})!VcAck}(m) \]

Reasoning (3): Eliminate case by contradiction: Definition VcAck shows \( \neg \)

\[ \text{LL!Replica(\text{cohort})!!AmPrimary} \]

Case 4. of 5

\[ \text{LL!Replica(\text{cohort})!Crash} \]

Reasoning (4): Eliminate case by contradiction: Definition Crash shows \( \neg \)

\[ \text{LL!Replica(\text{cohort})!!AmPrimary} \]

Default Case 5. of 5

Step 5.1. of 1

\[ \land \text{UNCHANGED} \quad \text{LL!Replica(\text{cohort})!CurView} \]

\[ \land \text{UNCHANGED} \quad \text{LL!Replica(\text{cohort})!!AmPrimary} \]

\[ \land \text{UNCHANGED} \quad \text{LL!Replica(\text{cohort})!LastProposed} \]

\[ \land \text{UNCHANGED} \quad \text{ProposedAs(view, cohort, opn, opv)} \]

Reasoning (5.1): inspection of remaining actions

Reasoning (5): induction hypothesis

Reasoning: Proof by case analysis

Invarante \textit{PrimaryDesignatedPrecludesDesignationNeeded}

Introduce \( \vdash \text{view} \in \text{ViewIds} \)

Assume \( \land \text{PrimaryDesignated(view)} \)

\[ \land \text{LL!Replica(view.viewInitiator)!CurView = view} \]

\[ \Rightarrow \]

\[ (\neg \text{LL!Replica(view.viewInitiator)!DesignationNeeded}) \]

Assume \( (\land \text{PrimaryDesignated(view)} \land \text{LL!Replica(view.viewInitiator)!CurView = view}) \)

Prove \( (\neg \text{LL!Replica(view.viewInitiator)!DesignationNeeded}) \)

Reasoning: Basic action analysis; probably some monotonicity; induction hypothesis

Invarante \textit{OneDesignationPerView}
Introduce \( \text{cohort} \in \text{Cohorts} \)

Assume

PrimaryDesignated(\( LL!\text{Replica(\text{cohort})}!\text{CurView} \)) \implies
(\( \neg LL!\text{Replica(\text{cohort})}!\text{DesignationNeeded} \))

Assume PrimaryDesignated(\( LL!\text{Replica(\text{cohort})}!\text{CurView} \))

Prove \( \neg LL!\text{Replica(\text{cohort})}!\text{DesignationNeeded} \)

Case 1. of 3
\( \exists \text{config} \in \text{DesignationConfigurations} : \)
\( \land LL!\text{Replica(config.designator)}!\text{DesignatePrimary(config)} \)
\( \land \text{config.designator} = \text{cohort} \)
\( \land \text{config.view} = LL!\text{Replica(\text{cohort})}!\text{CurView} \)

Reasoning [1.]: Define DesignatePrimary action

Case 2. of 3
\( \exists \text{m} \in \text{VcInitiatedMsg} : LL!\text{Replica(\text{cohort})}!\text{VcAck(m)} \)

Define \( \text{m} \triangleq \text{CHOOSE m} \in \text{VcInitiatedMsg} : LL!\text{Replica(\text{cohort})}!\text{VcAck(m)} \)

Case 2.1. of 2
\( \text{cohort} = \text{m.view.viewInitiator} \)

Step 2.1.1. of 4
\( LL!\text{Replica(\text{cohort})}!\text{CurView'.viewInitiator} = \text{cohort} \)

Step 2.1.1. of 1
\( LL!\text{Replica(\text{cohort})}!\text{CurView'} = \text{m.view} \)

Reasoning [2.1.1.]: Define VcAck

Reasoning [2.1.1.]: substitution with Case assumption

Step 2.1.2. of 4
PrimaryDesignated(\( LL!\text{Replica(\text{cohort})}!\text{CurView'} \))

Reasoning [2.1.2.]: VcAck doesn’t send a PrimaryDesignatedMsg

Step 2.1.3. of 4
\( LL!\text{Replica(\text{cohort})}!\text{CurView'} \leq LL!\text{Replica(\text{cohort})}!\text{CurView} \)

Step 2.1.3.1. of 1
\( LL!\text{Replica(\text{cohort})}!\text{CurView'} \leq LL!\text{Replica(\text{cohort})}!\text{CurView'}!\text{viewInitiator}!\text{CurView} \)

Reasoning [2.1.3.1.]: Ref:CurView OfInitiator Later Than AllPrimaryDesignateds

Reasoning [2.1.3.]: substitution with Ref:Step 2.1.1.

Step 2.1.4. of 4
\( LL!\text{Replica(\text{cohort})}!\text{CurView'} = LL!\text{Replica(\text{cohort})}!\text{CurView} \)

Reasoning [2.1.4.]: Forced by Ref:CurViewsMonotonic

Reasoning [2.1.]: Contradicts define VcAck action, eliminating the case.

Default Case 2.2. of 2

Reasoning [2.2.]: Define VcAck sets Designation Needed' = false, satisfying the goal.

Reasoning [2.]: Proof by case analysis

Default Case 3. of 3

Reasoning (3.): No other action could have sent a message that would make PrimaryDesignated true; hence it was true before. Apply induction hypothesis. No action besides VcAck can set Designation Needed true, so Designation Needed' = false.
Reasoning: Proof by case analysis

Invariant \textit{UniquePrimaryDesignatedMessage}

Hypotheses of \textit{OneDesignationPerView}

\begin{align*}
\text{Introduce} & \quad m_1 \in \text{PrimaryDesignatedMsg} \\
\text{Introduce} & \quad m_2 \in \text{PrimaryDesignatedMsg}
\end{align*}

Assume
\begin{align*}
\land m_1 \in \text{SentMessages} \\
\land m_2 \in \text{SentMessages} \\
\land m_1.\text{view} = m_2.\text{view}
\end{align*}

\Rightarrow

m_1 = m_2

Assume
\begin{align*}
(\land m_1 \in \text{SentMessages} \\
\land m_2 \in \text{SentMessages} \\
\land m_1.\text{view} = m_2.\text{view})
\end{align*}

Prove
\begin{align*}
(m_1 = m_2)
\end{align*}

Step 1. of 1

Assume (1.1.) \quad m_1 \in \text{SentMessages} \Rightarrow m_2 \in \text{SentMessages}

Prove \quad m_1 = m_2

Case 1.1. of 2

m_1 \notin \text{SentMessages}

\text{Defn} \quad \textbf{config} \overset{\Delta}{=} 

\text{\textbf{CHOOSE} config} \in \text{DesignationConfigurations}:
\begin{align*}
\land \text{LL!Replica}(m_1.\text{sender})!\text{DesignatePrimary}(\text{config}) \\
\land \text{config.view} = m_1.\text{view} \\
\land \text{config.newPrimary} = m_1.\text{newPrimary} \\
\land \text{MaxTruncationPoint}(\text{config.msgs}) = m_1.\text{maxTruncationPoint} \\
\land \text{AggregatePreparedOps}(\text{config.msgs}) = m_1.\text{preparedOps}
\end{align*}

Step 1.1.1. of 4

\begin{align*}
\land \text{LL!Replica}(m_1.\text{sender})!\text{DesignatePrimary}(\text{config}) \\
\land \text{config.view} = m_1.\text{view} \\
\land \text{config.newPrimary} = m_1.\text{newPrimary} \\
\land \text{MaxTruncationPoint}(\text{config.msgs}) = m_1.\text{maxTruncationPoint} \\
\land \text{AggregatePreparedOps}(\text{config.msgs}) = m_1.\text{preparedOps}
\end{align*}

Reasoning (1.1.1.): m_1 was sent in this step; this configuration must have done it.

Step 1.1.2. of 4

\begin{align*}
\text{LL!Replica}(m_1.\text{sender})!\text{DesignationNeeded}
\end{align*}

Reasoning (1.1.2.): Defn DesignatePrimary action

Step 1.1.3. of 4

\begin{align*}
\neg\text{PrimaryDesignated(LL!Replica}(m_1.\text{sender})!\text{Cur View})
\end{align*}
Reasoning (1.1.3.): Contrapositive of Ref hypothesis: OneDesignationPerView

Step 1.1.4. of 4
  \( m_2 \notin \text{SentMessages} \)

Reasoning (1.1.4.): Defn PrimaryDesignated

Reasoning (1.1.): Message \( m_2 \) was sent this step, and this action sent only one message \( (m_1) \). So they must be the same message.

DefaultCase 1.2. of 2

Step 1.2.1. of 2
  \( m_1 \in \text{SentMessages} \)

Reasoning (1.2.1.): No other action could send \( m_1 \)

Step 1.2.2. of 2
  \( m_2 \in \text{SentMessages} \)

Reasoning (1.2.2.): Assumption Ref: Assumption 1.4.
Reasoning (1.2.): induction hypothesis; Ref: PrimaryDesignatedMonotonic

Reasoning (1.): Proof by case analysis

Reasoning: without loss of generality, we can apply the substep with \( m_1 \) and \( m_2 \) swapped.

Theorem UniquePrimaryDesignated

Hypotheses of UniquePrimaryDesignation Message
Intro \( \text{view} \in \text{Viewlds} \)
Intro \( \text{cohort1} \in \text{Cohorts} \)
Intro \( \text{cohort2} \in \text{Cohorts} \)
Assume PrimaryDesignatedAs(\text{view, cohort1})
Assume PrimaryDesignatedAs(\text{view, cohort2})
Prove cohort1 = cohort2

Summary: Easily falls out of Ref hypothesis: UniquePrimaryDesignation Message.

Defn \( m_1 \equiv \)
  \( \text{CHOOSE } m \in \text{SentMessages} \cap \text{PrimaryDesignatedMsg} : \)
  \( \land m.\text{view} = \text{view} \)
  \( \land m.\text{newPrimary} = \text{cohort1} \)

Defn \( m_2 \equiv \)
  \( \text{CHOOSE } m \in \text{SentMessages} \cap \text{PrimaryDesignatedMsg} : \)
  \( \land m.\text{view} = \text{view} \)
  \( \land m.\text{newPrimary} = \text{cohort2} \)

Step 1. of 1
  \( m_1 = m_2 \)

Reasoning (1.): Assumptions guarantee \( \text{CHOOSEs} \) succeed; Ref hypothesis: UniquePrimaryDesignation Message

Reasoning: cohort1 = m1.newPrimary = m2.newPrimary = cohort2
Invariants

\text{PreparedOps} \times \text{PreparedImpliesPrepared}

\text{Introduce: } v2 \in \text{ViewIds}
\text{Introduce: } cohort \in \text{Cohorts}
\text{Introduce: } \text{opn} \in \text{Opns}
\text{Assume: }
\text{opn} \in \text{DOMAIN } LL!\text{Replica}(\text{cohort})!\text{PreparedOps} \Rightarrow
\text{Prepared}(LL!\text{Replica}(\text{cohort})!\text{PreparedOps}[, \text{view}, \text{cohort}, \text{opn}])

\text{Assume: } (\text{opn} \in \text{DOMAIN } LL!\text{Replica}(\text{cohort})!\text{PreparedOps})'

\text{Prove: }\text{Prepared}(LL!\text{Replica}(\text{cohort})!\text{PreparedOps}[\text{opn}, \text{view}, \text{cohort}, \text{opn}])'

\text{Case 1 of 3}
\exists m \in \text{ProposedMsg} : LL!\text{Replica}(\text{cohort})!\text{Prepare}(m)
\text{Defn: } m \triangleq \text{CHOOSE } m \in \text{ProposedMsg} : LL!\text{Replica}(\text{cohort})!\text{Prepare}(m)

\text{Case 1.1 of 2}
\text{m.opn} = \text{opn}

\text{Step 1.1.1 of 2}
\text{Prepared}(LL!\text{Replica}(\text{cohort})!\text{CurView}, \text{cohort}, \text{opn})'

\text{Reasoning (1.1.1): } \text{Defn } \text{Prepare } \text{action arguments to } \text{MakeProposedMsg}

\text{Step 1.1.2 of 2}
(\text{LL!Replica(cohort)\text{PreparedOps}[\text{opn}.\text{view}'] } = \text{LL!Replica(cohort)\text{CurView}}

\text{Reasoning (1.1.2): } \text{Defn } \text{Prepare } \text{action construction of } \text{PreparedOps'}

\text{Reasoning (1.1): } \text{substitution satisfies the proof goal}

\text{Default Case 1.2 of 2}

\text{Step 1.2.1 of 1}
\text{opn} \in \text{DOMAIN } LL!\text{Replica}(\text{cohort})!\text{PreparedOps}

\text{Reasoning (1.2.1): } \text{Defn } \text{Prepare } \text{defines } \text{DOMAIN } \text{PreparedOps'} \text{ with a union on old value}

\text{Reasoning (1.2): } \text{Apply induction hypothesis}

\text{Reasoning (1): } \text{Proof by case analysis}

\text{Case 2 of 3}
\exists m \in \text{PrimaryDesignatedMsg} :
\land m.\text{view} = v2
\land LL!\text{Replica}(\text{cohort})!\text{BecomePrimary}(m)

\text{Defn: } m \triangleq 
\text{CHOOSE } m \in \text{PrimaryDesignatedMsg} :
\land m.\text{view} = v2
\land LL!\text{Replica}(\text{cohort})!\text{BecomePrimary}(m)

\text{Step 2.1 of 1}
\text{opn} \in \text{DOMAIN } m.\text{prevPrepares}

\text{Reasoning (2.1): } \text{Defn } \text{BecomePrimary } \text{sets } \text{PreparedOps'} = \text{m.prevPrepares}

\text{Reasoning (2.1): } \text{Defn } \text{BecomePrimary } \text{action sends the required message (argument to SendMessageSet)}

\text{Default Case 3 of 3}

\text{Step 3.1 of 1}
\text{opn} \in \text{DOMAIN } LL!\text{Replica}(\text{cohort})!\text{PreparedOps}

73
Reasoning [3.1.]: All other actions leave PreparedOps unchanged
Reasoning [3.3.]: induction hypothesis; Ref: PreparedAsMonotonic
Reasoning: Proof by case analysis

Invariant VcAck\text{PreparedImpilesPrepared}

Hypotheses of PreparedOps\text{PreparedImpilesPrepared}

- \text{Introduction} v2 \in \text{ViewIds}
- \text{Introduction} cohort \in \text{Cohorts}
- \text{Introduction} opn \in \text{Ops}
- \text{Introduction} \text{preparedOpInfo} \in \text{PreparedOpInfo}

Assume

\text{VcAckPreparedOpAs}(v2, cohort, opn, \text{preparedOpInfo}) \Rightarrow \text{Prepared}(\text{preparedOpInfo}.\text{view}, cohort, opn)

Assume \text{VcAckPreparedOpAs}(v2, cohort, opn, \text{preparedOpInfo})

Prove \text{Prepared}(\text{preparedOpInfo}.\text{view}, cohort, opn)

Case 1. of 2

\land (\exists m \in \text{SentMessagesMatching}(cohort, \text{VcInitMsg}):
\land (\text{LL!Replica}(cohort)!\text{VcAck}(m)
\land m.\text{view} = v2))

\land (\text{LL!Replica}(cohort)!\text{PreparedOps}') = \text{preparedOpInfo}

Step 1.1. of 4

\text{PreparedOpInfo}\text{FromPreparedOps}(\text{LL!Replica}(cohort)!\text{PreparedOps}', opn) = \text{preparedOpInfo}

Reasoning [1.1.]: Defn VcAck action

Step 1.2. of 4

opn \in \text{DOMAIN} (\text{LL!Replica}(cohort)!\text{PreparedOps}')

Step 1.2.1. of 1

\text{preparedOpInfo} \neq \text{PreparedOpZero}

Reasoning [1.2.1.]: as defined when it was introduced

Reasoning [1.2.]: Definition \text{PreparedOpInfo}\text{FromPreparedOps}

Step 1.3. of 4

\text{Prepared}(\text{LL!Replica}(cohort)!\text{PreparedOps}[opn].\text{view}, cohort, opn)

Reasoning [1.3.]: Ref hypothesis: PreparedOps\text{PreparedImpilesPrepared} supports Ref: PreparedOps\text{PreparedImpilesPrepared}

Step 1.4. of 4

(\text{LL!Replica}(cohort)!\text{PreparedOps}'[opn].\text{view} = \text{preparedOpInfo}.\text{view}

Step 1.4.1. of 1

(\text{LL!Replica}(cohort)!\text{PreparedOps}')[opn] = \text{preparedOpInfo}

Reasoning [1.4.1.]: Last conjunct of Case condition; Defn VcAck

Reasoning [1.4.]: Substitution

Reasoning [1.]: Substitution

DefaultCase 2. of 2
Step 2.1. of 2
\( \text{VcAckPreparedOpAs}(v_2, \text{cohort}, \text{opn}, \text{preparedOpInfo}) \)

Reasoning (2.1.): No actions in this case send a message that could make the statement transition to true.

Step 2.2. of 2
\( \text{Prepared}(\text{preparedOpInfo}. \text{view}, \text{cohort}, \text{opn}) \)

Reasoning (2.2.): induction hypothesis

Reasoning (2.1.): \text{Ref:PreparedAsMonotonic}

Reasoning: Case analysis on actions

Invariant \( \text{IAmPrimaryImpliesPrimaryDesignated} \)

Introduce \( \text{view} \in \text{ViewIds} \)

Introduce \( \text{cohort} \in \text{Cohorts} \)

Assume
\[
\wedge \text{LL!Replica(cohort)!CurView} = \text{view} \\
\wedge \text{LL!Replica(cohort)!IAmPrimary} \\
\Rightarrow
\text{PrimaryDesignatedAs(view, cohort)}
\]

Assume
\[
( \wedge \text{LL!Replica(cohort)!CurView} = \text{view} \\
\wedge \text{LL!Replica(cohort)!IAmPrimary})'
\]

Prove \( \text{PrimaryDesignatedAs(view, cohort)}' \)

Step 1. of 1
\( \text{PrimaryDesignatedAs(view, cohort)} \)

Case 1.1. of 2
\[
\exists m \in \text{PrimaryDesignatedMsg} : \\
\wedge m.\text{view} = \text{view} \\
\wedge \text{LL!Replica(cohort)!BecomePrimary}(m)
\]

Reasoning (1.1.): Message \( m \) is a witness to \( \text{PrimaryDesignatedAs} \). Defn \( \text{BecomePrimary} \); Defn \( \text{ReceiveMessage} \); Defn \( \text{PrimaryDesignatedAs} \)

Default Case 1.2. of 2

Step 1.2.1. of 3
\( \text{LL!Replica(cohort)!IAmPrimary} \)

Reasoning (1.2.1.): No actions on cohort other than \( \text{BecomePrimary} \) make \( \text{IAmPrimary} \) transition to true

Step 1.2.2. of 3

Introduce \( m \in \text{VcInitiatedMsg} \)

Prove \( \neg \text{LL!Replica(cohort)!VcAck}(m) \)

Reasoning (1.2.2.): \( \text{VcAck} \) sets \( \text{IAmPrimary}' = \text{false} \), which contradicts antecedent conjunct \( \text{IAmPrimary} \)

75
Step 1.2.3. of 3

\[ \text{LL}1. \text{Replica(cohort)} \parallel \text{CurView} = \text{view} \]

Summary: No other actions on cohort change CurView

Step 1.2.3.1. of 1

UNCHANGED \[ \text{LL}1. \text{Replica(cohort)} \parallel \text{CurView} \]

Reasoning (1.2.3.1.): No other actions on cohort change CurView

Reasoning (1.2.3.): Antecedent conjunct CurView = view

Reasoning (1.2.): induction hypothesis

Reasoning (1.1.): Case analysis.

