Formal Verification of Multi-Paxos for Distributed Consensus

Saksham Chand, Yanhong A. Liu, and Scott D. Stoller

Stony Brook University, Stony Brook, New York, 11794, USA
{schand, liu, stoller}@cs.stonybrook.edu

January 4, 2019

Abstract

This paper describes formal specification and verification of Lamport's Multi-Paxos algorithm for distributed consensus. The specification is written in TLA$^+$, Lamport's Temporal Logic of Actions. The proof is written and automatically checked using TLAPS, a proof system for TLA$^+$. The proof checks safety property of the specification. Building on Lamport, Merz, and Doligez's specification and proof for Basic Paxos, we aim to facilitate the understanding of Multi-Paxos and its proof by minimizing the difference from those for Basic Paxos, and to demonstrate a general way of proving other variants of Paxos and other sophisticated distributed algorithms. We also discuss our general strategies for proving properties about sets and tuples that helped the proof check succeed in significantly reduced time.

Keywords. Distributed Algorithms, Formal Methods, Verification

1 Introduction

Distributed consensus is a fundamental problem in distributed computing. It requires that a set of processes agree on some value or values. Consensus is essential when distributed services are replicated for fault-tolerance, because non-faulty replicas must agree. Examples include leader election, atomic broadcast, and state machine replication in replicated data storage services like Google File System, Apache ZooKeeper, Amazon DynamoDB, etc. Unfortunately, consensus is difficult when processes or communication channels may fail.

Paxos [19] is an important algorithm, developed by Lamport, for solving distributed consensus. Basic Paxos is for agreeing on a single value, such as whether to commit a database transaction. Multi-Paxos is for agreeing on a continuing sequence of values, for example, a stream of commands to execute. Multi-Paxos has been used in many important distributed services, for example, Google's Chubby [1, 3] and Microsoft's Autopilot [13]. There are other Paxos variants, for example, variants that reduce a message delay [22] or add preemption [20], but Multi-Paxos is the most important in making Paxos practical for distributed services that must perform a continuing sequence of operations.

Paxos handles processes that run concurrently without shared memory, where processes may crash and may later recover, and messages may be lost or delayed indefinitely. In Basic

Paxos, each process may repeatedly propose some value, and wait for appropriate replies from appropriate subsets of the processes while also replying appropriately to other processes; consensus is reached eventually if enough processes and channels are non-faulty to vote on some proposal thus agreeing on the proposed value. In Multi-Paxos, many more different attempts, proposals, and replies may happen in overlapping fashions to reach consensus on values in different slots in the continuing sequence.

Paxos has often been difficult to understand since it was created in the late 1980s [24]. Lamport later wrote a much simpler description of the phases of the algorithm but only for Basic Paxos [20]. Lamport et al. [25] wrote a formal specification and proof of Basic Paxos in TLA$^+$ [21] and TLAPS [34]. Many efforts, especially in recent years, have been spent on formal specification and verification of Multi-Paxos, but they use more restricted or less direct language models, some mixed in large systems with many unrelated functionalities, or handle other variants of Paxos than Multi-Paxos, as discussed in Section 7. What is lacking is formal specification and proof of the exact phases of Multi-Paxos, in a most direct and general language like TLA$^+$ [21], with a complete proof that is mechanically checked, and a general method for doing such specifications and proofs in a more feasible way.

This article addresses this challenge. We describe a formal specification of Multi-Paxos written in TLA$^+$, and a complete proof written and automatically checked using TLAPS. Building on Lamport et al.'s specification and proof for Basic Paxos, we aim to facilitate the understanding of Multi-Paxos and its proof by minimizing the difference from those for Basic Paxos. The key change in the specification is to replace operations involving two numbers with those involving a set of 3-tuples, for each of a set of processes, exactly capturing the minimum conceptual difference between Basic Paxos and Multi-Paxos. However, the proof becomes significantly more difficult because of the handling of sets and tuples in place of two numbers.

This work also aims to show the minimum-change approach as a general way of specifying and verifying other variants of Paxos, and more generally of specifying and verifying other sophisticated algorithms by starting from the basics. We demonstrate this by further showing the extension of the specification and proof of Multi-Paxos to add preemption—letting processes abandon proposals that are already preempted by other proposals [20, 37]. We also extended the specification and proof of Basic Paxos with preemption, which is even easier.

Finally, we discuss a general method that we followed to tackle tedious and difficult proof obligations involving sets and tuples, a well-known significant complication in general. For difficult properties involving sets, we use induction and direct the prover to focus on the changes in the set values. For properties involving tuples, we change the ways of accessing and testing the elements to yield significantly reduced proof-checking time. Overall, we were able to keep the specification minimally changed, and keep the proof-checking time to about 3 minutes for both specifications while the prover checks the proofs for 779 obligations for Multi-Paxos and over 825 obligations for Multi-Paxos with Preemption.

This article is a corrected, improved, and extended version of [2]. The main changes are as follows.

1. The claim of a complete proof in [2] was incorrect, and the problem discovered is now fixed. The problem was due to an undocumented bug [29] in TLAPS that we discovered after the work in [2], which made us realize that the proof of Multi-Paxos with Preemption was incomplete. We fixed this by adding the missing proof. This is described in the new Section 5.1.

2. The proofs are simplified, shortened by 19% for Multi-Paxos and 17% for Multi-Paxos

with Preemption, even with the added proof to overcome the TLAPS bug discovered. In fact, it was during simplification of the proof that we discovered the bug. The simplifications are described in the new Section 5.2.

3. Sections 4.1, 4.3 and 4.4 are extended to define and explain all auxiliary predicates, acceptor invariants, and message invariants, respectively, that are used in the proof. Section 6 is extended with detailed results about the different versions of proofs. Section 7 is expanded with additional related works and more details about other proofs.

4. Section 5 is extended with a summary of the overall proof, in the new Section 5.3.

5. The complete, revised, and simplified TLA$^+$ specification and TLAPS-checked proof of Multi-Paxos with Preemption are added in the new Appendix C, including about 1.5 pages of specification, 1 page of invariants, and 9 pages of proof.

The rest of the paper is organized as follows. Section 2 covers preliminaries: distributed consensus (Section 2.1), Paxos (Section 2.2), TLA$^+$ (Section 2.3), and TLA Proof System, TLAPS (Section 2.4). Section 3 presents the TLA$^+$ specification of Multi-Paxos and compares it with Lamport et al.'s specification of Basic Paxos [25]. Section 4 presents the auxiliary predicates used throughout the proof (Section 4.1), the invariants proved (Sections 4.2, 4.3 and 4.4), and an overview of the main strategy used in the proof (Section 4.5). Section 5 describes the parts added to the specification of Multi-Paxos to support Preemption, and the changes enumerated above. Section 6 summarizes the results from our specification and proof. Section 7 discusses related work and concludes. The complete, cleaned up specification, invariants, and proof can be found in Appendices A, B, and C, respectively.

2 Preliminaries

2.1 Distributed consensus

A distributed system is a set of processes that process data locally and communicate with each other by sending and receiving messages. The processes may crash and may later recover, and the messages may be delayed indefinitely or lost. The basic consensus problem, called single-value consensus, is to ensure that at most a single value is chosen from among the values proposed by the processes. Formally, it is defined as

$$Safe_{basic} \triangleq \forall v_1, v_2 \in \mathcal{V} : Chosen(v_1) \wedge Chosen(v_2) \Rightarrow v_1 = v_2 \qquad (1)$$

where $\mathcal{V}$ is the set of possible proposed values, and $Chosen$ is a predicate that given a value v evaluates to true iff v was chosen by the algorithm. The specification of $Chosen$ is part of the algorithm.

The more general consensus problem, called multi-value consensus, is to choose a sequence of values, instead of a single value. Here we have

$$Safe_{multi} \triangleq \forall v_1, v_2 \in \mathcal{V}, s \in \mathcal{S} : Chosen(s, v_1) \wedge Chosen(s, v_2) \Rightarrow v_1 = v_2 \qquad (2)$$

where $\mathcal{V}$ is as above, $\mathcal{S}$ is a set of *slots* used to index the sequence of chosen values, and $Chosen$ is a predicate that given a slot s and a value v evaluates to true iff for slot s, value v was chosen by the algorithm.

2.2 Basic Paxos and Multi-Paxos

Paxos solves the problem of consensus. Two main roles of the algorithm are performed by two kinds of processes:

- $\mathcal{P}$, the set of proposers that propose values that can be chosen.

- $\mathcal{A}$, the set of acceptors that vote for proposed values. A value is chosen when there are enough votes for it.

A set $\mathcal{Q}$ of subsets of the acceptors, that is, $\mathcal{Q} \subseteq 2^{\mathcal{A}}$, is used as a quorum system. It must satisfy the following properties:

- $\mathcal{Q}$ is a set cover for $\mathcal{A}$, that is, $\bigcup_{Q \in \mathcal{Q}} Q = \mathcal{A}$.

- Any two quorums overlap, that is, $\forall Q_1, Q_2 \in \mathcal{Q} : Q_1 \cap Q_2 \neq \emptyset$.

The most commonly used quorum system $\mathcal{Q}$ takes any majority of acceptors as an element in $\mathcal{Q}$.

Basic Paxos solves the problem of single-value consensus. It defines predicate *Chosen* as

$$Chosen(v) \quad \triangleq \quad \exists\, Q \in \mathcal{Q} : \forall\, a \in Q : \exists\, b \in \mathcal{B} : sent(\text{``2b''}, a, b, v) \tag{3}$$

where $\mathcal{B}$ is the set of proposal numbers, also called ballot numbers, which is any set that can be totally ordered. $sent(\text{``2b''}, a, b, v)$ means that a message of type 2b with ballot number b and value v was sent by acceptor a. An acceptor votes by sending such a message.

Multi-Paxos solves the problem of multi-value consensus. It extends predicate *Chosen* to decide a value for each slot s in $\mathcal{S}$:

$$Chosen(s, v) \quad \triangleq \quad \exists\, Q \in \mathcal{Q} : \forall\, a \in Q : \exists\, b \in \mathcal{B} : sent(\text{``2b''}, a, b, s, v) \tag{4}$$

To satisfy the *Safe* property, $\mathcal{S}$ can be any set. In practice, $\mathcal{S}$ is usually the natural numbers.

Figure 1 shows Lamport's description of Basic Paxos [20]. It uses any majority of acceptors as a quorum. Following Lamport et al. [25], in the specifications presented in this paper, the *prepare* requests and responses have been renamed to 1a and 1b messages, respectively, the *accept* requests and responses have been renamed to 2a and 2b messages, respectively, and the number n is renamed to b and *bal*.

Multi-Paxos can be built from Basic Paxos by carefully adding slots. In Basic Paxos, acceptors cache the value they have accepted with the highest ballot number. In Multi-Paxos, we have a sequence of these values indexed by slot.

1. Phase 1a is unchanged.

2. In Phase 1b, the acceptors now respond with a set of triples in $\mathcal{B} \times \mathcal{S} \times \mathcal{V}$ as opposed to just one ballot in $\mathcal{B}$ and one value in $\mathcal{V}$.

3. In Phase 2a, the proposers now propose a set of pairs in $\mathcal{S} \times \mathcal{V}$ instead of just one value in $\mathcal{V}$. Similar to Basic Paxos, the proposer executes Phase 2a once it has received a set of responses for its 1a message from a quorum of acceptors and picks the value with highest ballot number. But this is now performed separately for each slot in the set of the triples received in the responses.

4. In Phase 2b, the acceptors now respond with a set of pairs in $\mathcal{S} \times \mathcal{V}$ as opposed to just one value in $\mathcal{V}$.

Putting the actions of the proposer and acceptor together, we see that the algorithm operates in the following two phases.

Phase 1. (a) A proposer selects a proposal number n and sends a *prepare* request with number n to a majority of acceptors.

(b) If an acceptor receives a *prepare* request with number n greater than that of any *prepare* request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.

Phase 2. (a) If the proposer receives a response to its *prepare* requests (numbered n) from a majority of acceptors, then it sends an *accept* request to each of those acceptors for a proposal numbered n with a value v, where v is the value of the highest-numbered proposal among the responses, or is any value if the responses reported no proposals.

(b) If an acceptor receives an *accept* request for a proposal numbered n, it accepts the proposal unless it has already responded to a *prepare* request having a number greater than n.

A proposer can make multiple proposals, so long as it follows the algorithm for each one. ... It is probably a good idea to abandon a proposal if some proposer has begun trying to issue a high-numbered one. Therefore, if an acceptor ignores a *prepare* or *accept* request because it has already received a *prepare* request with a higher number, then it should probably inform the propose, who should then abandon its proposal. This is a performance optimization that does not affect correctness.

To learn that a value has been chosen, a learner must find out that a proposal has been accepted by a majority of acceptors. The obvious algorithm is to have each acceptor, whenever it accepts a proposal, respond to all learners, sending them the proposal.

Figure 1: Lamport's description of Basic Paxos in English [20].

5. Learning, as described in the last part of Figure 1, is changed to consider different slots separately—a process learns that a value is chosen for a slot if a quorum of acceptors accepted it for that slot in 2b messages.

2.3 TLA$^+$

The specifications presented in this paper are written in TLA$^+$, an extension of the Temporal Logic of Actions (TLA) [28], a logic for specifying concurrent and distributed programs and reasoning about their properties. In TLA, a *state* is an instantiation of the variables of the program to values. An *action* is a relation between a current state and a new state, specifying the effect of executing a sequence of instructions. For example, the instruction $x := x + 1$ is represented in TLA and TLA$^+$ by the action $x' = x + 1$. An action is represented by a formula over unprimed and primed variables where unprimed variables refer to the values of the variables in the current state and primed variables refer to the values of the variables in the new state.

A program is specified by its actions and initial states. Formally, a program is specified

as $Spec \triangleq Init \land \Box[Next]_{vars}$ where $Init$ is a predicate that holds for initial states of the program, $Next$ is a disjunction of all the actions of the program, and $vars$ is the tuple of all the variables. The expression $[Next]_{vars}$ is true if either $Next$ is true, implying some action is true and therefore executed, or $vars$ stutters, that is, the values of the variables are same in the current and new states. $\Box$ is the temporal operator *always*.

As a simple example, consider this specification of a clock based on Lamport's logical clock [27] but on a shared memory system:

$$
\begin{array}{ll}
\text{VARIABLE } c & \\
Max(S) & \triangleq \text{CHOOSE } e \in S : \forall f \in S : e \geq f \\
Init & \triangleq c = [p \in \{0, 1\} \mapsto 0] \\
LocalEvent(p) & \triangleq c' = [c \text{ EXCEPT } ![p] = c[p] + 1] \\
ReceiveEvent(p) & \triangleq c' = [c \text{ EXCEPT } ![p] = Max(\{c[p], c[1 - p]\}) + 1] \\
Next & \triangleq \exists p \in \{0, 1\} : LocalEvent(p) \lor ReceiveEvent(p) \\
Spec & \triangleq Init \land \Box[Next]_{\langle c \rangle}
\end{array}
\tag{5}
$$

The system has two processes numbered 0 and 1. Variable c stores their current clock values as a function from process numbers to clock values. Both processes start with clock value 0, as specified in $Init$. $LocalEvent(p)$ specifies that process p has executed some local action and therefore increments its clock value. The expression $c' = [c \text{ EXCEPT } ![p] = c[p]+1]$ means that function c' is the same as function c except that $c'[p]$ is $c[p]+1$. $ReceiveEvent(p)$ specifies that process p updates its clock value to 1 greater than the higher of its and the other process' clock value. We define operator Max to obtain the highest of a set of values. CHOOSE denotes Hilbert's ϵ operator that returns some nondeterministically chosen term satisfying the body of the CHOOSE expression if it exists, otherwise an error is raised.

2.4 TLAPS

TLA$^+$ Proof System (TLAPS) is a tool that mechanically checks proofs of properties of systems specified in TLA$^+$. Proofs are written in a hierarchical style [26], and are transformed to individual proof obligations that are sent to backend theorem provers. The primary backend provers are Isabelle and Zenon, with the SMT solvers CVC3, Z3, veriT, and Yices as backups. Temporal formulas are proved using LS4, a PTL (Propsitional Temporal Logic) prover. Users can specify which prover they want to use by using its name and can specify the timeout for each obligation separately.

As an example, we present the proof of a simple type invariant about the clock specification in (5) — It is always the case that $c \in [\{0, 1\} \to \mathbb{N}]$:

$$
\begin{array}{l}
TypeOK \triangleq c \in [\{0, 1\} \to \mathbb{N}] \\
\text{THEOREM } Inv \triangleq Spec \Rightarrow \Box(TypeOK) \\
\langle 1 \rangle. \text{ USE DEF } TypeOK \\
\langle 1 \rangle 1. Init \Rightarrow TypeOK \text{ BY DEF } Init \\
\langle 1 \rangle 2. TypeOK \land [Next]_{\langle c \rangle} \Rightarrow TypeOK' \text{ BY DEF } Next, LocalEvent, ReceiveEvent \\
\langle 1 \rangle. \text{ QED BY } \langle 1 \rangle 1, \langle 1 \rangle 2, \text{PTL DEF } Spec
\end{array}
\tag{6}
$$

The proof of theorem Inv is written in a hierarchical fashion. It is proved by two steps, named $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$, and RuleINV1 by Lamport [28]. Proof steps in TLAPS are typically written as:

$$
\langle x \rangle y. \; Assertion \; \text{BY } e_1, \ldots, e_m \text{ DEF } d_1, \ldots, d_n
\tag{7}
$$

Basic Paxos	Multi-Paxos
$Phase1a(b \in \mathcal{B}) \triangleq$ $\quad \wedge \nexists\, m \in msgs \,:\, \wedge m.type = \text{``1a''}$   $\qquad\qquad\qquad\quad \wedge m.bal = b$   $\quad \wedge Send([type \mapsto \text{``1a''},$ $\qquad bal \mapsto b)$ $\quad \wedge \textsc{unchanged}\ \langle maxVBal, maxBal, maxVal \rangle$	$Phase1a(p \in \mathcal{P}) \triangleq$ $\quad \wedge \nexists\, m \in msgs \,:\, \wedge m.type = \text{``1a''}$   $\qquad\qquad\qquad\quad \wedge m.bal = pBal[p]$   $\quad \wedge Send([type \mapsto \text{``1a''}, from \mapsto p,$ $\qquad bal \mapsto pBal[p]])$ $\quad \wedge \textsc{unchanged}\ \langle pBal, aBal, aVoted \rangle$

Figure 2: Phase 1a of Basic Paxos and Multi-Paxos

which states that step number $\langle x \rangle y$ proves *Assertion* by assuming $e_1, \ldots, e_m$, and expanding the definitions of $d_1, \ldots, d_n$. For example, step $\langle 1 \rangle 1$ proves $Init \Rightarrow TypeOK$ by expanding the definition of *Init*. The QED step for $\langle 1 \rangle$ requires us to invoke a PTL prover because *Inv* is a temporal formula.

3   Specification of Multi-Paxos

We give a formal specification of Multi-Paxos by minimally extending that of Basic Paxos by Lamport et al [25].

Variables.   The specification of Multi-Paxos has four global variables.

msgs: the set of messages that have been sent. Processes read from or add to this set. This is the same as in the specification of Basic Paxos except that the contents of messages are more complex.

pBal: per proposer, the current ballot number of the proposer. This is not in the specification of Basic Paxos; it is added to support preemption.

aBal: per acceptor, the highest ballot number seen by the acceptor. This is named *maxBal* in the specification of Basic Paxos.

aVoted: per acceptor, a set of triples in $\mathcal{B} \times \mathcal{S} \times \mathcal{V}$ voted by the acceptor. For each slot only the triple with the highest ballot number is stored. This contrasts with two numbers per acceptor, in two variables, *maxVBal* and *maxVal*, in the specification of Basic Paxos.

Algorithm steps.   The algorithm consists of repeatedly executing two phases.

Phase 1a.   Figure 2 shows the specifications of Phase 1a for Basic Paxos and Multi-Paxos, which are in essence the same. Parameter ballot number b in Basic Paxos is replaced with proposer p executing this phase in Multi-Paxos, to allow extensions such as Preemption that need to know the proposer of a ballot number; uses of b are changed to $pBal[p]$; and $from \mapsto p$ is added in *Send*. *Send* is a macro that adds its argument to *msgs*, i.e., $Send(m) \triangleq msgs' = msgs \cup \{m\}$. In this specification, 1a messages do not have a receiver, making them accessible to all processes. However, this is not required. For safety, it is enough to send this message to *any* subset of $\mathcal{A}$, even $\emptyset$. For liveness, the receiving set should contain at least one quorum.

Basic Paxos	Multi-Paxos
$Phase1b(a \in \mathcal{A}) \quad \triangleq$	$Phase1b(a \in \mathcal{A}) \quad \triangleq$
$\exists\, m \in msgs\ :$	$\exists\, m \in msgs\ :$
$\quad \wedge m.type = \text{“1a”}$	$\quad \wedge m.type = \text{“1a”}$
$\quad \wedge m.bal > maxBal[a]$	$\quad \wedge m.bal > aBal[a]$
$\quad \wedge Send([type \mapsto \text{“1b”},$	$\quad \wedge Send([type \mapsto \text{“1b”},$
$\qquad acc \mapsto a,$	$\qquad from \mapsto a,$
$\qquad bal \mapsto m.bal,$	$\qquad bal \mapsto m.bal,$
$\qquad maxVBal \mapsto maxVBal[a],$	$\qquad voted \mapsto aVoted[a]])$
$\qquad maxVal \mapsto maxVal[a]])$	
$\quad \wedge maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal]$	$\quad \wedge aBal' = [aBal \text{ EXCEPT } ![a] = m.bal]$
$\quad \wedge \text{UNCHANGED } \langle maxVBal, maxVal \rangle$	$\quad \wedge \text{UNCHANGED } \langle pBal, aVoted \rangle$

Figure 3: Phase 1b of Basic Paxos and Multi-Paxos

Phase 1b. Figure 3 shows the specifications of Phase 1b. Parameter acceptor a executes this phase. The only key difference between the specifications is the set $aVoted[a]$ of triples in $Send$ of Multi-Paxos vs. the two numbers $maxVBal[a]$ and $maxVal[a]$ in Basic Paxos.

