MODULE *VoteProof*

This is a high-level consensus algorithm in which a set of processes called *acceptors* cooperatively choose a value. The algorithm uses numbered ballots, where a ballot is a round of voting. Acceptors cast votes in ballots, casting at most one vote per ballot. A value is chosen when a large enough set of acceptors, called a *quorum*, have all voted for the same value in the same ballot.

Ballots are not executed in order. Different acceptors may be concurrently performing actions for different ballots.

EXTENDS *Integers*, *NaturalsInduction*, *FiniteSets*, *FiniteSetTheorems*, *WellFoundedInduction*, *TLC*, *TLAPS*

CONSTANT *Value*, — As in module *Consensus*, the set of choosable values.
Acceptor, — The set of all acceptors.
Quorum — The set of all quorums.

The following assumption asserts that a quorum is a set of acceptors, and the fundamental assumption we make about quorums: any two quorums have a non-empty intersection.

ASSUME $QA \triangleq \land \forall Q \in Quorum : Q \subseteq Acceptor$
$\qquad\qquad\quad \land \forall Q1, Q2 \in Quorum : Q1 \cap Q2 \neq \{\}$

THEOREM $QuorumNonEmpty \triangleq \forall Q \in Quorum : Q \neq \{\}$
PROOF BY QA

Ballot is the set of all ballot numbers. For simplicity, we let it be the set of natural numbers. However, we write *Ballot* for that set to make it clear what the function of those natural numbers are.

The algorithm and its refinements work with *Ballot* any set with minimal element 0, -1 not an element of *Ballot*, and a well-founded total order $<$ on $Ballot \cup \{-1\}$ with minimal element -1, and $0 < b$ for all non-zero b in *Ballot*. In the proof, any set of the form $i \,..\, j$ must be replaced by the set of all elements b in $Ballot \cup \{-1\}$ with $i \leq b \leq j$, and $i \,..\, (j-1)$ by the set of such b with $i \leq b < j$.

$Ballot \triangleq Nat$

In the algorithm, each acceptor can cast one or more votes, where each vote cast by an acceptor has the form $\langle b, v \rangle$ indicating that the acceptor has voted for value v in ballot b. A value is chosen if a quorum of acceptors have voted for it in the same ballot.

The algorithm uses two variables, *votes* and *maxBal*, both arrays indexed by acceptor. Their meanings are:

- $votes[a]$ – The set of votes cast by acceptor a.
- $maxBal[a]$ – The number of the highest-numbered ballot in which a has cast a vote, or -1 if it has not yet voted.

The algorithm does not let acceptor a vote in any ballot less than $maxBal[a]$.

We specify our algorithm by the following *PlusCal* algorithm. The specification *Spec* defined by this algorithm describes only the safety properties of the algorithm. In other words, it specifies what steps the algorithm may take. It does not require that any (non-stuttering) steps be taken. We prove that this specification *Spec* implements the specification *Spec* of module *Consensus* under a refinement mapping defined below. This shows that the safety properties of the voting algorithm (and hence the algorithm with additional liveness requirements) imply the safety properties of the *Consensus* specification. Liveness is discussed later.

****

--algorithm *Voting* **{**
variables $votes = [a \in Acceptor \mapsto \{\}]$,
$maxBal = [a \in Acceptor \mapsto -1]$ **;**
define {

We now define the operator *SafeAt* so $SafeAt(b, v)$ is function of the state that equals TRUE if no value other than v has been chosen or can ever be chosen in the future (because the values of the variables votes and *maxBal* are such that the algorithm does not allow enough acceptors to vote for it). We say that value v is safe at ballot number b iff $Safe(b, v)$ is true. We define Safe in terms of the following two operators.

Note: This definition is weaker than would be necessary to allow a refinement of ordinary *Paxos* consensus, since it allows different quorums to "cooperate" in determining safety at b. This is used in algorithms like Vertical *Paxos* that are designed to allow reconfiguration within a single consensus instance, but not in ordinary *Paxos*. See

AUTHOR = "Leslie *Lamport* and Dahlia *Malkhi* and *Lidong Zhou* ",
TITLE = "Vertical *Paxos* and Primary-Backup Replication",
Journal = "*ACM SIGACT News* (Distributed Computing Column)",
editor = {*Srikanta Tirthapura and Lorenzo Alvisi*},
booktitle = {*PODC*},
publisher = {*ACM*}, *YEAR* = 2009, *PAGES* = "312–313"

$VotedFor(a, b, v) \triangleq \langle b, v \rangle \in votes[a]$

True iff acceptor a has voted for v in ballot b.

$DidNotVoteIn(a, b) \triangleq \forall v \in Value : \neg VotedFor(a, b, v)$

We now define *SafeAt*. We define it recursively. The nicest definition is

```
RECURSIVE SafeAt(_, _)
SafeAt(b, v) ≜
  ∨ b = 0
  ∨ ∃ Q ∈ Quorum :
      ∧ ∀ a ∈ Q : maxBal[a] > b
      ∧ ∃ c ∈ −1 .. (b − 1) :
          ∧ (c ≠ −1) ⇒ ∧ SafeAt(c, v)
                        ∧ ∀ a ∈ Q : ∀ w ∈ Value :
                              VotedFor(a, c, w) ⇒ (w = v)
        ∧ ∀ d ∈ (c + 1) .. (b − 1), a ∈ Q : DidNotVoteIn(a, d)
```

However, *TLAPS* does not currently support recursive operator definitions. We therefore define it as follows using a recursive function definition.