Reasoning: Ref: PrimaryDesignatedMonotonic

Invariant: ProposedImpliesPrimaryDesignated

Hypotheses of: IAmPrimaryImpliesPrimaryDesignated

Introduce \[ \text{view} \in \text{ViewIds} \]

Introduce \[ \text{cohort} \in \text{Cohorts} \]

Introduce \[ \text{opn} \in \text{Ops} \]

Introduce \[ \text{opv} \in \text{CsOps} \]

Assume \[ \text{ProposedAs(view, cohort, opn, opv)} \Rightarrow \text{PrimaryDesignatedAs(view, cohort)} \]

Assume \[ \text{ProposedAs(view, cohort, opn, opv)}' \]

Prove \[ \text{PrimaryDesignatedAs(view, cohort)}' \]

Step 1. of 1

PrimaryDesignatedAs(view, cohort)

Case 1.1. of 3

\[ \text{LL}1. \text{Replica(cohort)} \parallel \text{ProposeAction(view, opn, opv)} \]

Step 1.1.1. of 2

\[ \text{LL}1. \text{Replica(cohort)} \parallel \text{IAmPrimary} \]

Reasoning (1.1.1.): Definition of ProposeAction

Step 1.1.2. of 2

\[ \text{LL}1. \text{Replica(cohort)} \parallel \text{CurView} = \text{view} \]

Reasoning (1.1.2.): Definition of ProposeAction

Reasoning (1.1.): Ref hypothesis: IAmPrimaryImpliesPrimaryDesignated

Defn \[ \text{rec} \equiv \]

\[ \text{CHOOSE rec} \in [\text{cohort} : \text{Cohorts, m : PrimaryDesignatedMsg}] : \]

\[ \text{LL}1. \text{Replica(rec.cohort)} \parallel \text{BecomePrimary(rec.m)} \]

Case 1.2. of 3

\[ \text{LL}1. \text{Replica(rec.cohort)} \parallel \text{BecomePrimary(rec.m)} \]

Reasoning (1.2.): rec.m is the witness to PrimaryDesignatedAs(view, cohort)

DefaultCase 1.3. of 3

Step 1.3.1. of 1

ProposedAs(view, cohort, opn, opv)

Reasoning (1.3.1.): No other step emits a ProposedMsg for opn, which is needed for ProposedAs to transition from false to true.
Invariant \( \text{ProposedInSameViewDoNotConflict} \)

Hypotheses of \( \text{ProposedImpliesPrimaryDesignated} \)
Hypotheses of \( \text{LastProposedTracksProposals} \)
Hypotheses of \( \text{UniquePrimaryDesignationMessage} \)
Hypotheses of \( \text{ProposedImpliesActiveMember} \)
Hypotheses of \( \text{UniquePrimaryDesignated} \)

Introduce \( \text{view} \in \text{ViewIds} \)
Introduce \( \text{cohort1} \in \text{Cohorts} \)
Introduce \( \text{cohort2} \in \text{Cohorts} \)
Introduce \( \text{opn} \in \text{Opns} \)
Introduce \( \text{opv1} \in \text{CsOps} \)
Introduce \( \text{opv2} \in \text{CsOps} \)

Assume
\[ \land \text{ProposedAs(view, cohort1, opn, opv1)} \]
\[ \land \text{ProposedAs(view, cohort2, opn, opv2)} \]
\[ \Rightarrow \]
\[ \text{opv1} = \text{opv2} \]

Assume
\[ (\land \text{ProposedAs(view, cohort1, opn, opv1)} \]
\[ \land \text{ProposedAs(view, cohort2, opn, opv2)} \)

Prove \( (\text{opv1} = \text{opv2})' \)

Step 1. of 4
\( \text{cohort1} = \text{cohort2} \)

Step 1.1. of 2
\( \text{PrimaryDesignatedAs(view, cohort1)}' \)

Reasoning (1.1): Antecedent conjunct 1; Ref: ProposedImpliesPrimaryDesignated

Step 1.2. of 2
\( \text{PrimaryDesignatedAs(view, cohort2)}' \)

Reasoning (1.2): Antecedent conjunct 2; Ref: ProposedImpliesPrimaryDesignated

Reasoning (1): Ref: UniquePrimaryDesignated

Case 2. of 4
\[ \exists \text{opv} \in \text{CsOps} : \text{LL!Replica(cohort1)!ProposeAction(view, opn, opv)} \]

Step 2.1. of 3
\[ \text{LL!Replica(cohort1)!LastProposed} = \text{opn} - 1 \]

Reasoning (2.1): Defs ProposeAction

Step 2.2. of 3
\[ \land \text{LL!Replica(cohort1)!Cur View = view} \]
\[\text{LL}\!\text{Replica(cohort1)\!\text{IAmPrimary}}\]

Reasoning (2.2): Defn ProposeAction

Step 2.3. of 3
\[\neg \text{Proposed}(\text{view, cohort1, opn})\]

Reasoning (2.3): Ref hypothesis: LastProposedTracksProposals

Reasoning (2.): No proposals for \(\text{opn}\) in previous state, and action only proposes a single \(\text{opn}\), so the same proposal message must make both \(\text{ProposedAs}'\) statements true; hence \(\text{opn} = \text{opv} = \text{opv2}\).

Case 3. of 4
\[\exists m \in \text{PrimaryDesignatedMsg} : \]
\[\land \text{LL}\!\text{Replica(cohort1)\!BecomePrimary}(m)\]
\[\land m.\text{view} = \text{view}\]
\[\land \text{opn} \in \text{NotPrevPrepared}(m)\]

Summary: If cohort1 is just now becoming the primary, then it had proposed nothing (in this view) before this step. Therefore, whatever messages support the \(\text{ProposedAs}'\) assumptions must have been sent as a part of the \(\text{BecomePrimary}\) action.

Step 3.1. of 1
Introduce \(\text{opv} \in \text{CsOps}\)
Prove \(\neg \text{ProposedAs}(\text{view, cohort1, opn, opv})\)

Step 3.1.1. of 2
\[\text{LL}\!\text{Replica(cohort1)\!Cur View = view}\]
Reasoning (3.1.1.): Defn BecomePrimary

Step 3.1.2. of 2
\[\neg \text{LL}\!\text{Replica(cohort1)\!ActiveMember}\]
Reasoning (3.1.2.): Defn BecomePrimary

Reasoning (3.1.): Contrapositive of Ref hypothesis: ProposedImpliesActiveMember

Reasoning (3.): For each \(\text{opn}\), either \(\text{BecomePrimary}\) sends no proposal for it, or it sends a \(\text{NoOp}\), or it sends some \(\text{opn}\) from \(m.\text{prevPrepares}\); but in any case, a single message. That message is the only one that can witness to the two assumptions, so they must have the equal values for \(\text{opn}\).

Default Case 4. of 4

Reasoning (4.): No new proposals in this view for \(\text{opn} \text{sent}\); apply induction hypothesis and Ref: ProposedAsMonotonic

Reasoning: Proof by case analysis

Theorem PreparedsInSameViewDoNotConflict

Hypotheses of PreparedImpliesProposed
Hypotheses of ProposedsInSameViewDoNotConflict

Introduce \(v \in \text{ViewIds}\)
Introduce \(c \in \text{Cohorts}\)
Introduce \(\text{opn} \in \text{Ops}\)
Introduce \(\text{opn}1 \in \text{CsOps}\)
Introduce \( opv2 \in CsOps \)
Assume \( \text{PreparedAs}(v, c, opn, opv1) \)
Assume \( \text{PreparedAs}(v, c, opn, opv2) \)
Prove \( opv1 = opv2 \)

Step 1. of 2
\( \text{PreparedAs}(v, c, opn, opv1) \)
\begin{itemize}
\item Reasoning (1.): \text{Ref:PreparedImpliesProposed}
\end{itemize}

Step 2. of 2
\( \text{PreparedAs}(v, c, opn, opv2) \)
\begin{itemize}
\item Reasoning (2.): \text{Ref:PreparedImpliesProposed}
\end{itemize}
\begin{itemize}
\item Reasoning: \text{Ref:PrepotsInSameViewDoNotConflict}
\end{itemize}
\( \land LL\text{-}Replica(c) \mid \text{Prepare}(m) \)
\( \land m.\text{opn} = \text{opn} \)

Step 1.1. of 1
\( LL\text{-}Replica(c) \mid \text{PreparedOps}[\text{opn}] = [\text{view} \mapsto v1, \text{opv} \mapsto \text{opv}] \)

Case 1.1.1. of 3
\( LL\text{-}Replica(c) \mid \text{CurView} < v1 \)

Reasoning (1.1.1.): Ref hypothesis: CurViewLaterThanAllPreparads eliminates case by contradiction

Case 1.1.2. of 3
\( v1 < LL\text{-}Replica(c) \mid \text{CurView} \)

Reasoning (1.1.2.): Consider witness \( vi = LL\text{-}Replica(c) \mid \text{CurView} \) where Prepared[\( vi, c, \text{opn} \)] ‘because Prepare sends that message: it shows the second antecedent conjunct to be false. Case eliminated by contradiction.

Case 1.1.3. of 3
\( v1 = LL\text{-}Replica(c) \mid \text{CurView} \)

Step 1.1.3.1. of 3
\( \text{PreparedAs}(v1, c, \text{opn}, \text{opv})' \)

Reasoning (1.1.3.1.): Ref:PreparedAsMonotonic

Step 1.1.3.2. of 3
\( \text{PreparedAs}(v1, c, \text{opn}, m.\text{opv})' \)

Reasoning (1.1.3.2.): Defn ProposedMsg sends a message that is witness to PreparedAs

Step 1.1.3.3. of 3
\( m.\text{opv} = \text{opv} \)

Reasoning (1.1.3.3.): Ref:PreparedAsInSameViewDoNotConflict

Reasoning (1.1.): Conclusion follows from assignment to LLReplica(c)\mid PreapredOps[\text{opn}] in action defn

Reasoning (1.1.): Proof by case analysis

Reasoning (1.): Last step satisfies this obligation (it was down a level so it could use the Case pattern.)

DefaultCase 2. of 2

Step 2.1. of 3
\( \text{PreparedAs}(v1, c, \text{opn}, \text{opv}) \)

Reasoning (2.1.): No other action can send PreparedMsg, so unchanged PreparedAs

Step 2.2. of 3
Introduce \( vi \in \text{ViewIds} \)
Assume \( v1 < vi \)
Assume (2.2.1.)
\( vi \leq LL\text{-}Replica(c) \mid \text{CurView} \Rightarrow \neg(\text{Prepared}(vi, c, \text{opn}))' \)
Prove: \( vi \leq LL\text{-}Replica(c) \mid \text{CurView} \Rightarrow \neg(\text{Prepared}(vi, c, \text{opn})) \)

Step 2.2.1. of 1
\( vi \leq LL\text{-}Replica(c) \mid \text{CurView} \Rightarrow \neg(\text{Prepared}(vi, c, \text{opn}))' \)
Reasoning (2.2.1.): Ref:Assumption 22.1. Ref:CurViewsMonotonic

Reasoning (2.2.): Ref:PreparedAsMonotonic

80
Step 2.3. of 3

\( LL! \text{Replica}(c)!.\text{PreparedOps}[\text{opn}] = [\text{view} \mapsto v1, \text{opv} \mapsto \text{opv}] \)

Reasoning (2.3): induction hypothesis

Reasoning (2): In this case (not a Prepare of \( \text{opn} \)), \( \text{PreparedOps}[\text{opn}] \) cannot change. [Note: Truncate action could change \( \text{PreparedOps} \), but current spec explicitly ignores log truncation.]

Reasoning: Proof by case analysis

---

**Invariant**: \( \text{VcAckPreparesReflectViewRecentPrepare} \)

- Hypotheses of: \( \text{PreparedOpsReflectViewRecentPrepare} \)
- Hypotheses of: \( \text{CurViewLaterThanAllVcAcknowledged} \)

**Introduce**

- \( v1 \in \text{ViewIds} \)
- \( v2 \in \text{ViewIds} \)
- \( c \in \text{Cohorts} \)
- \( \text{opn} \in \text{Ops} \)
- \( \text{opv} \in \text{CsOps} \)

**Assume**

\( \land v1 < v2 \)

\( \land \text{PreparedAs}(v1, c, \text{opn}, \text{opv}) \)

\( \land (\forall v_i \in \text{ViewIds} : \)

\( \land v1 < v_i \)

\( \land v_i < v2 \)

\( \Rightarrow \)

\( \neg \text{Prepared}(v_i, c, \text{opn}) ) \)

\( \land \text{VcAckedView}(v2, c) \)

\( \Rightarrow \)

\( \text{VcAckPrepareOpAs}(v2, c, \text{opn}, [\text{opv} \mapsto \text{opv}, \text{view} \mapsto v1]) \)

**Assume**

\( (\land v1 < v2 \)

\( \land \text{PreparedAs}(v1, c, \text{opn}, \text{opv}) \)

\( (\forall v_i \in \text{ViewIds} : \)

\( \land v1 < v_i \)

\( \land v_i < v2 \)

\( \Rightarrow \)

\( \neg \text{Prepared}(v_i, c, \text{opn}) ) \)

\( \land \text{VcAckedView}(v2, c) \)

**Prove**

\( \text{VcAckPrepareOpAs}(v2, c, \text{opn}, [\text{opv} \mapsto \text{opv}, \text{view} \mapsto v1])' \)

**Case 1. of 3**

\( \exists \text{preparedOps} \in \text{PreparedOpsType} : LL! \text{Replica}(c)!.\text{VcAckAction}(v2, \text{preparedOps}) \)

**Step 1.1. of 4**

\( LL! \text{Replica}(c)!.\text{CurView}' = v2 \)

Reasoning (1.1): Define VcAck action

**Step 1.2. of 4**
\( \forall v_i \in \text{ViewIds} : \)
\( \land v_1 < v_i \)
\( \land (v_i < \langle \text{LL!Replica}(c) \cdot \text{Cur View}' \rangle) \Rightarrow (\neg (\text{Prepared}(v_i, c, \text{opn})'))) \)

Reasoning (1.2.): algebra applied to antecedent third conjunct

Step 1.3. of 4
\( \langle \text{LL!Replica}(c) \cdot \text{PreparedOps} \rangle[\text{opn}] = [\text{opn} \mapsto \text{opv}, \text{view} \mapsto v_1] \)

Reasoning (1.3.): Ref: PreparedOpsReflectView RecentPrepare

Step 1.4. of 4
\( \text{VcAcked}(v_2, c, \langle \text{LL!Replica}(c) \cdot \text{PreparedOps} \rangle[\text{opn}])' \)

Reasoning (1.4.): VcAck action puts a message into SentMessages that serves as a witness to VcAcked().

Reasoning [1]: Definition of VcAckPreparedOps

Case 2. of 3

\( \exists m \in \text{ProposedMsg} : \)
\( \land m.\text{view} = v_1 \)
\( \land m.\text{opn} = \text{opn} \)
\( \land \text{LL!Replica}(c) \cdot \text{Prepare}(m) \)

Step 2.1. of 2
\( \neg \text{VcAckedView}(v_2, c) \)

Step 2.1.1. of 2
\( \text{LL!Replica}(c) \cdot \text{Cur View} = v_1 \)

Reasoning (2.1.1.): Defn Prepare

Step 2.1.2. of 2
\( \text{LL!Replica}(c) \cdot \text{Cur View} < v_2 \)

Reasoning (2.1.2.): algebra

Reasoning (2.1.): Contrapositive of Ref hypothesis: CurViewLaterThanAllVcAcknowledged

Step 2.2. of 2
\( \neg (\text{VcAckedView}(v_2, c)') \)

Reasoning (2.2.): This action doesn’t send a VcAckedMsg

Reasoning (2.): case eliminated by contradiction

DefaultCase 3. of 3

Step 3.1. of 6
\( \text{UNCHANGED SentMessagesMatching}(c, \text{VcAckedMsg}) \)

Reasoning [3.1.]: No other action sends a VcAckedMsg

Step 3.2. of 6
\( \text{VcAckedView}(v_2, c) \)

Reasoning [3.2.]: VcAckedView only varies in SentMessagesMatching(c, VcAckedMsg)

Step 3.3. of 6
\( \text{UNCHANGED SentMessagesMatching}(c, \text{PreparedMsg}) \)

Reasoning [3.3.]: No other action sends a PreparedMsg

Step 3.4. of 6
\( \text{PreparedAs}(v_1, c, \text{opn}, \text{opv}) \)
Reasoning [3.4.]: $\text{PreparedAs}$ only varies in 
$\text{SentMessagesMatching}(c, \text{PreparedMsg})$

Step 3.5. of 6
$\forall vi \in \text{ViewIds} :$
$\land \forall i < vi$
$\land (vi < v2 \Rightarrow \neg \text{Prepared}(vi, c, opn)))$

Reasoning [3.5.]: Converse of $\text{Ref:PreparedAsMonotonic}$

Step 3.6. of 6
$\text{VcAckPreparedOpAs}(v2, c, opn, [opv \mapsto opv, \text{view} \mapsto v1])$

Reasoning [3.6.]: induction hypothesis

Reasoning [3.]: $\text{Ref:VcAckedMonotonic}$

Reasoning: Proof by case analysis

Theorem $\text{Plausible ElectionQuorumMonotonic}$

Introduce $\text{view} \in \text{ViewIds}$
Introduce $\text{quorum} \in \text{SUBSET Cohorts}$
Assume $\text{PlausibleElection Quorum}(\text{view}, \text{quorum})$

Prove: $\text{PlausibleElection Quorum}(\text{view}, \text{quorum})'$

Step 1. of 2
$\forall \text{cohort} \in \text{quorum} : (\text{VcAckedView}(\text{view}, \text{cohort})')$

Reasoning [1.]: $\text{Ref:VcAckedViewMonotonic}$

Step 2. of 2
$\text{Designation ReflectsVcAcks}(\text{view}, \text{quorum})'$

Reasoning [2.]: $\text{Ref:SentMessagesMonotonic}$; existential witnesses carry forward

Reasoning: Both conjuncts of $\text{Def: PlausibleElection Quorum}'$ are satisfied

Invariant $\text{Membership Map Domain}$

Introduce $\text{cohort} \in \text{Cohorts}$
Assume $\neg \text{LL!Replica(cohort)!Crash} \Rightarrow$

$\text{DOMAIN LL!Replica(cohort)!CsState.membershipMap = (1 . (LL!Replica(cohort)!CsState.numExecuted + Alpha))}$

Assume $\neg \text{LL!Replica(cohort)!Crash}'$

Prove $\text{DOMAIN LL!Replica(cohort)!CsState.membershipMap = (1 . (LL!Replica(cohort)!CsState.numExecuted + Alpha))}'$

Case 1. of 2
$\exists m \in \text{CommittedMsg} : \text{LL!Replica(cohort)!Execute}(m)$

83
Defn  \( m \triangleq \text{choose } m \in \text{CommittedMsg} : \text{LL!Replica(cohort)!Execute}(m) \)

Step 1.1. of 1

\( (\text{LL!Replica(cohort)!CsState'}) = \text{CsTx}[\text{LL!Replica(cohort)!CsState}, m, \text{opv}] \)

Reasoning (1.1.): Defn \text{Execute action}

Reasoning (1.1.): Defn newMembershipMap in Defs \text{CsTx}

Default Case 2. of 2

Reasoning (2.): Since we’ve ruled out \text{Crash} in the assumption, no other action updates \text{CsState}. Thus induction hypothesis carries forward into primed state.