Phase 2a. Figure 4 shows Phase 2a. The key difference is, in $Send$, the bloating of a single value v in $\mathcal{V}$ in Basic Paxos to a set of pairs in $\mathcal{S} \times \mathcal{V}$ given by $PropSV$ in Multi-Paxos. A proposal is a $\langle s, v \rangle$ pair. The operation of finding the value with the highest ballot in Basic Paxos is performed for each slot by $MaxSV$ in Multi-Paxos; $MaxSV$ takes a set T of triples in $\mathcal{B} \times \mathcal{S} \times \mathcal{V}$ and returns a set of pairs in $\mathcal{S} \times \mathcal{V}$. $NewSV$ generates a set of pairs in $\mathcal{S} \times \mathcal{V}$ where values are proposed for slots not in $MaxSV$. This is significantly more sophisticated than running Basic Paxos for each slot, because the ballots are shared and changing for all slots, and slots are paired with values dynamically where slots that failed to reach consensus values earlier are also detected and reused.

Phase 2b. Figure 5 shows Phase 2b. In Basic Paxos, the acceptor replies with the value received in the 2a message whereas in Multi-Paxos, it replies with a set of pairs in $\mathcal{S} \times \mathcal{V}$ received in the 2a message. Also, in Basic Paxos, the acceptor updates its voted pair $maxVBal[a]$ and $maxVal[a]$ upon receipt of a 2a message of the highest ballot; in Multi-Paxos, this is performed for each slot. The acceptor updates $aVoted$ to have all proposals in the received 2a message and all previous values in $aVoted$ for slots not mentioned in that message.

Complete algorithm specification. To complete the algorithm specification, we define *vars*, *Init*, *Next*, and *Spec*, typical TLA$^+$ macro names for the set of variables, the initial state, possible actions leading to the next state, and the system specification, respectively:

$$
\begin{aligned}
vars\ &\triangleq\ \langle msgs, pBal, aBal, aVoted \rangle \\
Init\ &\triangleq\ msgs = \emptyset \wedge pBal = [p \in \mathcal{P} \mapsto 0] \wedge aBal = [a \in \mathcal{A} \mapsto -1] \wedge aVoted = [a \in \mathcal{A} \mapsto \emptyset] \\
Next\ &\triangleq\ (\exists\, p \in \mathcal{P} : Phase1a(p) \vee Phase2a(p)) \vee (\exists\, a \in \mathcal{A} : Phase1b(a) \vee Phase2b(a)) \\
Spec\ &\triangleq\ Init \wedge \Box[Next]_{vars}
\end{aligned}
$$

$$\tag{8}$$

Basic Paxos	Multi-Paxos
$Phase2a(b \in \mathcal{B}) \triangleq$	$Phase2a(p \in \mathcal{P}) \triangleq$
$\wedge \nexists m \in msgs : \wedge m.type = $ "2a"	$\wedge \nexists m \in msgs : \wedge m.type = $ "2a"
$\qquad\qquad \wedge m.bal = b$	$\qquad\qquad\quad \wedge m.bal = pBal[p]$
$\wedge \exists v \in \mathcal{V} :$	
$\quad \wedge \exists Q \in \mathcal{Q}, S \subseteq \{m \in msgs : m.type = $ "1b" $\wedge$	$\wedge \exists Q \in \mathcal{Q}, S \subseteq \{m \in msgs : m.type = $ "1b" $\wedge$
$\quad m.bal = b\} :$	$\quad m.bal = pBal[p]\} :$
$\quad \wedge \forall a \in Q : \exists m \in S : m.acc = a$	$\quad \wedge \forall a \in Q : \exists m \in S : m.from = a$
$\quad \wedge \vee \forall m \in S : m.maxVBal = -1$	$\quad \wedge Send([type \mapsto $ "2a",
$\quad\quad \vee \exists c \in 0..(b-1) :$	$\quad\quad from \mapsto p,$
$\quad\quad\quad \wedge \forall m \in S : m.maxVBal \leq c$	$\quad\quad bal \mapsto pBal[p],$
$\quad\quad\quad \wedge \exists m \in S : (m.maxVBal = c)$	$\quad\quad propSV \mapsto PropSV(\text{UNION}$
$\quad\quad\quad\quad \wedge m.maxVal = v$	$\quad\quad\quad \{m.voted : m \in S\})])$
$\quad \wedge Send([type \mapsto $ "2a"$, bal \mapsto b, val \mapsto v])$	
$\wedge \text{UNCHANGED } \langle maxBal, maxVBal, maxVal \rangle$	$\wedge \text{UNCHANGED } \langle pBal, aBal, aVoted \rangle$
	$MaxBSV(T) \triangleq \{t \in T : \forall t2 \in T :$
	$\quad t2.slot = t.slot \Rightarrow t2.bal \leq t.bal\}$
	$MaxSV(T) \triangleq \{[slot \mapsto t.slot,$
	$\quad val \mapsto t.val] : t \in MaxBSV(T)\}$
	$UnusedS(T) \triangleq \{s \in \mathcal{S} : \nexists t \in T : t.slot = s\}$
	$NewSV(T) \triangleq \text{CHOOSE } D \subseteq [slot :$
	$\quad UnusedS(T), val : \mathcal{V}] : \forall d1, d2 \in D :$
	$\quad\quad d1.slot = d2.slot \Rightarrow d1 = d2 \wedge D \neq \emptyset$
	$PropSV(T) \triangleq MaxSV(T) \cup NewSV(T)$

Figure 4: Phase 2a of Basic Paxos and Multi-Paxos

Basic Paxos	Multi-Paxos
$Phase2b(a \in \mathcal{A}) \triangleq$	$Phase2b(a \in \mathcal{A}) \triangleq$
$\exists m \in msgs :$	$\exists m \in msgs :$
$\quad \wedge m.type = $ "2a"	$\quad \wedge m.type = $ "2a"
$\quad \wedge m.bal \geq maxBal[a]$	$\quad \wedge m.bal \geq aBal[a]$
$\quad \wedge Send([type \mapsto $ "2b",	$\quad \wedge Send([type \mapsto $ "2b",
$\quad\quad acc \mapsto a,$	$\quad\quad from \mapsto a,$
$\quad\quad bal \mapsto m.bal,$	$\quad\quad bal \mapsto m.bal,$
$\quad\quad val \mapsto m.val])$	$\quad\quad propSV \mapsto m.propSV])$
$\quad \wedge maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal]$	$\quad \wedge aBal' = [aBal \text{ EXCEPT } ![a] = m.bal]$
$\quad \wedge maxVBal' =$	$\quad \wedge aVoted' = [aVoted \text{ EXCEPT } ![a] =$
$\quad\quad [maxVBal \text{ EXCEPT } ![a] = m.bal]$	$\quad\quad \cup \{[bal \mapsto m.bal, slot \mapsto d.slot,$
$\quad \wedge maxVal' =$	$\quad\quad\quad val \mapsto d.val] : d \in m.propSV\}]$
$\quad\quad [maxVal \text{ EXCEPT } ![a] = m.val]$	$\quad\quad \cup \{e \in aVoted[a] :$
	$\quad\quad\quad \nexists r \in m.propSV : e.slot = r.slot\}$
	$\quad \wedge \text{UNCHANGED } \langle pBal \rangle$

Figure 5: Phase 2b of Basic Paxos and Multi-Paxos

The complete specification of Multi-Paxos with Preemption is given in Appendix A. Preemp-

tion is discussed in Section 5. We only provide specification of Multi-Paxos with Preemption because it is an extension of Multi-Paxos, thus also giving specification of Multi-Paxos would be redundant.

4   Verification of Multi-Paxos and Proof Strategy

We first define the auxiliary predicates and invariants used, by extending those for the proof of Basic Paxos with slots, and then describe our proof strategy, which proves *Safe* of Multi-Paxos.

4.1   Auxiliary predicates

These predicates are used throughout the proof. $Chosen(s, v)$ is true if there exists some ballot b such that $ChosenIn(b, s, v)$ holds.

$$Chosen(s \in \mathcal{S}, v \in \mathcal{V}) \;\triangleq\; \exists\, b \in \mathcal{B} : ChosenIn(b, s, v) \tag{9}$$

$ChosenIn(b, s, v)$ is true if there exists some quorum of acceptors such that for each acceptor in the quorum, $VotedForIn(a, b, s, v)$ holds.

$$ChosenIn(b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V}) \;\triangleq\; \exists\, Q \in \mathcal{Q} : \forall\, a \in Q : VotedForIn(a, b, s, v) \tag{10}$$

$VotedForIn(a, b, s, v)$ is true if acceptor a has voted value v for slot s in ballot b. This is realized in the algorithm by sending a 2b message with ballot b and pair $\langle s, v \rangle$ in the message's proposals. Putting everything together, the algorithm chooses value v for slot s if there exist some quorum Q and ballot b such that every acceptor in Q has voted value v for slot s in ballot b:

$$VotedForIn(a \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V}) \;\triangleq\; \exists\, m \in msgs :$$
$$m.type = \text{``2b''} \wedge m.from = a \wedge m.bal = b \wedge \exists\, d \in m.propSV \,:\, d.slot = s \wedge d.val = v \tag{11}$$

Predicate $SafeAt(b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V})$ means that no value except perhaps v has been or will be chosen in any ballot lower than b for slot s. This is realized by asserting that for each ballot $b2 < b$, there exists a quorum of acceptors such that for each acceptor in the quorum, either $VotedForIn(a, b2, s, v)$ holds or $WontVoteIn(a, b2, s)$ holds.

$$SafeAt(b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V}) \triangleq \forall\, b2 \in 0..(b-1) : \exists\, Q \in \mathcal{Q} :$$
$$\forall\, a \in Q : VotedForIn(a, v, b2, s) \vee WontVoteIn(a, b2, s) \tag{12}$$

$WontVoteIn(a, b, s)$ holds if acceptor a has seen a higher ballot than b, and did not and will not vote any value in b for slot s

$$WontVoteIn(a \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}) \triangleq aBal[a] > b \wedge \forall\, v \in \mathcal{V} : \neg VotedForIn(a, b, s, v) \tag{13}$$

Predicate $MaxBalInSlot(S \subseteq [bal : \mathcal{B}, slot : \mathcal{S}], s \in \mathcal{S})$ selects among set of elements in S with slot s, the highest ballot, or -1 if no element has slot s.

$$Max(T) \;\triangleq\; \text{CHOOSE } e \in T : \forall\, f \in T : e \geq f$$
$$MaxBalInSlot(T \subseteq [bal : \mathcal{B}, slot : \mathcal{S}], s \in \mathcal{S}) \;\triangleq\;$$
$$\text{LET } E \;\triangleq\; \{e \in T : e.slot = s\}$$
$$\text{IN} \quad \text{IF } E = \emptyset \text{ THEN } -1 \text{ ELSE } Max(\{e.bal : e \in E\}) \tag{14}$$

To prove the *Safe* property for the algorithm, we prove two properties:

1. Lemma *VotedInv*. If any acceptor votes any triple $\langle b, s, v \rangle$, then the predicate $SafeAt(b, s, v)$ holds. That is, $\forall\, a \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : VotedForIn(a, b, s, v) \Rightarrow SafeAt(b, s, v)$.

2. Lemma *VotedOnce*. If acceptor $a1$ votes triple $\langle b, s, v1 \rangle$ and acceptor $a2$ votes triple $\langle b, s, v2 \rangle$, then $v1 = v2$. That is, $VotedForIn(a1, b, s, v1) \wedge VotedForIn(a2, b, s, v2) \Rightarrow v1 = v2$.

In fact, for other consensus algorithms, either Paxos extensions like Fast Paxos [22] and Byzantine Paxos [23], or Paxos alternatives like Viewstamped Replication [30] and Raft [35], safety can also be proven by asserting these two properties.

To prove these properties, we write invariants about the algorithm. We, following Lamport et al. [25], use three kinds of invariants: type invariants, process invariants, and message invariants. These are sufficient because a distributed algorithm handles two kinds of data: messages communicated and process local data. Correspondingly, we obtain message invariants for the messages passed between processes and process invariants over the local data that these processes maintain. Type invariants ensure that the specification always uses data with correct types.

4.2 Type invariants

Type invariants are captured by *TypeOK*. They specify the sets of values that the variables of the system can hold. For example, $pBal \in [\mathcal{P} \to \mathcal{B}]$ specifies the set of values $pBal$ can hold: for any proposer p in $\mathcal{P}$, $pBal[p]$ is in $\mathcal{B}$.

$$
\begin{aligned}
Messages \;\triangleq\; & \\
& \cup\, [type \,:\, \{\text{``1a''}\}, bal \,:\, \mathcal{B}, from \,:\, \mathcal{P}] \\
& \cup\, [type \,:\, \{\text{``1b''}\}, bal \,:\, \mathcal{B}, voted \,:\, \text{SUBSET}\,[bal \,:\, \mathcal{B}, slot \,:\, \mathcal{S}, val \,:\, \mathcal{V}], from \,:\, \mathcal{A}] \\
& \cup\, [type \,:\, \{\text{``2a''}\}, bal \,:\, \mathcal{B}, propSV \,:\, \text{SUBSET}\,[slot \,:\, \mathcal{S}, val \,:\, \mathcal{V}], from \,:\, \mathcal{P}] \\
& \cup\, [type \,:\, \{\text{``2b''}\}, bal \,:\, \mathcal{B}, from \,:\, \mathcal{A}, propSV \,:\, \text{SUBSET}\,[slot \,:\, \mathcal{S}, val \,:\, \mathcal{V}]] \qquad (15) \\
& \cup\, [type \,:\, \{\text{``preempt''}\}, bal \,:\, \mathcal{B}, to \,:\, \mathcal{P}, maxBal \,:\, \mathcal{B}] \\
TypeOK \;\triangleq\; & \\
& \wedge\, msgs \subseteq Messages \wedge pBal \in [\mathcal{P} \to \mathcal{B}] \wedge aBal \in [\mathcal{A} \to \mathcal{B} \cup \{-1\}] \\
& \wedge\, aVoted \in [\mathcal{A} \to \text{SUBSET}\,[bal \,:\, \mathcal{B}, slot \,:\, \mathcal{S}, val \,:\, \mathcal{V}]]
\end{aligned}
$$

4.3 Invariants about acceptors

The following predicate specifies invariants about acceptor processes. For each acceptor a, the first conjunct establishes the initial condition. The second conjunct says that a has voted each triple in $aVoted[a]$ and $aBal[a]$ is higher than or equal to the ballot number of each triple in $aVoted[a]$. The third conjunct states that if acceptor a has voted any value v in ballot b for slot s, then there is some triple t in $aVoted[a]$ such that $t.bal \geq b$ and $t.slot = s$. The last conjunct says that acceptor a does not vote for a value in any ballot higher than the highest

it has seen per slot.

$$AccInv \triangleq \forall\, a \in \mathcal{A} :$$
$$\wedge\, (aBal[a] = -1) \Rightarrow (aVoted[a] = \emptyset)$$
$$\wedge\, \forall\, t \in aVoted[a] : aBal[a] \geq t.bal \wedge VotedForIn(a, t.bal, t.slot, t.val)$$
$$\wedge\, \forall\, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : VotedForIn(a, b, s, v) \Rightarrow \exists\, t \in aVoted[a] : t.bal \geq b \wedge t.slot = s$$
$$\wedge\, \forall\, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : b > MaxBalInSlot(aVoted[a], s) \Rightarrow \neg VotedForIn(a, b, s, v)$$
$$(16)$$

4.4 Invariants about messages

The following invariant is for a 1b message m, sent from an acceptor $m.from$. The first conjunct says that the ballot number in m is no higher than the highest ballot number seen by $m.from$. The second conjunct states that $m.from$ has voted every triple $\langle b, s, v \rangle$ in the set $m.voted$. The last conjunct asserts that for each slot s and ballot b higher than the highest ballot that $m.from$ has voted in for slot s, and lower than the ballot in m, $m.from$ has not voted any value in ballot b for slot s.

$$MsgInv1b(m) \triangleq$$
$$\wedge\, m.bal \leq aBal[m.from]$$
$$\wedge\, \forall\, t \in m.voted : VotedForIn(m.from, t.bal, t.slot, t.val) \qquad (17)$$
$$\wedge\, \forall\, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : b \in MaxBalInSlot(m.voted, s) + 1..m.bal - 1$$
$$\Rightarrow \neg VotedForIn(m.from, b, s, v)$$

The following invariant is for a 2a message m. The first conjunct establishes safety for each proposal d in m. The second conjunct says that two proposals in m with the same slot must be the same proposal. That is, for each slot, there is at most one proposal in m. The third conjunct says that there is at most one 2a message for each ballot.

$$MsgInv2a(m) \triangleq$$
$$\wedge\, \forall\, d \in m.propSV : SafeAt(m.bal, d.slot, d.val)$$
$$\wedge\, \forall\, d1, d2 \in m.propSV : d1.slot = d2.slot \Rightarrow d1 = d2 \qquad (18)$$
$$\wedge\, \forall\, m2 \in msgs : (m2.type = \text{``2a''}) \wedge (m2.bal = m.bal) \Rightarrow m2 = m$$

The following invariant is for a 2b message m sent by acceptor $m.from$. The first conjunct states that there exists a 2a message with the same ballot and same set of proposals as m. The second conjunct asserts that the ballot of m is no higher than the highest ballot seen by the sender of m.

$$MsgInv2b(m) \triangleq$$
$$\wedge\, \exists\, m2 \in msgs : m2.type = \text{``2a''} \wedge m2.bal = m.bal \wedge m2.propSV = m.propSV \qquad (19)$$
$$\wedge\, m.bal \leq aBal[m.from]$$

The complete message invariant is the conjunction of $MsgInv1b$, $MsgInv2a$, and $MsgInv2b$:

$$MsgInv \triangleq \forall\, m \in msgs : \wedge\, (m.type = \text{``1b''}) \Rightarrow MsgInv1b(m)$$
$$\wedge\, (m.type = \text{``2a''}) \Rightarrow MsgInv2a(m) \qquad (20)$$
$$\wedge\, (m.type = \text{``2b''}) \Rightarrow MsgInv2b(m)$$

The complete invariants, auxiliary operators, and the safety property to be proved can be found in Appendix B.

4.5   Proof strategy

The main theorem to prove is *Safety* as defined in Equation (21). For this, we define *Inv* and first prove $Inv \Rightarrow Safe$. Then, we prove $Spec \Rightarrow \Box Inv$ which by temporal logic, concludes $Spec \Rightarrow \Box Safe$. Note that property *Safety* is called *Consistent*, and invariant *Safe* is called *Consistency* by Lamport et al [25].

$$
\begin{aligned}
Inv &\triangleq TypeOK \wedge AccInv \wedge MsgInv \\
Safe &\triangleq \forall\, v1, v2 \in \mathcal{V}, s \in \mathcal{S} : Chosen(v1, s) \wedge Chosen(v2, s) \Rightarrow v1 = v2 \qquad (21) \\
\text{THEOREM } Safety &\triangleq Spec \Rightarrow \Box Safe
\end{aligned}
$$

The proof is developed following a standard hierarchical structure and uses proof by induction and contradiction. To prove $Spec \Rightarrow \Box Inv$, we employ a systematic proof strategy that works well for algorithms described in the event driven paradigm. Distributed algorithms are prime examples of this paradigm as they can be specified in blocks of code that are triggered upon receiving a certain set of messages and finish by sending a set of messages. We demonstrate the strategy for some invariants in *Inv*.

First, consider invariant *TypeOK*. The goal to prove is $Spec \Rightarrow \Box TypeOK$. Recall $Spec \triangleq Init \wedge \Box [Next]_{vars}$:

- The induction basis, $Init \Rightarrow TypeOK$, is trivial in this case because the set of sent messages is empty. TLAPS handles it automatically. This should be the case for almost all distributed algorithms.

- The induction step is to prove $TypeOK \wedge [Next]_{vars} \Rightarrow TypeOK'$, where the left side is the induction hypothesis, and right side is the goal to be proved. $[Next]_{vars}$ is a disjunction of phases, as for any distributed algorithm, and $TypeOK'$ is a conjunction of smaller invariants, as for many invariants.

Now, the hypothesis of the induction step can be stripped down to each disjunct of *Next* separately, and each smaller goal needs to be proved from all smaller disjuncts. This process is mechanical, and TLAPS provides a feature for precisely this expansion into smaller proof obligations. This breakdown is the first step in our proof strategy. For *TypeOK*, this expands to 4 smaller assertions; with 4 phases in *Next*, we obtain 16 small proof obligations.

Proving complex invariants using invariance lemmas and increments. *AccInv* and *MsgInv* are more involved. We proceed as we did for *TypeOK* and create a proof tree, each branch of which aims to prove an invariant for a disjunct in *Next*. To explain the rest of our strategy, we show a complex combination: *MsgInv* and Phase 1b. Equation (22) shows the skeleton of the proof. The goal is to prove $\langle 4 \rangle 2$, which states that $MsgInv'$ holds if acceptor a executes Phase 1b. $\langle 6 \rangle 1$ proves $MsgInv1b(m)'$ and $\langle 6 \rangle 2$ proves $MsgInv2a(m)'$ and $MsgInv2b(m)'$.