$SafeAt(b, v) \triangleq$
LET $SA[bb \in Ballot] \triangleq$

This recursively defines $SA[bb]$ to equal $SafeAt(bb, v)$.

$$
\begin{array}{l}
\lor\ bb = 0 \\
\lor\ \exists\, Q \in Quorum : \\
\quad \land\ \forall\, a \in Q \quad : maxBal[a] \geq bb \\
\quad \land\ \exists\, c \in -1 \,..\, (bb - 1) : \\
\qquad \land\ (c \neq -1) \Rightarrow \land\ SA[c] \\
\qquad \qquad \qquad \qquad \land\ \forall\, a \in Q : \\
\qquad \qquad \qquad \qquad \quad \forall\, w \in Value : \\
\qquad \qquad \qquad \qquad \qquad VotedFor(a, c, w) \Rightarrow (w = v) \\
\qquad \land\ \forall\, d \in (c + 1) \,..\, (bb - 1),\ a \in Q : DidNotVoteIn(a, d)
\end{array}
$$

IN $SA[b]$

}

There are two possible actions that an acceptor can perform, each defined by a macro. In these macros, *self* is the acceptor that is to perform the action. The first action, $IncreaseMaxBal(b)$ allows acceptor *self* to set $maxBal[self]$ to b if b is greater than the current value of $maxBal[self]$.

macro $IncreaseMaxBal(\ b\ )$ **{**
  when $b > maxBal[self]$ **;**
  $maxBal[self] := b$
  }

Action $VoteFor(b, v)$ allows acceptor *self* to vote for value v in ballot b if its *when* condition is satisfied.

macro $VoteFor(\ b, v\ )$ **{**

$$
\begin{array}{l}
\textbf{when}\ \land\ maxBal[self] \leq b \\
\qquad \land\ DidNotVoteIn(self, b) \\
\qquad \land\ \forall\, p \in Acceptor \setminus \{self\} : \\
\qquad \quad \forall\, w \in Value : VotedFor(p, b, w) \Rightarrow (w = v) \\
\qquad \land\ SafeAt(b, v)\ ; \\
votes[self] \quad := votes[self] \cup \{\langle b, v \rangle\}\ ; \\
maxBal[self] := b
\end{array}
$$

}

The following process declaration asserts that every process *self* in the set *Acceptor* executes its body, which loops forever nondeterministically choosing a *Ballot* b and executing either an $IncreaseMaxBal(b)$ action or nondeterministically choosing a value v and executing a $VoteFor(b, v)$ action. The single label indicates that an entire execution of the body of the *while* loop is performed as a single atomic action.

From this intuitive description of the process declaration, one might think that a process could be deadlocked by choosing a ballot b in which neither an $IncreaseMaxBal(b)$ action nor any $VoteFor(b, v)$ action is enabled. An examination of the TLA+ translation (and an elementary knowledge of the meaning of existential quantification) shows that this is not the case. You can think of all possible choices of b and of v being examined simultaneously, and one of the choices for which a step is possible being made.

process ($acceptor \in Acceptor$) **{**
  acc : **while** (TRUE) **{**
    with ($b \in Ballot$) **{**

```
            either IncreaseMaxBal(b)
            or     with ( v ∈ Value ) { VoteFor(b, v) }
      }
    }
   }
}
```

The following is the TLA+ specification produced by the translation. Blank lines, produced by the translation because of the comments, have been deleted.
****

BEGIN TRANSLATION

```
VARIABLES votes, maxBal

define statement
VotedFor(a, b, v) ≜ ⟨b, v⟩ ∈ votes[a]

DidNotVoteIn(a, b) ≜ ∀ v ∈ Value : ¬VotedFor(a, b, v)

SafeAt(b, v) ≜
  LET SA[bb ∈ Ballot] ≜
        ∨ bb = 0
        ∨ ∃ Q ∈ Quorum :
            ∧ ∀ a ∈ Q    : maxBal[a] ≥ bb
            ∧ ∃ c ∈ −1 .. (bb − 1) :
                ∧ (c ≠ −1) ⇒ ∧ SA[c]
                             ∧ ∀ a ∈ Q :
                                 ∀ w ∈ Value :
                                   VotedFor(a, c, w) ⇒ (w = v)
                ∧ ∀ d ∈ (c + 1) .. (bb − 1), a ∈ Q : DidNotVoteIn(a, d)
  IN  SA[b]

vars ≜ ⟨votes, maxBal⟩

ProcSet ≜ (Acceptor)

Init ≜ Global variables
       ∧ votes = [a ∈ Acceptor ↦ {}]
       ∧ maxBal = [a ∈ Acceptor ↦ −1]

acceptor(self) ≜ ∃ b   ∈ Ballot :
                   ∨ ∧ b > maxBal[self]
                     ∧ maxBal′ = [maxBal EXCEPT ![self] = b]
                     ∧ UNCHANGED votes
                   ∨ ∧ ∃ v ∈ Value :
                         ∧ ∧ maxBal[self] ≤ b
                           ∧ DidNotVoteIn(self, b)
                           ∧ ∀ p ∈ Acceptor \ {self} :
                               ∀ w ∈ Value : VotedFor(p, b, w) ⇒ (w = v)
```

$\land\ SafeAt(b, v)$
$\land\ votes' = [votes \text{ EXCEPT } ![self] = votes[self] \cup \{\langle b, v\rangle\}]$
$\land\ maxBal' = [maxBal \text{ EXCEPT } ![self] = b]$

$Next \triangleq (\exists\, self \in Acceptor : acceptor(self))$

$Spec \triangleq Init \land \Box[Next]_{vars}$

END TRANSLATION

To reason about a recursively-defined operator, one must prove a theorem about it. In particular, to reason about *SafeAt*, we need to prove that $SafeAt(b, v)$ equals the right-hand side of its definition, for $b \in Ballot$ and $v \in Value$. This is not automatically true for a recursive definition. For example, from the recursive definition

$Silly[n \in Nat] \triangleq \text{CHOOSE } v : v \neq Silly[n]$

we cannot deduce that

$Silly[42] = \text{CHOOSE } v : v \neq Silly[42]$

(From that, we could easily deduce $Silly[42] \neq Silly[42]$.)