Reasoning: Proof by case analysis

Invariant: MembershipMapChangesByExtension

Hypotheses of: MembershipMapDomain

Introduce \( \text{cohort} \in \text{Cohorts} \)

Assume

\( (\neg \text{LL!Replica(cohort)!Crash}) \Rightarrow \text{Fen Extends} ( \)

\( \text{LL!Replica(cohort)!CsState'.membershipMap}', \)

\( \text{LL!Replica(cohort)!CsState'.membershipMap} \)

Assume

\( (\neg \text{LL!Replica(cohort)!Crash}') \)

Prove

Fen Extends

\( \text{LL!Replica(cohort)!CsState'.membershipMap}', \)

\( \text{LL!Replica(cohort)!CsState'.membershipMap} \)

Case 1. of 2

\( \exists m \in \text{CommittedMsg} : \text{LL!Replica(cohort)!Execute}(m) \)

Defn  \( m \triangleq \text{choose } m \in \text{CommittedMsg} : \text{LL!Replica(cohort)!Execute}(m) \)

Step 1.1. of 3

\( (\text{LL!Replica(cohort)!CsState'}) = \text{CsTx}[\text{LL!Replica(cohort)!CsState}, m, \text{opv}] \)

Reasoning (1.1.): Defn \text{Execute action}

Step 1.2. of 3

\text{Domain} \text{LL!Replica(cohort)!CsState'.membershipMap} \subseteq

\text{Domain} \text{LL!Replica(cohort)!CsState'.membershipMap}'

Step 1.2.1. of 2

\text{Domain} \text{LL!Replica(cohort)!CsState'.membershipMap} =

(1 \ldots (\text{LL!Replica(cohort)!CsState'.numExecuted + Alpha}))

Reasoning (1.2.1.): Ref hypothesis: MembershipMapDomain

Step 1.2.2. of 2

\text{Domain} \text{LL!Replica(cohort)!CsState'.membershipMap}' =

(1 \ldots (((\text{LL!Replica(cohort)!CsState'.numExecuted + Alpha} + 1))

Reasoning (1.2.2.): Defn \text{CsTx}

Reasoning (1.2.): Defn .

Step 1.3. of 3

84
Introduce \( x \in \text{DOMAIN } LL!\text{Replica}(\text{cohort})!CsState.\text{membershipMap} \)
\[(LL!\text{Replica}(\text{cohort})!CsState.\text{membershipMap}')[x] =
LL!\text{Replica}(\text{cohort})!CsState.\text{membershipMap}[x] \]

Reasoning (1.3.): Defn \text{CsTz}

DefaultCase 2. of 2

Reasoning (1): We have satisfied Defn \text{FenExtends}

Reasoning (2.): Since we’ve ruled out \text{Crash} in the assumption, no other action updates \text{CsState}. Thus the reflexive \text{FenExtends} is easily satisfied.

Reasoning: Proof by case analysis

---

Theorem \text{VolatileMembershipMap Extends PersistentMembershipMap}

Hypotheses of \text{MembershipMap Changes By Extension}

Introduce \( \text{cohort} \in \text{Cohorts} \)

Assume \( \text{FenExtends} \)

Assume \( LL!\text{Replica}(\text{cohort})!CsState.\text{membershipMap}, LL!\text{Replica}(\text{cohort})!CsState\text{Snapshot.membershipMap} \)

Prove \( \text{FenExtends} \)

\( LL!\text{Replica}(\text{cohort})!CsState.\text{membershipMap}, LL!\text{Replica}(\text{cohort})!CsState\text{Snapshot.membershipMap} \)

Reasoning: Since we’re not doing log truncation, this theorem is really boring: \text{CsState\text{Snapshot.membershipMap} never changes. When CsState does, Ref \hspace{1mm} hypothesis: MembershipMap Changes By Extension is sufficient to show the theorem. If we had truncation, the Persist action is the only interesting case, and it’s not very interesting: it makes both states equal, so FenExtends follows because it is a reflexive relation.

---

Theorem \text{Membership As Monotonic}

Introduce \( \text{opn} \in \text{Ops} \)

Introduce \( \text{membership} \in \text{Memberships} \)

Assume \( \text{MembershipAs}(\text{opn}, \text{membership}, LL!SentMessages) \)

Prove \( \text{MembershipAs}(\text{opn}, \text{membership}, LL!SentMessages') \)

Reasoning:

---

Invariant \text{Membership Changes Are Broadcast}

Hypotheses of \text{MembershipMap Changes By Extension}

Introduce \( \text{opn} \in \text{Ops} \)


\[
\begin{align*}
\text{Introduce} & \quad \text{cohort} \in \text{Cohorts} \\
\text{Introduce} & \quad \text{membership} \in \text{Memberships} \\
\text{Assume} & \quad \land \ \text{opn} \in \text{DOMAIN} \quad LL!\text{Replica(cohort)!CsState. membership Map} \\
& \quad \land \ \text{membership} = LL!\text{Replica(cohort)!CsState. membership Map}[\text{opn}] \\
\Rightarrow & \quad \text{MembershipAs}(\text{opn, membership, LL!SentMessages}) \\
\text{Assume} & \quad (\land \ \text{opn} \in \text{DOMAIN} \quad LL!\text{Replica(cohort)!CsState. membership Map} \\
& \quad \land \ \text{membership} = LL!\text{Replica(cohort)!CsState. membership Map}[\text{opn}])' \\
\text{Prove} & \quad \text{MembershipAs}(\text{opn, membership, LL!SentMessages})' \\
\text{Defn} & \quad \text{state} \triangleq LL!\text{Replica(cohort)!CsState} \\
\text{Defn} & \quad \text{snapshot} \triangleq LL!\text{Replica(cohort)!CsStateSnapshot} \\
\text{Case 1. of 3} & \quad \exists \text{msg} \in \text{CommittedMsg : LL!Replica(cohort)!Execute(msg)} \\
\text{Defn} & \quad \text{msg} \triangleright \text{choose} \ m \in \text{CommittedMsg : LL!Replica(cohort)!Execute(msg)} \\
\text{Case 1.1. of 2} & \quad \text{opn} = m.\text{opn} + \text{Alpha} \\
\text{Step 1.1.1. of 2} & \quad \text{state'. membership Map}[(m.\text{opn} + \text{Alpha})] = \text{membership} \\
\text{Step 1.1.1. of 1} & \quad \text{state'. membership Map}[\text{opn}] = \text{membership} \\
& \quad \text{Reasoning (1.1.1.): Antecedent} \\
& \quad \text{Reasoning (1.1.1.): substitution} \\
\text{Defn} & \quad \text{sentMessage} = \text{MakeMembershipMsg(cohort, opn, membership)} \\
\text{Step 1.1.2. of 2} & \quad \text{sentMessage} \in (\text{SentMessages}') \\
& \quad \text{Reasoning (1.1.2.): Defn Execute sends a message} \\
& \quad \text{Reasoning (1.1.): Defn MembershipAs} \\
\text{Default Case 1.2. of 2} & \quad \text{opn} \in \text{DOMAIN} \quad LL!\text{Replica(cohort)!CsState. membership Map} \\
& \quad \text{Reasoning (1.2.1.): Ref hypothesis: MembershipMap ChangesByExtension ; Defn}\ Fcn\ Extends \\
\text{Step 1.2.2. of 3} & \quad \text{membership} = \text{state. membership Map}[\text{opn}] \\
& \quad \text{Reasoning (1.2.2.): \text{if} else in CsTy leaves unchanged any \ opn \neq m.\text{opn} + \text{Alpha}} \\
\text{Step 1.2.3. of 3} & \quad \text{MembershipAs}(\text{opn, membership, LL!SentMessages}) \\
& \quad \text{Reasoning (1.2.3.): induction hypothesis} \\
& \quad \text{Reasoning (1.2.): Ref: Membership As Monotonic} \\
& \quad \text{Reasoning (1.1.): Proof by case analysis} \\
\text{Case 2. of 3} & \quad LL!\text{Replica(cohort)!Crash} \\
\end{align*}
\]
Step 2.1. of 4

\[ \land \text{opn} \in \text{DOMAIN} \quad \text{snapshot}.\text{membership Map}[\text{opn}] \]
\[ \land \text{cohort} \in \text{snapshot}.\text{membership Map}[\text{opn}] \]

Reasoning: [2.1.]: \text{Defn Crash} equates \text{CsState' = CsState Snapshot' }

Step 2.2. of 4

\[ \land \text{opn} \in \text{DOMAIN} \quad \text{snapshot.membership Map}[\text{opn}] \]
\[ \land \text{cohort} \in \text{snapshot.membership Map}[\text{opn}] \]

Reasoning: [2.2.]: \text{Defn Crash leaves unchanged CsState Snapshot'}

Step 2.3. of 4

\[ \land \text{opn} \in \text{DOMAIN} \quad \text{state. membership Map}[\text{opn}] \]
\[ \land \text{cohort} \in \text{state. membership Map}[\text{opn}] \]

Reasoning: [2.3.]: \text{Ref VolatileMembership Map Extends PersistentMembership Map : Defn Fcn Extends}

Step 2.4. of 4

\[ \text{Membership As(opn, LL!Replica(cohort)!Membership, LL!SentMessages) } \]

Reasoning: [2.4.]: induction hypothesis

Reasoning: [2.]: \text{Ref:Membership As Monotonic}

DefaultCase 3. of 3

Reasoning: [3.]: No other actions update state \{CsState\} (this proof ignores the Transfer action); so we use the induction hypothesis and \text{Ref:Membership As Monotonic}.

Reasoning: Proof by case analysis

Theorem \text{Max Known Opn Grows}

\[ \text{Max Known Opn } \leq (\text{Max Known Opn'}) \]

Reasoning: \text{Ref:Committed Monotonic}: Anything committed before will still be committed after any legal action.

Theorem \text{Memberships Are Unique}

Hypotheses of \text{BroadcastMemberships Reflect Known State}

Introduce \[ \text{opn} \in \text{Opns} \]
Introduce \[ \text{membership1} \in \text{Memberships} \]
Introduce \[ \text{membership2} \in \text{Memberships} \]
Assume \[ \text{Membership As(opn, membership1, LL!SentMessages)} \]
Assume \[ \text{Membership As(opn, membership2, LL!SentMessages)} \]
Prove \[ \text{membership1} = \text{membership2} \]

Case 1. of 2

\[ \text{Alpha < opn} \]

Step 1.1. of 2

\[ \text{KnownState[}(\text{opn} - \text{Alpha})].\text{membership Map}[\text{opn}] = \text{membership1} \]
Reasoning (1.1): \( \text{Ref \ hypothesis: Broadcast Memberships \ Reflect Known State} \)

Step 1.2. of 2

\( \text{Known State}[\text{opn} \in \text{Alpha}], \text{membership Map}[\text{opn}] = \text{membership2} \)

Reasoning (1.2): \( \text{Ref \ hypothesis: Broadcast Memberships \ Reflect Known State} \)

Reasoning (1.): Substitution

Case 2. of 2

\( \text{opn} \leq \text{Alpha} \)

Step 2.1. of 2

\( \text{MembershipAs}(\text{opn}, \text{membership1}, \text{LL!SentMessages}) = \text{MakeMembership(InitialHosts, 1)} \)

Reasoning (2.1): \( \text{Defn MembershipAs} \)

Step 2.2. of 2

\( \text{MembershipAs}(\text{opn}, \text{membership2}, \text{LL!SentMessages'}) = \text{MakeMembership(InitialHosts, 1)} \)

Reasoning (2.2): \( \text{Defn MembershipAs} \)

Reasoning: \text{Proof by case analysis}

---

**NB** Unlike most, this theorem states properties about the primed state.

**Theorem MembershipsDetermineMembership**

Hypotheses of **Memberships Are Unique**

Introduce \( \text{opn} \in \text{Opns} \)

Introduce \( \text{membership} \in \text{Memberships} \)

Assume \( \text{MembershipAs}(\text{opn}, \text{membership}, \text{LL!SentMessages'}) \)

Prove \( (\text{Membership}(\text{opn}') = \text{membership}) \)

Defn \( \text{choices} \equiv \{ m \in \text{Memberships} : \text{MembershipAs}(\text{opn}, m, \text{LL!SentMessages'}) \} \)

Step 1. of 2

\( \text{membership} \in \text{choices} \)

Reasoning (1): \( \text{Defn choices} \)

Step 2. of 2

\( \text{Cardinality}(\text{choices}) = 1 \)

Reasoning (2): \( \text{Defn Memberships Are Unique \ (to get primed statement)} \)

Reasoning: \text{choose in Defn Membership(\text{opn}') is fully constrained}

---

**Theorem CTSIncrementEpochs**

Introduce \( \text{state} \in \text{CsStates} \)

Introduce \( \text{opv} \in \text{CsOps} \)

Prove

\( \forall \text{CsT}_{\text{opv}}[\text{state, opv}, \text{membership Map}[(\text{state.numExecuted} + \text{Alpha}) + 1]] = \text{state.menbership Map}[(\text{state.numExecuted} + \text{Alpha})] \)
\( \forall \text{EpochOf}(\text{CsTx}[\text{state}, \text{opn}].\text{membershipMap}[(\text{state}.\text{numExecuted} + \text{Alpha}) + 1]) = \text{EpochOf}(\text{state}.\text{membershipMap}[(\text{state}.\text{numExecuted} + \text{Alpha})]) + 1 \)

Reasoning: By construction of \text{CsTx}

---

**Invariant** \text{NumExecutedTicks}

**Introduce** \(\text{opn} \in \text{DOMAIN KnownState}\)

**Assume** \(\text{KnownState}[^{\text{opn}}].\text{numExecuted} = \text{opn}\)

**Prove** \((\text{KnownState}[^{\text{opn}}].\text{numExecuted} = \text{opn})'\)

Reasoning: Really boring induction induction hypothesis; \text{CsTx} shows the inductive step.

---

**Invariant** \text{LocalMembership EpochOrdering}

**Introduce** \(\text{cohort} \in \text{Cohorts}\)

**Assume** \(\land \text{EpochsOrdered}(\text{LL!Replica}(\text{cohort})!\text{CsState}.\text{membershipMap})\)
\(\land \text{EpochsOrdered}(\text{LL!Replica}(\text{cohort})!\text{CsStateSnapshot}.\text{membershipMap})\)

**Prove** \((\land \text{EpochsOrdered}(\text{LL!Replica}(\text{cohort})!\text{CsState}.\text{membershipMap})\)
\(\land \text{EpochsOrdered}(\text{LL!Replica}(\text{cohort})!\text{CsStateSnapshot}.\text{membershipMap})')'\)

Case 1. of 3:
\(\text{LL!Replica}(\text{cohort})!\text{Crash}\)

Reasoning (1.): \text{CsState}' = \text{CsStateSnapshot}' = \text{CsStateSnapshot}; apply induction hypothesis.

Case 2. of 3:
\(\exists m \in \text{CommittedMsg} : \text{LL!Replica}(\text{cohort})!\text{Execute}(m)\)

Summary: Only \text{CsState} changes, and it changes by extension by a single spot; we can apply \text{CsTx} there to show that the invariant holds.

**Define** \(m \triangleq \text{CHOOSE } m \in \text{CommittedMsg} : \text{LL!Replica}(\text{cohort})!\text{Execute}(m)\)

**Define** \(\text{map} \triangleq \text{LL!Replica}(\text{cohort})!\text{CsState}.\text{membershipMap}\)

**Step 2.1. of 2**

**Introduce** \(\text{opn1} \in \text{DOMAIN map}\)

**Introduce** \(\text{opn2} \in \text{DOMAIN map}\)

\(\land \text{EpochOf}(\text{map}[\text{opn1}]) \leq \text{EpochOf}(\text{map}[\text{opn2}])\)

\(\land (\text{EpochOf}(\text{map}[\text{opn1}]) = \text{EpochOf}(\text{map}[\text{opn2}]) \Rightarrow \text{map}[\text{opn1}] = \text{map}[\text{opn2}])\)

Summary: If \text{opn2} (and hence \text{opn1}) concern slots before the one being executed presently, then the induction hypothesis takes care of the proof. Otherwise, we use \text{CsTx}.

Case 2.1.1. of 2
\(\text{opn2} = m . \text{opn}\)

**Step 2.1.1. of 2**

\(\lor (\text{map}')(\text{opn2} - 1) = (\text{map}')[\text{opn2}]\)
\[ \forall \text{EpochOf}((\text{map}')(\text{opn}2-1)) < \text{EpochOf}((\text{map}')[\text{opn}2]) \]

Step 2.1.1.1. of 1

\[(\text{LL!Replica(}\text{cohort})\text{!CsSnap}')(\text{CsState}) = \text{CsTx}[\text{LL!Replica(}\text{cohort})\text{!CsState}, m.\text{opn}]\]

Reasoning (2.1.1.1.): Defn Execute

Reasoning (2.1.1.2.): Consider Defn CsTz, paying attention to the \( \text{LET newMembership IN variable} \text{newMembership} \)

Step 2.1.2. of 2

\[\forall (\text{map}')(\text{opn}2-1) = (\text{map}')[\text{opn}1] \]

\[\forall \text{EpochOf}((\text{map}')[\text{opn}1]) < \text{EpochOf}((\text{map}')[\text{opn}2-1]) \]

Reasoning (2.1.2.): Ref: MembershipMapChangesByExtension; induction hypothesis

Reasoning (2.1.): algebra relates \text{opn}2 to \text{opn}1 via \text{opn}2 - 1

Default Case 2.1.2. of 2

Reasoning (2.2.): Defn Execute implies \text{UNCHANGED CsStateSnapshot}; induction hypothesis

Reasoning (2.): First step proves DefnEpochOrdered in first conjunct of proof goal; Second step proves second conjunct.

Default Case 3. of 3

Step 3.1. of 1

\[\wedge \text{UNCHANGED LL!Replica(}\text{cohort})\text{!CsState} \]

\[\wedge \text{UNCHANGED LL!Replica(}\text{cohort})\text{!CsStateSnapshot} \]

Reasoning (3.1.): No other actions change \text{CsState} and \text{CsStateSnapshot} (besides Persist, but this proof is ignoring persistence and log truncation, and anyway, Persist is easy like Crash.)

Reasoning (3.2.): apply induction hypothesis

Reasoning: Proof by case analysis.