Phase 1b sends a 1b message. Intuitively, $\langle 6 \rangle 2$ should be easy because its goals are not invariants about 1b messages. However, this is not the case because of predicate *SafeAt*, which is used in *MsgInv2a*. At this point the prover needs an *invariance lemma*.

Invariance lemma. An invariance lemma is a lemma that asserts that a predicate continues to hold (or not hold) as the system goes from one state to the next in a single step. This happens when a step does not affect the part of system state asserted by the predicates, and requires more complex proofs otherwise. For example, the invariance lemma for *SafeAt* states that *SafeAt* continues to hold for any disjunct in *Next*, which includes Phase 1b. The

characteristic property of such lemmas is their reuse. In our proof of Multi-Paxos, we defined 5 invariance lemmas which are used in 27 places.

Lastly, we need to prove $\langle 6 \rangle 1$. Since $\langle 6 \rangle 1$ is about 1b messages, and Phase 1b generates such a message, the proof is more complicated, and the prover needs more manual intervention. Here we split the set of messages in the new state into two sets: $\langle 7 \rangle 1$ for the old messages, and $\langle 7 \rangle 2$ for the *increment*, the new message sent in this step. For the old messages, we use invariance lemmas. The most challenging case is for the increment.

Increment. An increment is a new message sent in a phase or generally a new element in a set. In the example of Equation (22), the increment is $m1$ and we focus on the cause of the increment—available here in the definition of Phase 1b—and prove each conjunct of the goal $MsgInv1b(m1)'$ separately in $\langle 5 \rangle 2, 4, 12$. The prover proves $\langle 5 \rangle 2$ by just the definition of Phase 1b. For $\langle 5 \rangle 4$, along with the definition of Phase 1b, the prover also needs invariance lemma for $VotedForIn$ for $Phase1b$. $\langle 5 \rangle 12$ requires, along with the definition of Phase 1b and invariance lemmas, case-specific manual intervention. Specifically, we helped the prover understand the change in limits of the set $MaxBalInSlot(m1.voted, s) + 1 .. m1.bal - 1$. The proof extended one more level for case $\langle 7 \rangle 2$.

$\langle 4 \rangle 2.\text{ASSUME NEW } a \in \mathcal{A}, Phase1b(a) \text{ PROVE } MsgInv'$

   $\langle 5 \rangle.\text{DEFINE } m1 \triangleq [type \mapsto \text{``1b''}, from \mapsto a, bal \mapsto m.bal, voted \mapsto aVoted[a]]$

   $\langle 5 \rangle 2.(m1.bal \leq aBal[m1.from])' \ldots$

   $\langle 5 \rangle 4.(\forall\, r \in m1.voted \,:\, VotedForIn(m1.from, r.bal, r.slot, r.val))' \ldots$

   $\langle 5 \rangle 12.(\forall\, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} \,:\, b \in MaxBalInSlot(m1.voted, s) + 1 .. m1.bal - 1 \Rightarrow$
      $\neg VotedForIn(m1.from, b, s, v))' \ldots$

$$\vdots$$

   $\langle 6 \rangle\text{ASSUME NEW } m2 \in msgs' \text{ PROVE } MsgInv'$

   $\langle 6 \rangle 1.(m2.type = \text{``1b''} \Rightarrow MsgInv1b(m2))'$

      $\langle 7 \rangle 1.\text{CASE } m2 \in msgs \ \ldots$

      $\langle 7 \rangle 2.\text{CASE } m2 \in msgs' \setminus msgs \ \ldots$

   $\langle 6 \rangle 2.((m2.type = \text{``2a''} \Rightarrow MsgInv2a(m2)) \wedge (m2.type = \text{``2b''} \Rightarrow MsgInv2b(m2)))' \ldots$

$$(22)$$

Induction for properties over sets, and ways of accessing elements of tuples. With the above strategy, we were still faced with certain assertions that were difficult to prove. One of the main difficulties lay in proving properties about tuples and sets of tuples for each of a set of processes in Multi-Paxos, as opposed to one or two values for each of a set of processes in Basic Paxos. It may appear that, in many places, this requires simply adding an extra parameter for the slot, but the proof became significantly more difficult: even in places where an explicit inductive proof is not needed, auxiliary facts had to be added to help TLAPS succeed or proceed sufficiently fast.

For example, adding slots to the proof of theorem *Safety* for Basic Paxos caused the prover to take about 90 seconds to check it. To aid the proof, we added $\exists\, a \in \mathcal{A} :$ $VotedForIn(a, b1, s, v1) \wedge VotedForIn(a, b1, s, v2)$ as an intermediate fact derivable from $b1 = b2 \wedge ChosenIn(b1, s, v1) \wedge ChosenIn(b2, s, v2)$ (step $\langle 3 \rangle 1$ of step $\langle 2 \rangle 1$ in the proof of theorem *Safety*). Following this, the prover asserted the conclusion $v_1 = v_2$ in a few milliseconds, a

10,000 time speedup.

Tuples have only a fixed number of components and therefore do not require separate inductive proofs, but they often turn out to be tricky and require special care in choosing the ways to access and test their elements, to sufficiently reduce TLAPS's proof-checking time and observe progress. For example, consider the definition of *VotedForIn* in Equation (11). Originally $[slot \mapsto s, val \mapsto v] \in m.propSV$ was written for the last conjunct with existential quantification, because it was natural, but it had to be changed to $\exists\, d \in m.propSV \,:\, d.slot = s \wedge d.val = v$, because the prover made more observable progress. With the original version, the proof did not carry through after 1 or 2 minutes. After the change, the proof proceeded quickly. One minute of waiting for such simple, small tests felt very long, making it uncertain whether the proof would carry through.

5   Multi-Paxos with Preemption, and overall proof

Preemption is described informally in Lamport's description of Basic Paxos in Figure 1, in the paragraph about abandoning a proposal number. Preemption has an acceptor reply to a proposer, in both Phases 1b and 2b, if the proposer's ballot is preempted i.e., the acceptor has seen a higher ballot than the one just received from the proposer. This reply informs the proposer of the highest ballot the acceptor has seen, and the proposer can abandon its lower ballot number.

$NewBal(b2 \in \mathcal{B}) \triangleq$ CHOOSE $b \in \mathcal{B} : b > b2 \wedge \nexists\, m \in msgs : m.type = $ "1a" $\wedge m.bal = b$ $Preempt(p \in \mathcal{P}) \triangleq \exists\, m \in msgs :$ $\quad\quad\quad \wedge m.type = $ "preempt" $\quad\quad\quad \wedge m.to = p$ $\quad\quad\quad \wedge m.bal > pBal[p]$ $\quad\quad\quad \wedge pBal' = [pBal \text{ EXCEPT } ![p] = NewBal(m.bal)]$ $\quad\quad\quad \wedge$ UNCHANGED $\langle msgs, aBal, aVoted \rangle$

Phase 1b without Preemption	Phase 1b with Preemption
$Phase1b(a \in \mathcal{A}) \quad\triangleq$ $\exists\, m \in msgs :$ $\quad \wedge m.type = $ "1a" $\quad \wedge m.bal > aBal[a]$ $\quad \wedge Send([type \mapsto $ "1b", $\quad\quad from \mapsto a,$ $\quad\quad bal \mapsto m.bal,$ $\quad\quad voted \mapsto aVoted[a]])$ $\quad \wedge aBal' = [aBal \text{ EXCEPT } ![a] = m.bal]$ $\quad \wedge$ UNCHANGED $\langle pBal, aVoted \rangle$	$Phase1b(a \in \mathcal{A}) \quad\triangleq$ $\exists\, m \in msgs :$ $\quad \wedge m.type = $ "1a" $\quad \wedge$ IF $m.bal > aBal[a]$ THEN $\quad\quad \wedge Send([type \mapsto $ "1b", $\quad\quad\quad from \mapsto a,$ $\quad\quad\quad bal \mapsto m.bal,$ $\quad\quad\quad voted \mapsto aVoted[a]])$ $\quad\quad \wedge aBal' = [aBal \text{ EXCEPT } ![a] = m.bal]$ $\quad\quad \wedge$ UNCHANGED $\langle pBal, aVoted \rangle$ $\quad$ ELSE $\quad\quad \wedge Send([type \mapsto $ "preempt", $\quad\quad\quad to \mapsto m.from, bal \mapsto aBal[a]])$ $\quad\quad \wedge$ UNCHANGED $\langle pBal, aBal, aVoted \rangle$

Figure 6: Extension of Multi-Paxos to Multi-Paxos with Preemption

To specify Preemption, each of Phases 1b and 2b adds a new case for when the acceptor receives a lower ballot than the highest ballot it has seen. We also define predicate *Preempt*

15

that specifies how proposers update *pBal* upon receiving a preempt message. Figure 6 shows Phase 1b with and without the modifications to add Preemption. Modifications to Phase 2b are similar.

Preemption adds a new phase, *Preempt*, as a passive action in *Next*, modifies definitions of existing phases, and adds a new type of message. This new phase increases the width of the proof tree. Except for *TypeOK*, this new branch of the proof was proven by asserting invariance lemmas established earlier. The entire task of adding the new parts of specification and proof, except for the proof of *TypeOK*, took less than an hour.

5.1   Remedial proof for TypeOK

The invariance of *TypeOK*, i.e., $\Box\,TypeOK$, was automatically checked previously [2] by instructing TLAPS to use its PTL (Propositional Temporal Logic) backend prover. However, we later discovered that due to an undocumented bug in TLAPS [29], using PTL in this way causes incorrect obligations to be marked correct. This bug lets one *prove* that a primed formula is true if its unprimed form is an assumption. A primed formula is one with only primed variables and constants. For example, $x' = 0$ is primed but $x' = x + 1$ and $x = 0$ are not. The unprimed form of $x' = 0$ is $x = 0$. Thus, TLAPS incorrectly verifies $TypeOK'$ under the assumption $TypeOK \wedge [Next]_{vars}$.

As a remedy, we here provide the missing proof of invariance of *TypeOK*. Using our proof strategy, writing this proof of invariance took a matter of minutes for Multi-Paxos. The remedial proof of *TypeOK* is longer by 39 lines (72%), increasing from 54 for the previous incomplete proof [2] to 93 , and took 3 seconds (30%) more for TLAPS to check, from 10 to 13 seconds here.

For Multi-Paxos with Preemption, our strategy failed at first because the invariants were not strong enough. Preemption adds the conjunct $pBal \in [\mathcal{P} \to \mathcal{B}]$ to *TypeOK*. Upon preemption, proposer p changes $pBal[p]$ to ballot b such that (i) no 1a message has been sent yet with b as ballot, and (ii) b is higher than the ballot of the preempting message. To prove that $\Box pBal \in [\mathcal{P} \to \mathcal{B}]$, we need to establish that such a b indeed exists.

To this end, we strengthen our invariants by adding the fact that *msgs* is always a finite set, i.e., *IsFiniteSet(msgs)*. We add this as a conjunct in *TypeOK*. Now, it can be proven that only a finite number of 1a messages exist. Thus, only a finite number of ballots can ever be used. Because $\mathcal{B}$ is infinite, there will always be some ballot available to be chosen. We prove this constructively by providing the prover with a witness: a ballot that is 1 greater than the highest ballot in 1a messages. This remedial proof of *TypeOK* is 63 lines (84%) longer, increasing from 75 for the previous incomplete proof [2] to 138, and took 29 seconds (242%) more for TLAPS to check, from 12 to 41 seconds here.

Table 1 summarizes the results.

5.2   Overall proof improvement

To make the proof easier to read and understand, we made three main kinds of improvements. These improvements apply to the proofs for both Multi-Paxos and Multi-Paxos with Preemption, but the numbers presented are for the proof of Multi-Paxos with Preemption.

1. **Refined invariants.** *MsgInv* is now defined as a conjunction of three new predicates *MsgInv1b*, *MsgInv2a*, and *MsgInv2b*, in (17)-(19), which were not separate predicates in the proof for [2], even though the names were used in the text for ease of presentation. This introduction of intermediate predicates reduced the proof size by 75 lines, because,

at 12 places, the new names are used in place of their expanded definitions. This also reduced the proof-checking time by about 2 seconds, because now only the portions of *MsgInv* that one needed for examination in the proof are instructed to be expanded.

2. **Merged proof cases.** Cases with similar proofs are merged into a single proof. For example, to prove that *AccInv* continues to hold when acceptor a executes Phase 1b with Preemption, there are three cases to consider for each acceptor $a2$ in $\mathcal{A}$: (1) if $a2$ is not a, (2) if $a2$ is a, and it sends a preempt message in this action, and (3) if $a2$ is a, and it sends a 1b message in this action. The proofs for cases (1) and (2) are similar because in both cases, $a2$'s state does not change. We found 5 instances in the proof described in [2] where such cases were handled separately, creating unnecessarily longer proof. Merging them resulted in an overall reduction of 27 lines of proof and slightly reduced (by less than 2 seconds) proof-checking times.

3. **Removed unnecessary obligations.** When TLAPS fails to prove an assertion, a proof must be manually written to be checked automatically by TLAPS. We discovered that parts of these manually written proofs described in [2] are unnecessary because they can be found automatically by TLAPS. About 25 instances of these were removed causing 110 lines of proof to be removed and the proof-checking times to reduce by around 20 seconds. An additional 22 instances were removed from the remedial proof for *TypeOK*, totalling to 61 lines of proof and reducing its checking time by 5 seconds.

Metric	Multi-Paxos			Multi- w/ Preemption		
	Old [2]	Remedial Proof	Improved Remedial Proof	Old [2]	Remedial Proof	Improved Remedial Proof
Proof size	54^^	93	54	75^^	138	77
CPU check time (seconds)	10^^	13	4	12^^	41	12
Elapsed check time (seconds)	20^^	22	8	21^^	35	25

Table 1: Proof statistics for remedial proof of TypeOK and improvements on it. Proof size is measured as non-empty lines excluding comments.
^^ indicates an incorrect number from the incomplete proof due to the TLAPS bug [29].

5.3   Overall proof summary

The proof for Multi-Paxos with Preemption spans about 9 pages, and is summarized below:

1. Auxiliary predicates and invariants span about 1 page.

2. Helper lemmas and their proofs are about 1.5 pages. They are used to prove helpful properties of operators *MaxBalInSlot*, *NewSV*, and *MaxSV*. This also includes proofs for lemmas *VotedInv* and *VotedOnce*.

3. The invariance lemmas and their proofs are less than 1 page.

4. The proof of type invariant *TypeOK* is 1 page, using only a 1-level proof for each action.

5. The proof of acceptor invariant *AccInv* is 1.5 pages, using a 5-level proof for action *Phase2b* because only *Phase2b* changes variable *aVoted* which affects *AccInv*. The proofs for other actions are only 1-level because of their independence from *AccInv*.

6. The proof of message invariant *MsgInv* is a little over 4 pages, using 4-level proofs for actions *Phase1b* and *Phase2b* taking about 1 page each, and a 6-level proof for *Phase2a* taking 2 pages, because *MsgInv* is over messages sent in these actions. The proof is long and complex in particular because each message is a record and its contents may contain sets of records.

7. The proof of theorem *Safety* using $Spec \Rightarrow \Box Inv$ is less than half a page with a straight-forward argument of $Inv \Rightarrow Safe$.

The complete TLAPS-checked proof is given in Appendix C.

6 Results of TLAPS-checked proofs

Table 2 summarizes the results from our specification and proof. Some mistakes in the counting logic of the result generating script caused incorrect numbers to be reported in [2] for specification and proof sizes and number of proofs by contradiction. These errors have been corrected here.

- The specification size grew by only 3 lines (6%), from 52 lines for Basic Paxos to 55 lines for Multi-Paxos; another 19 lines (35%) are added for Preemption.

- Overall, the proof size increased significantly by 477 lines (154%), from 310 for Basic Paxos to 787 for Multi-Paxos, due to the complex interaction between slots and ballots; only 44 more lines (6%) were added for Preemption, thanks to the reuse of all lemmas, especially invariance lemmas. As mentioned in Section 5.1, adding the remedial proof to the incomplete proof reported in [2] initially increased the proof size by 39 lines (4%) from 1010 to 1049 for Multi-Paxos and by 63 lines (6%) from 1054 to 1117 for Multi-Paxos with Preemption. However, the proof improvements decreased these numbers by 262 lines (25%) from 1049 to 787 for Multi-Paxos and by 286 lines (26%) from 1117 to 831 for Multi-Paxos with Preemption.

- The maximum level of proof tree nodes increased from 7 to 10 going from Basic Paxos to Multi-Paxos but remained 10 after adding Preemption; this contrast is even stronger for the maximum degree of proof tree nodes, consistent with the challenge of going to Multi-Paxos. The proof improvements reduced the maximum level of proof tree nodes by 1 for both Multi-Paxos and Multi-Paxos with Preemption compared with [2].

- The increase in number of lemmas is due to the change from *Max* in Basic Paxos to *MaxBalInSlot* in Multi-Paxos, defined in Equation (14). Five lemmas were needed for this predicate alone to aid the prover, as we moved from scalar values to a set of tuples for each acceptor.

- No proof by induction on set increment is used for Basic Paxos. Four such proofs are used for Multi-Paxos and for Multi-Paxos with Preemption.

- Proof by contradiction is used twice in the proof of Basic Paxos, and we extended them with slots in the proof of Multi-Paxos and Multi-Paxos with Preemption. We use contradiction proofs one more time in our proofs in lieu of longer constructive proofs.

18

- The number of proof obligations for the prover increased most significantly, by 540 (226%), from 239 for Basic Paxos to 779 for Multi-Paxos. Only another 46 (6%) proof obligations were generated for Multi-Paxos with Preemption. However, the number of obligations decreased by 139 (15%) from 918 in [2] to 779 for Multi-Paxos and by 134 (14%), from 959 in [2] to 825 for Multi-Paxos with Preemption.

- The checking time of invariance proof of *TypeOK* increased by 3 seconds (300%) from 1 for Basic Paxos to 4 for Multi-Paxos due to the more complicated structures involved in Multi-Paxos. It further increased by 8 seconds (200%) from 4 to 12 when Preemption was added. This is due to new aspects in the remedial proof of *TypeOK* as explained in Section 5.1.

- The checking time of the total proof increased by 135 seconds (338%), from 40 for Basic Paxos to 175 for Multi-Paxos, despite our continuous efforts to help the prover reduce it. This is because of the greatly increased size and complexity of inductions, leading to significantly more obligations for the prover. A small increase of 5 seconds (3%) is observed when Preemption is added. As a result of the proof improvements, we notice a 53 second decrease (23%, from 228 to 175) and a 42 second decrease (19%, from 222 to 180) in the proof checking time for Multi-Paxos and Multi-Paxos with Preemption.

7 Related work and conclusion

We discuss closest related results on verification of Paxos, categorized by the verification techniques.

Model checking. Lamport has written TLA$^+$ specifications for Basic Paxos and its variants, e.g., Fast Paxos [22], and checked them using the TLA$^+$ model checker TLC [33], but not for Multi-Paxos or its variants; a number of MS students at our university have also done this in course projects, including for Multi-Paxos. Delano et al. [8] modeled Basic Paxos in Promela and checked it using the Spin model checker [40]. To reduce the state space, they use counting guards to track majority, reset local variables after state operations, and use sorted *send* instead of FIFO *send* (with random *receive*, to model non-FIFO channels). They checked Basic Paxos for pairs of numbers of proposers and acceptors up to (2,8), (3,5), (4,4), (5,3), and (8,2). Yabandeh et al. [44] checked a C++ implementation of Basic Paxos using CrystalBall, a tool built on Mace [16], which includes a model checker. Yang et al. [45] used their model checker MoDist to check a Multi-Paxos-based service system developed by a Microsoft product team [31]. With dynamic partial-order reduction [10], they found 13 bugs including 2 bugs in the Paxos implementation, with as few as 3 replicas and a few slots. In all cases, existing work in model checking either does not check Multi-Paxos or can check it for only a very small number of processes and slots.

Deductive verification. Kellomäki [15] formally specified and verified Basic Paxos using PVS [41]. Charron-Bost and Schiper [5] expressed Basic Paxos in the Heard-Of model, and Charron-Bost and Merz [4] verified it formally using Isabelle/HOL [42]. Drăgoi et al. [9] specified and verified a version of Basic Paxos in PSync, which is based on the Heard-Of model, so the specification and proof are similar to [5, 4]. Lamport et al. [25] give a formal specification of Basic Paxos in TLA$^+$ and a TLAPS-checked proof of its correctness. Lamport [23] wrote a TLA$^+$ specification of Byzantine Paxos, a variant of Basic Paxos that tolerates arbitrary

Metric	Basic Paxos	Multi-Paxos		Multi- w/ Preemption	
		Old [2]	New	Old [2]	New
Spec size (lines, excl. comments)	-, 52	-, 56	55	-, 75	74, 52*
Proof size (lines, excl. comments)	-, 310	-, 1010^^	787	-, 1054^^	831, 528*
Spec size incl. comments (lines)	115^, 106	133^, 123	115	158^, 151	144, 76*
Proof size incl. comments (lines)	423^, 432	1106^, 1096^^	868	1136^, 1143^^	915, 613*
Max level of proof tree nodes	7	11	10	11	10
Max degree of proof tree nodes	3	17	17	17	17
# lemmas	4	11	11	12	12
# invariance lemmas	1	5	5	6	6
# uses of invariance lemmas	8	27	27	29	29
# proofs by induction on set increment	0	4	4	4	4
# proofs by contradiction	1^, 2	1^, 3	3	1^, 3	3
# obligations in TLAPS	239	918^^	779	959^^	825
TypeOK CPU check time (seconds)	-, 1	-, 10^^	4	-, 12	12
Total CPU check time (seconds)	-, 40	-, 228^^	175	-, 222^^	180
TypeOK elapsed check time (seconds)	-, 1	-, 20^^	7	-, 22^^	24
Total elapsed check time (seconds)	24**, 14	128**, 63^^	51	94**, 69^^	90

Table 2: Summary of results. An obligation is a condition that TLAPS checks. The check time is on a 4-core Intel i7-4720HQ 2.6 GHz CPU with 16 GB of memory, running 64-bit Ubuntu 17.10 and TLAPS 1.5.6.