Here is the theorem that essentially asserts that $SafeAt(b, v)$ equals the right-hand side of its definition.

```
THEOREM SafeAtProp ≜
  ∀ b ∈ Ballot, v ∈ Value :
    SafeAt(b, v) ≡
      ∨ b = 0
      ∨ ∃ Q ∈ Quorum :
          ∧ ∀ a ∈ Q    : maxBal[a] ≥ b
          ∧ ∃ c ∈ −1 . . (b − 1) :
              ∧ (c ≠ −1) ⇒ ∧ SafeAt(c, v)
                            ∧ ∀ a ∈ Q :
                                ∀ w ∈ Value :
                                  VotedFor(a, c, w) ⇒ (w = v)
              ∧ ∀ d ∈ (c + 1) . . (b − 1), a ∈ Q : DidNotVoteIn(a, d)
⟨1⟩1. SUFFICES ASSUME NEW v ∈ Value
               PROVE  ∀ b ∈ Ballot : SafeAtProp!(b, v)
  BY Zenon
⟨1⟩ USE DEF Ballot
⟨1⟩ DEFINE Def(SA, bb) ≜
        ∨  bb = 0
        ∨  ∃ Q ∈ Quorum :
             ∧ ∀ a ∈ Q    : maxBal[a] ≥ bb
             ∧ ∃ c ∈ −1 . . (bb − 1) :
                 ∧ (c ≠ −1) ⇒ ∧ SA[c]
                               ∧ ∀ a ∈ Q :
                                   ∀ w ∈ Value :
```

$VotedFor(a,\ c,\ w) \Rightarrow (w = v)$

$\land \forall d \in (c+1) \, .. \, (bb-1),\ a \in Q : DidNotVoteIn(a,\ d)$

$SA[bb \in Ballot] \triangleq Def(SA,\ bb)$

$\langle 1 \rangle 2.$ $\forall b : SafeAt(b,\ v) = SA[b]$

BY DEF $SafeAt$

$\langle 1 \rangle 3.$ ASSUME NEW $n \in Nat$, NEW g, NEW h,

$\forall i \in 0 \, .. \, (n-1) : g[i] = h[i]$

PROVE $Def(g,\ n) = Def(h,\ n)$

BY $\langle 1 \rangle 3$

$\langle 1 \rangle 4.$ $SA = [b \in Ballot \mapsto Def(SA,\ b)]$

$\langle 2 \rangle$ HIDE DEF Def

$\langle 2 \rangle$ QED

BY $\langle 1 \rangle 3$, $RecursiveFcnOfNat$, Isa

$\langle 1 \rangle 5.$ $\forall b \in Ballot : SA[b] = Def(SA,\ b)$

$\langle 2 \rangle$ HIDE DEF Def

$\langle 2 \rangle$ QED

BY $\langle 1 \rangle 4$, $Zenon$

$\langle 1 \rangle 6.$ QED

BY $\langle 1 \rangle 2$, $\langle 1 \rangle 5$, $Zenon$ DEF $SafeAt$

We now define *TypeOK* to be the type-correctness invariant.

$$TypeOK \triangleq \begin{array}{l} \land\ votes \in [Acceptor \rightarrow \text{SUBSET}\ (Ballot \times Value)] \\ \land\ maxBal \in [Acceptor \rightarrow Ballot \cup \{-1\}] \end{array}$$

We now define *chosen* to be the state function so that the algorithm specified by formula *Spec* conjoined with the liveness requirements described below implements the algorithm of module *Consensus* (satisfies the specification *LiveSpec* of that module) under a refinement mapping that substitutes this state function *chosen* for the variable *chosen* of module *Consensus*. The definition uses the following one, which defines $ChosenIn(b,\ v)$ to be true iff a quorum of acceptors have all voted for v in ballot b.

$$ChosenIn(b,\ v) \triangleq \exists Q \in Quorum : \forall a \in Q : VotedFor(a,\ b,\ v)$$

$$chosen \triangleq \{v \in Value : \exists b \in Ballot : ChosenIn(b,\ v)\}$$

The following lemma is used for reasoning about the operator *SafeAt*. It is proved from *SafeAtProp* by induction.

LEMMA $SafeLemma \triangleq$

$TypeOK \Rightarrow$

$\forall b \in Ballot :$

$\forall v \in Value :$

$SafeAt(b,\ v) \Rightarrow$

$\forall c \in 0 \, .. \, (b-1) :$

$\exists Q \in Quorum :$

$\forall a \in Q : \land\ maxBal[a] \geq c$

$\land \lor DidNotVoteIn(a,\ c)$

$\lor \mathit{VotedFor}(a, c, v)$

$\langle 1 \rangle$ SUFFICES ASSUME TypeOK
PROVE $\mathit{SafeLemma}!2$

OBVIOUS

$\langle 1 \rangle$ DEFINE $P(b) \triangleq \forall c \in 0 \,..\, b : \mathit{SafeLemma}!2!(c)$

$\langle 1 \rangle$ USE DEF Ballot

$\langle 1 \rangle 1.$ $P(0)$

OBVIOUS

$\langle 1 \rangle 2.$ ASSUME NEW $b \in \mathit{Ballot}$, $P(b)$
PROVE $P(b+1)$

$\langle 2 \rangle 1.$ $\land\ b+1 \in \mathit{Ballot} \setminus \{0\}$
$\land\ (b+1) - 1 = b$

OBVIOUS

$\langle 2 \rangle 2.$ $0 \,..\, (b+1) = (0 \,..\, b) \cup \{b+1\}$

OBVIOUS

$\langle 2 \rangle 3.$ SUFFICES ASSUME NEW $v \in \mathit{Value}$,
$\mathit{SafeAt}(b+1, v)$,
NEW $c \in 0 \,..\, b$
PROVE $\exists Q \in \mathit{Quorum} :$
$\forall a \in Q : \land\ \mathit{maxBal}[a] \geq c$
$\land\ \lor\ \mathit{DidNotVoteIn}(a, c)$
$\lor\ \mathit{VotedFor}(a, c, v)$

BY $\langle 1 \rangle 2$

$\langle 2 \rangle 4.$ PICK $Q \in \mathit{Quorum} :$
$\land\ \forall a \in Q : \mathit{maxBal}[a] \geq (b+1)$
$\land\ \exists cc \in -1 \,..\, b :$
$\land\ (cc \neq -1) \Rightarrow \land\ \mathit{SafeAt}(cc, v)$
$\land\ \forall a \in Q :$
$\forall w \in \mathit{Value} :$
$\mathit{VotedFor}(a, cc, w) \Rightarrow (w = v)$
$\land\ \forall d \in (cc+1) \,..\, b,\ a \in Q : \mathit{DidNotVoteIn}(a, d)$