Theorem MembershipEpochOrdering

Hypotheses of NumExecutedTicks

Introduce \( \text{opn}1 \in \text{Opns} \)

Introduce \( \text{opn}2 \in \text{Opns} \)

Assume \( \text{Alpha} \leq \text{opn}1 \)

Assume \( \text{opn}1 < \text{opn}2 \)

Assume \( \text{opn}2 \leq \text{MaxKnownOpn} + \text{Alpha} \)

Prove

\[\wedge \text{EpochOf}(\text{KnownMembership}(\text{opn}1)) \leq \text{EpochOf}(\text{KnownMembership}(\text{opn}2))\]

\[\wedge (\text{EpochOf}(\text{KnownMembership}(\text{opn}1)) = \text{EpochOf}(\text{KnownMembership}(\text{opn}2)) \Rightarrow \]

90
KnownMembership(opn1) = KnownMembership(opn2))

Step 1. of 2
Assume \( \text{opn2} = \text{opn1} + 1 \)
Proof
\[ \land \text{EpochOf}(\text{KnownMembership}(\text{opn1})) \leq \text{EpochOf}(\text{KnownMembership}(\text{opn2})) \]
\[ \land (\text{EpochOf}(\text{KnownMembership}(\text{opn1})) = \text{EpochOf}(\text{KnownMembership}(\text{opn2})) \Rightarrow \text{KnownMembership}(\text{opn1}) = \text{KnownMembership}(\text{opn2})) \]
Reasoning (1): Follows by algebra from \ref{CSThreeMembEpochs}

Step 2. of 2
Assume (2.1)
\[ \land \text{EpochOf}(\text{KnownMembership}(\text{opn1})) \leq \text{EpochOf}(\text{KnownMembership}(\text{opn2} + 1)) \]
\[ \land (\text{EpochOf}(\text{KnownMembership}(\text{opn1})) = \text{EpochOf}(\text{KnownMembership}(\text{opn2} + 1)) \Rightarrow \text{KnownMembership}(\text{opn1}) = \text{KnownMembership}(\text{opn2} + 1)) \]
Defn \(\text{state} \triangleq \text{KnownState}[(\text{opn2} - \text{Alpha})]\)
Step 2.1. of 5
\(\text{KnownMembership}(\text{opn2}) = \text{state}. \text{membershipMap}[(\text{state}. \text{numExecuted} + \text{Alpha})]\)
Reasoning (2.1): Defn KnownMembership; Defn state
Defn \(\text{opv} \triangleq \text{KnownOpv}[(\text{opn2} - \text{Alpha} + 1)]\)
Step 2.2. of 5
\(\text{KnownState}[(\text{opn2} - \text{Alpha} + 1)] = \text{CsTx}[\text{state}, \text{opv}]\)
Step 2.2.1. of 1
\(\text{KnownState}[(\text{opn2} - \text{Alpha} + 1)] = \text{CsTx}[(\text{KnownState}[(\text{opn2} - \text{Alpha} + 1)]), (\text{KnownOpv}[(\text{opn2} - \text{Alpha} + 1)])]\]
Reasoning (2.2.1): Defn KnownState
Reasoning (2.2.): algebra
Step 2.2. of 5
\(\text{KnownMembership}(\text{opn2} + 1) = \text{CsTx}[\text{state}, \text{opv}], \text{membershipMap}[(\text{state}. \text{numExecuted} + \text{Alpha}) + 1)]\]
Summary: Basically a boring bunch of algebra
Step 2.3.1. of 3
\(\text{KnownMembership}(\text{opn2} + 1) = \text{KnownState}[(\text{opn2} + 1 - \text{Alpha})], \text{membershipMap}[(\text{opn2} + 1)]\]
Reasoning (2.3.1.): Defn KnownMembership
Step 2.3.2. of 3
\(\text{KnownState}[(\text{opn2} + 1 - \text{Alpha}), \text{membershipMap}[(\text{opn2} + 1)] = \text{CsTx}[\text{state}, \text{opv}], \text{membershipMap}[(\text{opn2} + 1)]\]
Reasoning (2.3.2.): Defn KnownState; Defn state; Defn opv
Step 2.3.3. of 3
\(\text{CsTx}[\text{state}, \text{opv}], \text{membershipMap}[(\text{opn2} + 1)] = \text{CsTx}[\text{state}, \text{opv}], \text{membershipMap}[(\text{state}. \text{numExecuted} + \text{Alpha}) + 1]]\)
Step 2.3.3.1. of 1
\(\text{opn2} + 1 = (\text{state}. \text{numExecuted} + \text{Alpha}) + 1\)
Step 2.3.3.1. of 1

\[ state.\text{numExecuted} = \text{opn2} - \text{Alpha} \]

Reasoning [2.3.3.1.]: Defn state; Ref hypothesis: NumExecutedTicks

Reasoning [2.3.3.]: algebra

Reasoning (2.3.): transitivity

Case 2.4. of 5

\[ CsT\text{x}[state, opn].\text{membershipMap}[(state.\text{numExecuted} + \text{Alpha}) + 1] = state.\text{membershipMap}[(state.\text{numExecuted} + \text{Alpha})] \]

Step 2.4.1. of 1

\[ \text{KnownMembership}(\text{opn2} + 1) = \text{KnownMembership}(\text{opn2}) \]

Step 2.4.1.1. of 4

\[ \text{KnownMembership}(\text{opn2} + 1) = \]

\[ \text{KnownState}[(\text{opn2} - \text{Alpha}), \text{membershipMap}] \]

\[ (\text{KnownState}[(\text{opn2} - \text{Alpha})], \text{numExecuted} + \text{Alpha}) \]

Reasoning [2.4.1.1.]: Case condition, with substitutions from Ref: Step 2.3. and Defn state

Step 2.4.1.2. of 4

\[ \text{KnownState}[(\text{opn2} - \text{Alpha}), \text{membershipMap}] \]

\[ (\text{KnownState}[(\text{opn2} - \text{Alpha}), \text{numExecuted} + \text{Alpha})] \]

\[ = \]

\[ \text{KnownState}[(\text{opn2} - \text{Alpha}), \text{membershipMap}[(\text{opn2} - \text{Alpha} + \text{Alpha})] \]

Reasoning [2.4.1.2.]: Ref hypothesis: NumExecutedTicks

Step 2.4.1.3. of 4

\[ \text{KnownState}[(\text{opn2} - \text{Alpha}), \text{membershipMap}[(\text{opn2} - \text{Alpha} + \text{Alpha})]] = \]

\[ \text{KnownState}[(\text{opn2} - \text{Alpha}), \text{membershipMap}[(\text{opn2})] \]

Reasoning [2.4.1.3.]: algebra

Step 2.4.1.4. of 4

\[ \text{KnownState}[(\text{opn2} - \text{Alpha}), \text{membershipMap}[(\text{opn2})] = \text{KnownMembership}(\text{opn2}) \]

Reasoning [2.4.1.4.]: Defn KnownMembership

Reasoning (2.4.): We can substitute into Ref: Assumption 2.4.1. to produce the proof goal.

Case 2.5. of 5

\[ \text{EpochOf}(CsT\text{x}[state, opn].\text{membershipMap}[(state.\text{numExecuted} + \text{Alpha}) + 1]) = \]

\[ \text{EpochOf}(state.\text{membershipMap}[(state.\text{numExecuted} + \text{Alpha})]) + 1 \]

Step 2.5.1. of 2

\[ \text{EpochOf}(\text{KnownMembership}(\text{opn2} + 1)) = \text{EpochOf}(\text{KnownMembership}(\text{opn2})) + 1 \]

Reasoning [2.5.1.]: Ref:Step 2.1.; Ref:Step 2.3.

Step 2.5.2. of 2

\[ \text{EpochOf}(\text{KnownMembership}(\text{opn1})) < \text{EpochOf}(\text{KnownMembership}(\text{opn2} + 1)) \]
Reasoning (2.5.2.): Ref: Step 2.5.1.; inductive hypothesis
Reasoning (2.5.): The first conjunct of the goal is clearly satisfied by the previous step, and
the antecedent of the second conjunct of the goal is denied by the previous step.
Reasoning (2.): Case analysis; complete by Ref: CstCountsEpochs.
Reasoning: By induction over opn2.

Theorem NonconflictingViewMemberships
Hypotheses of MembershipAreUnique
Hypotheses of MembershipEpochOrdering
Hypotheses of BroadcastMembershipsReflectKnownState
Introduce $v \in \text{ViewIds}$
Introduce $\text{membership} \in \text{Memberships}$
Introduce $\text{opn} \in \text{Ops}$
Assume $v.\text{viewInitiator}.\text{epoch} = \text{EpochOf(\text{membership})}$
Assume $\text{MembershipAs}(\text{opn}, \text{membership}, \text{LL:SentMessages})$
Prove $\text{ViewMembership}(\text{view}) = \text{membership}$
Defn $\text{satisfyingMemberships} \triangleq$
\begin{align*}
\{ & \text{potentialMembership} \in \text{Memberships} : \\
& ( \land (\exists \text{opn2} \in \text{Ops} : \text{MembershipAs}(\text{opn2}, \text{potentialMembership}, \text{LL:SentMessages})) \\
& \land \text{EpochOf(\text{potentialMembership})} = v.\text{viewInitiator}.\text{epoch} ) \\
\} \\
\end{align*}

Step 1 of 2
memoryship $\in$ satisfyingMemberships
Reasoning (1.): Follows from assumptions and Defn Membership
Step 2 of 2
Introduce $m2 \in$ satisfyingMemberships
Assume $m2 \neq \text{membership}$
Prove FALSE
Defn $\text{opn2} \triangleq \text{CHOOSE opn2} \in \text{Ops} : \text{MembershipAs}(\text{opn2}, m2, \text{LL:SentMessages})$

Step 2.1 of 6
$\text{MembershipAs}(\text{opn2}, m2, \text{LL:SentMessages})$
Reasoning (2.1.): Defn satisfyingMemberships
Step 2.2 of 6
$\text{opn} \neq \text{opn2}$
Reasoning (2.2.): Ref: MembershipAreUnique
Step 2.3 of 6
$\text{KnownState}((\text{opn2} - \text{Alpha}), \text{membershipMap[\text{opn2}]} = \text{membership}$
Reasoning (2.3): Ref hypothesis: BroadcastMembershipsReflectKnownState
Step 2.4 of 6
$\text{KnownMembership(\text{opn2})} = \text{membership}$
Reasoning (2.4.): Defn KnownMembership
Step 2.5 of 6
\[\text{EpochOf}(m2) \neq \text{EpochOf}(\text{membership})\]

Reasoning (2.5): Algebra on Ref:MembershipEpochOrdering

Step 26 of 6

\[\text{EpochOf}(m2) \neq \text{viewViewInitiator.epoch}\]

Reasoning (2.6): That distinction is already claimed by membership

Reasoning (2): We have arrived at a contradiction.

Reasoning: choose in ViewMembership is fully constrained

---

**Theorem** NonconflictingViewMembershipsPrimed

**Hypotheses of** MembershipsAreUnique

**Hypotheses of** MembershipEpochOrdering

**Hypotheses of** BroadcastMembershipsReflectKnownState

**Introduce** view \(\in\) ViewIds

**Introduce** membership \(\in\) Memberships

**Introduce** \(opn\in\) Opns

**Assume** viewViewInitiator.epoch = EpochOf(membership)

**Assume** MembershipAs(opn, membership, LL!SentMessages)

**Prove** (ViewMembership(view)') = membership

Reasoning: Track each hypothesis back to the underlying invariants, use the invariants to push the statements into the primed state, apply Ref:NonconflictingViewMemberships to get the conclusion.

---

**Invariant** PrimaryDesignatedImpliesElectionQuorum

**Introduce** view \(\in\) ViewIds

**Introduce** primary \(\in\) CoHorts

**Assume** PrimaryDesignatedAs(view, primary) \(\Rightarrow\)

(\(\exists\) quorum \(\in\) QuoraOfMembership(ViewMembership(view)):

PlausibleElectionQuorum(view, quorum))

**Assume** PrimaryDesignatedAs(view, primary)

**Prove** (\(\exists\) quorum \(\in\) QuoraOfMembership(ViewMembership(view)):

PlausibleElectionQuorum(view, quorum))'

Summary: The interesting action is DesignatePrimary; all other actions fall out by monotonicity.
Case 1. of 2

\[ \exists \text{config} \in \text{Designation Configurations} : \]
\[ \land \text{config}. \text{view} = \text{view} \]
\[ \land \text{LL}.\text{Replica} (\text{config}. \text{designator})! \text{DesignatePrimary}(\text{config}) \]

Summary: If a primary was designated in this step, then we identify the VcAcks used to make that decision. The cohorts that sent these VcAcks (the electing quorum) must have formed a PlausibleElectionQuorum, so we prove the conjuncts of that definition.

\text{Defn} \quad \text{config} \overset{\Delta}{=} \quad \text{CHOOSE config} \in \text{Designation Configurations} :
\[ \land \text{config}. \text{view} = \text{view} \]
\[ \land \text{LL}.\text{Replica} (\text{config}. \text{designator})! \text{DesignatePrimary}(\text{config}) \]

Step 1.1. of 7
\[ \land \text{config}. \text{view} = \text{view} \]
\[ \land \text{LL}.\text{Replica} (\text{config}. \text{designator})! \text{DesignatePrimary}(\text{config}) \]

Reasoning (1.1.): choose Axiom

Step 1.2. of 7
\[ \text{LL}.\text{Replica} (\text{config}. \text{designator})! \text{CurView} = \text{view} \]

Summary: The configuration was chosen specifically to enforce this equality

\text{Defn} \quad \text{witnessMsg} \overset{\Delta}{=} \quad \text{CHOOSE} m \in \text{config}. \text{msgs} : \text{TRUE}

Step 1.2.1. of 3
\[ \text{witnessMsg} \in \text{config}. \text{msgs} \]

Reasoning (1.2.1.): Defn DesignationConfigurations explicitly disallows empty \text{msgs} fields.

Step 1.2.2. of 3
\[ \text{witnessMsg}. \text{view} = \text{LL}.\text{Replica} (\text{config}. \text{designator})! \text{CurView} \]

Reasoning (1.2.2.): universal quantifier in Defn DesignatePrimaryAction

Step 1.2.3. of 3
\[ \text{witnessMsg}. \text{view} = \text{view} \]

Reasoning (1.2.3.): Defn DesignationConfigurations; Ref: Step 1.1.

Reasoning (1.2.): substitution

Step 1.3. of 7
\[ \text{config}. \text{designator} = \text{view}. \text{viewInitiator} \]

Step 1.3.1. of 2
\[ \text{config}. \text{designator} = \text{LL}.\text{Replica} (\text{config}. \text{designator})! \text{ThisCohort} \]

Reasoning (1.3.1.): Defn DesignatePrimary

Step 1.3.2. of 2
\[ \text{LL}.\text{Replica} (\text{config}. \text{designator})! \text{ThisCohort} = \text{LL}.\text{Replica} (\text{config}. \text{designator})! \text{CurView. viewInitiator} \]

Reasoning (1.3.2.): Defn DesignatePrimaryAction

Reasoning (1.3.): Substitution, including Ref: Step 1.2.

Step 1.4. of 7
\[ \forall \text{cohort} \in \text{config. quorum} : (\text{VcAkedView}(\text{view}, \text{cohort})) \]

Step 1.4.1. of 1
Introduce \[ \text{cohort} \in \text{config. quorum} \]
Prove \( VcAckedView(\text{view}, \text{cohort}) \)

Summary: \( \text{configmsgs} \) provides the collection of \( VcAck \) messages.

Defn \( vcackMsg \xrightarrow{\Delta} \text{choose } vcackMsg \in \text{configmsgs} : vcackMsg.\text{sender} = \text{cohort} \)

Step 1.4.1.1. of 3
\( vcackMsg.\text{sender} = \text{cohort} \)

Reasoning (1.4.1.1.): 
\( \text{Defn DesignatePrimaryAction; } \text{Defn EachCohortSentLA Message; choose axiom} \)

Step 1.4.1.2. of 3
\( VcAcked(\text{view}, \text{cohort}, vcackMsg.\text{preparedOps}) \)

Step 1.4.1.2.1. of 2
\( vcackMsg \in \text{SentMessages} \)

Reasoning (1.4.1.2.1.): 
\( \text{Defn ReceiveMessageSet(configmsgs)} \)

Step 1.4.1.2.2. of 2
\( vcackMsg.\text{view} = \text{view} \)

Reasoning (1.4.1.2.2.): 
\( \text{Defn DesignationConfigurations} \)

Reasoning (1.4.1.2.2.): 
\( \text{Defn vcackMsg; Defn VcAcked} \)

Step 1.4.1.3. of 3
\( VcAckedView(\text{view}, \text{cohort}) \)

Reasoning (1.4.1.3.): 
\( vcackMsg.\text{preparedOps} \) is witness to the existential in \( \text{Defn VcAckedView} \)

Reasoning (1.4.1.): 
\( \text{Ref: VcAckedMonotonic} \)

Reasoning (1.4.): 
\( \text{expand universal quantifier} \)

Step 1.5. of 7
\( \text{DesignationReflectsVcAcks(\text{view}, \text{config.quorum})} \)

Summary: 
This step follows by the construction of the message sent in \( \text{DesignatePrimaryAction} \).

Step 1.5.1. of 5
\( \forall vcackMsg \in \text{configmsgs} : vcackMsg.\text{sender} \in \text{config.quorum} \)

Reasoning (1.5.1.): 
\( \text{Defn EachCohortSentLA Message} \)

Step 1.5.2. of 5
\( \forall vcackMsg \in \text{configmsgs} : vcackMsg.\text{view} = \text{view} \)

Reasoning (1.5.2.): 
\( \text{Defn DesignatePrimaryAction; } \text{Defn DesignationConfigurations} \)

Defn \( \text{designationMsg} \xrightarrow{\Delta} \text{MakePrimaryDesignatedMsg(} \)
\( \text{view, view.Initiator, primary, MaxTruncationPoint(configmsgs), AggregatePreparedOps(configmsgs)} \)

Step 1.5.3. of 5
\( \text{designationMsg.}\text{view} = \text{view} \)

Reasoning (1.5.3.): 
\( \text{Defn designationMsg; Defn MakePrimaryDesignatedMsg} \)

Step 1.5.4. of 5
\[\text{designation}\text{Msg.}\text{prevPrepares} = \text{AggregatePreparedOps}\left(\text{config.}\text{msgs}\right)\]

Reasoning (1.5.4.): \text{Defn}\ designation\text{Msg}; \text{Defn}\ MakePrimaryDesignated\text{Msg}

Step 1.5.5. of 5

\[\text{DesignationReflectsVcAcks}\left(\text{view, config.}\text{quorum}\right)\]

Reasoning (1.5.5.): With existential witness \text{designation}\text{Msg} \text{is designation}\text{Msg} and \text{vcAckMsgSet} \text{is config.}\text{msgs}, we have satisfied each conjunct of \text{DesignationReflectsVcAcks}\left(\text{view, config.}\text{quorum}\right).