- denotes an entry not in [2], followed by the now added measurement.

^ indicates a number with a count oversight in [2], followed by the now correct count.

^^ indicates an incorrect number from the incomplete proof due to the TLAPS bug [29].

** indicates a number that used TLAPS version 1.5.3 in [2], followed by the number using the new version 1.5.6.

* indicates a number for the specification or proof in Appendix C, after removing unnecessary line breaks from default latex generated by TLA+ Tools.

failures, and a TLAPS-checked proof that it refines Basic Paxos. Küfner et al. [18] exhibit a methodology to develop machine-checkable parameterized proofs of correctness of fault-tolerant round-based distributed algorithms with Basic Paxos as a case study. Their proof is approximately 10,000 lines in Isabelle/HOL.

With IronFleet, Hawblitzel et al. [11] verified a state machine replication system that uses Multi-Paxos at its core. Their specification mimics TLA$^+$ models but is written in Dafny [38], which has no direct concurrency support but has more automated proof support than TLAPS. This work is superior to its peers by proving not only safety but also liveness properties. However, it is a complex system, with 3 levels and many components of specifications, over 1000 lines, and proofs, over 30,000 lines. Schiper et al. [39] used EventML [36] to specify Multi-Paxos and used NuPRL [7] to verify safety. Using the Verdi framework, Wilcox et al. [43] expressed Raft [35], an algorithm similar to Multi-Paxos, in OCAML and verified it using Coq [12]. The proof is over 50,000 lines and takes almost 30 minutes to verify. Padon et al. [32] specify many variants of Paxos including Basic and Multi-Paxos in first-order logic. They present a methodology aiming at automatic verification based on effectively propositional logic (EPR).

All these works either do not handle Multi-Paxos or handle it using more restricted or less direct language models than TLA$^+$, some mixed in large systems, making the essence of the algorithm's proof harder to find and understand.

In contrast, our work is the first to specify the exact phases of Multi-Paxos in a most direct and general language model, TLA$^+$, with a complete correctness proof automatically checked using TLAPS. Building on Lamport et al.'s specification and proof for Basic Paxos [25], we aim to facilitate the understanding of Multi-Paxos and its proof by minimizing the difference from those for basic Paxos. We further show this as a general way for specifying and proving variants of Multi-Paxos, by doing so for Multi-Paxos extended with preemption. We also discuss the significantly more complex but necessary subproofs by induction. Future work may automate inductive proofs and support the verification of variants that improve and extend Multi-Paxos, by extending specifications of variants of Paxos, e.g., Fast Paxos [22] and Byzantine Paxos [23], to Multi-Paxos and verifying these variants of Multi-Paxos as well as Raft [35].

References

[1] Burrows, M.: The Chubby lock service for loosely-coupled distributed systems. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation. pp. 335–350. USENIX Association (2006)

[2] Chand, S., Liu Y.A., Stoller, S.D.: Formal verification of multi-Paxos for distributed consensus. In: Proceedings of the 21st International Symposium on FormalMethods, pp. 119–136. Springer (2016)

[3] Chandra, T.D., Griesemer, R., Redstone, J.: Paxos made live—An engineering perspective. In: Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing. pp. 398–407 (2007)

[4] Charron-Bost, B., Merz, S.: Formal verification of a consensus algorithm in the Heard-Of model. International Journal of Software and Informatics 3(2-3), 273–303 (2009)

[5] Charron-Bost, B., Schiper, A.: The Heard-Of model: Computing in distributed systems with benign faults. Distributed Computing 22(1), 49–71 (2009)

[6] Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)

[7] Constable, R.L., Allen, S.F., Bromley, H.M., Cleaveland, W.R., Cremer, J.F., Harper, R.W., Howe, D.J., Knoblock, T.B., Mendler, N.P., Panangaden, P., Sasaki, J.T., Smith, S.F.: Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall (1986)

[8] Delzanno, G., Tatarek, M., Traverso, R.: Model checking Paxos in Spin. In: Proceedings of the 5th International Symposium on Games, Automata, Logics and Formal Verification. pp. 131–146 (2014)

[9] Drăgoi, C., Henzinger, T.A., Zufferey, D.: Psync: A partially synchronous language for fault-tolerant distributed algorithms. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 400–415 (2016)

[10] Flanagan, C., Godefroid, P.: Dynamic partial-order reduction for model checking software. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 110–121 (2005)

[11] Hawblitzel, C., Howell, J., Kapritsos, M., Lorch, J.R., Parno, B., Roberts, M.L., Setty, S., Zill, B.: IronFleet: Proving practical distributed systems correct. In: Proceedings of the 25th Symposium on Operating Systems Principles. pp. 1–17 (2015)

[12] INRIA: The Coq Proof Assistant. http://coq.inria.fr/ (Last released January 2016)

[13] Isard, M.: Autopilot: Automatic data center management. ACM SIGOPS Operating Systems Review 41(2), 60–67 (2007)

[14] Jamshidi, M., Betancourt, A. J., Gomez, J.: Cyber-physical control of unmanned aerial vehicles. Scientia Iranica 18(3). pp. 663–668. (2011)

[15] Kellomäki, P.: An annotated specification of the consensus protocol of Paxos using superposition in PVS. Report 36, Institute of Software Systems, Tampere University of Technology (2004)

[16] Killian, C.E., Anderson, J.W., Braud, R., Jhala, R., Vahdat, A.M.: Mace: language support for building distributed systems. In: Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 179–188 (2007)

[17] Kirsch, J., Goose, S., Amir, Y., Wei, D., Skare, P.: Survivable SCADA via intrusion-tolerant replication. In: IEEE Transactions on Smart Grid 5(1). pp. 60–70. (2014)

[18] Küfner P., Nestmann U., Rickmann C.: Formal verification of distributed algorithms. In: Theoretical Computer Science. pp. 209-224 (2012)

[19] Lamport, L.: The part-time parliament. ACM Transactions on Computer Systems 16(2), 133–169 (1998)

[20] Lamport, L.: Paxos made simple. SIGACT News (Distributed Computing Column) 32(4), 51–58 (2001)

[21] Lamport, L.: Specifying Systems: The TLA$^+$ Language and Tools for Hardware and Software Engineers. Addison-Wesley (2002)

[22] Lamport, L.: Fast Paxos. Distributed Computing 19(2), 79–103 (2006)

[23] Lamport, L.: Byzantizing Paxos by refinement. In: Proceedings of the 25th International Symposium on Distributed Computing. pp. 211–224. Springer (2011)

[24] Lamport, L.: My writings. http://research.microsoft.com/en-us/um/people/lamport/pubs/p (Accessed January 24, 2016), Lamport's history of paper [19]

[25] Lamport, L., Merz, S., Doligez, D.: A TLA$^+$ specification of the Paxos Consensus algorithm described in Paxos Made Simple and a TLAPS-checked proof of its correctness. https://github.com/tlaplus/v1-tlapm/blob/master/examples/paxos/Paxos.tla (Last modified Fri Nov 28 10:39:17 PST 2014 by Lamport. Accessed Feb 6, 2018)

[26] Lamport, L.: How to write a 21st century proof. In: Journal of Fixed Point Theory and Applications (JFPTA) 11(1). pp. 43–63. Springer (2012)

[27] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. In: Communications of the ACM 21(7). pp. 558–565. (1978)

[28] Lamport, L.: The temporal logic of actions. In: ACM Transactions on Programming Languages and Systems (TOPLAS) 16(3). pp. 872–923. (1994)

[29] Merz, S.: Coalescing bug in TLAPS. https://github.com/tlaplus/tlaplus/issues/40. Accessed March 16, 2018.

[30] Liskov B.: From viewstamped replication to byzantine fault tolerance. In: Replication. pp. 121–149. Springer Berlin Heidelberg (2010)

[31] Liu, X., Guo, Z., Wang, X., Chen, F., Lian, X., Tang, J., Wu, M., Kaashoek, M.F., Zhang, Z.: D3S: Debugging deployed distributed systems. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation. pp. 423–437. USENIX Association (2008)

[32] Padon, O., Losa, G., Sagiv, M., Shoham, S.: Paxos made EPR: Decidable Reasoning about Distributed Protocols. In: Proceedings of the ACM on Programming Languages, 1(OOPSLA). pp. 108. (2017)

[33] Microsoft Research: The TLA Toolbox. http://research.microsoft.com/en-us/um/people/lam (Last modified January 4, 2016)

[34] Microsoft Research-Inria Joint Center: TLA$^+$ Proof System (TLAPS). http://tla.msr-inria.inria.fr/tlaps/ (Last released June 2015)

[35] Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 USENIX Annual Technical Conference (USENIX ATC 14). pp. 305–319. USENIX Association (2014)

[36] PRL Project: EventML. http://www.nuprl.org/software/#WhatisEventML (Last released September 21, 2012)

[37] van Renesse, R., Altinbuken, D.: Paxos made moderately complex. ACM Computing Surveys 47(3), 42:1–42:36 (Feb 2015)

[38] Microsoft Research: Dafny: A language and program verifier for functional correctness. http://research.microsoft.com/en-us/projects/dafny/ (Last released October 12, 2015)

[39] Schiper, N., Rahli, V., van Renesse, R., Bickford, M., Constable, R.L.: Developing correctly replicated databases using formal tools. In: Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 395–406. IEEE CS Press (2014)

[40] Spin Community: Verifying Multi-threaded Software with Spin. http://spinroot.com/spin/whatispin.html (Last released January 1, 2016)

[41] SRI: PVS Specification and Verification System. http://pvs.csl.sri.com/ (Last released February 11, 2013)

[42] University of Cambridge: Isabelle (a generic proof assistant). http://isabelle.in.tum.de/ (Last released May 25, 2015)

[43] Wilcox, J.R., Woos, D., Panchekha, P., Tatlock, Z., Wang, X., Ernst, M.D., Anderson, T.: Verdi: A framework for implementing and formally verifying distributed systems. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 357–368 (2015)

[44] Yabandeh, M., Knezevic, N., Kostic, D., Kuncak, V.: CrystalBall: Predicting and preventing inconsistencies in deployed distributed systems. In: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation. pp. 229–244. USENIX Association (2009)

[45] Yang, J., Chen, T., Wu, M., Xu, Z., Liu, X., Lin, H., Yang, M., Long, F., Zhang, L., Zhou, L.: MoDist: Transparent model checking of unmodified distributed systems. In: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation. pp. 213–228. USENIX Association (2009)

A TLA$^+$ specification of Multi-Paxos with Preemption

─────────────── MODULE *MultiPaxosSpec* ───────────────

This is a specification in TLA+ and machine checked proof in TLAPS of Multi-Paxos with Preemption.

EXTENDS *Integers*, *TLAPS*, *FiniteSets*, *FiniteSetTheorems*
CONSTANTS $\mathcal{P}$, $\mathcal{A}$, $\mathcal{Q}$, $\mathcal{V}$ Sets of proposers, acceptors, quorums of acceptors, and values to propose

VARIABLES *msgs*,   Set of sent messages
         pBal,   For each proposer, the current ballot of the proposer
         aBal,   For each acceptor, the highest ballot seen by the acceptor
         aVoted   For each acceptor, a subset of $\langle ballot,\ slot,\ value \rangle$ triples that the acceptor has voted

ASSUME *QuorumAssumption* $\triangleq \mathcal{Q} \subseteq$ SUBSET $\mathcal{A} \land \forall\, Q1, Q2 \in \mathcal{Q} : Q1 \cap Q2 \neq \emptyset$

$\mathcal{B} \triangleq \mathbb{N}$ Set of ballots
$\mathcal{S} \triangleq \mathbb{N}$ Set of slots
$vars \triangleq \langle msgs,\ pBal,\ aBal,\ aVoted \rangle$
$Send(m) \triangleq msgs' = msgs \cup \{m\}$

Phase 1*a*: For a proposer p, this phase selects some ballot number $pBal[p]$ with which a 1a message has not been sent, and sends it (to all processes).

$Phase1a(p) \triangleq \ \land \nexists\, m \in msgs : m.type =$ "1a" $\land m.bal = pBal[p]$
               $\land Send([type \mapsto$ "1a"$,\ from \mapsto p,\ bal \mapsto pBal[p]])$
               $\land$ UNCHANGED $\langle pBal,\ aBal,\ aVoted \rangle$

Phase 1*b*: For an acceptor a, if there is a 1a message m with ballot $m.bal$ that is higher than the highest it has seen, a sends a 1b message with $m.bal$ and with the set of highest-numbered triples it has voted for each slot, and it updates the highest ballot it has seen to be $m.bal$; otherwise it sends a preempt message back with the highest ballot it has seen.

$Phase1b(a) \triangleq \exists\, m \in msgs : m.type =$ "1a" $\land$
IF $m.bal > aBal[a]$ THEN
     $\land Send([type \mapsto$ "1b"$,\ from \mapsto a,\ bal \mapsto m.bal,\ voted \mapsto aVoted[a]])$
     $\land aBal' = [aBal$ EXCEPT $![a] = m.bal]$
     $\land$ UNCHANGED $\langle pBal,\ aVoted \rangle$
ELSE
     $\land Send([type \mapsto$ "preempt"$,\ to \mapsto m.from,\ bal \mapsto aBal[a]])$
     $\land$ UNCHANGED $\langle pBal,\ aBal,\ aVoted \rangle$

Phase 2*a*: For a proposer p, if there is no 2a message with current ballot $pBal[p]$, and a quorum of acceptors has sent a set S of 1b messages with $pBal[p]$, p sends a 2a message with $pBal[p]$ and a set of proposals $PropSV(T)$, where T is the union of all voted triples in messages in S. $PropSV(T)$ includes $MaxSV(T)$, the set of slot-value pairs with the highest ballot for each slot in T, and $NewSV(T)$, a set of new slot-value pairs for slots not in T.

$MaxBSV(T) \quad \triangleq \{t \in T : \forall\, t2 \in T : t2.slot = t.slot \Rightarrow t2.bal \leq t.bal\}$
$MaxSV(T) \quad\ \triangleq \{[slot \mapsto t.slot,\ val \mapsto t.val] : t \in MaxBSV(T)\}$
$UnusedS(T) \quad \triangleq \{s \in \mathcal{S} : \nexists\, t \in T : t.slot = s\}$
$NewSV(T) \quad\ \triangleq$ CHOOSE $D \subseteq [slot : UnusedS(T),\ val : \mathcal{V}] : \forall\, d1,\ d2\ \in D : d1.slot = d2.slot \Rightarrow d1 = d2 \land D \neq \emptyset$
$PropSV(T) \quad\ \triangleq MaxSV(T) \cup NewSV(T)$

$Phase2a(p) \triangleq$
   $\land \nexists\, m \in msgs : (m.type =$ "2a"$) \land (m.bal = pBal[p])$
   $\land \exists\, Q \in \mathcal{Q},\ S \subseteq \{m \in msgs : m.type =$ "1b" $\land m.bal = pBal[p]\} :$
       $\land \forall\, a \in Q : \exists\, m \in S : m.from = a$
       $\land Send([type \mapsto$ "2a"$,\ from \mapsto p,\ bal \mapsto pBal[p],\ propSV \mapsto PropSV($UNION $\{m.voted : m \in S\})])$
   $\land$ UNCHANGED $\langle pBal,\ aBal,\ aVoted \rangle$

Phase 2*b*: For an acceptor a, if there is a 2a message m with ballot $m.bal$ that is higher than or equal to the highest it has seen, a sends a 2b message with $m.bal$ and $m.propSV$, updates the highest ballot it has seen to $m.bal$, and updates set of voted triples using $m.propSV$; otherwise it sends a preempt message back with the highest ballot it has seen.

$Phase2b(a) \triangleq \exists\, m \in msgs : m.type =$ "2a" $\land$
IF $m.bal \geq aBal[a]$ THEN

$\wedge Send([type \mapsto \text{"2b"}, from \mapsto a, bal \mapsto m.bal, propSV \mapsto m.propSV])$
$\wedge aBal' = [aBal \text{ EXCEPT } ![a] = m.bal]$
$\wedge aVoted' = [aVoted \text{ EXCEPT } ![a] = \{[bal \mapsto m.bal, slot \mapsto d.slot, val \mapsto d.val] : d \in m.propSV\} \cup$
$\qquad\qquad\qquad\qquad \{e \in aVoted[a] : \nexists r \in m.propSV : e.slot = r.slot\}]$
$\wedge \text{UNCHANGED } \langle pBal \rangle$
ELSE
$\quad \wedge Send([type \mapsto \text{"preempt"}, to \mapsto m.from, bal \mapsto aBal[a]])$
$\quad \wedge \text{UNCHANGED } \langle pBal, aBal, aVoted \rangle$

Preempt: For a proposer p, if there is a preempt message m with ballot $m.bal$ that is higher than p's current ballot, p updates its current ballot to a new ballot that is higher than $m.bal$ and with which no 1a message has been sent.

$NewBal(b2) \triangleq \text{CHOOSE } b \in \mathcal{B} : b > b2 \wedge \nexists m \in msgs : m.type = \text{"1a"} \wedge m.bal = b$
$Preempt(p) \triangleq \exists m \in msgs :$
$\quad \wedge m.type = \text{"preempt"} \wedge m.to = p \wedge m.bal > pBal[p]$
$\quad \wedge pBal' = [pBal \text{ EXCEPT } ![p] = NewBal(m.bal)]$
$\quad \wedge \text{UNCHANGED } \langle msgs, aBal, aVoted \rangle$

$Init \triangleq msgs = \emptyset \wedge pBal = [p \in \mathcal{P} \mapsto 0] \wedge aBal = [a \in \mathcal{A} \mapsto -1] \wedge aVoted = [a \in \mathcal{A} \mapsto \emptyset]$
$Next \triangleq \vee \exists p \in \mathcal{P} : Phase1a(p) \vee Phase2a(p) \vee Preempt(p)$
$\qquad \vee \exists a \in \mathcal{A} : Phase1b(a) \vee Phase2b(a)$
$Spec \triangleq Init \wedge \Box[Next]_{vars}$

B Safety property to prove for Multi-Paxos with Preemption and invariants used in proof

$\quad$ MODULE $\textit{MultiPaxosProp}$ $\quad$

$VotedForIn(a, b, s, v)$ means that acceptor a has sent some 2b message m with $m.bal$ equal to b and some proposal in $m.propSV$ with $slot$ equal to s and val equal to v. This specifies that acceptor a has voted the triple $\langle b, s, v \rangle$.

$VotedForIn(a, b, s, v) \triangleq \exists\, m \in msgs :$
$m.type = \text{``2b''} \land m.from = a \land m.bal = b \land \exists\, d \in m.propSV : d.slot = s \land d.val = v$

$ChosenIn(b, s, v)$ means that every acceptor in some quorum Q has voted the triple $\langle b, s, v \rangle$.

$ChosenIn(b, s, v) \triangleq \exists\, Q \in \mathcal{Q} : \forall\, a \in Q : VotedForIn(a, b, s, v)$

$Chosen(s, v)$ means that for some ballot b, $ChosenIn(b, s, v)$ holds.

$Chosen(s, v) \triangleq \exists\, b \in \mathcal{B} : ChosenIn(b, s, v)$

$WontVoteIn(a, b, s)$ means that acceptor a has seen a higher ballot than b, and did not and will not vote any value with b for slot s.

$WontVoteIn(a, b, s) \triangleq aBal[a] > b \land \forall\, v \in \mathcal{V} : \neg\, VotedForIn(a, b, s, v)$

$SafeAt(b, s, v)$ means that no value except perhaps v has been or will be chosen in any ballot lower than b for slot s.

$SafeAt(b, s, v) \triangleq \forall\, b2 \in 0\,..\,(b-1) : \exists\, Q \in \mathcal{Q} : \forall\, a \in Q : VotedForIn(a, b2, s, v) \lor WontVoteIn(a, b2, s)$

$Safe$ states that at most one value can be chosen for each slot.

$Safe \triangleq \forall\, v1, v2 \in \mathcal{V}, s \in \mathcal{S} : Chosen(s, v1) \land Chosen(s, v2) \Rightarrow v1 = v2$

$Messages$ defines the set of valid messages. $TypeOK$ defines invariants for the types of the variables.

$Messages \triangleq [type : \{\text{``1a''}\}, from : \mathcal{P}, bal : \mathcal{B}] \cup$
$\qquad [type : \{\text{``1b''}\}, from : \mathcal{A}, bal : \mathcal{B}, voted : \text{SUBSET}\ [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}]] \cup$
$\qquad [type : \{\text{``2a''}\}, from : \mathcal{P}, bal : \mathcal{B}, propSV : \text{SUBSET}\ [slot : \mathcal{S}, val : \mathcal{V}]] \cup$
$\qquad [type : \{\text{``2b''}\}, from : \mathcal{A}, bal : \mathcal{B}, propSV : \text{SUBSET}\ [slot : \mathcal{S}, val : \mathcal{V}]] \cup$
$\qquad [type : \{\text{``preempt''}\}, to : \mathcal{P}, bal : \mathcal{B}]$
$TypeOK \triangleq \land\ msgs \subseteq Messages \land IsFiniteSet(msgs) \land pBal \in [\mathcal{P} \to \mathcal{B}]$
$\qquad\quad \land\ aBal \in [\mathcal{A} \to \mathcal{B} \cup \{-1\}] \land aVoted \in [\mathcal{A} \to \text{SUBSET}\ [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}]]$

$Max(T)$ selects the largest element in nonempty set T.