BY $\mathit{SafeAtProp}$, $\langle 2 \rangle 3$, $\langle 2 \rangle 1$, Zenon

$\langle 2 \rangle 5.$ PICK $cc \in -1 \,..\, b :$
$\land\ (cc \neq -1) \Rightarrow \land\ \mathit{SafeAt}(cc, v)$
$\land\ \forall a \in Q :$
$\forall w \in \mathit{Value} :$
$\mathit{VotedFor}(a, cc, w) \Rightarrow (w = v)$
$\land\ \forall d \in (cc+1) \,..\, b,\ a \in Q : \mathit{DidNotVoteIn}(a, d)$

BY $\langle 2 \rangle 4$

$\langle 2 \rangle 6.$CASE $c > cc$

BY $\langle 2 \rangle 4$, $\langle 2 \rangle 5$, $\langle 2 \rangle 6$, QA DEF TypeOK

$\langle 2 \rangle 7.$CASE $c = cc$

$\langle 3 \rangle 2.$ $\forall a \in Q : \mathit{maxBal}[a] \in \mathit{Ballot} \cup \{-1\}$

BY QA DEF TypeOK

$\langle 3 \rangle 3.$ $\forall a \in Q : \mathit{maxBal}[a] \geq c$

BY $\langle 2\rangle 4$, $\langle 2\rangle 7$, $\langle 3\rangle 2$

$\langle 3\rangle 4$. $\forall a \in Q : \lor DidNotVoteIn(a, c)$
$\qquad\qquad\qquad \lor VotedFor(a, c, v)$

BY $\langle 2\rangle 7$, $\langle 2\rangle 5$ DEF $DidNotVoteIn$

$\langle 3\rangle 5$. QED

BY $\langle 3\rangle 3$, $\langle 3\rangle 4$

$\langle 2\rangle 8$.CASE $c < cc$

BY $\langle 2\rangle 8$, $\langle 1\rangle 2$, $\langle 2\rangle 5$

$\langle 2\rangle 9$. QED

BY $\langle 2\rangle 6$, $\langle 2\rangle 7$, $\langle 2\rangle 8$

$\langle 1\rangle 3$. $\forall b \in Ballot : P(b)$

BY $\langle 1\rangle 1$, $\langle 1\rangle 2$, $NatInduction$, Isa

$\langle 1\rangle 4$. QED

BY $\langle 1\rangle 3$

We now define the invariant that is used to prove the correctness of our algorithm–meaning that specification *Spec* implements specification Spec of module *Consensus* under our refinement mapping. Correctness of the voting algorithm follows from the the following three invariants:

*VInv*1: In any ballot, an acceptor can vote for at most one value.

*VInv*2: An acceptor can vote for a value v in ballot b iff v is safe at b.

*VInv*3: Two different acceptors cannot vote for different values in the same ballot.

Their precise definitions are as follows.

$$VInv1 \triangleq \forall a \in Acceptor, b \in Ballot, v, w \in Value : VotedFor(a, b, v) \land VotedFor(a, b, w) \Rightarrow (v = w)$$

$$VInv2 \triangleq \forall a \in Acceptor, b \in Ballot, v \in Value : VotedFor(a, b, v) \Rightarrow SafeAt(b, v)$$

$$VInv3 \triangleq \forall a1, a2 \in Acceptor, b \in Ballot, v1, v2 \in Value : VotedFor(a1, b, v1) \land VotedFor(a2, b, v2) \Rightarrow (v1 = v2)$$

It is obvious, that *VInv*3 implies *VInv*1–a fact that we now let *TLAPS* prove as a little check that we haven't made a mistake in our definitions. (Actually, we used *TLC* to check everything before attempting any proofs.) We define *VInv*1 separately because *VInv*3 is not needed for proving safety, only for liveness.

THEOREM $VInv3 \Rightarrow VInv1$

BY DEF $VInv1$, $VInv3$

The following lemma proves that $SafeAt(b, v)$ implies that no value other than v can have been chosen in any ballot numbered less than b. The fact that it also implies that no value other than v can ever be chosen in the future follows from this and the fact that $SafeAt(b, v)$ is stable–meaning that once it becomes true, it remains true forever. The stability of $SafeAt(b, v)$ is proved as step $\langle 1\rangle 6$ of theorem *InductiveInvariance* below.

This lemma is used only in the proof of theorem *VT*1 below.

LEMMA $VT0 \triangleq \land TypeOK$

$\wedge\ VInv1$

$\wedge\ VInv2$

$\Rightarrow \forall\, v,\, w \in Value,\ b,\ c \in Ballot :$

$\quad (b > c) \wedge SafeAt(b,\ v) \wedge ChosenIn(c,\ w) \Rightarrow (v = w)$

$\langle 1 \rangle$ SUFFICES ASSUME $TypeOK,\ VInv1,\ VInv2,$

NEW $v \in Value$, NEW $w \in Value$

PROVE $\forall\, b,\ c \in Ballot :$

$\quad (b > c) \wedge SafeAt(b,\ v) \wedge ChosenIn(c,\ w) \Rightarrow (v = w)$

OBVIOUS

$\langle 1 \rangle\ P(b) \triangleq \forall\, c \in Ballot :$

$\quad (b > c) \wedge SafeAt(b,\ v) \wedge ChosenIn(c,\ w) \Rightarrow (v = w)$

$\langle 1 \rangle$ USE DEF $Ballot$

$\langle 1 \rangle 1.\ P(0)$

OBVIOUS

$\langle 1 \rangle 2.$ ASSUME NEW $b \in Ballot,\ \forall\, i \in 0 \,..\, (b-1) : P(i)$

PROVE $P(b)$

$\langle 2 \rangle 1.$CASE $b = 0$

BY $\langle 2 \rangle 1$

$\langle 2 \rangle 2.$CASE $b \neq 0$

$\langle 3 \rangle 1.$ SUFFICES ASSUME NEW $c \in Ballot,\ b > c,\ SafeAt(b,\ v),\ ChosenIn(c,\ w)$

PROVE $v = w$

OBVIOUS

$\langle 3 \rangle 2.$ PICK $Q \in Quorum : \forall\, a \in Q : VotedFor(a,\ c,\ w)$