Reasoning (1.5.): \text{Ref: DesignationReflectsVcAcks Monotonic}

Step 1.6. of 7

\[\text{PlausibleElectionQuorum}\left(\text{view, config.}\text{quorum}\right)\]

Reasoning (1.6.): Prior two steps satisfy \text{Defn}\ PlausibleElectionQuorum

Step 1.7. of 7

\[\text{config.}\text{quorum} \in \text{QuororumMembership}\left(\text{ViewMembership}\left(\text{view}'\right)\right)\]

\[\text{Defn}\ \text{membershipOpn} \triangleq \]

\[\text{CHOOSE}\ \text{membershipOpn} \in \text{Opns} : \]

\[\text{view.}\text{viewInitiator} \in \]

\[\text{LL.}\text{Replica}\left(\text{view.}\text{viewInitiator}\right)!CsState'.\text{membershipMap}[\text{membershipOpn}]\]

\[\text{Defn}\ \text{membership} \triangleq \]

\[\text{LL.}\text{Replica}\left(\text{view.}\text{viewInitiator}\right)!CsState'.\text{membershipMap}[\text{membershipOpn}]\]

Step 1.7.1. of 1

\[\left(\text{ViewMembership}\left(\text{view}'\right)\right) = \text{membership}\]

Summary: Sketch: Active\text{Member} \Rightarrow \text{LL.}\text{Membership} is defined. \Rightarrow it's been recorded in the message history \Rightarrow \text{LL.}\text{Membership} is in the set of globally-known memberships [recorded in message history] (an invariant; not sure how many steps) \Rightarrow it's the only one [a different invariant, coincident with nonconflicting-commits] \Rightarrow it's the one chosen by \text{Defn}\ View\text{Membership}

Step 1.7.1.1. of 3

\[\text{view.}\text{viewInitiator} \in \text{membership}\]

Reasoning (1.7.1.1.): \text{Defn}\ Active\text{Member'}; \text{choose}\ \text{axiom}

Step 1.7.1.2. of 3

\[\text{MembershipAs}\left(\text{membershipOpn}, \text{membership}, \text{LL.}\text{SentMessages}'\right)\]

Reasoning (1.7.1.2.): \text{Ref}\ \text{hypothesis: MembershipChangesAreBroadcast} \left(\text{opn} = \text{membershipOpn}, \text{cohort} = \text{view.}\text{viewInitiator}, \text{membership} = \text{membership}\right)

Step 1.7.1.3. of 3

\[\text{view.}\text{viewInitiator.}\text{epoch} = \text{EpochOf}\left(\text{membership}\right)\]

Step 1.7.1.3.1. of 2

\[\left(\text{Membership}\left(\text{membershipOpn}'\right)\right) = \text{membership}\]

Reasoning (1.7.1.3.1.): \text{Ref: MembershipAsDetermineMembership}

Step 1.7.1.3.2. of 2

\[\text{membership} \in \text{Memberships}\]

Reasoning (1.7.1.3.2.): \text{Ref: CsStateTypeInvariant}

Reasoning (1.7.1.3.3.): \text{Defn}\ EpochOf; \text{Defn}\ Memberships

Reasoning (1.7.1.): We have satisfied the antecedent of Ref: NonconflictingView\text{MembershipsPruned} \left(\text{view} = \text{view}, \text{membership} = \text{membership}, \text{opn} = \text{membershipOpn}\right)
Reasoning (1.7.): 
\[
\text{Defn } \text{DesignatePrimaryAction provides } \text{config.quorum} = \\
\text{LL!Replica(view.viewInitiator)!Membership; expand}
\]

Defn Quora

Reasoning (1.): We have exhibited a witness \text{config.quorum} to existential variable \text{quorum}

Default Case 2. of 2

Summary: When no "interesting" action has occurred, the relevant predicates are monotonic.

Step 21. of 3
\[
\text{PrimaryDesignatedAs(view, primary)}
\]

Reasoning [2.1.]: No other actions send \text{PrimaryDesignated} messages for this view, so \text{PrimaryDesignatedAs(\_)} cannot have changed.

Step 22. of 3
\[
\exists \text{quorum } \in \text{QuoraOfMembership(ViewMembership(view))} : \\
\text{PlausibleElectionQuorum(view, quorum)}
\]

Reasoning [2.2.]: induction hypothesis

Defn \[
\text{quorum } \triangleq \\
\text{choose quorum } \in \text{QuoraOfMembership(ViewMembership(view))} : \\
\text{PlausibleElectionQuorum(view, quorum)}
\]

Step 23. of 3
\[
\text{PlausibleElectionQuorum(view, quorum)'}
\]

Reasoning [2.3.]: choose: axiom; \text{Ref:PlausibleElectionQuorumMonotonic}

Reasoning: Case analysis

---

**Invariant: IAmPrimaryImpliesE electing Quorum**

Hypotheses of \text{PrimaryDesignatedImpliesE lectingQuorum}

Introduce \text{view } \in \text{Viewlds}

Introduce \text{primary } \in \text{Cohorts}

Assume
\[
\land \text{LL!Replica(primary)!Cur View = view} \\
\land \text{LL!Replica(primary)!IAmPrimary}
\]

\[
\Rightarrow \\
(\exists \text{quorum } \in \text{QuoraOfMembership(ViewMembership(view))} : \\
\text{PlausibleElectionQuorum(view, quorum)})
\]

Assume
\[
(\land \text{LL!Replica(primary)!Cur View = view} \\
\land \text{LL!Replica(primary)!IAmPrimary})'
\]

Prove
\[
(\exists \text{quorum } \in \text{QuoraOfMembership(ViewMembership(view))} : \\
\text{PlausibleElectionQuorum(view, quorum)})'
\]

Case 1. of 2
\[\exists m \in \text{PrimaryDesignatedMsg} : \text{llReplica(primary)}!\text{BecomePrimary}(m)\]

\[\text{PrimaryDesignatedAs}(\text{view}, \text{primary})\]

Reasoning (1.1): Defn. \text{BecomePrimary} shows that \( m \) is a witness to Defn. \text{PrimaryDesignatedAs}.

Reasoning (1.1): Ref. hypothesis: \text{PrimaryDesignatedImpliesElectionQuorum}

DefaultCase 2. of 2

Step 2.1. of 3

\[\land \text{llReplica(primary)}!\text{CurView} = \text{view}\]

\[\land \text{llReplica(primary)}!\text{IAmPrimary}\]

Reasoning (2.1): No actions on cohort other than \text{BecomePrimary} make \text{IAmPrimary} transition to true. (VeAck changes \text{CurView}, but it also assigns \text{IAmPrimary} = false, which we have assumed isn’t the case.)

Defn. \text{quorum} \triangleq \text{choose quorum} \in \text{QuoraOfMembership}(\text{ViewMembership}(\text{view})) : \text{PlausibleElectionQuorum}(\text{view}, \text{quorum})

Step 2.2. of 3

\text{PlausibleElectionQuorum}(\text{view}, \text{quorum})

Reasoning (2.2): induction hypothesis

Step 2.3. of 3

\text{PlausibleElectionQuorum}(\text{view}, \text{quorum})'

Reasoning (2.3): Ref. \text{PlausibleElectionQuorumMonotonic}

Reasoning (2.): We have exhibited a witness variable \text{quorum}

Reasoning: Case analysis

\[\text{Invariant} \text{ ProposedImpliesElectionQuorum} \]

which also incorporates the hypotheses of \text{ProposedImpliesElectionQuorum}

Hypotheses of \text{IAmPrimaryImpliesElectionQuorum}

Hypotheses of \text{PrimaryDesignatedImpliesElectionQuorum}

Introduce \text{view} \in \text{ViewIds}

Introduce \text{opn} \in \text{Opsns}

Assume \text{ProposedByAny}(\text{view}, \text{opn}) \Rightarrow \text{Proposed}(\text{view}, \text{primary}, \text{opn})'

Assume \text{ProposedByAny}(\text{view}, \text{opn})'

Prove \text{ProposedByAny}(\text{view}, \text{opn})\text{'}

\text{Defn.} \text{primary} \triangleq \text{choose primary} \in \text{Cohorts} : \text{Proposed}(\text{view}, \text{primary}, \text{opn})\text{'}

Step 1. of 4

\text{Proposed}(\text{view}, \text{primary}, \text{opn})\text{'}
Reasoning (1.): Defn ProposedByAny; choose axiom
Case 2. of 4
\[ \exists \text{opv} \in \text{Ops} : \text{LL!Replica(primary)!ProposeAction(view, opn, opv)} \]
Step 2.1. of 1
\[ \wedge \text{LL!Replica(primary)!CurView} = \text{view} \]
\[ \wedge \text{LL!Replica(primary)!IAMPrimary} \]
Reasoning (2.1.): Defn ProposeAction
Reasoning (2.): Ref hypothesis: IAMPrimaryImpliesElectionQuorum
Case 3. of 4
\[ \exists m \in \text{PrimaryDesignatedMsg} : \text{LL!Replica(primary)!BecomePrimary(m)} \]
Step 3.1. of 1
\[ \text{PrimaryDesignatedAs}(\text{view, primary}) \]
Reasoning (3.1.): Defn BecomePrimary shows that \( m \) is a witness to Defn PrimaryDesignatedAs.
Reasoning (3.): Ref hypothesis: PrimaryDesignatedImpliesElectionQuorum
DefaultCase 4. of 4
Step 4.1. of 3
\[ \text{Proposed(view, primary, opn)} \]
Reasoning (4.1.): No other actions change proposals
Defn quorum \( \triangleq \)
\[ \text{chose quorum} \in \text{QuorumOfMembership( ViewMembership(view))} : \]
\[ \text{PlausibleElectionQuorum(view, quorum)} \]
Step 4.2. of 3
\[ \text{PlausibleElectionQuorum(view, quorum)} \]
Reasoning (4.2.): Ref hypothesis: ProposedImpliesElectionQuorum
Step 4.3. of 3
\[ \text{PlausibleElectionQuorum(view, quorum)} \]
Reasoning (4.3.): Ref: PlausibleElectionQuorumMonotonic
Reasoning (4.): We have exhibited a witness variable quorum
Reasoning: Case analysis

Invariant CsStateTypeInvariant
Introduce cohort \( \in \text{Cohorts} \)
Assume
\[ \wedge \text{LL!Replica(cohort)!CsState} \in \text{CsStates} \]
\[ \wedge \text{LL!Replica(cohort)!CsStateSnapshot} \in \text{CsStates} \]
Prove
\[ ( \wedge \text{LL!Replica(cohort)!CsState} \in \text{CsStates} \]
\[ \wedge \text{LL!Replica(cohort)!CsStateSnapshot} \in \text{CsStates} ) \]
Reasoning:
Theorem OpInMembershipMapImpliesMembershipDefined

Hypotheses of
- MembershipChangesAreBroadcast
- CsStateTypeInvariant

Introduction
- \( opn \in \text{Ops} \)
- \( cohort \in \text{Cohorts} \)

Assume
- \( opn \in \text{DOMAIN} \ LL!\text{Replica}(cohort)\)\(\times\)CsState.membershipMap

Prove
- MembershipDefined\(\text{(opn)}\)

Define
- \( \text{membership} \triangleq LL!\text{Replica}(cohort)\)\(\times\)CsState.membershipMap\[opn]\)

Step 1 of 2
- Membership As \( (opn, \text{membership}, LL!\text{Sent Messages}) \)
  - Reasoning (1): Ref hypothesis: MembershipChangesAreBroadcast

Step 2 of 2
- \( \text{membership} \in \text{Memberships} \)
  - Reasoning (2): Ref hypothesis: CsStateTypeInvariant

Reasoning: membership is witness to Defn: MembershipDefined

Invariant ProposedImpliesMembershipDefined

Hypotheses of
- OpInMembershipMap Implies MembershipDefined

Introduction
- \( \text{view} \in \text{Viewlds} \)
- \( cohort \in \text{Cohorts} \)
- \( opn \in \text{Ops} \)

Assume
- \( \text{Proposed} (\text{view, cohort, opn}) \Rightarrow \text{MembershipDefined} (\text{opn}) \)

Assume
- \( \text{Proposed} (\text{view, cohort, opn}) \)

Prove
- MembershipDefined\(\text{(opn)}\)

Step 1 of 1
- MembershipDefined\(\text{(opn)}\)

Case 1.1. of 2
- \( \exists \text{opv} \in \text{CsOps} : LL!\text{Replica}(cohort)!\text{ProposeAction}(\text{view, opn, opv}) \)
  - Step 1.1.1. of 1
    - \( opn \in \text{DOMAIN} \ LL!\text{Replica}(cohort)\)\(\times\)CsState.membershipMap
      - Reasoning (1,1,): Def ActMember
      - Reasoning (1,1,): Ref: OpInMembershipMap Implies MembershipDefined

Default Case 1.2. of 2
- Step 1.2.1. of 1
  - Proposed\(\text{(view, cohort, opn)}\)
    - Reasoning (1,2,): No other action changes Proposed
    - Reasoning (1,2,): Induction hypothesis
    - Reasoning (1,): Case analysis
    - Reasoning: Ref: MembershipDefinedMonotonic

101
Invariant: LastProposedReflectsPrevPreps

Hypotheses of UniquePrimaryDesignationMessage
Hypotheses of PrimaryDesignatedPrecludesDesignationNeeded
Hypotheses of IAmPrimaryImpliesPrimaryDesignated

Introduce view ∈ Viewlds
Introduce primary ∈ Cohorts
Introduce opn ∈ Opns
Introduce opv ∈ CsOps

Assume
\( \land \text{PrimaryDesignatedPrevPrep}(\text{view}, \text{opn}, \text{opv}) \)
\( \land \text{LL!Replica(primary)!Cur View = view} \)
\( \land \text{LL!Replica(primary)!IAmPrimary} \)
\( \Rightarrow \)
\( \text{opn} \leq \text{LL!Replica(primary)!LastProposed} \)

Assume
\( (\land \text{PrimaryDesignatedPrevPrep}(\text{view}, \text{opn}, \text{opv})) \)
\( \land \text{LL!Replica(primary)!Cur View = view} \)
\( \land \text{LL!Replica(primary)!IAmPrimary} \)

Proof: \( (\text{opn} \leq \text{LL!Replica(primary)!LastProposed}) \)

Case 1. of 5
\( \exists m ∈ \text{PrimaryDesignatedMsg} : \text{LL!Replica(primary)!BecomePrimary}(m) \)

Summary: Action assigns LastProposed suitably.

Define \( m_1 \) as
\[ \text{CHOOSE } m_1 \in \text{SentMessages} \cap \text{PrimaryDesignatedMsg} : \]
\( \land m_1.\text{view} = \text{view} \)
\( \land \text{opn} \in \text{DOM} \text{AIN } m_1.\text{prevPrepares} \)
\( \land m_1.\text{prevPrepares}[\text{opn}] = \text{opv} \)

Define \( m_2 \) as
\[ \text{CHOOSE } m_2 ∈ \text{PrimaryDesignatedMsg} : \text{LL!Replica(primary)!BecomePrimary}(m_2) \]

Step 1.1. of 2
\( m_1 = m_2 \)

Reasoning [1.1.]: Antecedent makes first choose succeed; Case condition makes second choose succeed; Ref hypothesis: UniquePrimaryDesignationMessage

Step 1.2. of 2
\( \text{opn} \leq \text{MaxPreparedOpn}(m_1) \)

Reasoning [1.2.]: Defn MaxPreparedOpn

Reasoning [1.]: substitution

Case 2. of 5
\( \exists \text{opn}_2 ∈ \text{Opns}, \text{opn}_2 ∈ \text{CsOps} : \)
\( \text{LL!Replica(primary)!ProposeAction}(\text{view}, \text{opn}_2, \text{opn}_2) \)

Summary: Induction hypothesis holds in unprimed state, and action increments LastProposed, so things only get better.
Step 2.1. of 3
\( \land \ PrimaryDesignatedPrevPrep(view, \ opn, \ opv) \)
\( \land \ LL.Replica(primary)!CurView = view \)
\( \land \ LL.Replica(primary)!IAmPrimary \)

Reasoning (2.1.): Propose doesn’t send a PrimaryDesignatedMsg, and leaves CurView and IAmPrimary unchanged

Step 2.2. of 3
\( \ opn \leq LL.Replica(primary)!LastProposed \)
Reasoning (2.2.): induction hypothesis

Step 2.3. of 3
\( LL.Replica(primary)!LastProposed < (LL.Replica(primary)!LastProposed') \)
Reasoning (2.3.): Defn Propose
Reasoning (2.): transitivity

Case 3. of 5
\( \exists \ config \in Designation Configurations : \)
\( \land \ config.view = view \)
\( \land \ LL.Replica(config.designator)!DesignatePrimary(config) \)

Summary: If the cohort is already operating as a primary, we won’t see another DesignatePrimary action on this view.

Step 3.1. of 3
\( \land \ LL.Replica(view.viewInitiator)!DesignationNeeded \)
\( \land \ LL.Replica(view.viewInitiator)!CurView = view \)
Reasoning (3.1.): Defn DesignatePrimary: substitution

Step 3.2. of 3
\( \neg PrimaryDesignated(view) \)
Reasoning (3.2.): Ref hypothesis: PrimaryDesignatedPrecedesDesignationNeeded ; algebra

Step 3.3. of 3
PrimaryDesignatedAs(view, primary)
Reasoning (3.3.): Defn hypothesis: IAmPrimaryImpliesPrimaryDesignated

Reasoning (3.): Case eliminated by contradiction

Case 4. of 5
LL.Replica(primary)!Crash

Summary: This action cannot have happened if IAmPrimary is true.

Step 4.1. of 1
\( \neg (LL.Replica(primary)!IAmPrimary) \)
Reasoning (4.1.): Defn Crash action

Reasoning (4.): Case eliminated by contradiction

DefaultCase 5. of 5

Step 5.1. of 5
PrimaryDesignatedPrevPrep(view, opn, opv)
Reasoning (5.1.): No action other than DesignatePrimary sends a PrimaryDesignatedMsg

Step 5.2. of 5
LL.Replica(primary)!CurView = view
Reasoning (5.2.) Only \textit{VcAck} action updates \textit{CurView}, and it requires \textit{\textnormal{NAAmPrimary}}, so it cannot have happened.

Step 5.3. of 5
\[ \text{LLReplica( primary)!IAmPrimary} \]

Reasoning (5.3.) No action other than \textit{BecomePrimary} changes \textit{IAmPrimary} to \textit{true}.

Step 5.4. of 5
\[ \text{opn} \leq \text{LLReplica( primary)!LastProposed} \]
Reasoning (5.4.): induction hypothesis

Step 5.5. of 5
\[ \text{opn} \leq (\text{LLReplica( primary)!LastProposed}') \]

Reasoning (5.5.): No action other than \textit{Propose}, \textit{Crash}, and \textit{BecomePrimary} changes \textit{LastProposed}.

Reasoning (5.6.): Done.

Reasoning: case analysis

\begin{align*}
\text{Invariant: ProposalsRespectPresPreps} \quad & \\
\text{Hypotheses of } & \text{ProposedImpliesPrimaryDesignated} \\
\text{Hypotheses of } & \text{PrimaryDesignatedPrecludesDesignationNeeded} \\
\text{Hypotheses of } & \text{IAmPrimaryImpliesPrimaryDesignated} \\
\text{Hypotheses of } & \text{UniquePrimaryDesignated} \\
\text{Hypotheses of } & \text{LastProposedReflectsPresPreps} \\
\text{Introduce } & \text{view} \in \text{Viewlds} \\
\text{Introduce } & \text{opn} \in \text{Opns} \\
\text{Introduce } & \text{opv1} \in \text{CsOps} \\
\text{Introduce } & \text{opv2} \in \text{CsOps} \\
\text{Assume } & \text{PrimaryDesignatedPrevPrep( view, opn, opv1) } \\
& \text{ProposedByAnyAs( view, opn, opv2) } \\
& \Rightarrow \\
& \text{opv1} = \text{opv2} \\
\text{Assume } & (\text{PrimaryDesignatedPrevPrep( view, opn, opv1) } \\
& \text{ProposedByAnyAs( view, opn, opv2)') } \\
\text{Prove } & (\text{opv1} = \text{opv2}') \\
\end{align*}

Summary: Three actions are interesting: We show designation cannot occur (again) if a proposal has already been made. A Propose action cannot occur, because \textit{LastProposed} will prevent it. A \textit{BecomePrimary} action will respect \textit{PresPreps}.