$Max(T) \triangleq \text{CHOOSE}\ e \in T : \forall\, f \in T : e \geq f$

$MaxBalInSlot(T, s)$ selects, among set of elements in T with slot s, the highest ballot, or -1 if no element has slot s.

$MaxBalInSlot(T, s) \triangleq \text{LET}\ E \triangleq \{e \in T : e.slot = s\}\ \text{IN}\quad \text{IF}\ E = \emptyset\ \text{THEN}\ -1\ \text{ELSE}\ Max(\{e.bal : e \in E\})$

$MsgInv$ defines properties satisfied by the contents of messages, for 1b, 2a, and 2b messages.

$MsgInv1b(m) \triangleq \land\ m.bal \leq aBal[m.from]$
$\qquad\qquad\quad \land\ \forall\, r \in m.voted : VotedForIn(m.from, r.bal, r.slot, r.val)$
$\qquad\qquad\quad \land\ \forall\, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : b \in MaxBalInSlot(m.voted, s) + 1\,..\,m.bal - 1 \Rightarrow$
$\qquad\qquad\qquad \neg\, VotedForIn(m.from, b, s, v)$

$MsgInv2a(m) \triangleq \land\ \forall\, d \in m.propSV : SafeAt(m.bal, d.slot, d.val)$
$\qquad\qquad\quad \land\ \forall\, d1, d2 \in m.propSV : d1.slot = d2.slot \Rightarrow d1 = d2$
$\qquad\qquad\quad \land\ \forall\, m2 \in msgs : (m2.type = \text{``2a''} \land m2.bal = m.bal) \Rightarrow m2 = m$

$MsgInv2b(m) \triangleq \land\ \exists\, m2 \in msgs : m2.type = \text{``2a''} \land m2.bal = m.bal \land m2.propSV = m.propSV$
$\qquad\qquad\quad \land\ m.bal \leq aBal[m.from]$

$MsgInv \triangleq \forall\, m \in msgs : \land\ (m.type = \text{``1b''}) \Rightarrow MsgInv1b(m)$
$\qquad\qquad\qquad\quad \land\ (m.type = \text{``2a''}) \Rightarrow MsgInv2a(m)$
$\qquad\qquad\qquad\quad \land\ (m.type = \text{``2b''}) \Rightarrow MsgInv2b(m)$

AccInv defines properties satisfied by the data maintained by the acceptors.

$AccInv \triangleq \forall\, a \in \mathcal{A}:$
$\quad \wedge\, aBal[a] = -1 \Rightarrow aVoted[a] = \emptyset$
$\quad \wedge\, \forall\, r \in aVoted[a] : aBal[a] \geq r.bal \wedge VotedForIn(a,\, r.bal,\, r.slot,\, r.val)$
$\quad \wedge\, \forall\, b \in \mathcal{B},\, s \in \mathcal{S},\, v \in \mathcal{V} : VotedForIn(a,\, b,\, s,\, v) \Rightarrow \exists\, r \in aVoted[a] : r.bal \geq b \wedge r.slot = s$
$\quad \wedge\, \forall\, b \in \mathcal{B},\, s \in \mathcal{S},\, v \in \mathcal{V} : b > MaxBalInSlot(aVoted[a],\, s) \Rightarrow \neg VotedForIn(a,\, b,\, s,\, v)$

Inv is the complete inductive invariant.

$Inv \triangleq TypeOK \wedge AccInv \wedge MsgInv$

C TLAPS checked proof of Multi-Paxos with Preemption

The following 2 axioms and 10 lemmas are straightforward consequences of the predicates defined above.

AXIOM $MaxInSet \triangleq \forall S \in (\text{SUBSET } \mathbb{N}) \setminus \emptyset : Max(S) \in S$
AXIOM $MaxOnNat \triangleq \forall S \in \text{SUBSET } \mathbb{N} : \nexists s \in S : Max(S) < s$

LEMMA $MaxOnNatS \triangleq \forall S1, S2 \in (\text{SUBSET } \mathbb{N}) \setminus \emptyset : S1 \subseteq S2 \Rightarrow Max(S1) \leq Max(S2)$ BY $MaxInSet$

LEMMA $MaxBInSType \triangleq \forall S \in \text{SUBSET } [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}], s \in \mathcal{S} : MaxBalInSlot(S, s) \in \mathcal{B} \cup \{-1\}$
BY $MaxInSet$ DEF $MaxBalInSlot$

LEMMA $MaxBInSSubsets \triangleq \forall S1, S2 \in \text{SUBSET } [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}], s \in \mathcal{S} : S1 \subseteq S2 \Rightarrow$
$\qquad MaxBalInSlot(S1, s) \leq MaxBalInSlot(S2, s)$
⟨1⟩ SUFFICES ASSUME NEW $S1 \in \text{SUBSET } [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}]$, NEW $s \in \mathcal{S}$,
$\qquad$ NEW $S2 \in \text{SUBSET } [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}], S1 \subseteq S2$
$\qquad$ PROVE $\quad MaxBalInSlot(S1, s) \leq MaxBalInSlot(S2, s)$ OBVIOUS
⟨1⟩1. CASE $\nexists d \in S1 : d.slot = s$
$\quad$ ⟨2⟩1. $MaxBalInSlot(S1, s) = -1$ BY ⟨1⟩1 DEF $MaxBalInSlot$
$\quad$ ⟨2⟩ QED BY ⟨2⟩1, $MaxBInSType$ DEF $\mathcal{B}$
⟨1⟩2. CASE $\exists d \in S1 : d.slot = s$
$\quad$ ⟨2⟩1. CASE $\nexists d \in S2 \setminus S1 : d.slot = s$
$\qquad$ ⟨3⟩1. $MaxBalInSlot(S1, s) = MaxBalInSlot(S2, s)$ BY ⟨2⟩1, ⟨1⟩2 DEF $MaxBalInSlot$
$\qquad$ ⟨3⟩ QED BY ⟨3⟩1, $MaxBInSType$ DEF $\mathcal{B}$
$\quad$ ⟨2⟩2. CASE $\exists d \in S2 \setminus S1 : d.slot = s$ BY ⟨2⟩2, ⟨1⟩2, $MaxBInSType, MaxOnNatS$ DEF $\mathcal{B}, MaxBalInSlot$
$\quad$ ⟨2⟩ QED BY ⟨2⟩1, ⟨2⟩2
⟨1⟩ QED BY ⟨1⟩1, ⟨1⟩2

LEMMA $MaxBInSNoSlot \triangleq \forall S \in \text{SUBSET } [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}], s \in \mathcal{S} :$
$\qquad (\nexists d \in S : d.slot = s) \equiv MaxBalInSlot(S, s) = -1$
BY $MaxInSet$ DEF $MaxBalInSlot, \mathcal{B}$

LEMMA $MaxBInSExists \triangleq \forall S \in \text{SUBSET } [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}], s \in \mathcal{S} : MaxBalInSlot(S, s) \in \mathcal{B} \Rightarrow$
$\qquad \exists d \in S : d.slot = s \wedge d.bal = MaxBalInSlot(S, s)$
⟨1⟩ SUFFICES ASSUME NEW $S \in \text{SUBSET } [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}]$,
$\qquad$ NEW $s \in \mathcal{S}, MaxBalInSlot(S, s) \in \mathcal{B}$
$\qquad$ PROVE $\quad \exists d \in S : d.bal = MaxBalInSlot(S, s) \wedge d.slot = s$ OBVIOUS
⟨1⟩1. $\exists d \in S : d.slot = s$ BY DEF $MaxBalInSlot, \mathcal{B}$
⟨1⟩2. $MaxBalInSlot(S, s) = Max(\{d.bal : d \in \{d \in S : d.slot = s\}\})$ BY ⟨1⟩1 DEF $MaxBalInSlot$
⟨1⟩ QED BY ⟨1⟩1, ⟨1⟩2, $MaxInSet$

LEMMA $MaxBInSNoMore \triangleq \forall S \in \text{SUBSET } [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}], s \in \mathcal{S} :$
$\qquad \nexists d \in S : d.bal > MaxBalInSlot(S, s) \wedge d.slot = s$
⟨1⟩ SUFFICES ASSUME NEW $S \in \text{SUBSET } [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}]$, NEW $s \in \mathcal{S}$
$\qquad$ PROVE $\quad \nexists d \in S : d.bal > MaxBalInSlot(S, s) \wedge d.slot = s$ OBVIOUS
⟨1⟩1. CASE $\nexists d \in S : d.slot = s$ BY ⟨1⟩1
⟨1⟩2. CASE $\exists d \in S : d.slot = s$
$\quad$ ⟨2⟩1. $\nexists b \in \{d.bal : d \in \{d \in S : d.slot = s\}\} : b > MaxBalInSlot(S, s)$
$\qquad$ BY ⟨1⟩2, $MaxOnNat$ DEF $MaxBalInSlot, \mathcal{B}, \mathcal{S}$
$\quad$ ⟨2⟩2. $\nexists d \in S : (d.slot = s \wedge \neg(d.bal \leq MaxBalInSlot(S, s)))$ BY ⟨2⟩1
$\quad$ ⟨2⟩ QED BY ⟨2⟩2
⟨1⟩ QED BY ⟨1⟩1, ⟨1⟩2

LEMMA $Misc \triangleq \forall S : \wedge NewSV(S) \in (\text{SUBSET } [slot : UnusedS(S), val : \mathcal{V}]) \setminus \emptyset$
$\qquad\qquad \wedge \forall t1, t2 \in NewSV(S) : t1.slot = t2.slot \Rightarrow t1 = t2$
$\qquad\qquad \wedge \nexists t1 \in MaxSV(S), t2 \in NewSV(S) : t1.slot = t2.slot$
⟨1⟩ SUFFICES ASSUME NEW S
$\qquad$ PROVE $\quad \wedge NewSV(S) \in (\text{SUBSET } [slot : UnusedS(S), val : \mathcal{V}]) \setminus \emptyset$
$\qquad\qquad \wedge \forall t1, t2 \in NewSV(S) : t1.slot = t2.slot \Rightarrow t1 = t2$
$\qquad\qquad \wedge \nexists t1 \in MaxSV(S), t2 \in NewSV(S) : t1.slot = t2.slot$ OBVIOUS
⟨1⟩1. $\exists T \in (\text{SUBSET } [slot : UnusedS(S), val : \mathcal{V}]) \setminus \emptyset : \forall t1, t2 \in T : t1.slot = t2.slot \Rightarrow t1 = t2$
$\quad$ BY DEF $UnusedS$
⟨1⟩2. $NewSV(S) \in (\text{SUBSET } [slot : UnusedS(S), val : \mathcal{V}]) \setminus \emptyset$ BY ⟨1⟩1 DEF $NewSV$
⟨1⟩3. $\forall t1, t2 \in NewSV(S) : t1.slot = t2.slot \Rightarrow t1 = t2$ BY ⟨1⟩1 DEF $NewSV$
⟨1⟩4. $\nexists t1 \in MaxSV(S), t2 \in ([slot : UnusedS(S), val : \mathcal{V}] \setminus \emptyset) : t1.slot = t2.slot$
$\quad$ BY DEF $MaxSV, MaxBSV, UnusedS$
⟨1⟩ QED BY ⟨1⟩2, ⟨1⟩3, ⟨1⟩4

VotedInv asserts that if any acceptor a voted any triple $\langle b, s, v \rangle$, then that triple is safe.

LEMMA $VotedInv \triangleq MsgInv \land TypeOK \Rightarrow \forall a \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V}:$
$\qquad VotedForIn(a, b, s, v) \Rightarrow SafeAt(b, s, v) \land b \leq aBal[a]$
BY DEF $VotedForIn, MsgInv, Messages, TypeOK, MsgInv2a, MsgInv1b, MsgInv2b$

LEMMA $VotedOnce \triangleq MsgInv \Rightarrow \forall a1, a2 \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}, v1, v2 \in \mathcal{V}:$
$\qquad VotedForIn(a1, b, s, v1) \land VotedForIn(a2, b, s, v2) \Rightarrow (v1 = v2)$
BY DEF $MsgInv, VotedForIn, MsgInv2a, MsgInv1b, MsgInv2b$

$VotedUnion$ asserts that, in any two $1b$ messages' $voted$ field, triples with the same ballot and slot have the same value.

LEMMA $VotedUnion \triangleq MsgInv \land TypeOK \Rightarrow \forall m1, m2 \in msgs : m1.type = \text{"1b"} \land m2.type = \text{"1b"} \Rightarrow$
$\qquad \forall d1 \in m1.voted, d2 \in m2.voted : (d1.bal = d2.bal \land d1.slot = d2.slot) \Rightarrow$
$\qquad d1.val = d2.val$
$\langle 1 \rangle$ SUFFICES ASSUME $MsgInv, TypeOK$, NEW $m1 \in msgs$, NEW $m2 \in msgs$, $m1.type = \text{"1b"}$, $m2.type = \text{"1b"}$,
$\qquad$ NEW $d1 \in m1.voted$, NEW $d2 \in m2.voted$, $d1.bal = d2.bal$, $d1.slot = d2.slot$
$\qquad$ PROVE $d1.val = d2.val$ OBVIOUS
$\langle 1 \rangle 1.$ $VotedForIn(m1.from, d1.bal, d1.slot, d1.val)$ BY DEF $MsgInv, MsgInv1b$
$\langle 1 \rangle 2.$ $VotedForIn(m2.from, d2.bal, d2.slot, d2.val)$ BY DEF $MsgInv, MsgInv1b$
$\langle 1 \rangle$ QED BY $\langle 1 \rangle 1, \langle 1 \rangle 2, VotedOnce$ DEF $TypeOK, Messages$

The following 5 invariance lemmas assert that, for acceptor a, ballot b, slot s, and value v, and for all phases and preempt, if $VotedForIn(a, b, s, v)$ holds then $VotedForIn(a, b, s, v)'$ holds; for all except $Phase2b$, the inverse also holds.

LEMMA $Phase1aVotedForInv \triangleq TypeOK \Rightarrow \forall p \in \mathcal{P} : Phase1a(p) \Rightarrow \forall a \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V}:$
$\qquad VotedForIn(a, b, s, v) \equiv VotedForIn(a, b, s, v)'$
BY DEF $VotedForIn, Send, TypeOK, Messages, Phase1a$

LEMMA $Phase1bVotedForInv \triangleq TypeOK \Rightarrow \forall a \in \mathcal{A} : Phase1b(a) \Rightarrow \forall a2 \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V}:$
$\qquad VotedForIn(a2, b, s, v) \equiv VotedForIn(a2, b, s, v)'$
BY DEF $VotedForIn, Send, TypeOK, Messages, Phase1b$

LEMMA $Phase2aVotedForInv \triangleq TypeOK \Rightarrow \forall p \in \mathcal{P} : Phase2a(p) \Rightarrow \forall a \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V}:$
$\qquad VotedForIn(a, b, s, v) \equiv VotedForIn(a, b, s, v)'$
BY $\forall p \in \mathcal{P} : Phase2a(p) \Rightarrow \forall m \in msgs' \setminus msgs : m.type = \text{"2a"}$ DEF $VotedForIn$,
$Send, TypeOK, Messages, Phase2a$

LEMMA $Phase2bVotedForInv \triangleq TypeOK \Rightarrow \forall a \in \mathcal{A} : Phase2b(a) \Rightarrow \forall a2 \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V}:$
$\qquad VotedForIn(a2, b, s, v) \Rightarrow VotedForIn(a2, b, s, v)'$
BY DEF $VotedForIn, Send, TypeOK, Messages, Phase2b$

LEMMA $PreemptVotedForInv \triangleq TypeOK \Rightarrow \forall p \in \mathcal{P} : Preempt(p) \Rightarrow \forall a \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V}:$
$\qquad VotedForIn(a, b, s, v) \equiv VotedForIn(a, b, s, v)'$
BY DEF $VotedForIn, Send, TypeOK, Messages, Preempt$

Invariance lemma $SafeAtStable$ asserts that if $SafeAt(b, s, v)$ holds then $SafeAt(b, s, v)'$ holds in the next state.

LEMMA $SafeAtStable \triangleq Inv \land Next \land TypeOK' \Rightarrow \forall b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : SafeAt(b, s, v) \Rightarrow SafeAt(b, s, v)'$
$\langle 1 \rangle$ SUFFICES ASSUME $Inv, Next, TypeOK'$, NEW $b \in \mathcal{B}$, NEW $s \in \mathcal{S}$, NEW $v \in \mathcal{V}$, $SafeAt(b, s, v)$
$\qquad$ PROVE $SafeAt(b, s, v)'$ OBVIOUS
$\langle 1 \rangle$ USE DEF $Send, Inv, \mathcal{B}$
$\langle 1 \rangle 1.$ CASE $\exists p \in \mathcal{P} : Phase1a(p)$ BY $\langle 1 \rangle 1$ DEF $SafeAt, Phase1a, VotedForIn, WontVoteIn$
$\langle 1 \rangle 2.$ CASE $\exists a \in \mathcal{A} : Phase1b(a)$
$\qquad$ BY $\langle 1 \rangle 2, QuorumAssumption$ DEF $TypeOK, SafeAt, WontVoteIn, VotedForIn, Phase1b$
$\langle 1 \rangle 3.$ ASSUME NEW $p \in \mathcal{P}$, $Phase2a(p)$ PROVE $SafeAt(b, s, v)'$
$\quad \langle 2 \rangle 1.$ $\forall a \in \mathcal{A}, b2 \in \mathcal{B}, s2 \in \mathcal{S} : WontVoteIn(a, b2, s2) \equiv WontVoteIn(a, b2, s2)'$
$\qquad$ BY $\langle 1 \rangle 3, Phase2aVotedForInv$ DEF $WontVoteIn, Send, Phase2a$
$\quad \langle 2 \rangle$ QED BY $\langle 2 \rangle 1, QuorumAssumption, Phase2aVotedForInv, \langle 1 \rangle 3$ DEF $SafeAt$
$\langle 1 \rangle 4.$ ASSUME NEW $a \in \mathcal{A}$, $Phase2b(a)$ PROVE $SafeAt(b, s, v)'$
$\quad \langle 2 \rangle 1.$ PICK $m \in msgs : Phase2b(a)!(m)$ BY $\langle 1 \rangle 4$ DEF $Phase2b$
$\quad \langle 2 \rangle 2.$ $\forall a2 \in \mathcal{A}, b2 \in \mathcal{B} : aBal[a2] > b2 \Rightarrow aBal'[a2] > b2$ BY $\langle 2 \rangle 1$ DEF $TypeOK$
$\quad \langle 2 \rangle 3.$ ASSUME NEW $a2 \in \mathcal{A}$, NEW $b2 \in \mathcal{B}$, NEW $s2 \in \mathcal{S}$, NEW $v2 \in \mathcal{V}$, $WontVoteIn(a2, b2, s2)$,
$\qquad VotedForIn(a2, b2, s2, v2)'$, NEW $S \in$ SUBSET $[slot : \mathcal{S} \setminus \{s2\}, val : \mathcal{V}]$
$\qquad$ PROVE FALSE
$\quad \langle 3 \rangle 1.$ $\exists m1 \in msgs' \setminus msgs : \land m1.type = \text{"2b"} \land m1.bal = b2 \land m1.from = a2$
$\qquad \qquad \land \exists d \in m1.propSV : d.slot = s2 \land d.val = v2$
$\qquad$ BY $\langle 2 \rangle 3$ DEF $VotedForIn, WontVoteIn$

$\langle 3\rangle 2.\ a2 = a \wedge m.bal = b2\ \text{BY}\ \langle 2\rangle 1,\ \langle 3\rangle 1\ \ \text{DEF}\ TypeOK$
$\langle 3\rangle\ \text{QED}\ \ \text{BY}\ \langle 2\rangle 1,\ \langle 2\rangle 3,\ \langle 3\rangle 2,\ \langle 3\rangle 1\ \ \text{DEF}\ Phase2b,\ WontVoteIn,\ TypeOK$
$\langle 2\rangle 4.\ \forall\, a2 \in \mathcal{A},\ b2 \in \mathcal{B},\ s2 \in \mathcal{S} : WontVoteIn(a2,\ b2,\ s2) \Rightarrow WontVoteIn(a2,\ b2,\ s2)'$
$\quad \text{BY}\ \langle 2\rangle 2,\ \langle 2\rangle 3\ \ \text{DEF}\ WontVoteIn$
$\langle 2\rangle\ \text{QED BY}\ Phase2bVotedForInv,\ \langle 2\rangle 4,\ QuorumAssumption,\ \langle 1\rangle 4\ \ \text{DEF}\ SafeAt$
$\langle 1\rangle 5.\ \text{CASE}\ \exists\, p \in \mathcal{P} : Preempt(p)\ \text{BY}\ \langle 1\rangle 5\ \ \text{DEF}\ SafeAt,\ Preempt,\ VotedForIn,\ WontVoteIn$
$\langle 1\rangle\ \text{QED BY}\ \langle 1\rangle 1,\ \langle 1\rangle 2,\ \langle 1\rangle 3,\ \langle 1\rangle 4,\ \langle 1\rangle 5\ \ \text{DEF}\ Next$

Invariant asserts the temporal formula that if *Spec* holds then *Inv* always holds.