BY $\langle 3 \rangle 1$ DEF $ChosenIn$

$\langle 3 \rangle 3.$ PICK $QQ \in Quorum,$

$d \in -1 \,..\, (b-1) :$

$\wedge\ (d \neq -1) \Rightarrow \wedge\ SafeAt(d,\ v)$

$\wedge\ \forall\, a \in QQ :$

$\forall\, x \in Value :$

$VotedFor(a,\ d,\ x) \Rightarrow (x = v)$

$\wedge\ \forall\, e \in (d+1) \,..\, (b-1),\ a \in QQ : DidNotVoteIn(a,\ e)$

BY $\langle 2 \rangle 2,\ \langle 3 \rangle 1,\ SafeAtProp,\ Zenon$

$\langle 3 \rangle$ PICK $aa \in QQ \cap Q$: TRUE

BY QA

$\langle 3 \rangle 4.\ c \leq d$

BY $\langle 3 \rangle 1,\ \langle 3 \rangle 2,\ \langle 3 \rangle 3$ DEF $DidNotVoteIn$

$\langle 3 \rangle 5.$CASE $c = d$

BY $\langle 3 \rangle 2,\ \langle 3 \rangle 3,\ \langle 3 \rangle 4,\ \langle 3 \rangle 5$

$\langle 3 \rangle 6.$CASE $d > c$

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

$\langle 3 \rangle 7.$ QED

BY $\langle 3 \rangle 4,\ \langle 3 \rangle 5,\ \langle 3 \rangle 6$

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

$\langle 1 \rangle 3.\ \forall\, b \in Ballot : P(b)$

$\langle 2 \rangle$.HIDE DEF P
$\langle 2 \rangle$.QED BY $\langle 1 \rangle 2$, *GeneralNatInduction*, *Isa*
$\langle 1 \rangle 4$. QED
BY $\langle 1 \rangle 3$

The following theorem asserts that the invariance of *TypeOK*, *VInv*1, and *VInv*2 implies that the algorithm satisfies the basic consensus property that at most one value is chosen (at any time). If you can prove it, then you understand why the *Paxos* consensus algorithm allows only a single value to be chosen. Note that *VInv*3 is not needed to prove this property.

THEOREM $VT1 \triangleq \land TypeOK$
$\land VInv1$
$\land VInv2$
$\Rightarrow \forall v, w :$
$(v \in chosen) \land (w \in chosen) \Rightarrow (v = w)$
$\langle 1 \rangle 1$. SUFFICES ASSUME $TypeOK$, $VInv1$, $VInv2$,
NEW v, NEW w,
$v \in chosen$, $w \in chosen$
PROVE $v = w$
OBVIOUS
$\langle 1 \rangle 2$. $v \in Value \land w \in Value$
BY $\langle 1 \rangle 1$ DEF *chosen*
$\langle 1 \rangle 3$. PICK $b \in Ballot$, $c \in Ballot : ChosenIn(b, v) \land ChosenIn(c, w)$
BY $\langle 1 \rangle 1$ DEF *chosen*
$\langle 1 \rangle 4$. PICK $Q \in Quorum$, $R \in Quorum$:
$\land \forall a \in Q : VotedFor(a, b, v)$
$\land \forall a \in R : VotedFor(a, c, w)$
BY $\langle 1 \rangle 3$ DEF *ChosenIn*
$\langle 1 \rangle 5$. PICK $av \in Q$, $aw \in R$: $\land VotedFor(av, b, v)$
$\land VotedFor(aw, c, w)$
BY $\langle 1 \rangle 4$, *QuorumNonEmpty*
$\langle 1 \rangle 6$. $SafeAt(b, v) \land SafeAt(c, w)$
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 5$, *QA* DEF *VInv*2
$\langle 1 \rangle 7$.CASE $b = c$
$\langle 2 \rangle$ PICK $a \in Q \cap R$: TRUE
BY *QA*
$\langle 2 \rangle 1$. $\land VotedFor(a, b, v)$
$\land VotedFor(a, c, w)$
BY $\langle 1 \rangle 4$
$\langle 2 \rangle 2$. QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 7$, $\langle 2 \rangle 1$, *QA* DEF *VInv*1
$\langle 1 \rangle 8$.CASE $b > c$
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 6$, $\langle 1 \rangle 3$, $\langle 1 \rangle 8$, $VT0$, $\langle 1 \rangle 2$
$\langle 1 \rangle 9$.CASE $c > b$
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 6$, $\langle 1 \rangle 3$, $\langle 1 \rangle 9$, $VT0$, $\langle 1 \rangle 2$
$\langle 1 \rangle 10$. QED
BY $\langle 1 \rangle 7$, $\langle 1 \rangle 8$, $\langle 1 \rangle 9$ DEF *Ballot*

The rest of the proof uses only the primed version of $VT1$–that is, the theorem whose statement is $VT1'$. (Remember that $VT1$ names the formula being asserted by the theorem we call $VT1$.) The formula $VT1'$ asserts that $VT1$ is true in the second state of any transition (pair of states). We derive that theorem from $VT1$ by simple temporal logic, and similarly for $VT0$ and *SafeAtProp*.

THEOREM *SafeAtPropPrime* $\triangleq$

$$
\begin{array}{l}
\forall b \in Ballot, v \in Value : \\
\quad SafeAt(b, v)' \equiv \\
\quad\quad \lor b = 0 \\
\quad\quad \lor \exists Q \in Quorum : \\
\quad\quad\quad \land \forall a \in Q \quad : maxBal'[a] \geq b \\
\quad\quad\quad \land \exists c \in -1 \,..\, (b-1) : \\
\quad\quad\quad\quad \land (c \neq -1) \Rightarrow \land SafeAt(c, v)' \\
\quad\quad\quad\quad\quad\quad\quad\quad \land \forall a \in Q : \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \forall w \in Value : \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad VotedFor(a, c, w)' \Rightarrow (w = v) \\
\quad\quad\quad\quad \land \forall d \in (c+1) \,..\, (b-1), a \in Q : DidNotVoteIn(a, d)'
\end{array}
$$

$\langle 1 \rangle 1.$ $SafeAtProp'$ BY *SafeAtProp*, *PTL*

$\langle 1 \rangle$.QED BY $\langle 1 \rangle 1$

LEMMA *VT0Prime* $\triangleq$

$$
\begin{array}{l}
\land TypeOK' \\
\land VInv1' \\
\land VInv2' \\
\Rightarrow \forall v, w \in Value, b, c \in Ballot : \\
\quad\quad (b > c) \land SafeAt(b, v)' \land ChosenIn(c, w)' \Rightarrow (v = w)
\end{array}
$$