\textbf{Def}: \quad \textit{primary} \equiv \text{CHOOSE primary} \in \text{Cohorts} : \text{PrimaryDesignatedAs( view, primary) }

Step 1. of 5
\[ \text{PrimaryDesignatedAs( view, primary) } \]

Reasoning (1.): Ref \textbf{hypothesis: ProposedImpliesPrimaryDesignated} and some expansion of quantifiers
Case 2. of 5
\[ \exists \text{config} \in \text{Designation Configurations} : \]
\[ \text{LL!Replica}(\text{config.designator})!\text{DesignatePrimary}(\text{config}) \]

Summary: Since there has already been a proposal in the view, this action cannot be enabled.

Defn \[ \text{config} \triangleq \]
\[ \text{LL!Replica}(\text{config.designator})!\text{DesignatePrimary}(\text{config}) \]

Step 2.1. of 1

\[-\text{LL!Replica}(\text{config.designator})!\text{Designation Needed} \]

Reasoning (2.1.): A bunch of substitutions on Defn \text{DesignatePrimary}; then apply Ref hypothesis:PrimaryDesignatedPrecludesDesignationNeeded

Reasoning (2.): Case eliminated by contradiction with Defns \text{DesignatePrimary}

Case 3. of 5
\[ \exists \text{cohort} \in \text{Cohorts} : \text{LL!Replica}(\text{cohort})!\text{ProposeAction}(\text{view}, \text{opn}, \text{opn}2) \]

Summary: A Propose action cannot occur, because \text{LastProposed} will prevent it.

Defn \[ \text{cohort} \triangleq \]
\[ \text{LL!Replica}(\text{primary})!\text{ProposeAction}(\text{view}, \text{opn}, \text{opn}2) \]

Step 3.1. of 2

\[ \text{LL!Replica}(\text{primary})!\text{ProposeAction}(\text{view}, \text{opn}, \text{opn}2) \]

Step 3.1.1. of 3

\[ \text{LL!Replica}(\text{cohort})!\text{IAmPrimary} \]

Reasoning (3.1.1.): Defn \text{Propose}

Step 3.1.2. of 3

\[ \text{PrimaryDesignatedAs}(\text{view}, \text{cohort}) \]

Reasoning (3.1.2.): Ref hypothesis:IAmPrimaryImplyPrimaryDesignated

Step 3.1.3. of 3

\[ \text{cohort} = \text{primary} \]

Reasoning (3.1.3.): Ref:UniquePrimaryDesignated

Step 3.2. of 2

\[ \text{opn} < \text{LL!Replica}(\text{cohort})!\text{LastProposed} \]

Reasoning (3.2.): Case eliminated by contradiction with Defn \text{ProposeAction} (opn = \text{LastProposed} + 1)

Case 4. of 5
\[ \exists m \in \text{PrimaryDesignatedMsg} : \]
\[ \land \text{LL!Replica}(\text{m.newPrimary})!\text{BecomePrimary}(\text{m}) \]
\[ \land \text{m.view} = \text{view} \]
\[ \land \text{opn} \in \text{DOMAIN m.prevPrepares} \]

Defn \[ m \triangleq \]
\[ \text{LL!Replica}(\text{m.newPrimary})!\text{BecomePrimary}(\text{m}) \]
\[ \land \text{m.view} = \text{view} \]
\( \land \, \text{opn} \in \text{DOMAIN} \, m.\text{prevPrepares} \)

**Step 4.1. of 1**

*ProposedAs* \((\text{view, } m.\text{newPrimary}, \text{opn}, m.\text{prevPrepares}[\text{opn}])\)'

*Reasoning* (4.1.): *Defs* *BecomePrimary* sends proposal message

*Reasoning* (4.1.): *Refs* *ProposalsInSameViewDoNotConflict*

**DefaultCase** 5. of 5

**Step 5.1. of 2**

*PrimaryDesignatedPrevPrep*(\(\text{view, opn, opn1}\))

*Reasoning* (5.1.): *PrimaryDesignatedPrevPrep* cannot change without a *DesignatePrimary* action

**Step 5.2. of 2**

*ProposedByAnyAs*(\(\text{view, opn, opn2}\))

*Reasoning* (5.2.): *ProposedByAnyAs*(\(\text{view, opn, opn2}\)) cannot change without a suitable *Propose* or *BecomePrimary* action

*Reasoning* (5.): induction hypothesis ; conclusion is a constant expression

*Reasoning*: case analysis

**Invariant** *ViewInitiatorElectedPrimaryInSameEpoch*

**Hypotheses of** *CsStateTypeInvariant*

**Introduce** *view \in ViewIds*

**Introduce** *primary \in Cohorts*

**Assume** *PrimaryDesignatedAs*(\(\text{view, primary}\)) \(\Rightarrow\) \(\text{view.viewInitiator.epoch} = \text{primary.epoch}\)

**Assume** *PrimaryDesignatedAs*(\(\text{view, primary}\))'

**Prove** *PrimaryDesignatedAs*(\(\text{view, primary}\))'

**Case 1. of 2**

\(\exists\, \text{config} \in \text{Designation Configurations} :\)

\(\land\, \text{config.view} = \text{view}\)

\(\land\, \text{config.newPrimary} = \text{primary}\)

\(\land\, \text{LL!Replica(config.designator)!DesignatePrimary(config)}\)

*Defs*

\(\text{config} \triangleq \)

**CHOOSE config \in Designation Configurations :**

\(\land\, \text{config.view} = \text{view}\)

\(\land\, \text{config.newPrimary} = \text{primary}\)

\(\land\, \text{LL!Replica(config.designator)!DesignatePrimary(config)}\)

**Step 1.1. of 3**

\(\text{view.viewInitiator} \in \text{LL!Replica(config.designator)!Membership}\)

**Step 1.1.1. of 2**

\(\text{config.designator} \in \text{LL!Replica(config.designator)!Membership}\)

*Reasoning* (1.1.1.): *Defs* *DesignatePrimary, Defn ActiveMember*

**Step 1.1.2. of 2**

106
\[
\text{config.designator} = \text{view.viewInitiator}
\]

Reasoning (1.1.): Defn DesignatePrimary

Reasoning (1.1.): substitution

Step 1.2. of 3

\[
\text{primary} \in \text{LL!Replica(config.designator)!.Membership}
\]

Step 1.2.1. of 5

\[
\text{config.quorum} \in \text{LL!Replica(config.designator)!.Quorum}
\]

Reasoning (1.2.1.): Defn DesignatePrimary

Step 1.2.2. of 5

\[
\text{config.quorum} \in \text{QuoraOfMembership(LL!Replica(config.designator)!.Membership)}
\]

Reasoning (1.2.2.): Defn Quora

Step 1.2.3. of 5

\[
\text{config.quorum} \subseteq \text{LL!Replica(config.designator)!.Membership}
\]

Reasoning (1.2.3.): Defn QuoraOfMembership

Step 1.2.4. of 5

\[
\text{config.newPrimary} \in \text{config.quorum}
\]

Reasoning (1.2.4.): Defn DesignationConfigurations

Step 1.2.5. of 5

\[
\text{config.newPrimary} \in \text{LL!Replica(config.designator)!.Membership}
\]

Reasoning (1.2.5.): Substitution

Reasoning (1.2.): Substitution

Step 1.3. of 3

\[
\text{LL!Replica(config.designator)!.Membership} \in \text{Memberships}
\]

Step 1.3.1. of 1

\[
\text{Range(} \text{LL!Replica(config.designator)!.CsStatemembershipMap)} = \text{Memberships}
\]

Reasoning (1.3.1.): Ref hypothesis: CsStateTypeInvariant

Reasoning (1.3.): Defn Membership

Reasoning (1.): Defn Memberships requires all members to share a common epoch.

DefaultCase 2. of 2

Step 2.1. of 1

PrimaryDesignatedAs(view, primary)

Reasoning (2.1.): No other action could have sent a witness PrimaryDesignatedMsg, so it must already have been in SentMessages

Reasoning (2): induction hypothesis

Reasoning: Proof by case analysis

Invariant PrimaryAndViewInitiatorInSameEpoch

Hypotheses of ViewInitiatorElectsPrimaryInSameEpoch

Introduce primary \in Cohorts

Assume

LL!Replica(primary)!IAmPrimary
\[LL! \text{Replica}(\text{primary})! \text{Cur View}.\text{viewInitiator}.\text{epoch} = \text{primary}.\text{epoch}\]

Assume \(LL! \text{Replica}(\text{primary})! \text{IAmPrimary}'\)

Prove \((LL! \text{Replica}(\text{primary})! \text{Cur View}.\text{viewInitiator}.\text{epoch} = \text{primary}.\text{epoch})'\)

Case 1, of 2

\(\exists m \in \text{PrimaryDesignatedMsg} : LL! \text{Replica}(\text{primary})! \text{BecomePrimary}(m)\)

Define \(m \triangleq \text{PrimaryDesignatedMsg} : LL! \text{Replica}(\text{primary})! \text{BecomePrimary}(m)\)

Step 1.1, of 1

\(\text{PrimaryDesignatedAs}(LL! \text{Replica}(\text{primary})! \text{Cur View}, \text{primary})\)

Reasoning (1.1.): Defn \text{BecomePrimary} constraints \text{Cur View}

Reasoning (1.): Ref hypothesis \text{ViewInitiatorElectionInSameEpoch}

Default Case 2, of 2

Step 2.1, of 4

\(LL! \text{Replica}(\text{primary})! \text{IAmPrimary}\)

Reasoning (2.1.): No actions besides \text{BecomePrimary} change \text{IAmPrimary} to true, so it must have stayed true.

Step 2.2, of 4

\(LL! \text{Replica}(\text{primary})! \text{Cur View}.\text{viewInitiator}.\text{epoch} = \text{primary}.\text{epoch}\)

Reasoning (2.2.): induction hypothesis

Step 2.3, of 4

\(- (\exists m \in \text{VeInitiatedMsg} : LL! \text{Replica}(\text{primary})! \text{VeAck}(m))\)

Reasoning (2.3.): \text{VeAck} action requires

\(LL! \text{Replica}(\text{primary})! \text{IAmPrimary}' = \text{FALSE}\)

Step 2.4, of 4

\(\text{UNCHANGED LL! Replica(primary)! Cur View}\)

Reasoning (2.4.): inspection of remaining actions

Reasoning (2.): substitution

Reasoning: Proof by case analysis

\[\text{Invariant} \ Proposed \text{Constraints ViewMembership}\]

Hypotheses of \(\text{MembershipChangesAreBroadcast}\)

Hypotheses of \(\text{PrimaryAndViewInitiatorInSameEpoch}\)

Hypotheses of \(\text{MembershipAsDeterminesMembership}\)

Hypotheses of \(\text{ProposedImpliesMembershipAs}\)

Hypotheses of \(\text{NonconflictingViewMembershipsPrimed}\)

Introduce \(\text{view} \in \text{ViewIds}\)

Introduce \(\text{opn} \in \text{Ops}\)

Assume \(\text{ProposedByAny}(\text{view, opn}) \Rightarrow \text{ViewMembership(view)} = \text{Membership(opn)}\)

Assume \(\text{ProposedByAny}(\text{view, opn})'\)

Prove \((\text{ViewMembership(view)} = \text{Membership(opn)})'\)

Define \(\text{rec} \triangleq \)
CHOOSE rec ∈ [cohort : Cohorts, opv : CsOps] :
  LLI.Replica(rec.cohort)!ProposeAction(view, opn, rec.opv)
Case 1. of 2
  LLI.Replica(rec.cohort)!ProposeAction(view, opn, rec.opv)
    
Step 1.1. of 8
  ∧ opn ∈ DOMAIN LLI.Replica(rec.cohort)!CsState.membershipMap
  ∧ rec.cohort ∈ LLI.Replica(rec.cohort)!CsState.membershipMap[opn]
    Reasoning (1.1.): Defn Propose

Step 1.2. of 8
  ∃ membership ∈ Memberships :
    ∧ opn ∈ DOMAIN LLI.Replica(rec.cohort)!CsState.membershipMap
    ∧ membership = LLI.Replica(rec.cohort)!CsState.membershipMap[opn]
    ∧ rec.cohort ∈ membership
    Reasoning (1.2.): logical rewrite

Step 1.3. of 8
  ∃ membership ∈ Memberships :
    ∧ MembershipAs(opn, membership, LLI.SentMessages)
    ∧ rec.cohort ∈ membership
  Reasoning (1.3.): Replace first two conjuncts using Ref
  hypothesis: MembershipChangesAreBroadcast

Defn membership ≡
  ∨ MembershipAs(opn, membership, LLI.SentMessages)
  ∧ rec.cohort ∈ membership

Step 1.4. of 8
  EpochOf(membership) = view.viewInitiator.epoch
  Reasoning (1.4.1.): Defn Memberships; Defn EpochOf

Step 1.5. of 8
  ViewMembership(view) = membership
  Reasoning (1.5.): Defn ViewMembership

Step 1.6. of 8
  MembershipAs(opn, membership, LLI.SentMessages')
  Reasoning (1.6.): Ref: MembershipAsMonotonic

Step 1.7. of 8
  (Membership(opn')) = membership
  Reasoning (1.7.): Ref: MembershipAsDeterminesMembership

Step 1.8. of 8
  (ViewMembership(view')) = membership
  Reasoning (1.8.): Ref: NonconflictingViewMembership

Reasoning (1.): transitivity
DefaultCase 2. of 2
Step 2.1. of 5

\(\text{ProposedByAny}(\text{view, opn})\)

Step 2.1.1. of 1

UNCHANGED \(\text{ProposedByAny}(\text{view, opn})\)

Reasoning (2.1.1.): Case condition

Reasoning (2.1.): antecedent

Step 2.2. of 5

\(\text{ViewMembership(view)} = \text{Membership(opn)}\)

Reasoning (2.2.): induction hypothesis

Step 23. of 5

\(\text{MembershipAs(opn, Membership(opn), LL!SentMessages')}\)

Step 23.1. of 1

\(\text{MembershipAs(opn, Membership(opn), LL!SentMessages)}\)

Reasoning (23.1.): \(\text{Ref:ProposedImpliesMembershipAs}\)

Reasoning (23.2.): \(\text{Ref:MembershipAsMonotonic}\)

Step 24. of 5

\((\text{Membership(opn')} = \text{Membership(opn)})\)

Reasoning (24.): \(\text{Ref:MembershipAsDeterminesMembership}\)

Step 25. of 5

\((\text{ViewMembership(view')} = \text{Membership(opn)})\)

Reasoning (25.): \(\text{Ref:NonconflictingViewMembershipPruned}\)

Reasoning (25.): case analysis

Theorem QuorumPreparationPreventsConflictingProposal

Hypotheses of \(\text{VcAckPreparesReflectViewRecentPrepare}\)

Hypotheses of \(\text{ProposedInSameViewDoNotConflict}\)

Hypotheses of \(\text{VcAckPreparedImpliesPrepared}\)

Hypotheses of \(\text{ProposalsRespectPrevPrepares}\)

Hypotheses of \(\text{ProposedImpliesElectedQuorum}\)

Hypotheses of \(\text{ProposedConstrainsViewMembership}\)

Hypotheses of \(\text{PreparedImpliesProposed}\)

Introduce \(v_1 \in \text{ViewIds}\)

Introduce \(v_2 \in \text{ViewIds}\)

Introduce \(\text{opn} \in \text{Ops}\)

Introduce \(\text{opv}1 \in \text{CsOps}\)

Introduce \(\text{opv}2 \in \text{CsOps}\)

Assume [A1] \(\text{QuorumPreparedAs}(v_1, \text{opn}, \text{opv}1)\)

Assume [A2] \(v_1 < v_2\)

Assume [A2] \(\text{opv}2 \neq \text{opv}1\)

Prove \(\neg \text{ProposedByAnyAs}(v_2, \text{opn}, \text{opv}2)\)
Summary: The proof of this theorem is the core of the Pazos proof. We proceed by contradiction. We are given one witness view \( v_1 \) in which a quorum prepares \( opn \) as \( opv_1 \), and a later view \( v_2 \) in which a primary manages to propose \( opn \) as \( opv_2 \). Once we have those witnesses, we know that there is some earliest view with such a conflicting proposal (which view this proof calls \( vm \)). The quorum that elected the primary of \( vm \) must have allowed the conflicting proposal, and the quorum that prepared \( opv_1 \) in \( v_1 \) should not have. We identify a spoiler cohort that belongs to both quorums (that's the point of a quorum), show that he must have quorum prepared in \( v_1 \) before electing in \( vm \), and show that he must have maintained and relayed to the primary of \( vm \) prepared\( opn \) info that would have precluded the conflicting proposal.

Step 1. of 1
Assume \([A.1.1.]. \quad ProposedByAnyAs(v_2, \, opn, \, opv_2)\)
Prove \quad FALSE.
Define \quad InconsistentProposalView(v_i) \triangleq \exists opv_3 \in CsOps:
\quad \land v_1 < v_i
\quad \land ProposedByAnyAs(v_i, \, opn, \, opv_3)
\quad \land opv_3 \neq opv_1
Define \quad vm \triangleq \text{Minimum}(\{v \in ViewIds : InconsistentProposalView(v)\})
Step 1.1. of 10
InconsistentProposalView(vm)
Reasoning (1.1.): True when \( vm = v_2 \); maybe for some earlier view.
Define \quad PreparingQuorum \triangleq \{c \in Cohorts : PreparedAs(v_1, \, c, \, opn, \, opv_1)\}
Step 1.2. of 10
PreparingQuorum \in Quora(opn)
Reasoning (1.2.): Ref: Assumption A1; def QuorumPreparedAs
Define \quad ElectionQuorum \triangleq \text{CHOICE} quorum \in Quora(opn) : PlausibleElectionQuorum(vm, \, quorum)
Step 1.3. of 10
PlausibleElectionQuorum(vm, \, ElectionQuorum)
Step 1.3.1. of 3
ProposedByAny(vm, \, opn)
Reasoning (1.3.1.): Ref: Step 1.1; Defn InconsistentProposalView
Step 1.3.2. of 3
\exists quorum \in QuoraOfMembership(ViewMembership(vm)) :
\quad PlausibleElectionQuorum(vm, \, quorum)
Reasoning (1.3.2.): Ref hypothesis: ProposedImpliesElectionQuorum
Step 1.3.3. of 3
QuoraOfMembership(ViewMembership(vm)) = Quora(opn)
Step 1.3.3.1. of 1
ViewMembership(vm) = Membership(opn)
Reasoning (1.3.3.1.): Ref hypothesis: Proposed Constrains View Membership
Reasoning (1.3.3.): Defn Quora
Reasoning (1.3.): choose: axiom
Define \quad spoiler \triangleq \text{CHOICE} c \in PreparingQuorum \cap ElectionQuorum ; \text{TRUE}
Step 1.4. of 10
spoiler \in PreparingQuorum \cap ElectionQuorum

111
Summary: The definition of QuoraOfMembership and the pigeon-hole principle ensure that the two quora overlap.

Step 1.4.1. of 1
Assume \( \text{PreparingQuorum} \cap \text{ElectionQuorum} = \{\} \)
Prove FALSE
Defn \( \text{bothQuora} \triangleq \text{PreparingQuorum} \cup \text{ElectionQuorum} \)

Step 1.4.1. of 2
\( \text{bothQuora} \subseteq \text{Membership(opn)} \)

Reasoning (1.4.1.1): Defn \( \text{PreparingQuorum} \); Defn \( \text{ElectionQuorum} \); Defn \( \text{Quora(opn)} \); Defn \( \text{QuoraOfMembership} \); Defn \( \text{subset} \); property of \( \cup \)

Step 1.4.1.2. of 2
Cardinality(bothQuora) > Cardinality(Membership(opn))

Reasoning (1.4.1.2): Since bothQuora is composed of a union of disjoint sets, its size is the sum of the sizes of the operands of the union. Defn QuoraOfMembership provides a minimum on the size of each operand.