$\text{THEOREM}\ Invariant\ \triangleq\ Spec \Rightarrow \Box Inv$
$\langle 1\rangle\ \text{USE}\ \ \text{DEF}\ \mathcal{B},\ \mathcal{S}$
$\langle 1\rangle 1.\ Init \Rightarrow Inv\ \text{BY}\ FS\_EmptySet\ \text{DEF}\ Init,\ Inv,\ TypeOK,\ AccInv,\ MsgInv,\ VotedForIn$
$\langle 1\rangle 2.\ Inv \wedge [Next]_{vars} \Rightarrow Inv'$
$\langle 2\rangle\ \text{SUFFICES ASSUME}\ Inv,\ [Next]_{vars}\,\text{PROVE}\ \ \ Inv'\ \text{OBVIOUS}$
$\langle 2\rangle\ \text{USE}\ \ \text{DEF}\ Inv$
$\langle 2\rangle 1.\ \text{CASE}\ Next$

$\langle 3\rangle 1$ proves $TypeOK'$ for *Next*. Each of $\langle 4\rangle 1$-4 assumes the action of a phase and proves $TypeOK'$ for that case.

$\langle 3\rangle 1.\ TypeOK'$
$\langle 4\rangle 1.\ \text{ASSUME NEW}\ p \in \mathcal{P},\ Phase1a(p)\ \text{PROVE}\ \ \ TypeOK'$
$\quad \text{BY}\ \langle 4\rangle 1,\ msgs' \setminus msgs \subseteq Messages,\ FS\_AddElement\ \text{DEF}\ Phase1a,\ TypeOK,\ Send,\ Messages$
$\langle 4\rangle 2.\ \text{ASSUME NEW}\ p \in \mathcal{P},\ Phase2a(p)\ \text{PROVE}\ \ \ TypeOK'$
$\langle 5\rangle 1.\ \text{PICK}\ Q \in \mathcal{Q},\ S \in \text{SUBSET}\ \{m \in msgs : (m.type = \text{``1b''}) \wedge (m.bal = pBal[p])\} :$
$\qquad \wedge\, \forall\, a \in Q : \exists\, m \in S : m.from = a$
$\qquad \wedge\, Send([type \mapsto \text{``2a''},\ from \mapsto p,\ bal \mapsto pBal[p],$
$\qquad\quad propSV \mapsto PropSV(\text{UNION}\ \{m.voted : m \in S\})])\ \text{BY}\ \langle 4\rangle 2\ \ \text{DEF}\ Phase2a$
$\langle 5\rangle 2.\ UnusedS(\text{UNION}\ \{m.voted : m \in S\}) \subseteq \mathcal{S}\ \text{BY}\ \langle 5\rangle 1\ \ \text{DEF}\ UnusedS,\ TypeOK,\ Messages$
$\langle 5\rangle 3.\ MaxBSV(\text{UNION}\ \{m.voted : m \in S\}) \subseteq [bal : \mathcal{B},\ slot : \mathcal{S},\ val : \mathcal{V}]$
$\qquad \text{BY}\ \ \text{DEF}\ MaxBSV,\ TypeOK,\ Messages$
$\langle 5\rangle 4.\ \wedge\, NewSV(\text{UNION}\ \{m.voted : m \in S\}) \subseteq [slot : \mathcal{S},\ val : \mathcal{V}]$
$\qquad \wedge\, MaxSV(\text{UNION}\ \{m.voted : m \in S\}) \subseteq [slot : \mathcal{S},\ val : \mathcal{V}]\ \text{BY}\ \langle 5\rangle 3,\ \langle 5\rangle 2,\ Misc\ \text{DEF}\ MaxSV$
$\langle 5\rangle 5.\ PropSV(\text{UNION}\ \{m.voted : m \in S\}) \subseteq [slot : \mathcal{S},\ val : \mathcal{V}]\ \text{BY}\ \langle 5\rangle 4\ \ \text{DEF}\ PropSV$
$\langle 5\rangle 6.\ \forall\, m2 \in msgs' \setminus msgs : \wedge\, m2.type = \text{``2a''} \wedge m2.from = p \wedge m2.bal = pBal[p]$
$\qquad\qquad\qquad\qquad\qquad\qquad \wedge\, m2.propSV = PropSV(\text{UNION}\ \{m.voted : m \in S\})$
$\qquad \text{BY}\ \langle 5\rangle 1,\ \langle 5\rangle 5\ \ \text{DEF}\ Send,\ TypeOK,\ Messages$
$\langle 5\rangle 7.\ (msgs \subseteq Messages)'\ \text{BY}\ \langle 4\rangle 2,\ \langle 5\rangle 6,\ \langle 5\rangle 1,\ \langle 5\rangle 5,\ msgs' \setminus msgs \subseteq Messages\ \text{DEF}\ Phase2a,$
$\qquad TypeOK,\ Send,\ Messages$
$\langle 5\rangle\ \text{QED BY}\ \langle 5\rangle 7,\ \langle 4\rangle 2,\ FS\_AddElement\ \text{DEF}\ Phase2a,\ TypeOK,\ Send$
$\langle 4\rangle 3.\ \text{ASSUME NEW}\ a \in \mathcal{A},\ Phase1b(a)\ \text{PROVE}\ \ \ TypeOK'$
$\langle 5\rangle 1.\ \text{PICK}\ m \in msgs : Phase1b(a)!(m)\ \text{BY}\ \langle 4\rangle 3\ \ \text{DEF}\ Phase1b$
$\langle 5\rangle 2.\ \vee\, msgs' = msgs \cup \{[type \mapsto \text{``1b''},\ from \mapsto a,\ bal \mapsto m.bal,\ voted \mapsto aVoted[a]]\}$
$\qquad \vee\, msgs' = msgs \cup \{[type \mapsto \text{``preempt''},\ to \mapsto m.from,\ bal \mapsto aBal[a]]\}\ \text{BY}\ \langle 5\rangle 1\ \ \text{DEF}\ Send$
$\langle 5\rangle 3.\ \forall\, m2 \in msgs' \setminus msgs : \vee\, m2.type = \text{``1b''} \wedge m2.from = a \wedge m2.bal = m.bal \wedge m2.voted = aVoted[a]$
$\qquad\qquad\qquad\qquad\qquad \vee\, m2.type = \text{``preempt''} \wedge m2.to = m.from \wedge m2.bal = aBal[a]$
$\qquad \text{BY}\ \langle 5\rangle 1\ \ \text{DEF}\ Send$
$\langle 5\rangle 4.\ (msgs \subseteq Messages)'\ \text{BY}\ \langle 5\rangle 1,\ \langle 5\rangle 3\ \ \text{DEF}\ TypeOK,\ Messages,\ Send$
$\langle 5\rangle 5.\ IsFiniteSet(msgs)'\ \text{BY}\ \langle 5\rangle 2,\ FS\_AddElement\ \text{DEF}\ TypeOK$
$\langle 5\rangle\ \text{QED BY}\ \langle 5\rangle 4,\ \langle 5\rangle 5,\ \langle 4\rangle 3\ \ \text{DEF}\ Phase1b,\ TypeOK$
$\langle 4\rangle 4.\ \text{ASSUME NEW}\ a \in \mathcal{A},\ Phase2b(a)\ \text{PROVE}\ \ \ TypeOK'$
$\langle 5\rangle 1.\ \text{PICK}\ m \in msgs : Phase2b(a)!(m)\ \text{BY}\ \langle 4\rangle 4\ \ \text{DEF}\ Phase2b$
$\langle 5\rangle 2.\ \vee\, msgs' = msgs \cup \{[type \mapsto \text{``2b''},\ bal \mapsto m.bal,\ from \mapsto a,\ propSV \mapsto m.propSV]\}$
$\qquad \vee\, msgs' = msgs \cup \{[type \mapsto \text{``preempt''},\ to \mapsto m.from,\ bal \mapsto aBal[a]]\}\ \text{BY}\ \langle 5\rangle 1\ \ \text{DEF}\ Send$
$\langle 5\rangle 3.\ IsFiniteSet(msgs)'\ \text{BY}\ \langle 5\rangle 2,\ FS\_AddElement\ \text{DEF}\ TypeOK$
$\langle 5\rangle 4.\ \forall\, m2 \in msgs' \setminus msgs : \vee\, m2.type = \text{``2b''} \wedge m2.from = a \wedge m2.bal = m.bal \wedge m2.propSV = m.propSV$
$\qquad\qquad\qquad\qquad\qquad \vee\, m2.type = \text{``preempt''} \wedge m2.to = m.from \wedge m2.bal = aBal[a]$
$\qquad \text{BY}\ \langle 5\rangle 1\ \ \text{DEF}\ Send$
$\langle 5\rangle 5.\ (msgs \subseteq Messages)'\ \text{BY}\ \langle 5\rangle 1,\ \langle 5\rangle 4\ \ \text{DEF}\ TypeOK,\ Send,\ Messages$
$\langle 5\rangle 6.\ \vee\, aVoted = aVoted'$
$\qquad \vee\, \wedge \text{DOMAIN}\ aVoted = \text{DOMAIN}\ aVoted'$
$\qquad\quad \wedge\, aVoted'[a] = \{d \in aVoted[a] : \nexists\, d2 \in m.propSV : d.slot = d2.slot\}\ \cup$
$\qquad\qquad\quad \{[bal \mapsto m.bal,\ slot \mapsto d.slot,\ val \mapsto d.val] : d \in m.propSV\}$
$\qquad\quad \wedge\, \forall\, a2 \in \mathcal{A} \setminus \{a\} : aVoted[a2] = aVoted'[a2]\ \text{BY}\ \langle 5\rangle 1\ \ \text{DEF}\ TypeOK$
$\langle 5\rangle 7.\ (aVoted \in [\mathcal{A} \to \text{SUBSET}\ [bal : \mathcal{B},\ slot : \mathcal{S},\ val : \mathcal{V}]])'\ \text{BY}\ \langle 5\rangle 1,\ \langle 5\rangle 6,$
$\qquad \{[bal \mapsto m.bal,\ slot \mapsto d.slot,\ val \mapsto d.val] : d \in m.propSV\} \subseteq [bal : \mathcal{B},\ slot : \mathcal{S},\ val : \mathcal{V}],$
$\qquad \{d \in aVoted[a] : \nexists\, d2 \in m.propSV : d.slot = d2.slot\} \subseteq [bal : \mathcal{B},\ slot : \mathcal{S},\ val : \mathcal{V}],$
$\qquad aVoted'[a] \subseteq [bal : \mathcal{B},\ slot : \mathcal{S},\ val : \mathcal{V}]\ \text{DEF}\ TypeOK,\ Messages$

⟨5⟩ QED BY ⟨5⟩5, ⟨5⟩7, ⟨4⟩4, ⟨5⟩3 DEF *Phase2b*, *TypeOK*

⟨4⟩5. ASSUME NEW $p \in \mathcal{P}$, *Preempt(p)* PROVE   *TypeOK′*
  ⟨5⟩ DEFINE $S \triangleq \{m1 \in msgs : m1.type = \text{"1a"}\}$ $T \triangleq \{s.bal : s \in S\}$ $f \triangleq [s \in S \mapsto s.bal]$
  ⟨5⟩ HIDE DEF S, T, f
  ⟨5⟩1. PICK $m \in msgs : Preempt(p)!(m)$ BY ⟨4⟩5 DEF *Preempt*
  ⟨5⟩2. $T \subseteq \mathcal{B}$ BY DEF T, S, *TypeOK*, *Messages*
  ⟨5⟩3. $\exists\, b \in \mathcal{B} : b > m.bal \wedge b \notin T$
      BY ⟨5⟩2, *MaxInSet*, $Max(T \cup \{m.bal\}) + 1 > m.bal$, ⟨5⟩1 DEF *Max*, *TypeOK*, *Messages*
  ⟨5⟩4. $NewBal(m.bal) \in \mathcal{B}$ BY ⟨5⟩3 DEF *NewBal*, *TypeOK*, *Messages*, T, S
  ⟨5⟩5. $(pBal \in [\mathcal{P} \rightarrow \mathcal{B}])'$ BY ⟨5⟩1, ⟨5⟩4 DEF *TypeOK*, *Messages*
  ⟨5⟩ QED BY ⟨4⟩5, ⟨5⟩5 DEF *Preempt*, *TypeOK*
⟨4⟩ QED BY ⟨2⟩1, ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5 DEF *Next*

⟨3⟩2. *AccInv′*
⟨4⟩1. CASE $\exists\, p \in \mathcal{P} : Phase1a(p)$ BY ⟨4⟩1, ⟨3⟩1, *Phase1aVotedForInv* DEF *AccInv*, *TypeOK*, *Phase1a*, *Send*
⟨4⟩2. ASSUME NEW $p \in \mathcal{P}$, *Phase2a(p)* PROVE   *AccInv′*
  ⟨5⟩1. $\forall\, a \in \mathcal{A}, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : VotedForIn(a, b, s, v) \equiv VotedForIn(a, b, s, v)'$
      BY ⟨4⟩2, *Phase2aVotedForInv*
  ⟨5⟩ QED BY ⟨3⟩1, ⟨4⟩2, ⟨5⟩1 DEF *AccInv*, *TypeOK*, *Phase2a*, *Send*, *Messages*
⟨4⟩3. CASE $\exists\, a \in \mathcal{A} : Phase1b(a)$ BY ⟨4⟩3, ⟨3⟩1, *Phase1bVotedForInv* DEF *AccInv*, *TypeOK*, *Phase1b*, *Send*
⟨4⟩4. ASSUME NEW $a \in \mathcal{A}$, *Phase2b(a)* PROVE   *AccInv′*
  ⟨5⟩ SUFFICES ASSUME NEW $a2 \in \mathcal{A}'$
          PROVE $(\wedge\ aBal[a2] = -1 \Rightarrow aVoted[a2] = \emptyset$
              $\wedge\ \forall\, r \in aVoted[a2] : VotedForIn(a2, r.bal, r.slot, r.val) \wedge r.bal \leq aBal[a2]$
              $\wedge\ \forall\, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} :$
                  $\wedge\ VotedForIn(a2, b, s, v) \Rightarrow \exists\, r \in aVoted[a2] : r.bal \geq b \wedge r.slot = s$
                  $\wedge\ b > MaxBalInSlot(aVoted[a2], s) \Rightarrow \neg VotedForIn(a2, b, s, v))'$
          BY DEF *AccInv*
  ⟨5⟩1. PICK $m \in msgs : Phase2b(a)!(m)$ BY ⟨4⟩4 DEF *Phase2b*

  ⟨5⟩2. CASE $(a2 = a \wedge \neg(m.bal \geq aBal[a])) \vee a2 \neq a$
    ⟨6⟩1. $\forall\, b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : VotedForIn(a2, b, s, v) \equiv VotedForIn(a2, b, s, v)'$
        BY ⟨3⟩1, ⟨5⟩1, ⟨5⟩2 DEF *Phase2b*, *TypeOK*, $\mathcal{B}$, *Messages*, *VotedForIn*, *Send*
    ⟨6⟩ QED BY ⟨6⟩1, ⟨5⟩1, ⟨5⟩2, ⟨4⟩4, ⟨3⟩1 DEF *Phase2b*, *Send*, *AccInv*, *TypeOK*, *Messages*

  ⟨5⟩3. CASE $a2 = a \wedge (m.bal \geq aBal[a])$
    ⟨6⟩1. $(aBal[a2] = -1 \Rightarrow aVoted[a2] = \emptyset)'$ BY ⟨5⟩3, ⟨4⟩4, ⟨3⟩1 DEF *AccInv*, *Phase2b*, *Send*,
        TypeOK, *Messages*
    ⟨6⟩2. $(\forall\, r \in aVoted[a2] : VotedForIn(a2, r.bal, r.slot, r.val) \wedge r.bal \leq aBal[a2])'$
      ⟨7⟩ SUFFICES ASSUME NEW $r \in (aVoted[a2])'$
              PROVE $(VotedForIn(a2, r.bal, r.slot, r.val) \wedge r.bal \leq aBal[a2])'$ OBVIOUS

      ⟨7⟩1. $VotedForIn(a2, r.bal, r.slot, r.val)'$
        ⟨8⟩1. CASE $r \in aVoted[a2]$
          ⟨9⟩1. $VotedForIn(a2, r.bal, r.slot, r.val)$ BY ⟨5⟩3, ⟨4⟩4, ⟨8⟩1 DEF *AccInv*
          ⟨9⟩ QED BY ⟨9⟩1, *Phase2bVotedForInv*, ⟨3⟩1, ⟨4⟩4 DEF *TypeOK*, *Messages*
        ⟨8⟩2. CASE $r \in aVoted'[a2] \setminus aVoted[a2]$
          ⟨9⟩1. $\exists\, m2 \in msgs' : m2.type = \text{"2b"} \wedge m2.from = a2 \wedge m2.bal = m.bal \wedge m2.propSV = m.propSV$
              BY ⟨3⟩1, ⟨8⟩2, ⟨5⟩1, ⟨5⟩3 DEF *Send*
          ⟨9⟩2. $r.bal = m.bal \wedge \exists\, d \in m.propSV : r.slot = d.slot \wedge r.val = d.val$ BY ⟨5⟩1, ⟨5⟩3, ⟨3⟩1, ⟨8⟩2
          ⟨9⟩ QED BY ⟨9⟩1, ⟨9⟩2 DEF *Send*, *TypeOK*, *Messages*, *VotedForIn*
        ⟨8⟩ QED BY ⟨8⟩1, ⟨8⟩2
      ⟨7⟩2. $(r.bal \leq aBal[a2])'$ BY ⟨5⟩1, ⟨5⟩3, ⟨3⟩1, $aBal[a] \leq aBal'[a]$, $r \in aVoted[a] \Rightarrow r.bal \leq aBal'[a]$,
          $aVoted'[a] = \{d \in aVoted[a] : \nexists\, d2 \in m.propSV : d.slot = d2.slot\} \cup$
                  $\{[bal \mapsto m.bal, slot \mapsto d.slot, val \mapsto d.val] : d \in m.propSV\}$,
          $r \in aVoted'[a2] \setminus aVoted[a2] \Rightarrow r.bal = m.bal$

DEF *AccInv, Send, TypeOK, Messages*
$\langle 7 \rangle$ QED BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$
$\langle 6 \rangle 3$. $(\forall b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : VotedForIn(a2, b, s, v) \Rightarrow \exists r \in aVoted[a2] : r.bal \geq b \land r.slot = s)'$
$\langle 7 \rangle$ SUFFICES ASSUME NEW $b \in \mathcal{B}'$, NEW $s \in \mathcal{S}'$, NEW $v \in \mathcal{V}'$, $VotedForIn(a2, b, s, v)'$
$\qquad$ PROVE $(\exists r \in aVoted[a2] : r.bal \geq b \land r.slot = s)'$ OBVIOUS
$\langle 7 \rangle 1$. CASE $VotedForIn(a2, b, s, v)$
$\quad \langle 8 \rangle 1$. PICK $r \in aVoted[a2] : r.bal \geq b \land r.slot = s$ BY $\langle 7 \rangle 1$ DEF *AccInv, TypeOK, Messages*
$\quad \langle 8 \rangle 2$. $m.bal \geq b$ BY $\langle 8 \rangle 1$, $\langle 5 \rangle 3$ DEF *TypeOK, Messages, AccInv*
$\quad \langle 8 \rangle 3$. CASE $\exists d \in m.propSV : d.slot = s$
$\quad \quad \langle 9 \rangle 1$. $\exists r2 \in aVoted'[a] : r2.bal = m.bal \land r2.slot = s$ BY $\langle 5 \rangle 1$, $\langle 5 \rangle 3$,
$\qquad aVoted'[a] = \{d \in aVoted[a] : \nexists d2 \in m.propSV : d.slot = d2.slot\} \cup$
$\qquad \{[bal \mapsto m.bal, slot \mapsto d.slot, val \mapsto d.val] : d \in m.propSV\}$, $\langle 8 \rangle 3$ DEF *TypeOK*
$\quad \langle 9 \rangle$ QED BY $\langle 9 \rangle 1$, $\langle 5 \rangle 3$, $\langle 8 \rangle 2$
$\quad \langle 8 \rangle 4$. CASE $\nexists d \in m.propSV : d.slot = s$ BY $\langle 8 \rangle 4$, $\langle 8 \rangle 1$, $\langle 5 \rangle 1$, $\langle 5 \rangle 3$, $r \in aVoted'[a2]$
$\quad \langle 8 \rangle$ QED BY $\langle 8 \rangle 3$, $\langle 8 \rangle 4$
$\langle 7 \rangle 2$. CASE $\neg VotedForIn(a2, b, s, v)$
$\quad \langle 8 \rangle 1$. $\exists d \in m.propSV : d.slot = s$ BY $\langle 5 \rangle 1$, $\langle 7 \rangle 2$ DEF *Send, TypeOK, Messages, VotedForIn*
$\quad \langle 8 \rangle 2$. $\exists r \in aVoted'[a2] : r.bal = m.bal \land r.slot = s$ BY $\langle 5 \rangle 1$, $\langle 8 \rangle 1$, $\langle 2 \rangle 1$, $\langle 5 \rangle 3$ DEF *Send, TypeOK,*
$\qquad$ *Messages*
$\quad \langle 8 \rangle 3$. $b = m.bal$ BY $\langle 5 \rangle 1$, $\langle 7 \rangle 2$, $\langle 5 \rangle 3$ DEF *VotedForIn, Send*
$\quad \langle 8 \rangle$ QED BY $\langle 8 \rangle 2$, $\langle 8 \rangle 3$
$\langle 7 \rangle$ QED BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$
$\langle 6 \rangle 4$. $(\forall b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : b > MaxBalInSlot(aVoted[a2], s) \Rightarrow \neg VotedForIn(a2, b, s, v))'$
$\langle 7 \rangle$ SUFFICES ASSUME NEW $b \in \mathcal{B}'$, NEW $s \in \mathcal{S}'$, NEW $v \in \mathcal{V}'$, $(VotedForIn(a2, b, s, v))'$,
$\qquad (b > MaxBalInSlot(aVoted[a2], s))'$
$\qquad$ PROVE FALSE OBVIOUS
$\langle 7 \rangle 1$. $\nexists d \in aVoted'[a2] : d.slot = s \land d.bal > MaxBalInSlot(aVoted[a2], s)'$
$\qquad$ BY *MaxBinSNoMore*, $\langle 3 \rangle 1$ DEF *TypeOK*
$\langle 7 \rangle$ QED BY $\langle 7 \rangle 1$, $\langle 6 \rangle 3$, $\exists r \in aVoted'[a2] : r.bal \geq b \land r.slot = s$, *MaxBinSType*, $\langle 3 \rangle 1$ DEF *Send,*
$\qquad$ *TypeOK, Messages*
$\langle 6 \rangle$ QED BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $\langle 6 \rangle 4$
$\langle 5 \rangle$ QED BY $\langle 5 \rangle 2$, $\langle 5 \rangle 3$
$\langle 4 \rangle 5$. CASE $\exists p \in \mathcal{P} : Preempt(p)$
$\langle 5 \rangle 1$. $\forall a \in \mathcal{A}, s \in \mathcal{S} : MaxBalInSlot(aVoted[a], s) = MaxBalInSlot(aVoted[a], s)'$
$\qquad$ BY $\langle 3 \rangle 1$, $\langle 4 \rangle 5$ DEF *Preempt, MaxBalInSlot*
$\langle 5 \rangle$ QED BY $\langle 4 \rangle 5$, $\langle 3 \rangle 1$, *PreemptVotedForInv*, $\langle 5 \rangle 1$ DEF *AccInv, TypeOK, Preempt, Send*
$\langle 4 \rangle$. QED BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 2 \rangle 1$ DEF *Next*

$\langle 3 \rangle 3$ proves $MsgInv'$ for *Next*. Each of $\langle 4 \rangle 1$-4 assumes the action of a phase and proves $MsgInv'$ for that case.