$\langle 1 \rangle 1.$ $VT0'$ BY $VT0$, *PTL*

$\langle 1 \rangle$.QED BY $\langle 1 \rangle 1$

THEOREM *VT1Prime* $\triangleq$

$$
\begin{array}{l}
\land TypeOK' \\
\land VInv1' \\
\land VInv2' \\
\Rightarrow \forall v, w : \\
\quad\quad (v \in chosen') \land (w \in chosen') \Rightarrow (v = w)
\end{array}
$$

$\langle 1 \rangle 1.$ $VT1'$ BY $VT1$, *PTL*

$\langle 1 \rangle$.QED BY $\langle 1 \rangle 1$

The invariance of $VInv2$ depends on $SafeAt(b, v)$ being stable, meaning that once it becomes true it remains true forever. Stability of $SafeAt(b, v)$ depends on the following invariant.

$$
\begin{array}{l}
VInv4 \triangleq \forall a \in Acceptor, b \in Ballot : \\
\quad\quad maxBal[a] < b \Rightarrow DidNotVoteIn(a, b)
\end{array}
$$

The inductive invariant that we use to prove correctness of this algorithm is *VInv*, defined as follows.

$$VInv \triangleq TypeOK \land VInv2 \land VInv3 \land VInv4$$

To simplify reasoning about the next-state action *Next*, we want to express it in a more convenient form. This is done by lemma *NextDef* below, which shows that *Next* equals an action defined in terms of the following subactions.

$IncreaseMaxBal(self, b) \triangleq$
$\quad \land\ b > maxBal[self]$
$\quad \land\ maxBal' = [maxBal \text{ EXCEPT } ![self] = b]$
$\quad \land\ \text{UNCHANGED } votes$

$VoteFor(self, b, v) \triangleq$
$\quad \land\ maxBal[self] \leq b$
$\quad \land\ DidNotVoteIn(self, b)$
$\quad \land\ \forall p \in Acceptor \setminus \{self\} :$
$\qquad \forall w \in Value : VotedFor(p, b, w) \Rightarrow (w = v)$
$\quad \land\ SafeAt(b, v)$
$\quad \land\ votes' = [votes \text{ EXCEPT } ![self] = votes[self] \cup \{\langle b, v\rangle\}]$
$\quad \land\ maxBal' = [maxBal \text{ EXCEPT } ![self] = b]$

$BallotAction(self, b) \triangleq$
$\quad \lor\ IncreaseMaxBal(self, b)$
$\quad \lor\ \exists v \in Value : VoteFor(self, b, v)$

When proving lemma *NextDef*, we were surprised to discover that it required the assumption that the set of acceptors is non-empty. This assumption isn't necessary for safety, since if there are no acceptors there can be no quorums (see theorem *QuorumNonEmpty* above) so no value is ever chosen and the *Consensus* specification is trivially implemented under our refinement mapping. However, the assumption is necessary for liveness and it allows us to lemma *NextDef* for the safety proof as well, so we assert it now.

ASSUME $AcceptorNonempty \triangleq Acceptor \neq \{\}$

The proof of the lemma itself is quite simple.

LEMMA $NextDef \triangleq$
$\quad TypeOK \Rightarrow$
$\quad (Next = \exists self \in Acceptor :$
$\qquad\qquad \exists b \in Ballot : BallotAction(self, b))$
$\langle 1\rangle$ HAVE $TypeOK$
$\langle 1\rangle 2.\ Next = \exists self \in Acceptor : acceptor(self)$
BY $AcceptorNonempty$ DEF $Next, ProcSet$
$\langle 1\rangle 3.\ @ = NextDef!2!2$
BY DEF $Next, BallotAction, IncreaseMaxBal, VoteFor, ProcSet, acceptor$
$\langle 1\rangle 4.$ QED
BY $\langle 1\rangle 2, \langle 1\rangle 3$

We now come to the proof that *VInv* is an invariant of the specification. This follows from the following result, which asserts that it is an inductive invariant of the next-state action. This fact is used in the liveness proof as well.

THEOREM $InductiveInvariance \triangleq VInv \land [Next]_{vars} \Rightarrow VInv'$
$\langle 1\rangle 1.\ VInv \land (vars' = vars) \Rightarrow VInv'$

```
  BY Isa
      DEF VInv, vars, TypeOK, VInv2, VotedFor, SafeAt, DidNotVoteIn, VInv3, VInv4
⟨1⟩ SUFFICES ASSUME VInv,
                    NEW self ∈ Acceptor,
                    NEW b ∈ Ballot,
                    BallotAction(self, b)
             PROVE  VInv'
  BY ⟨1⟩1, NextDef DEF VInv

⟨1⟩2. TypeOK'
  ⟨2⟩1.CASE IncreaseMaxBal(self, b)
    BY ⟨2⟩1 DEF IncreaseMaxBal, VInv, TypeOK
  ⟨2⟩2.CASE ∃ v ∈ Value : VoteFor(self, b, v)
    BY ⟨2⟩2 DEF VInv, TypeOK, VoteFor
  ⟨2⟩3. QED
    BY ⟨2⟩1, ⟨2⟩2 DEF BallotAction

⟨1⟩3. ASSUME NEW a ∈ Acceptor, NEW c ∈ Ballot, NEW w ∈ Value,
             VotedFor(a, c, w)
      PROVE  VotedFor(a, c, w)'
  ⟨2⟩1.CASE IncreaseMaxBal(self, b)
    BY ⟨2⟩1, ⟨1⟩3 DEF IncreaseMaxBal, VotedFor
  ⟨2⟩2.CASE ∃ v ∈ Value : VoteFor(self, b, v)
    ⟨3⟩1. PICK v ∈ Value : VoteFor(self, b, v)
      BY ⟨2⟩2
    ⟨3⟩2.CASE a = self
      ⟨4⟩1. votes'[a] = votes[a] ∪ {⟨b, v⟩}
        BY ⟨3⟩1, ⟨3⟩2 DEF VoteFor, VInv, TypeOK
      ⟨4⟩2. QED
        BY ⟨1⟩3, ⟨4⟩1 DEF VotedFor
    ⟨3⟩3.CASE a ≠ self
      ⟨4⟩1. votes[a] = votes'[a]
        BY ⟨3⟩1, ⟨3⟩3 DEF VoteFor, VInv, TypeOK
      ⟨4⟩2. QED
        BY ⟨1⟩3, ⟨4⟩1 DEF VotedFor
    ⟨3⟩4. QED
     BY ⟨3⟩2, ⟨3⟩3 DEF VoteFor
  ⟨2⟩3. QED
    BY ⟨2⟩1, ⟨2⟩2 DEF BallotAction

⟨1⟩4. ASSUME NEW a ∈ Acceptor, NEW c ∈ Ballot, NEW w ∈ Value,
             ¬VotedFor(a, c, w), VotedFor(a, c, w)'
      PROVE  (a = self) ∧ (c = b) ∧ VoteFor(self, b, w)
  ⟨2⟩1.CASE IncreaseMaxBal(self, b)
      BY ⟨2⟩1, ⟨1⟩4 DEF IncreaseMaxBal, VInv, TypeOK, VotedFor
  ⟨2⟩2.CASE ∃ v ∈ Value : VoteFor(self, b, v)
```