Reasoning (1.4.1.): Strangely, the size of the set is bigger than the size of its superset.

Reasoning (1.4.): Proof by contradiction

Step 1.5. of 10
\( \text{VcAcknowledged(iter, spoiler)} \)

Reasoning (1.5.): Defn \( \text{ElectionQuorum} \); Flasible\( \text{ElectionQuorum} \)

Step 1.6. of 10
\( \text{PreparedAs}(\text{v1, spoiler, opn, opv1}) \)

Reasoning (1.6.): Defn \( \text{PreparingQuorum} \); Quorum\( \text{Prepared} \)

Step 1.7. of 10
\( \text{VcAcknowledgedIter(iter, spoiler, opn, [view \mapsto v1, opv \mapsto opv1])} \)

Defn \( \text{ViewPreparesOpn}(\text{va}) \triangleq \exists \text{opv} \in \text{Ops} : \text{va} < \text{vm} \land \text{PreparedAs}(\text{va, spoiler, opn, opv}) \)

Defn \( \text{lastViewPreparingOpnBeforeConflict} \triangleq \text{Maximum}([v \in \text{ViewIds} : \text{ViewPreparesOpn}(v)]) \)

Step 1.7.1. of 6
\( \text{ViewPreparesOpn(lastViewPreparingOpnBeforeConflict)} \)

Step 1.7.1.1. of 1
\( \exists \text{v} \in \text{ViewIds} : \text{ViewPreparesOpn}(\text{v}) \)

Reasoning (1.7.1.1.): assumption 1 provides a witness \text{ViewPreparesOpn}(\text{v1})

Reasoning (1.7.1.): choose axiom

Step 1.7.2. of 6
\( \text{v1} \leq \text{lastViewPreparingOpnBeforeConflict} \)

Reasoning (1.7.2.): \( \text{v1} \) satisfies \text{ViewPreparesOpn}, and hence provides a lower bound for \text{Maximum}()

Defn \( \text{lastOpnPreparedBeforeConflict} \triangleq \text{choose opv4} : \text{PreparedAs(lastViewPreparingOpnBeforeConflict, spoiler, opn, opv4)} \)

Step 1.7.3. of 6
\( \text{PreparedAs(lastViewPreparingOpnBeforeConflict, spoiler, opn, lastOpnPreparedBeforeConflict)} \)

112
Reasoning (1.7.3.): choose axiom

Step 1.7.4. of 6
\[ Vc.Ack\text{PreparedOp As} ( \]
\[ \text{vm, } \]
\[ \text{spoiler, } \]
\[ \text{opn, } \]
\[ [ \]
\[ \text{opv \leftrightarrow lastOpePreparedBeforeConflict, } \]
\[ \text{view \leftrightarrow lastViewPreparingOpnBeforeConflict} \]
\] \]

Reasoning (1.7.4.): \text{Ref hypothesis: Vc.AckPreparesRecentViewRecentPrepare} \text{ (v1 = lastViewPreparingOpnBeforeConflict, v2 = vm, opv = lastOpePreparedBeforeConflict)}

Step 1.7.5. of 6
\[ v1 \leq lastViewPreparingOpnBeforeConflict \]

Reasoning (1.7.5.): assumption provides a witness to a max value of \text{lastViewPreparingOpnBeforeConflict}

Step 1.7.6. of 6
\[ \text{lastOpePreparedBeforeConflict} = opv1 \]

Case 1.7.6.1. of 2
\[ v1 < lastViewPreparingOpnBeforeConflict \]

Reasoning (1.7.6.1.): \text{Defn vm requires lastViewPreparingOpnBeforeConflict to prepare opv1.} \text{ (By contradiction: if it prepares something else, then Ref hypothesis PreparedImpliesProposed requires it to be proposed there, which contradicts the definition of vm as the minimum view in which an opv other than opv1 was proposed.)}

Case 1.7.6.2. of 2
\[ v1 = lastViewPreparingOpnBeforeConflict \]

Step 1.7.6.2.1. of 2
\[ \text{ProposedByAnyAs (v1, opn, lastOpePreparedBeforeConflict)} \]

Reasoning (1.7.6.2.1.): \text{Ref hypothesis: PreparedImpliesProposed}

Step 1.7.6.2.2. of 2
\[ \text{ProposedByAnyAs (v1, opn, opv1)} \]

Step 1.7.6.2.2.1. of 1
\[ \exists \text{ c \in Cohorts : PreparedAs (v1, c, opn, opv1)} \]

Reasoning (1.7.6.2.2.1.): \text{Ref Assumption A1, defn QuorumPreparedAs}

Reasoning (1.7.6.2.2.): \text{Ref hypothesis: PreparedImpliesProposed}

Reasoning (1.7.6.2.): \text{Ref hypothesis: ProposedInSameView DoNotConflict}

Reasoning (1.7.6.): \text{Proof by cases}

Reasoning (1.7.): \text{Ref: Step 1.7.4. ; Ref: Step 1.7.6.}

Step 18. of 10
Introduce \[ c \in Cohorts \]
Assume \[ Vc.AckedView (vm, c) \]
Prove \[ \forall \text{ Choose Vc.AckPreparedOpInfo (vm, c, opn).opv = opv1 } \]
\[ \forall \text{ Choose Vc.AckPreparedOpInfo (vm, c, opn).view < v1 } \]
Defn \( \text{latestPreparedView} \triangleq \text{Choose VcAckPreparedOpInfo}(vm, c, opn).view \)

Case 1.8.1. of 2
\( \text{latestPreparedView} < v1 \)
Reasoning [1.8.1.]: satisfies second disjunct of prove goal

Case 1.8.2. of 2
\( v1 < \text{latestPreparedView} \)

Step 1.8.2.1. of 5
\( \text{Prepared}(\text{latestPreparedView}, c, opn) \)
Reasoning [1.8.2.1.]: Ref hypothesis: VcAckPreparedImpliesPrepared

Step 1.8.2.2. of 5
\( \forall \, vi \in \text{ViewIds} : v1 < vi \land vi < v2 \Rightarrow (\neg \text{Prepared}(vi, c, opn)) \)
Reasoning [1.8.2.2.]: Contrapositive of Ref hypothesis: VcAckPreparedReflectViewRecentPrepare

Step 1.8.2.3. of 5
Introduce \( opv3 \in \text{ViewIds} \)
Assume \( opv3 \neq opv1 \)
Prove \( \neg \text{PreparedAs}(\text{latestPreparedView}, c, opn, opv3) \)

Step 1.8.2.3.1. of 1
Assume \( \text{PreparedAs}(\text{latestPreparedView}, c, opn, opv3) \)
Prove \( \text{FALSE} \)

Step 1.8.2.3.1.1. of 2
\( \text{ProposedByAnyAs}(\text{latestPreparedView}, opn, opv3) \)
Reasoning [1.8.2.3.1.1.]: Ref hypothesis: PreparedImpliesProposed

Step 1.8.2.3.1.2. of 2
\( \text{InconsistentProposalView}(\text{latestPreparedView}) \)
Reasoning [1.8.2.3.1.2.]: Defn InconsistentProposalView

Reasoning [1.8.2.3.]: \( \text{latestPreparedView} \) is a witness to the non-minimality of \( vm \)

Reasoning [1.8.2.3.]: By contradiction

Step 1.8.2.4. of 5
\( \text{PreparedAs}(\text{latestPreparedView}, c, opn, opv1) \)
Reasoning [1.8.2.4.]: Defn Prepared gives witness \( opv1 \); only \( opv1 \) satisfies previous step

Step 1.8.2.5. of 5
\( \text{Choose VcAckPreparedOpInfo}(vm, c, opn).opv = opv1 \)
Reasoning [1.8.2.5.]: Ref hypothesis: VcAckPreparedReflectViewRecentPrepare

Reasoning [1.8.2.]: satisfies first disjunct of prove goal

Reasoning [1.8.]: Proof by case analysis

Step 1.9. of 10
\( \text{PrimaryDesignatedPrevPrep}(vm, opn, opv1) \)
Reasoning [1.9.]: No conflicting VcAckPrevPreps have views later than \( v1 \), so any conflict is dominated by VcAckPreparedOp\( (vm, spoiler, opn) \)

Step 1.10. of 10
\( opv1 = opv2 \)
Reasoning (1.10): Ref: Assumption 1.1. and Ref: Step 1.9. satisfy Ref hypothesis: Proposals Respect Prevent Prepare

Reasoning (1.1): We have arrived at a contradiction with Ref: Assumption A2.

Reasoning: Proof by contradiction.

Theorem QuorumPreparedImpliesProposed

Hypotheses of PreparedImpliesProposed
Introduce \( v \in \text{ViewIds} \)
Introduce \( opn \in \text{Opns} \)
Introduce \( opv \in \text{CsOps} \)
Assume \( \text{QuorumPreparedAs}(v, opn, opv) \)
Prove \( \text{ProposedByAnyAs}(v, opn, opv) \)

Step 1 of 1

\[ \exists c \in \text{Cohorts} : \text{PreparedAs}(v, c, opn, opv) \]

Reasoning (1.1): Defn QuorumPreparedAs

Reasoning: Ref hypothesis: PreparedImpliesProposed

Theorem No Conflicting Quorum Preparation In Ordered Views

Hypotheses of QuorumPreparedImpliesProposed
Hypotheses of QuorumPreparation Prevents Conflicting Proposal
Hypotheses of ProposedImpliesElectionQuorum
Hypotheses of ProposedConstrainsViewMembership
Introduce \( v1 \in \text{ViewIds} \)
Introduce \( v2 \in \text{ViewIds} \)
Introduce \( opn \in \text{Opns} \)
Introduce \( opv1 \in \text{CsOps} \)
Introduce \( opv2 \in \text{CsOps} \)
Assume \( v1 < v2 \)
Assume \( \text{QuorumPreparedAs}(v1, opn, opv1) \)
Assume \( \text{QuorumPreparedAs}(v2, opn, opv2) \)
Prove \( opv1 = opv2 \)

Step 1 of 1

\( \text{ProposedByAnyAs}(v2, opn, opv) \)

Reasoning (1.1): Ref: QuorumPreparedImpliesProposed

Reasoning: Ref: QuorumPreparation Prevents Conflicting Proposal and some algebra
Theorem \textit{No Conflicting Quorum Preparation}

Hypotheses of \textit{QuorumPreparedImpliesProposed}
Hypotheses of \textit{ProposedInSameViewDoNotConflict}
Hypotheses of \textit{No Conflicting Quorum Preparation In Ordered Views}

Introduce \( v_1 \in \text{ViewIds} \)
Introduce \( v_2 \in \text{ViewIds} \)
Introduce \( \text{opn} \in \text{Opns} \)
Introduce \( \text{opv} \in \text{CsOpns} \)
Introduce \( \text{opv1} \in \text{CsOpns} \)
Introduce \( \text{opv2} \in \text{CsOpns} \)

Assume \( \text{QuorumPreparedAs}(v_1, \text{opn}, \text{opv}) \)
Assume \( \text{QuorumPreparedAs}(v_2, \text{opn}, \text{opv2}) \)

\textbf{Proof:} \( \text{opv1} = \text{opv2} \)

Case 1. of 3
\( v_1 < v_2 \)

Reasoning (1): \textit{Ref: No Conflicting Quorum Preparation In Ordered Views (v1 = v1, v2 = v2)}

Case 2. of 3
\( v_1 = v_2 \)

Step 21. of 2
\( \text{ProposedByAnyAs}(v_1, \text{opn}, \text{opv1}) \)

Reasoning (21): \textit{Ref: QuorumPreparedImpliesProposed}

Step 22. of 2
\( \text{ProposedByAnyAs}(v_2, \text{opn}, \text{opv2}) \)

Reasoning (22): \textit{Ref: QuorumPreparedImpliesProposed}

Reasoning (2): \textit{Ref hypothesis: ProposedInSameViewDoNotConflict}

Case 3. of 3
\( v_1 > v_2 \)

Reasoning (3): \textit{Ref: No Conflicting Quorum Preparation In Ordered Views (v1 = v2, v2 = v1)}

Reasoning: Proof by case analysis

\textbf{Invariant} \textit{CommittedImpliesQuorumPrepared}

Hypotheses of \textit{LocalMembershipEpochOrdering}

Introduce \( \text{opn} \in \text{Opns} \)
Introduce \( \text{opv} \in \text{CsOpns} \)

Assume \( \text{CommittedByAnyAs}(\text{opn}, \text{opv}) \Rightarrow (\exists v \in \text{ViewIds} : \text{QuorumPreparedAs}(v, \text{opn}, \text{opv})) \)

Assume \( \text{CommittedByAnyAs}(\text{opn}, \text{opv}) \)

Prove: \( (\exists v \in \text{ViewIds} : \text{QuorumPreparedAs}(v, \text{opn}, \text{opv})) \)

Defn \( r \triangleq \)

\textbf{CHOOSE} \( r \in [\text{cohort} : \text{Cohorts}, \text{preparedMsgProto} : \text{PreparedMsg}] : \)
\( r.\text{preparedMsgProto.\text{opn}} = \text{opn} \)
\( r.\text{LL!Replica}(r.\text{cohort})!\text{Commit}(r.\text{preparedMsgProto}) \)
Case 1. of 2
\[ \text{rec.preparedMsgProto.opn} = \text{opn} \]
\[ \land \text{LL!Replica(rec.cohort)!Commit(rec.preparedMsgProto)} \]

\textbf{Defn:} \(\text{rec2} \triangleq \)

\textbf{CHOOSE}

\(\text{rec2} \in [mSet : \text{SUBSET ConsensusMessage, quorum : LL!Replica(rec.cohort)!Quora}] : \)

\(\land \text{LL!Replica(rec.cohort)!ReceiveMessageSet(rec2.mSet)} \)
\(\land \text{LL!Replica(rec.cohort)!MessagesMatchProtoType(rec2.mSet, rec.preparedMsgProto)} \)
\(\land \text{LL!Replica(rec.cohort)!EachCohortSentAMessage(rec2.quorum, rec2.mSet)} \)

\textbf{Step 1.1. of 3}
\(\land \text{LL!Replica(rec.cohort)!ReceiveMessageSet(rec2.mSet)} \)
\(\land \text{LL!Replica(rec.cohort)!MessagesMatchProtoType(rec2.mSet, rec.preparedMsgProto)} \)
\(\land \text{LL!Replica(rec.cohort)!EachCohortSentAMessage(rec2.quorum, rec2.mSet)} \)

\textbf{Reasoning [1.1.]: Defn Commit, Defn ReceivePromQuorum}

\textbf{Step 1.2. of 3}
\(\text{Quora(opn)} = \text{LL!Replica(rec.cohort)!Quora} \)

\textbf{Step 1.2.1. of 1}
\(\text{Membership(opn)} = \text{LL!Replica(rec.cohort)!Membership} \)

\textbf{Step 1.2.1.1. of 4}
\(\text{opn} \in \text{DOMAIN LL!Replica(rec.cohort)!CsState.membershipMap} \)

\textbf{Reasoning [1.2.1.1.]: Defn Commit implies ActiveMember}

\textbf{Step 1.2.1.2. of 4}
\(\text{MembershipAs(} \)
\(\text{opn, LL!Replica(rec.cohort)!CsState.membershipMap[opn], LL!SentMessages}) \)

\textbf{Reasoning [1.2.1.2.]: Ref hypothesis: MembershipChangesAreBroadcast}

\textbf{Step 1.2.1.3. of 4}
\(\text{Membership(opn)} = \text{LL!Replica(rec.cohort)!CsState.membershipMap[opn]} \)

\textbf{Reasoning [1.2.1.3.]: UNPRIMED version of Ref: MembershipAsDeterminesMembership}

\textbf{Step 1.2.1.4. of 4}
\(\text{LL!Replica(rec.cohort)!CsState.membershipMap[opn]} = \)
\(\text{LL!Replica(rec.cohort)!Membership} \)

\textbf{Step 1.2.1.4.1. of 2}
\(\text{rec.cohort} \in \text{LL!Replica(rec.cohort)!CsState.membershipMap[opn]} \)

\textbf{Reasoning [1.2.1.4.1.]: ActiveMember}

\textbf{Step 1.2.1.4.2. of 2}
\(\forall \text{opn2} \in \text{DOMAIN LL!Replica(rec.cohort)!CsState.membershipMap : rec.cohort} \in \text{LL!Replica(rec.cohort)!CsState.membershipMap} \Rightarrow \)
\(\text{LL!Replica(rec.cohort)!CsState.membershipMap[opn2]} = \)
\(\text{LL!Replica(rec.cohort)!CsState.membershipMap[opn]} \)

\textbf{Reasoning [1.2.1.4.2.]: Any two memberships both containing rec.cohort must have the same epoch; with Ref hypothesis: LocalMembershipEpochOrdering, they must be the same membership.}
Reasoning (1.2.1.4.): choose in Defn LL!Replica(rec.cohort)!Membership is fully-constrained

Reasoning (1.2.1.4.): substitution
Reasoning (1.2.): Defn Quorum; Defn LL!Replica(rec.cohort)!Quorum

Step 1.3. of 3
Introduce: \( \text{member} \in \text{rec.2:quorum} \)
Prove: \( \text{PreparedAs}(\text{rec.preparedMsgProto.view, member, opn, opv}) \)
Defn: \( \text{memberMessage} \triangleq \text{CHOOSE} \ m \in \text{rec2.mSet} : \text{m.sender} = \text{member} \)

Step 1.3.1. of 3
\( \text{memberMessage.sender} = \text{member} \)
Reasoning (1.3.1.): Defn EachCohortSentAMessage: choose axiom

Step 1.3.2. of 3
\( \wedge \text{memberMessage.view} = \text{rec.preparedMsgProto.view} \)
\( \wedge \text{memberMessage.opn} = \text{opn} \)
\( \wedge \text{memberMessage.opv} = \text{opv} \)
Reasoning (1.3.2.): Defn MessagesMatchPrototype

Step 1.3.3. of 3
\( \text{memberMessage} \in \text{SentMessages} \)
Reasoning (1.3.3.): Defn ReceiveMessages
Reasoning (1.3.): Defn PreparedAs

Reasoning (1.): rec2:quorum is witness to QuorumPreparedAs:quorum, and rec.preparedMsgProto.view is witness to v in proof obligation.

DefaultCase 2. of 2

Step 2.1. of 2
\( \text{CommittedByAnyAs(opn, opv)} \)

Step 2.1.1. of 1
\( \text{UNCHANGED} \text{CommittedByAnyAs(opn, opv)} \)

Reasoning (2.1.1.): CommittedByAnyAs only changes when a CommittedMsg is sent; Inspection of actions shows that only happens in Case 1.