$\langle 3 \rangle 3$. $MsgInv'$
$\langle 4 \rangle 1$. CASE $\exists p \in \mathcal{P} : Phase1a(p)$ BY $\langle 4 \rangle 1$, *Phase1aVotedForInv*, $\langle 3 \rangle 1$, *SafeAtStable*, $\langle 2 \rangle 1$
$\quad$ DEF *Phase1a, MsgInv, Send, TypeOK, Messages, MsgInv1b, MsgInv2a, MsgInv2b*

$\langle 4 \rangle 2$ proves $MsgInv'$ for *Phase1b*. $\langle 5 \rangle 13,14$ conclude the proof while $\langle 5 \rangle 1$-12 prove intermediate facts. Each of $\langle 5 \rangle 2,4,12$ proves a conjunct of $MsgInv1b$ for the increment $m1$—the new $1b$ message sent in *Phase1b(a)*.

$\langle 4 \rangle 2$. ASSUME NEW $a \in \mathcal{A}$, $Phase1b(a)$ PROVE $MsgInv'$
$\langle 5 \rangle$ DEFINE $m1 \triangleq [type \mapsto \text{"1b"}, from \mapsto a, bal \mapsto m.bal, voted \mapsto aVoted[a]]$
$\langle 5 \rangle 1$. PICK $m \in msgs : Phase1b(a)!(m)$ BY $\langle 4 \rangle 2$ DEF *Phase1b*
$\langle 5 \rangle 2$. $(m1.bal \leq aBal[m1.from])'$ BY $\langle 5 \rangle 1$, $\langle 3 \rangle 1$ DEF *Phase1b, Send, TypeOK, MsgInv, Messages*
$\langle 5 \rangle 3$. $m1.voted = aVoted[m1.from]$ BY $\langle 5 \rangle 1$, $\langle 3 \rangle 1$ DEF *Phase1b, Send, TypeOK, Messages*
$\langle 5 \rangle 4$. $(\forall r \in m1.voted : VotedForIn(m1.from, r.bal, r.slot, r.val))'$
$\qquad$ BY $\langle 5 \rangle 1$, $\langle 4 \rangle 2$, *Phase1bVotedForInv*, $\langle 3 \rangle 1$, $\langle 5 \rangle 3$ DEF *TypeOK, Messages, AccInv*
$\langle 5 \rangle 5$. $(\forall b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : b > MaxBalInSlot(m1.voted, s) \Rightarrow \neg VotedForIn(m1.from, b, s, v))'$
$\qquad$ BY *Phase1bVotedForInv*, $\langle 4 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 1$ DEF *AccInv, MsgInv, Send, TypeOK, Messages*
$\langle 5 \rangle 6$. $\forall s \in \mathcal{S} : MaxBalInSlot(m1.voted, s) \in \mathcal{B} \cup \{-1\}$ BY $\langle 3 \rangle 1$, *MaxBinSType* DEF *TypeOK, Messages*
$\langle 5 \rangle 7$. $\forall s \in \mathcal{S} : \land MaxBalInSlot(m1.voted, s) + 1 \in \mathcal{B}$
$\qquad \qquad \land MaxBalInSlot(m1.voted, s) + 1 > MaxBalInSlot(m1.voted, s)$
$\quad \langle 6 \rangle$ SUFFICES ASSUME NEW $s \in \mathcal{S}$
$\qquad$ PROVE $\land MaxBalInSlot(m1.voted, s) + 1 > MaxBalInSlot(m1.voted, s)$
$\qquad \qquad \land MaxBalInSlot(m1.voted, s) + 1 \in \mathcal{B}$ OBVIOUS
$\quad \langle 6 \rangle 1$. CASE $MaxBalInSlot(m1.voted, s) = -1$ BY $\langle 6 \rangle 1$
$\quad \langle 6 \rangle 2$. CASE $MaxBalInSlot(m1.voted, s) \in \mathcal{B}$ BY $\forall x \in \mathcal{B} : x + 1 > x$, $\langle 6 \rangle 2$
$\quad \langle 6 \rangle$ QED BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 5 \rangle 6$
$\langle 5 \rangle 9$. $m1.bal \in \mathcal{B}$ BY $\langle 3 \rangle 1$ DEF *TypeOK, Messages*
$\langle 5 \rangle 10$. $\forall b \in \mathcal{B}, s \in \mathcal{S} : b \in MaxBalInSlot(m1.voted, s) + 1 .. m1.bal - 1 \Rightarrow b > MaxBalInSlot(m1.voted, s)$
$\quad \langle 6 \rangle$ SUFFICES ASSUME NEW $b \in \mathcal{B}$, NEW $s \in \mathcal{S}$, $b \in MaxBalInSlot(m1.voted, s) + 1 .. m1.bal - 1$

33

$$\text{PROVE} \quad b > MaxBalInSlot(m1.voted, s) \; \text{OBVIOUS}$$

$\langle 6 \rangle$ HIDE DEF $m1$

$\langle 6 \rangle$ DEFINE $x \triangleq MaxBalInSlot(m1.voted, s) \; y \triangleq m1.bal - 1$

$\langle 6 \rangle 1.\ x \in \mathcal{B} \cup \{-1\}$ BY $\langle 5 \rangle 6$

$\langle 6 \rangle 2.\ y \in \mathcal{B} \cup \{-1\}$ BY $\langle 5 \rangle 9$

$\langle 6 \rangle$ HIDE DEF x, y

$\langle 6 \rangle 3.$ CASE $x + 1 > y$

$\quad \langle 7 \rangle 1.\ \forall\, e \in \mathcal{B} : e \notin x + 1 \,..\, y$ BY $\langle 6 \rangle 3, \langle 6 \rangle 1, \langle 6 \rangle 2$

$\quad \langle 7 \rangle$ QED BY $\langle 6 \rangle 3, \langle 7 \rangle 1$ DEF x, y

$\langle 6 \rangle 4.$ CASE $x + 1 = y$ BY $\langle 6 \rangle 4, \langle 5 \rangle 7, \langle 3 \rangle 1, \langle 5 \rangle 9$ DEF x, y

$\langle 6 \rangle 5.$ CASE $x + 1 < y$

$\quad \langle 7 \rangle 1.\ \forall\, e \in \mathcal{B} : e \in x + 1 \,..\, y \Rightarrow e > x$ BY $\langle 6 \rangle 5, \langle 6 \rangle 1, \langle 6 \rangle 2$

$\quad \langle 7 \rangle 2.\ b \in x + 1 \,..\, y$ BY DEF x, y

$\quad \langle 7 \rangle 3.\ b > x$ BY $\langle 7 \rangle 2, \langle 7 \rangle 1$

$\quad \langle 7 \rangle$ QED BY $\langle 7 \rangle 3$ DEF x

$\langle 6 \rangle$ QED BY $\langle 6 \rangle 3, \langle 6 \rangle 4, \langle 6 \rangle 5, \langle 6 \rangle 1, \langle 6 \rangle 2$

$\langle 5 \rangle 11.\ (\forall\, b \in \mathcal{B},\, s \in \mathcal{S} : b \in MaxBalInSlot(m1.voted, s) + 1 \,..\, m1.bal - 1 \Rightarrow$
$\qquad b > MaxBalInSlot(m1.voted, s)))'$ BY $\langle 5 \rangle 10, \langle 5 \rangle 1$

$\langle 5 \rangle 12.\ (\forall\, b \in \mathcal{B},\, s \in \mathcal{S},\, v \in \mathcal{V} : b \in MaxBalInSlot(m1.voted, s) + 1 \,..\, m1.bal - 1 \Rightarrow$
$\qquad \neg VotedForIn(m1.from, b, s, v))'$

$\quad \langle 6 \rangle$ SUFFICES ASSUME NEW $b \in \mathcal{B}'$, NEW $s \in \mathcal{S}'$, NEW $v \in \mathcal{V}'$
$\qquad\qquad$ PROVE $(b \in MaxBalInSlot(m1.voted, s) + 1 \,..\, m1.bal - 1 \Rightarrow$
$\qquad\qquad\qquad \neg VotedForIn(m1.from, b, s, v))'$ OBVIOUS

$\quad \langle 6 \rangle 1.$ CASE $\nexists\, x \in \mathcal{B} : x \in (MaxBalInSlot(m1.voted, s) + 1 \,..\, m1.bal - 1)'$ BY $\langle 6 \rangle 1$

$\quad \langle 6 \rangle 2.$ CASE $\exists\, x \in \mathcal{B} : x \in (MaxBalInSlot(m1.voted, s) + 1 \,..\, m1.bal - 1)'$ BY $\langle 6 \rangle 2, \langle 5 \rangle 5, \langle 5 \rangle 11$

$\quad \langle 6 \rangle$ QED BY $\langle 6 \rangle 1, \langle 6 \rangle 2$

$\langle 5 \rangle 13$ is for **1a** message m having a higher ballot than the highest seen, thus generating a **1b** message.

$\langle 5 \rangle 13.$ CASE $m.bal > aBal[a]$

$\quad \langle 6 \rangle$ SUFFICES ASSUME NEW $m2 \in msgs'$
$\qquad\qquad$ PROVE $(\land (m2.type = \text{``1b''}) \Rightarrow MsgInv1b(m2) \land (m2.type = \text{``2a''}) \Rightarrow MsgInv2a(m2)$
$\qquad\qquad\qquad \land (m2.type = \text{``2b''}) \Rightarrow MsgInv2b(m2))'$ BY DEF $MsgInv$

Proves $MsgInv1b$ using two cases. $\langle 7 \rangle 1$ is for $m2 \in msgs$. $\langle 7 \rangle 2$ is for the increment $m2 \in msgs' \setminus msgs$.

$\quad \langle 6 \rangle 1.\ (m2.type = \text{``1b''} \Rightarrow MsgInv1b(m2))'$

$\qquad \langle 7 \rangle 1.$ CASE $m2 \in msgs$ BY $\langle 7 \rangle 1, \langle 5 \rangle 13, \langle 5 \rangle 1, Phase1bVotedForInv, \langle 4 \rangle 2$ DEF $MsgInv, MsgInv1b,$
$\qquad\qquad TypeOK, Messages$

$\qquad \langle 7 \rangle 2.$ CASE $m2 \in msgs' \setminus msgs$

$\qquad\quad \langle 8 \rangle 1.\ m2 = m1$ BY $\langle 5 \rangle 1, \langle 7 \rangle 2, \langle 5 \rangle 13$ DEF $Send$

$\qquad\quad \langle 8 \rangle$ QED BY $\langle 7 \rangle 2, \langle 5 \rangle 13, Phase1bVotedForInv, \langle 4 \rangle 2, \langle 5 \rangle 2, \langle 5 \rangle 4, \langle 5 \rangle 12, \langle 5 \rangle 1, \langle 3 \rangle 1, \langle 2 \rangle 1,$
$\qquad\qquad \langle 8 \rangle 1$ DEF $Send, TypeOK, MsgInv, Messages, MsgInv1b$

$\qquad \langle 7 \rangle$ QED BY $\langle 7 \rangle 1, \langle 7 \rangle 2$

Proves $MsgInv2a$ and $MsgInv2b$ using invariance lemma for $Phase1b$ because it does not send **2a** or **2b** messages.

$\quad \langle 6 \rangle 2.\ ((m2.type = \text{``2a''} \Rightarrow MsgInv2a(m2)) \land (m2.type = \text{``2b''} \Rightarrow MsgInv2b(m2)))'$
$\qquad$ BY $\langle 5 \rangle 13, Phase1bVotedForInv, \langle 5 \rangle 1, \langle 4 \rangle 2, \langle 3 \rangle 1, SafeAtStable, \langle 2 \rangle 1$ DEF $Send, TypeOK,$
$\qquad MsgInv, Messages, MsgInv2a, MsgInv2b$

$\quad \langle 6 \rangle$ QED BY $\langle 6 \rangle 1, \langle 6 \rangle 2$

$\langle 5 \rangle 14$ is the simple case of preemption and uses the invariance lemmas for $Phase1b$ and $SafeAt$.

$\langle 5 \rangle 14.$ CASE $\neg(m.bal > aBal[a])$ BY $\langle 5 \rangle 14, Phase1bVotedForInv, \langle 4 \rangle 2, \langle 5 \rangle 2, \langle 5 \rangle 4, \langle 5 \rangle 12, \langle 5 \rangle 1,$
$\qquad SafeAtStable, \langle 3 \rangle 1, \langle 2 \rangle 1$ DEF $Send, TypeOK, MsgInv, Messages, MsgInv1b, MsgInv2a, MsgInv2b$

$\langle 5 \rangle$ QED BY $\langle 5 \rangle 13, \langle 5 \rangle 14$

$\langle 4 \rangle 3$ proves $MsgInv'$ for $Phase2a$. Each of $\langle 5 \rangle 4$-6 proves a conjunct of $MsgInv$.

$\langle 4 \rangle 3.$ ASSUME NEW $p \in \mathcal{P},\, Phase2a(p)$ PROVE $MsgInv'$

$\langle 5 \rangle$ SUFFICES ASSUME NEW $m \in msgs'$
$\qquad\qquad$ PROVE $(\land (m.type = \text{``1b''} \Rightarrow MsgInv1b(m)) \land (m.type = \text{``2a''} \Rightarrow MsgInv2a(m))$
$\qquad\qquad\qquad \land (m.type = \text{``2b''} \Rightarrow MsgInv2b(m)))'$ BY DEF $MsgInv$

$\langle 5 \rangle$ DEFINE $b \triangleq pBal[p]$

$\langle 5 \rangle 1.$ PICK $Q \in \mathcal{Q},\, S \in$ SUBSET $\{m2 \in msgs : (m2.type = \text{``1b''}) \land (m2.bal = b)\} :$
$\qquad \land \forall\, a \in Q : \exists\, m2 \in S : m2.from = a$
$\qquad \land Send([type \mapsto \text{``2a''},\, bal \mapsto b,\, from \mapsto p,\, propSV \mapsto PropSV(\text{UNION } \{m2.voted : m2 \in S\})])$
$\quad$ BY $\langle 4 \rangle 3$ DEF $Phase2a$

$\langle 5 \rangle 2.\ b = pBal'[p] \land b \in \mathcal{B}$ BY $\langle 4 \rangle 3$ DEF $Phase2a, TypeOK$

$\langle 5 \rangle 3.\ \forall\, m2 \in msgs' \setminus msgs : m2.type = \text{``2a''} \land m2.bal = b$ BY $\langle 5 \rangle 1$ DEF $Send$

$\langle 5 \rangle 4$ proves $MsgInv1b'$. It uses invariance lemma for $Phase2a$, because $Phase2a$ does not send **1b** messages.

34

⟨5⟩4. $(m.type = \text{"1b"} \Rightarrow MsgInv1b(m))'$
  ⟨6⟩ SUFFICES ASSUME $(m.type = \text{"1b"})'$ PROVE $MsgInv1b(m)'$ OBVIOUS
  ⟨6⟩1. $(m.bal \leq aBal[m.from])'$ BY ⟨4⟩3, ⟨5⟩3, ⟨3⟩1, $Phase2aVotedForInv$ DEF $TypeOK$, $Messages$,
    $MsgInv$, $Phase2a$, $MsgInv1b$
  ⟨6⟩2. $(\forall\, r \in m.voted : VotedForIn(m.from, r.bal, r.slot, r.val))'$ BY ⟨4⟩3, ⟨5⟩3, ⟨3⟩1,
    $Phase2aVotedForInv$ DEF $TypeOK$, $Messages$, $MsgInv$, $MsgInv1b$
  ⟨6⟩3. $\forall\, s \in \mathcal{S} : MaxBalInSlot(m.voted, s) = MaxBalInSlot(m.voted, s)'$ BY DEF $MaxBalInSlot$
  ⟨6⟩4. $(\forall\, b2 \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : b2 \in MaxBalInSlot(m.voted, s) + 1 \,.\,.\, m.bal - 1 \Rightarrow$
    $\neg\, VotedForIn(m.from, b2, s, v))'$
      BY ⟨4⟩3, ⟨5⟩3, ⟨3⟩1, $Phase2aVotedForInv$, ⟨6⟩3 DEF $TypeOK$, $Messages$, $MsgInv$, $MsgInv1b$
  ⟨6⟩ QED BY ⟨6⟩1, ⟨6⟩2, ⟨6⟩4 DEF $MsgInv1b$

⟨5⟩5 proves $MsgInv2a'$. Each of ⟨6⟩2-4 proves a conjunct of $MsgInv2a$ using the increment approach. The increment is $m2$ in ⟨8⟩2 of ⟨7⟩9.