$\langle 3\rangle 1.$ PICK $v \in \mathit{Value} : \mathit{VoteFor}(\mathit{self}, b, v)$
BY $\langle 2\rangle 2$
$\langle 3\rangle 2.$ $a = \mathit{self}$
BY $\langle 3\rangle 1$, $\langle 1\rangle 4$ DEF $\mathit{VoteFor}$, VInv, TypeOK, $\mathit{VotedFor}$
$\langle 3\rangle 3.$ $\mathit{votes}'[a] = \mathit{votes}[a] \cup \{\langle b, v\rangle\}$
BY $\langle 3\rangle 1$, $\langle 3\rangle 2$ DEF $\mathit{VoteFor}$, VInv, TypeOK
$\langle 3\rangle 4.$ $c = b \land v = w$
BY $\langle 1\rangle 4$, $\langle 3\rangle 3$ DEF $\mathit{VotedFor}$
$\langle 3\rangle 5.$ QED
BY $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 4$
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$ DEF $\mathit{BallotAction}$

$\langle 1\rangle 5.$ ASSUME NEW $a \in \mathit{Acceptor}$
PROVE $\land\ \mathit{maxBal}[a] \in \mathit{Ballot} \cup \{-1\}$
$\land\ \mathit{maxBal}'[a] \in \mathit{Ballot} \cup \{-1\}$
$\land\ \mathit{maxBal}'[a] \geq \mathit{maxBal}[a]$
BY DEF VInv, TypeOK, $\mathit{IncreaseMaxBal}$, VInv, $\mathit{VoteFor}$, $\mathit{BallotAction}$, $\mathit{DidNotVoteIn}$, $\mathit{VotedFor}$, Ballot

$\langle 1\rangle 6.$ ASSUME NEW $c \in \mathit{Ballot}$, NEW $w \in \mathit{Value}$,
$\mathit{SafeAt}(c, w)$
PROVE $\mathit{SafeAt}(c, w)'$
$\langle 2\rangle$ USE DEF Ballot
$\langle 2\rangle$ DEFINE $P(i) \triangleq \forall j \in 0 \,..\, i : \mathit{SafeAt}(j, w) \Rightarrow \mathit{SafeAt}(j, w)'$
$\langle 2\rangle 1.$ $P(0)$
BY $\mathit{SafeAtPropPrime}$, $0 \,..\, 0 = \{0\}$, Zenon
$\langle 2\rangle 2.$ ASSUME NEW $d \in \mathit{Ballot}$, $P(d)$
PROVE $P(d+1)$
$\langle 3\rangle 1.$ SUFFICES ASSUME NEW $e \in 0 \,..\, (d+1)$, $\mathit{SafeAt}(e, w)$
PROVE $\mathit{SafeAt}(e, w)'$
OBVIOUS
$\langle 3\rangle 2.$CASE $e \in 0 \,..\, d$
BY $\langle 2\rangle 2$, $\langle 3\rangle 1$, $\langle 3\rangle 2$
$\langle 3\rangle 3.$CASE $e = d + 1$
$\langle 4\rangle.e \in \mathit{Ballot} \setminus \{0\}$
BY $\langle 3\rangle 3$
$\langle 4\rangle 1.$ PICK $Q \in \mathit{Quorum} : \mathit{SafeAtProp}!(e, w)!2!2!(Q)$
BY $\langle 3\rangle 1$, $\mathit{SafeAtProp}$, Zenon
$\langle 4\rangle 2.$ $\forall\, aa \in Q : \mathit{maxBal}'[aa] \geq e$
BY $\langle 1\rangle 5$, $\langle 4\rangle 1$, QA
$\langle 4\rangle 3.$ $\exists\, cc \in -1 \,..\, (e-1) :$
$\land\ (cc \neq -1) \Rightarrow \land\ \mathit{SafeAt}(cc, w)'$
$\land\ \forall\, ax \in Q :$
$\forall\, z \in \mathit{Value} :$

$\qquad\qquad\qquad VotedFor(ax, cc, z)' \Rightarrow (z = w)$

$\quad \land \forall\, dd \in (cc+1) \,..\, (e-1),\ ax \in Q : DidNotVoteIn(ax, dd)'$

$\langle 5\rangle 1.$ ASSUME NEW $cc \in 0 \,..\, (e-1)$,
NEW $ax \in Q$, NEW $z \in Value$,
$VotedFor(ax, cc, z)', \neg VotedFor(ax, cc, z)$
PROVE FALSE

$\langle 6\rangle 1.$ $(ax = self) \land (cc = b) \land VoteFor(self, b, z)$
BY $\langle 5\rangle 1, \langle 1\rangle 4, QA$

$\langle 6\rangle 2.$ $\land\ maxBal[ax] \geq e$
$\land\ maxBal[self] \leq b$
BY $\langle 4\rangle 1, \langle 6\rangle 1$ DEF $VoteFor$

$\langle 6\rangle$.QED BY $\langle 3\rangle 3, \langle 6\rangle 1, \langle 6\rangle 2$ DEF $VInv, TypeOK$