Reasoning (2.1.): Assumption; Defn UNCHANGED
Step 2.2. of 2
\( \exists v \in \text{ViewIds} : \text{QuorumPreparedAs}(v, \text{opn, opv}) \)
Reasoning (2.2.): induction hypothesis
Reasoning (2.): Ref:QuorumPreparedAsMonotonic

Reasoning: Proof by case analysis

Theorem NoConflictingCommits

Hypotheses of: CommittedImpiesQuorumPrepared
Hypotheses of: NoConflictingQuorumPreparation
Introduce: \( \text{opn} \in \text{Ops} \)
Introduce: \( \text{opv1} \in \text{CsOps} \)
Introduce \( \text{opv} 2 \in C\text{sOps} \)
Assume \( \text{CommittedByAnyAs(opn, opv 1)} \)
Assume \( \text{CommittedByAnyAs(opn, opv 2)} \)
Prove \( \text{opv} 1 = \text{opv} 2 \)

Summary: We push the problem back from when the operations were Committed to when they were QuorumPrepared, and invoke \text{NoConflictingQuorumPreparation}.

Define \( \text{QuorumPreparedViews(opv)} \triangleq \{ v \in \text{Views} : \text{QuorumPreparedAs}(v, \text{opn}, \text{opv}) \} \)
Define \( v 1 \triangleq \text{CHOOSE } v 1 : v 1 \in \text{QuorumPreparedViews(opv} 1) \)
Define \( v 2 \triangleq \text{CHOOSE } v 2 : v 2 \in \text{QuorumPreparedViews(opv} 2) \)

Step 1. of 2
\( v 1 \in \text{QuorumPreparedViews(opv} 1) \)
Reasoning (1): \text{Ref: hypothesis; CommittedImplyCommittedPrepared}

Step 2. of 2
\( v 2 \in \text{QuorumPreparedViews(opv} 2) \)
Reasoning (2): \text{Ref: hypothesis; CommittedImplyCommittedPrepared}
Reasoning: \text{Apply Ref: NoConflictingQuorumPreparation}

Theorem \( \text{KnownOpvFcnExtended} \)

Hypotheses of \( \text{NoConflictingCommits} \)
\( \text{FcnExtends(KnownOpv', KnownOpv)} \)

Step 1. of 2
\( \text{DOMAIN KnownOpv} \subseteq \text{DOMAIN (KnownOpv')} \)
Reasoning (1): \text{Ref: Max KnownOpvGrows}

Step 2. of 2
Introduce \( \text{opn} \in \text{DOMAIN KnownOpv} \)
\( \text{KnownOpv[opn]} = (\text{KnownOpv'})[\text{opn}] \)

Summary: If an operation was known in the previous step, we’ll commit no conflicting operations in this step, so we can be sure that \( \text{KnownOpv}' \) makes the same operation assignment.

Step 21. of 3
\( \text{CommittedByAnyAs(opn, KnownOpv[opn])} \)
Reasoning (2.1): \text{Defn KnownOpv; Defn Max KnownOpn}

Step 22. of 3
\( \text{CommittedByAnyAs(opn, KnownOpv[opn])}' \)
Reasoning (2.2): \text{Ref: CommittedMonotonic}

Step 23. of 3
\( \forall \text{opv} \in \text{CsOps} : (\text{CommittedByAnyAs(opn, opv)})' = \text{opv} = \text{KnownOpv[opn]} \)
Reasoning (2.3): \text{Ref: NoConflictingCommits}
Reasoning (2.): \text{Defn KnownOpv; choose constrained to singleton set}
Reasoning: \text{Defn FcnExtends}
Theorem \textit{KnownStateFcnExtended}

Hypotheses of \textit{KnownOpvFcnExtended}\\
\textit{FcnExtends(KnownState', KnownState)}

\textbf{Step 1. of 2}\\
\textbf{Domain} \textit{KnownState} \subseteq \textbf{Domain} (\textit{KnownState}')\\
\textbf{Reasoning} (1): Ref: \textit{Max KnownOpvGrows}\\

\textbf{Step 2. of 2}\\
\textbf{Introduce} \ opn \in \textbf{Domain} \textit{KnownState}\\
\textbf{Prove} \ \textit{KnownState[0]} = (\textit{KnownState}')[0]\\
\textbf{Reasoning} [2.1]: \textit{CInit} = \textit{CInit}\\

\textbf{Step 2.2. of 2}\\
\textbf{Assume} \ opn < \textit{opn}\\
\textbf{Assume} \ \textit{KnownState}[(\textit{opn} - 1)] = (\textit{KnownState}')[(\textit{opn} - 1)]\\
\textbf{Prove} \ \textit{KnownState[opn]} = (\textit{KnownState}')[opn]\\
\textbf{Step 2.2.1. of 1}\\
\textbf{opn} \in \textbf{Domain} \textit{KnownOpv}\\
\textbf{Reasoning} [2.2.1.1]: \textit{Defn KnownOpv}\\
\textbf{Reasoning} [2.2.1.2]: \textit{Ref. KnownOpvFcnExtended}\\
\textbf{Reasoning} [2.2.]: Substitution of equal terms in \textit{else} clause of \textit{Defn KnownState}.\\
\textbf{Reasoning} [2.]: Proof by induction on \textit{opn}\\
\textbf{Reasoning}: \textit{Defn FcnExtends}

\hline

\textbf{Invariant} \textit{LocalStateConsistentWithKnownState}

\textbf{Hypotheses of} \textit{NoConflictingCommits}\\
\textbf{Introduce} \ \textit{cohort} \in \textbf{Cohorts}\\
\textbf{Defn} \ \textit{state} \triangleq \textit{LL}!\textit{Replica(cohort)}!\textit{CsState}\\
\textbf{Defn} \ \textit{snapshot} \triangleq \textit{LL}!\textit{Replica(cohort)}!\textit{CsStateSnapshot}\\
\textbf{Assume}\\
\ \land \textit{Consonant(state)}\\
\ \land \textit{Consonant(snapshot)}\\
\textbf{Prove}\\
\ \land \textit{Consonant(snapshot)}'\\

\textbf{Summary}: We first establish lemmas showing that if either \textit{state} or \textit{snapshot} doesn't change, the corresponding variable holds its consonance. With those lemmas, we can charge through a case analysis of the three actions that touch \textit{state} or \textit{snapshot}.\hline

120
Step 1. of 6
Assume \( \text{UNCHANGED state} \)
Prove \( \text{Consonant(state)}' \)

Step 1.1. of 1
\( \land (\text{state}') \in \text{Range(KnownState)} \)
\( \land (\text{state}') = \text{KnownState[state'].numExecuted} \)
Reasoning [1.1.]: UNCHANGED assumption; induction hypothesis

Reasoning [1.1.]: Ref:KnownState Fen Extended; Defn Consonant

Step 2. of 6
Assume \( \text{UNCHANGED snapshot} \)
Prove \( \text{Consonant(snapshot)}' \)

Step 2.1. of 1
\( \land (\text{snapshot}') \in \text{Range(KnownState)} \)
\( \land (\text{snapshot}') = \text{KnownState[snapshot'.numExecuted]} \)
Reasoning [2.1.]: UNCHANGED assumption; induction hypothesis

Reasoning [2.1.]: Ref:KnownState Fen Extended; Defn Consonant

Case 3. of 6
\( \text{LL! Replica(cohort)}!Persist \)

Summary: The state is UNCHANGED by Persist; the snapshot part relies on the consonance of state in the prior state.

Step 3.1. of 2
\( \text{Consonant(state)}' \)

Step 3.1.1. of 1
\( \text{UNCHANGED state} \)
Reasoning [3.1.1.]: Defn Crash action
Reasoning [3.1.1.]: Ref: Step 1.

Step 3.2. of 2
\( \text{Consonant(snapshot)}' \)

Step 3.2.1. of 2
\( (\text{snapshot}') = \text{state} \)
Reasoning [3.2.1.]: Defn Persist action

Step 3.2.2. of 2
\( \land (\text{snapshot}') \in \text{Range(KnownState)} \)
\( \land (\text{snapshot}') = \text{KnownState[snapshot'.numExecuted]} \)
Reasoning [3.2.2.]: UNCHANGED assumption; induction hypothesis
Reasoning [3.2.2.]: Ref:KnownState Fen Extended; Defn Consonant

Reasoning [3.2.2.]: We’ve shown both conjuncts of the proof goal

Case 4. of 6
\( \text{LL! Replica(cohort)!Crash} \)

Summary: This case is the mirror of the previous. The snapshot is UNCHANGED by a Crash; the state part relies on the consonance of snapshot in the prior state.

Step 4.1. of 2
\( \text{Consonant(snapshot)}' \)

Step 4.1.1. of 1
\( \text{UNCHANGED snapshot} \)
Reasoning (4.1.1): Defn Crash action
Reasoning (4.1.): Ref: Step 2
Step 4.2. of 2
Consonant (state)
Step 4.2.1. of 2
(state') = snapshot
Reasoning (4.2.1.): Defn Crash action
Step 4.2.2. of 2
∧ (state') ∈ Range(KnownState)
∧ (state') = KnownState[state'.numExecuted]
Reasoning (4.2.2.): induction hypothesis
Reasoning (4.2.): Ref: KnownStateFenExtended; Defn Consonant
Reasoning (4.): We've shown both conjuncts of the proof goal

Case 5. of 6
∃ m ∈ CommittedMsg : LL!Replica(cohort)!Execute(m)
Defn m ≡ CHOOSE m ∈ CommittedMsg : LL!Replica(cohort)!Execute(m)
Step 5.1. of 2
Consonant (snapshot)
Step 5.1.1. of 1
UNCHANGED snapshot
Reasoning (5.1.1.): Defn Crash action
Reasoning (5.1.): Ref: Step 2
Step 5.2. of 2
Consonant (state)
Step 5.2.1. of 2
state'.numExecuted ∈ DOMAIN KnownState
Step 5.2.1.1. of 2
state.numExecuted + 1 ≤ MaxKnownOpn
Step 5.2.1.1.1. of 2
∀ opn ∈ 1..state.numExecuted : CommittedByAny(opn)
Step 5.2.1.1.1.1. of 1
state.numExecuted ≤ MaxKnownOpn
Step 5.2.1.1.1.1.1. of 1
state.numExecuted ∈ DOMAIN KnownState
Reasoning (5.2.1.1.1.1.1.): induction hypothesis; Defn Consonant
Reasoning (5.2.1.1.1.1.): Defn KnownState
Reasoning (5.2.1.1.1.): Defn MaxKnownOpn
Step 5.2.1.1.2. of 2
CommittedByAny(state.numExecuted + 1)
Step 5.2.1.1.2.1. of 1
CommittedByAnyAs(m.opn, m.opv)
Reasoning (5.2.1.1.2.1.): According to Defn Commit, Message m is a witness
Reasoning (5.2.1.1.2.): Defn Execute action
Reasoning (5.2.1.1.): Defn MaxKnownOpn

122
Step 5.2.1.2. of 2
\[ \text{state'.numExecuted} + 1 \in \text{DOMAIN KnownState} \]

Reasoning (5.2.1.2.): Defn KnownState

Step 5.2.2. of 2
\[ (\text{state'}) = \text{KnownState}[\text{state'.numExecuted}] \]

Step 5.2.2.1. of 4
\[ \text{KnownState}[(\text{state'}).numExecuted - 1)] = \text{state} \]

Step 5.2.2.1. of 2
\[ \text{state} = \text{KnownState}[\text{state}.numExecuted] \]

Reasoning (5.2.2.1.1.): induction hypothesis; Defn Consonant

Step 5.2.2.2. of 2
\[ \text{state}.numExecuted = \text{state}'.numExecuted - 1 \]

Reasoning (5.2.2.2.1.): Defn Execute; Defn CsTz

Reasoning (5.2.2.1.): Substitution

Step 5.2.2.2. of 4
\[ \text{KnownOpv}[\text{state}'.numExecuted] = \text{opv} \]

Step 5.2.2.2.1. of 2
\[ \text{CommittedByAnyAs}(\text{opn}, \text{opv}) \]

Reasoning (5.2.2.2.1.): m is a witness

Step 5.2.2.2.2. of 2
\[ \forall \text{opv} \in \text{Ops} : \text{CommittedByAnyAs}(\text{opn}, \text{opv}) \Rightarrow \text{opv} = \text{op} \]

Reasoning (5.2.2.2.2.): Ref: NoConflictingCommits

Reasoning (5.2.2.2.): choose in Defn KnownOpv is fully constrained.

Step 5.2.2.3. of 4
\[ \text{KnownState}[\text{state}'.numExecuted] = \text{CsTz}[\text{state}, \text{opv}] \]

Reasoning (5.2.2.3.): Defn KnownState; Ref: Step 5.2.2.1. ; Ref: Step 5.2.2.2.

Step 5.2.2.4. of 4
\[ (\text{state'}) = \text{CsTz}[\text{state}, \text{opv}] \]

Reasoning (5.2.2.4.): Defn Execute action

Reasoning (5.2.2.): Substitution

Reasoning (5.): We’ve shown both conjuncts of the proof goal

DefaultCase 6. of 6

Step 6.1. of 1
\[ \land \text{UNCHANGED state} \]
\[ \land \text{UNCHANGED snapshot} \]

Reasoning (6.1.): All other actions leave UNCHANGED CsState and CsStateSnapshot.

Reasoning (6.): Ref: Step 1.; Ref: Step 2.

Reasoning: Proof by case analysis
Invariant: BroadcastMembershipReflectKnownState
Hypotheses:
LocalStateConsonantWithKnownState
Introduce: \( \text{opn} \in \text{Ops} \)
Introduce: \( \text{membership} \in \text{Memberships} \)
Assume:
\( \wedge \text{Alpha} < \text{opn} \)
\( \wedge \text{MembershipAs(opn, membership, LL!SentMessages)} \)
\( \Rightarrow \)
\( \wedge \text{opn} - \text{Alpha} \in \text{DOMAIN KnownState} \)
\( \wedge \text{opn} \in \text{DOMAIN KnownState}[\text{opn} - \text{Alpha}], \text{membershipMap} \)
\( \wedge \text{KnownState}[\text{opn} - \text{Alpha}], \text{membershipMap} \text{[opn]} = \text{membership} \)
Assume:
\( ( \wedge \text{Alpha} < \text{opn} \)
\( \wedge \text{MembershipAs(opn, membership, LL!SentMessages)} )^\prime \)
Prove:
\( ( \wedge \text{opn} - \text{Alpha} \in \text{DOMAIN KnownState} \)
\( \wedge \text{opn} \in \text{DOMAIN KnownState}[\text{opn} - \text{Alpha}], \text{membershipMap} \)
\( \wedge \text{KnownState}[\text{opn} - \text{Alpha}], \text{membershipMap} \text{[opn]} = \text{membership} )^\prime \)

Case 1. of 2
\( \exists \text{membershipMsg} \in \text{MembershipMsg} : \)
\( \wedge \text{membershipMsg} \text{.opn} = \text{opn} \)
\( \wedge \text{membershipMsg} \text{.membership} = \text{membership} \)
\( \wedge \text{membershipMsg} \notin \text{SentMessages} \)
\( \wedge \text{membershipMsg} \in \text{(SentMessages')} \)

Define: \( \text{rec} \triangleq \)

\( \text{choose rec} \in [\text{cohort} : \text{Cohorts}, \text{commitMsg} : \text{CommittedMsg}] : \)
\( \wedge \text{LL!Replica(rec.cohort)!Execute(rec.commitMsg)} \)
\( \wedge \text{rec.commitMsg} \text{.opn} = \text{opn} - \text{Alpha} \)

Step 1.1. of 6
\( \wedge \text{LL!Replica(rec.cohort)!Execute(rec.commitMsg)} \)
\( \wedge \text{rec.commitMsg} \text{.opn} = \text{opn} - \text{Alpha} \)

Reasoning: Only such an \text{Execute} action sends a \text{MembershipMessage} matching the Case condition.

Step 1.2. of 6
\( \wedge \text{rec.commitMsg} \text{.opn} \in \text{DOMAIN KnownState} \)
\( \wedge (\text{LL!Replica(rec.cohort)!CsState'}) = \text{KnownState}[\text{rec.commitMsg} \text{.opn}] \)

Step 1.2.1. of 3
\( \text{rec.commitMsg} \text{.opn} = \text{LL!Replica(rec.cohort)!CsState', numExecuted} \)

Reasoning: Only such an \text{Execute} action sends a \text{MembershipMessage} matching the Case condition.

Step 1.2.2. of 3
\( \wedge \text{LL!Replica(rec.cohort)!CsState', numExecuted} \in \text{DOMAIN (KnownState')} \)
\( \wedge (\text{LL!Replica(rec.cohort)!CsState'}) = \)

Step 1.2.3. of 3
\( \text{(KnownState')[LL!Replica(rec.cohort)!CsState', numExecuted}] \)

Reasoning: Only such an \text{Execute} action sends a \text{MembershipMessage} matching the Case condition.
**UNCHANGED KnownState**

**Reasoning (1.2.3.):** Execute action sends no CommittedMsgs; KnownState only varies over SentMessages \( \cap \) CommitedMsgs.

**Step 1.3. of 6**

opn \( \in \text{DOMAIN } LL! \text{Replica}\{\text{rec.coort}\}!\text{CsState'}\cdot \text{membership Map} \)

**Reasoning (1.3.):** Defn Execute; Defn CsTo

**Step 1.4. of 6**

\( LL! \text{Replica}\{\text{rec.coort}\}!\text{CsState'}\cdot \text{membership Map}[\text{opn}] = \text{membership} \)

**Reasoning (1.4.):** This action was responsible for sending membershipMsg (Defn SendMessage), and Defn Execute constrains what message we send to match the membership in CsState'.

**Step 1.5. of 6**

opn \( \in \text{DOMAIN } \text{KnownState}\{(\text{opn} - \text{Alpha})\}\cdot \text{membership Map} \)

**Reasoning (1.5.):** Substitute into Ref:Step 1.3. second conjunct of Ref:Step 1.2.; substitute in opn-Alpha from second conjunct of Ref:Step 1.1.

**Step 1.6. of 6**

\( \text{KnownState}\{(\text{opn} - \text{Alpha})\}\cdot \text{membership Map}[\text{opn}] = \text{membership} \)

**Step 1.6.1. of 2**

\( \text{KnownState}\{\text{rec.commitMsg. opn}\}\cdot \text{membership Map}[\text{opn}] = LL! \text{Replica}\{\text{rec.coort}\}!\text{CsState'}\cdot \text{membership Map}[\text{opn}] \)

**Reasoning (1.6.1.):** Ref:Step 1.2

**Step 1.6.2. of 2**

\( \text{KnownState}\{\text{rec.commitMsg. opn}\}\cdot \text{membership Map}[\text{opn}] = \text{membership} \)

**Reasoning (1.6.2.):** Ref:Step 1.4

**Reasoning (1.6.):** substitute in opn-Alpha from second conjunct of Ref:Step 1.1.

**Reasoning (1.1.):** We've satisfied each required conjunct.

**DefaultCase 2. of 2**

**Step 2.1. of 1**

**UNCHANGED Membership As (opn , membership , LL! SentMessages)**

**Reasoning (2.1.):** Case condition

**Reasoning (2.):** induction hypothesis ; Ref:KnownStateFenExtended

**Reasoning:** Proof by case analysis