⟨5⟩5. $(m.type = \text{"2a"} \Rightarrow MsgInv2a(m))'$
  ⟨6⟩ SUFFICES ASSUME $(m.type = \text{"2a"})'$ PROVE $MsgInv2a(m)'$ OBVIOUS
  ⟨6⟩ DEFINE $VS \triangleq$ UNION $\{m2.voted : m2 \in S\}$
  ⟨6⟩1. $\forall\, a \in Q : aBal[a] \geq b$ BY ⟨5⟩1, ⟨3⟩2, ⟨3⟩1 DEF $MsgInv$, $TypeOK$, $Messages$, $MsgInv1b$
  ⟨6⟩2. $(\forall\, d \in m.propSV : SafeAt(m.bal, d.slot, d.val))'$
    ⟨7⟩1. $\forall\, d \in [slot : UnusedS(VS), val : \mathcal{V}] \setminus \emptyset : SafeAt(b, d.slot, d.val)$
      ⟨8⟩ SUFFICES ASSUME NEW $d \in [slot : UnusedS(VS), val : \mathcal{V}] \setminus \emptyset$ PROVE $SafeAt(b, d.slot, d.val)$
        OBVIOUS
      ⟨8⟩1. $\forall\, m2 \in S : \nexists\, d2 \in m2.voted : d.slot = d2.slot$ BY DEF $UnusedS$
      ⟨8⟩2. $\forall\, m2 \in S : MaxBalInSlot(m2.voted, d.slot) + 1 = 0$ BY ⟨8⟩1 DEF $MaxBalInSlot$
      ⟨8⟩3. $\forall\, m2 \in S, b2 \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : b2 \in MaxBalInSlot(m2.voted, s) + 1 \,.\,.\, m2.bal - 1 \Rightarrow$
        $\neg\, VotedForIn(m2.from, b2, s, v)$ BY DEF $MsgInv$, $MsgInv1b$
      ⟨8⟩4. $\forall\, v \in \mathcal{V}, b2 \in \mathcal{B}, a \in Q : b2 \in 0 \,.\,.\, b - 1 \Rightarrow \neg\, VotedForIn(a, b2, d.slot, v)$
        BY ⟨5⟩1, ⟨8⟩2, ⟨8⟩3 DEF $UnusedS$, $TypeOK$, $Messages$
      ⟨8⟩ QED BY ⟨8⟩4, ⟨3⟩1, ⟨6⟩1 DEF $SafeAt$, $NewSV$, $UnusedS$, $WontVoteIn$, $TypeOK$, $Messages$
    ⟨7⟩2. $\forall\, d \in [slot : UnusedS(VS), val : \mathcal{V}] \setminus \emptyset : SafeAt(b, d.slot, d.val)'$ BY ⟨7⟩1, $SafeAtStable$,
      ⟨3⟩1, ⟨2⟩1, ⟨5⟩2 DEF $NewSV$, $UnusedS$, $TypeOK$, $Messages$
    ⟨7⟩3. $\forall\, d \in NewSV(VS) : SafeAt(b, d.slot, d.val)'$ BY ⟨7⟩2, $Misc$ DEF $NewSV$
    ⟨7⟩4. $\forall\, d \in MaxBSV(VS) : SafeAt(b, d.slot, d.val)$
      ⟨8⟩ SUFFICES ASSUME NEW $d \in MaxBSV(VS)$, NEW $b2 \in \mathcal{B}$, $b2 \in 0 \,.\,.\, (b - 1)$
          PROVE $\exists\, Q2 \in \mathcal{Q} : \forall\, a \in Q2 : \vee\, VotedForIn(a, b2, d.slot, d.val)$
                           $\vee\, WontVoteIn(a, b2, d.slot)$ BY DEF $SafeAt$
      ⟨8⟩ DEFINE $max \triangleq MaxBalInSlot(VS, d.slot)$
      ⟨8⟩ USE DEF $MaxBSV$
      ⟨8⟩1. $max \in \mathcal{B}$ BY $MaxBinSType$, $MaxBinSNoSlot$ DEF $TypeOK$, $Messages$
      ⟨8⟩2. $\forall\, m2 \in S : MaxBalInSlot(m2.voted, d.slot) \leq max$
        BY $\forall\, m2 \in S : m2.voted \subseteq VS$, $MaxBinSSubsets$ DEF $MaxBSV$, $TypeOK$, $Messages$
      ⟨8⟩3. $\nexists\, d2 \in VS : (d2.bal > d.bal \wedge d2.slot = d.slot)$
        BY $\forall\, d2 \in VS : \neg(\neg(d2.bal \leq d.bal) \wedge d2.slot = d.slot)$ DEF $MaxBSV$, $TypeOK$, $Messages$
      ⟨8⟩4. $VS \subseteq [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}]$ BY ⟨3⟩1 DEF $TypeOK$, $Messages$
      ⟨8⟩5. $max = d.bal$
        ⟨9⟩ SUFFICES ASSUME $max \neq d.bal$ PROVE   FALSE OBVIOUS
        ⟨9⟩1. CASE $max > d.bal$
          ⟨10⟩ HIDE DEF VS
          ⟨10⟩1. $\exists\, d2 \in VS : d2.bal = max \wedge d2.slot = d.slot$ BY ⟨8⟩4, ⟨8⟩1, $MaxBinSExists$
          ⟨10⟩ QED BY ⟨10⟩1, ⟨8⟩3, ⟨9⟩1
        ⟨9⟩2. CASE $max < d.bal$ BY $MaxBinSNoMore$, ⟨9⟩2 DEF $MaxBSV$, $TypeOK$, $Messages$
        ⟨9⟩ QED BY ⟨9⟩1, ⟨9⟩2, ⟨8⟩1 DEF $\mathcal{B}$, $TypeOK$, $Messages$
      ⟨8⟩6. CASE $b2 \in max + 1 \,.\,.\, b - 1$
        ⟨9⟩ HIDE DEF max
        ⟨9⟩1. $\forall\, m2 \in S, b3 \in \mathcal{B}, v \in \mathcal{V} : b3 \in MaxBalInSlot(m2.voted, d.slot) + 1 \,.\,.\, b - 1 \Rightarrow$
          $\neg\, VotedForIn(m2.from, b3, d.slot, v)$
          BY DEF $MsgInv$, $TypeOK$, $Messages$, $MsgInv1b$
        ⟨9⟩2. $\forall\, m2 \in S, v \in \mathcal{V} : \neg\, VotedForIn(m2.from, b2, d.slot, v)$
          BY ⟨8⟩6, ⟨9⟩1, ⟨8⟩2, ⟨8⟩1, $MaxBinSType$ DEF $TypeOK$, $Messages$, $Send$
        ⟨9⟩ QED BY ⟨5⟩1, ⟨8⟩6, ⟨6⟩1, ⟨9⟩2 DEF $MsgInv$, $MsgInv1b$, $TypeOK$, $Messages$, $WontVoteIn$,
        $MaxBSV$
      ⟨8⟩7. CASE $b2 = max$
        ⟨9⟩1. $\exists\, a \in \mathcal{A}, m2 \in S : m2.from = a \wedge \exists\, d2 \in m2.voted :$
          $d2.bal = d.bal \wedge d2.slot = d.slot \wedge d2.val = d.val$
          BY DEF $MaxBalInSlot$, $TypeOK$, $Messages$
        ⟨9⟩2. $\exists\, a \in \mathcal{A} : VotedForIn(a, b2, d.slot, d.val)$
          BY ⟨8⟩7, ⟨9⟩1, ⟨8⟩5 DEF $MsgInv$, $TypeOK$, $Messages$, $MsgInv1b$
        ⟨9⟩3. $\forall\, q \in Q, v2 \in \mathcal{V} : VotedForIn(q, b2, d.slot, v2) \Rightarrow v2 = d.val$

35

BY $\langle 9 \rangle 2$, *VotedOnce*, *QuorumAssumption* DEF *TypeOK*, *Messages*

$\langle 9 \rangle 4$. $\forall q \in Q : aBal[q] > b2$ BY $\langle 5 \rangle 1$ DEF *MsgInv*, *TypeOK*, *Messages*, *MsgInv1b*

$\langle 9 \rangle$ QED BY $\langle 8 \rangle 7$, $\langle 9 \rangle 3$, $\langle 9 \rangle 4$ DEF *WontVoteIn*

$\langle 8 \rangle 8$. CASE $b2 \in 0 \mathrel{..} max - 1$

$\langle 9 \rangle 1$. $\exists a \in \mathcal{A} : VotedForIn(a, d.bal, d.slot, d.val)$

BY $\langle 8 \rangle 8$, $\langle 8 \rangle 2$ DEF *MsgInv*, *TypeOK*, *Messages*, *MsgInv1b*

$\langle 9 \rangle 2$. *SafeAt*$(d.bal, d.slot, d.val)$ BY $\langle 9 \rangle 1$, *VotedInv* DEF *TypeOK*, *Messages*

$\langle 9 \rangle$ QED BY $\langle 8 \rangle 8$, $\langle 9 \rangle 2$, $\langle 8 \rangle 5$ DEF *SafeAt*, *MsgInv*, *TypeOK*, *Messages*, *MaxBalInSlot*

$\langle 8 \rangle$ QED BY $\langle 8 \rangle 6$, $\langle 8 \rangle 7$, $\langle 8 \rangle 8$, $\langle 8 \rangle 1$

$\langle 7 \rangle 5$. *MaxBSV*$(VS) \subseteq [bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}]$ BY DEF *MaxBSV*, *TypeOK*, *Messages*

$\langle 7 \rangle 6$. $\forall d \in MaxBSV(VS) : SafeAt(b, d.slot, d.val)'$ BY $\langle 7 \rangle 4$, *SafeAtStable*, $\langle 3 \rangle 1$, $\langle 7 \rangle 5$, $\langle 2 \rangle 1$, $\langle 5 \rangle 2$

$\langle 7 \rangle 7$. $\forall d \in MaxSV(VS) : SafeAt(b, d.slot, d.val)'$ BY $\langle 7 \rangle 6$, $\langle 7 \rangle 5$ DEF *MaxSV*

$\langle 7 \rangle 8$. $\forall d \in MaxSV(VS) \cup NewSV(VS) : SafeAt(b, d.slot, d.val)'$ BY $\langle 7 \rangle 7$, $\langle 7 \rangle 3$

$\langle 7 \rangle 9$. $(\forall m2 \in msgs : m2.type = \text{``2a''} \Rightarrow \forall d \in m2.propSV : SafeAt(m2.bal, d.slot, d.val))'$

$\langle 8 \rangle$ SUFFICES ASSUME NEW $m2 \in msgs'$, $(m2.type = \text{``2a''})'$, NEW $d \in m2.propSV$

PROVE $(SafeAt(m2.bal, d.slot, d.val))'$ OBVIOUS

$\langle 8 \rangle 1$. CASE $m2 \in msgs$ BY $\langle 3 \rangle 1$, *SafeAtStable*, $\langle 8 \rangle 1$, $\langle 2 \rangle 1$ DEF *MsgInv*, *MsgInv2a*, *Messages*, *TypeOK*

$\langle 8 \rangle 2$. CASE $m2 \in msgs' \setminus msgs$

$\langle 9 \rangle 1$. *SafeAt*$(m2.bal, d.slot, d.val)'$ BY $\langle 7 \rangle 8$, $\langle 8 \rangle 2$, $\langle 5 \rangle 1$, $\langle 5 \rangle 2$ DEF *Send*, *PropSV*

$\langle 9 \rangle$ QED BY $\langle 3 \rangle 1$, $\langle 9 \rangle 1$ DEF *Send*, *TypeOK*, *Messages*

$\langle 8 \rangle$ QED BY $\langle 8 \rangle 1$, $\langle 8 \rangle 2$

$\langle 7 \rangle$ QED BY $\langle 7 \rangle 9$

The increment is $m2$ in $\langle 7 \rangle 1$.

$\langle 6 \rangle 3$. $(\forall d1, d2 \in m.propSV : d1.slot = d2.slot \Rightarrow d1 = d2)'$

$\langle 7 \rangle 1$. $\forall m2 \in msgs' \setminus msgs : \forall d1, d2 \in m2.propSV : d1.slot = d2.slot \Rightarrow d1 = d2$

$\langle 8 \rangle 1$. $VS \in$ SUBSET $[bal : \mathcal{B}, slot : \mathcal{S}, val : \mathcal{V}]$ BY DEF *Messages*, *TypeOK*

$\langle 8 \rangle 2$. $\forall r1, r2 \in MaxBSV(VS) : r1.slot = r2.slot \Rightarrow r1.bal = r2.bal$ BY $\langle 8 \rangle 1$ DEF *MaxBSV*

$\langle 8 \rangle 3$. *MaxBSV*$(VS) \subseteq VS$ BY $\langle 8 \rangle 1$ DEF *MaxBSV*

$\langle 8 \rangle 4$. $\forall r1, r2 \in MaxBSV(VS) : r1.bal = r2.bal \wedge r1.slot = r2.slot \Rightarrow r1.val = r2.val$

BY $\langle 8 \rangle 3$, *VotedUnion*

$\langle 8 \rangle 5$. $\forall r1, r2 \in MaxBSV(VS) : r1.slot = r2.slot \Rightarrow r1.bal = r2.bal \wedge r1.val = r2.val$

BY $\langle 8 \rangle 4$, $\langle 8 \rangle 2$, $\langle 8 \rangle 3$, $\langle 8 \rangle 1$

$\langle 8 \rangle 6$. $\forall r1, r2 \in MaxSV(VS) : r1.slot = r2.slot \Rightarrow r1 = r2$ BY $\langle 8 \rangle 5$ DEF *MaxSV*

$\langle 8 \rangle$ QED BY $\langle 8 \rangle 6$, *Misc*, $\langle 5 \rangle 1$ DEF *PropSV*, *Send*

$\langle 7 \rangle$ QED BY $\langle 7 \rangle 1$ DEF *MsgInv*, *MsgInv2a*

The increment is $m1, m2$ in $\langle 7 \rangle 2$.

$\langle 6 \rangle 4$. $(\forall m2 \in msgs : (m2.type = \text{``2a''} \wedge m2.bal = m.bal) \Rightarrow (m2 = m))'$

$\langle 7 \rangle 1$. $\forall m2 \in msgs : (m2.type = \text{``2a''}) \Rightarrow (m2.bal \neq b)$ BY $\langle 4 \rangle 3$ DEF *Phase2a*

$\langle 7 \rangle 2$. $\forall m1, m2 \in msgs' \setminus msgs : m1 = m2$ BY $\langle 4 \rangle 3$ DEF *Phase2a*, *Send*

$\langle 7 \rangle$ QED BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, $\langle 5 \rangle 3$, $\langle 2 \rangle 1$, $\langle 3 \rangle 1$ DEF *MsgInv*, *MsgInv2a*

$\langle 6 \rangle$ QED BY $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $\langle 6 \rangle 4$ DEF *MsgInv2a*

$\langle 5 \rangle 6$. $((m.type = \text{``2b''}) \Rightarrow MsgInv2b(m))'$ BY $\langle 5 \rangle 3$, $\langle 5 \rangle 1$, $m.type = \text{``2b''} \Rightarrow m \in msgs$,

$\langle 3 \rangle 1$, $\langle 4 \rangle 3$ DEF *TypeOK*, *Messages*, *MsgInv*, *Phase2a*, *Send*, *MsgInv2b*

$\langle 5 \rangle$ QED BY $\langle 5 \rangle 4$, $\langle 5 \rangle 5$, $\langle 5 \rangle 6$

$\langle 4 \rangle 4$ proves *MsgInv'* for *Phase2b*. Each of $\langle 5 \rangle$2-4 proves a conjunct of *MsgInv*.

$\langle 4 \rangle 4$. ASSUME NEW $a \in \mathcal{A}$, *Phase2b*(a) PROVE *MsgInv'*

$\langle 5 \rangle$ SUFFICES ASSUME NEW $m \in msgs'$

PROVE $(\wedge (m.type = \text{``1b''}) \Rightarrow MsgInv1b(m) \wedge (m.type = \text{``2a''}) \Rightarrow MsgInv2a(m)$

$\wedge (m.type = \text{``2b''}) \Rightarrow MsgInv2b(m))'$ BY DEF *MsgInv*

$\langle 5 \rangle 1$. PICK $m1 \in msgs : Phase2b(a)!(m1)$ BY $\langle 4 \rangle 4$ DEF *Phase2b*

$\langle 5 \rangle 2$ proves *MsgInv1b'* for *Phase2b*. Invariance lemmas do not apply because the 3rd conjunct in *MsgInv1b* quantifies over 2b messages negatively — *VotedForIn*(a, b, s, v) means acceptor a has sent a 2b message voting $\langle b, s, v \rangle$.

$\langle 5 \rangle 2$. $((m.type = \text{``1b''}) \Rightarrow MsgInv1b(m))'$

$\langle 6 \rangle$ SUFFICES ASSUME $(m.type = \text{``1b''})'$ PROVE *MsgInv1b*$(m)'$ OBVIOUS

$\langle 6 \rangle 1$. $(m.bal \leq aBal[m.from] \wedge \forall r \in m.voted : VotedForIn(m.from, r.bal, r.slot, r.val))'$

BY $\langle 5 \rangle 1$, $\langle 3 \rangle 1$, $\langle 4 \rangle 4$, *Phase2bVotedForInv* DEF *MsgInv*, *MsgInv1b*, *TypeOK*,

Messages, *Send*

$\langle 6 \rangle 2$. $(\forall b \in \mathcal{B}, s \in \mathcal{S}, v \in \mathcal{V} : b \in MaxBalInSlot(m.voted, s) + 1 \mathrel{..} m.bal - 1 \Rightarrow$

$\neg VotedForIn(m.from, b, s, v))'$

$\langle 7 \rangle$ SUFFICES ASSUME NEW $b \in \mathcal{B}'$, NEW $s \in \mathcal{S}'$, NEW $v \in \mathcal{V}'$,

$(b \in MaxBalInSlot(m.voted, s) + 1 \mathrel{..} m.bal - 1)'$

PROVE $(\neg VotedForIn(m.from, b, s, v))'$ OBVIOUS

$\langle 7 \rangle 1$. $\neg VotedForIn(m.from, b, s, v)$ BY $\langle 5 \rangle 1$ DEF *Send*, *MsgInv*, *TypeOK*, *Messages*, *MsgInv1b*

$\langle 7 \rangle 2$. CASE $m.from \neq a \vee \neg (m1.bal \geq aBal[a])$ BY $\langle 5 \rangle 1$, $\langle 3 \rangle 1$, $\langle 7 \rangle 2$, $\langle 7 \rangle 1$ DEF *VotedForIn*,

$TypeOK$, $Messages$, $Send$

$\langle 7 \rangle 3$. CASE $m.from = a \wedge (m1.bal \geq aBal[a])$

$\langle 8 \rangle 1$. $\forall\, m2 \in msgs' \setminus msgs : m2.bal = m1.bal$ BY $\langle 5 \rangle 1$, $\langle 7 \rangle 3$, $\langle 6 \rangle 1$, $\langle 7 \rangle 3$ DEF $Send$, $TypeOK$

$\langle 8 \rangle 2$. $\forall\, m2 \in msgs' \setminus msgs : m2.bal \neq b$ BY $\langle 5 \rangle 1$, $\langle 7 \rangle 3$, $\langle 6 \rangle 1$, $\langle 7 \rangle 3$, $\langle 8 \rangle 1$ DEF $TypeOK$, $Messages$

$\langle 8 \rangle$ QED BY $\langle 7 \rangle 3$, $\langle 7 \rangle 1$, $\langle 8 \rangle 2$ DEF $VotedForIn$, $TypeOK$, $Messages$

$\langle 7 \rangle$ QED BY $\langle 7 \rangle 2$, $\langle 7 \rangle 3$ DEF $TypeOK$, $Messages$

$\langle 6 \rangle$ QED BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$ DEF $MsgInv1b$

$\langle 5 \rangle 3$. $((m.type = \text{``2a''}) \Rightarrow MsgInv2a(m))'$ BY $SafeAtStable$, $\langle 3 \rangle 1$, $\langle 4 \rangle 4$, $\langle 2 \rangle 1$ DEF $MsgInv$, $MsgInv2a$, $TypeOK$, $Messages$, $Phase2b$, $Send$

$\langle 5 \rangle 4$ proves $MsgInv2b'$. It uses two cases: the second case, $\langle 6 \rangle 2$, is for the increment m.

$\langle 5 \rangle 4$. $((m.type = \text{``2b''}) \Rightarrow MsgInv2b(m))'$

$\langle 6 \rangle 1$. CASE $\neg(m1.bal \geq aBal[a]) \vee m \in msgs$ BY $\langle 5 \rangle 1$, $\langle 3 \rangle 1$, $\langle 6 \rangle 1$ DEF $TypeOK$, $Messages$, $Send$, $MsgInv$, $MsgInv2b$

$\langle 6 \rangle 2$. CASE $m1.bal \geq aBal[a] \wedge m \in msgs' \setminus msgs$ BY $\langle 5 \rangle 1$, $\langle 3 \rangle 1$, $\langle 6 \rangle 2$ DEF $TypeOK$, $Send$, $MsgInv2b$

$\langle 6 \rangle$ QED BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$

$\langle 5 \rangle$ QED BY $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$ DEF $MsgInv2b$

$\langle 4 \rangle 5$ proves $MsgInv'$ for $Preempt$. It uses the invariance lemma for $Preempt$, since $Preempt$ does not send messages.

$\langle 4 \rangle 5$. CASE $\exists\, p \in \mathcal{P} : Preempt(p)$ BY $\langle 4 \rangle 5$, $PreemptVotedForInv$, $\langle 3 \rangle 1$, $SafeAtStable$, $\langle 2 \rangle 1$ DEF $Preempt$, $MsgInv$, $TypeOK$, $Messages$, $MsgInv1b$, $MsgInv2a$, $MsgInv2b$

$\langle 4 \rangle$ QED BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 2 \rangle 1$ DEF $Next$

$\langle 3 \rangle$ QED BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$ DEF Inv, $vars$, $Next$

$\langle 2 \rangle 2$. CASE UNCHANGED $vars$ BY $\langle 2 \rangle 2$ DEF $vars$, Inv, $TypeOK$, $AccInv$, $MsgInv$, $VotedForIn$, $SafeAt$, $WontVoteIn$, $MaxBalInSlot$, $MsgInv1b$, $MsgInv2a$, $MsgInv2b$

$\langle 2 \rangle$ QED BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$

$\langle 1 \rangle$ QED BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, PTL DEF $Spec$

$Safety$ asserts that $Spec$ implies that $Safe$ always holds.

THEOREM $Safety \triangleq Spec \Rightarrow \Box Safe$

$\langle 1 \rangle$ USE DEF $\mathcal{B}$

$\langle 1 \rangle 1$. $Inv \Rightarrow Safe$

$\langle 2 \rangle$ SUFFICES ASSUME Inv, NEW $v1 \in \mathcal{V}$, NEW $v2 \in \mathcal{V}$, NEW $s \in \mathcal{S}$, NEW $b1 \in \mathcal{B}$, NEW $b2 \in \mathcal{B}$, $ChosenIn(b1, s, v1)$, $ChosenIn(b2, s, v2)$, $b1 \leq b2$

PROVE $v1 = v2$ BY DEF $Safe$, $Chosen$

$\langle 2 \rangle 1$. CASE $b1 = b2$

$\langle 3 \rangle 1$. $\exists\, a \in \mathcal{A} : VotedForIn(a, b1, s, v1) \wedge VotedForIn(a, b1, s, v2)$

BY $\langle 2 \rangle 1$, $QuorumAssumption$ DEF $ChosenIn$

$\langle 3 \rangle$ QED BY $\langle 3 \rangle 1$, $VotedOnce$ DEF Inv

$\langle 2 \rangle 2$. CASE $b1 < b2$

$\langle 3 \rangle 1$. $SafeAt(b2, s, v2)$ BY $VotedInv$, $QuorumAssumption$ DEF $ChosenIn$, Inv

$\langle 3 \rangle 2$. PICK $Q1 \in \mathcal{Q} : \forall\, a \in Q1 : VotedForIn(a, b1, s, v1)$ BY DEF $ChosenIn$

$\langle 3 \rangle 3$. PICK $Q2 \in \mathcal{Q} : \forall\, a \in Q2 : VotedForIn(a, b1, s, v2) \vee WontVoteIn(a, b1, s)$ BY $\langle 3 \rangle 1$, $\langle 2 \rangle 2$ DEF $SafeAt$

$\langle 3 \rangle$ QED BY $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $QuorumAssumption$, $VotedOnce$ DEF $WontVoteIn$, Inv

$\langle 2 \rangle$ QED BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$

$\langle 1 \rangle$ QED BY $Invariant$, $\langle 1 \rangle 1$, PTL