$\langle 5\rangle 2.$ PICK $cc \in -1 \,..\, (e-1) : SafeAtProp!(e, w)!2!2!(Q)!2!(cc)$
BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ ASSUME $cc \neq -1$
PROVE $\land\ SafeAt(cc, w)'$
$\land \forall\, ax \in Q : \forall\, z \in Value :$
$\quad VotedFor(ax, cc, z)' \Rightarrow (z = w)$

$\langle 6\rangle 1.$ $\land\ SafeAt(cc, w)$
$\land \forall\, ax \in Q :$
$\quad \forall\, z \in Value : VotedFor(ax, cc, z) \Rightarrow (z = w)$
BY $\langle 5\rangle 2, \langle 5\rangle 3$

$\langle 6\rangle 2.$ $SafeAt(cc, w)'$
BY $\langle 6\rangle 1, \langle 5\rangle 3, \langle 3\rangle 3, \langle 2\rangle 2$

$\langle 6\rangle 3.$ ASSUME NEW $ax \in Q$, NEW $z \in Value$, $VotedFor(ax, cc, z)'$
PROVE $z = w$

$\langle 7\rangle 1.$CASE $VotedFor(ax, cc, z)$
BY $\langle 6\rangle 1, \langle 7\rangle 1$

$\langle 7\rangle 2.$CASE $\neg VotedFor(ax, cc, z)$
BY $\langle 7\rangle 2, \langle 6\rangle 3, \langle 5\rangle 1, \langle 5\rangle 3$

$\langle 7\rangle 3.$ QED
BY $\langle 7\rangle 1, \langle 7\rangle 2$

$\langle 6\rangle 4.$ QED
BY $\langle 6\rangle 2, \langle 6\rangle 3$

$\langle 5\rangle 4.$ ASSUME NEW $dd \in (cc+1) \,..\, (e-1)$, NEW $ax \in Q$,
$\neg DidNotVoteIn(ax, dd)'$
PROVE FALSE
BY $\langle 5\rangle 2, \langle 5\rangle 1, \langle 5\rangle 4$ DEF $DidNotVoteIn$

$\langle 5\rangle 5.$ QED
BY $\langle 5\rangle 3, \langle 5\rangle 4$

$\langle 4\rangle 4.$ $\lor\ e = 0$
$\lor \exists\, Q\_1 \in Quorum :$
$\quad \land \forall\, aa \in Q\_1 : maxBal'[aa] \geq e$
$\quad \land \exists\, c\_1 \in -1 \,..\, e-1 :$
$\qquad \land\ c\_1 \neq -1$

$\Rightarrow (\land\ SafeAt(c\_1,\ w)'$
$\land\ \forall\ aa \in Q\_1 :$
$\forall\ w\_1 \in Value :$
$VotedFor(aa,\ c\_1,\ w\_1)' \Rightarrow w\_1 = w)$
$\land\ \forall\ d\_1 \in c\_1 + 1 \,..\, e - 1,\ aa \in Q\_1 :$
$DidNotVoteIn(aa,\ d\_1)'$

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

$\langle 4\rangle 6$. $SafeAt(e,\ w)' \equiv \langle 4\rangle 4$
BY $SafeAtPropPrime$, $\langle 3\rangle 3$, $Zenon$

$\langle 4\rangle 7$. QED
BY $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 6$

$\langle 3\rangle 4$. QED
BY $\langle 3\rangle 2$, $\langle 3\rangle 3$

$\langle 2\rangle 3$. $\forall\ d \in Ballot : P(d)$
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $NatInduction$, Isa

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

$\langle 1\rangle 7$. $VInv2'$

$\langle 2\rangle 1$. SUFFICES ASSUME NEW $a \in Acceptor$, NEW $c \in Ballot$, NEW $v \in Value$,
$VotedFor(a,\ c,\ v)'$
PROVE $SafeAt(c,\ v)'$
BY DEF $VInv2$

$\langle 2\rangle 2$.CASE $VotedFor(a,\ c,\ v)$
BY $\langle 1\rangle 6$, $\langle 2\rangle 2$ DEF $VInv$, $VInv2$

$\langle 2\rangle 3$.CASE $\neg VotedFor(a,\ c,\ v)$
BY $\langle 1\rangle 6$, $\langle 2\rangle 1$, $\langle 2\rangle 3$, $\langle 1\rangle 4$ DEF $VoteFor$

$\langle 2\rangle 4$. QED
BY $\langle 2\rangle 2$, $\langle 2\rangle 3$

$\langle 1\rangle 8$. $VInv3'$

$\langle 2\rangle 1$. ASSUME NEW $a1 \in Acceptor$, NEW $a2 \in Acceptor$,
NEW $c \in Ballot$, NEW $v1 \in Value$, NEW $v2 \in Value$,
$VotedFor(a1,\ c,\ v1)'$,
$VotedFor(a2,\ c,\ v2)'$,
$VotedFor(a1,\ c,\ v1)$,
$VotedFor(a2,\ c,\ v2)$
PROVE $v1 = v2$
BY $\langle 2\rangle 1$ DEF $VInv$, $VInv3$

$\langle 2\rangle 2$. ASSUME NEW $a1 \in Acceptor$, NEW $a2 \in Acceptor$,
NEW $c \in Ballot$, NEW $v1 \in Value$, NEW $v2 \in Value$,
$VotedFor(a1,\ c,\ v1)'$,
$VotedFor(a2,\ c,\ v2)'$,
$\neg VotedFor(a1,\ c,\ v1)$
PROVE $v1 = v2$

$\langle 3\rangle 1.\ (a1 = \mathit{self}) \land (c = b) \land \mathit{VoteFor}(\mathit{self},\ b,\ v1)$
BY $\langle 2\rangle 2,\ \langle 1\rangle 4$
$\langle 3\rangle 2.$CASE $a2 = \mathit{self}$
$\langle 4\rangle 1.\ \neg \mathit{VotedFor}(\mathit{self},\ b,\ v2)$
BY $\langle 3\rangle 1$ DEF $\mathit{VoteFor},\ \mathit{DidNotVoteIn}$
$\langle 4\rangle 2.\ \mathit{VoteFor}(\mathit{self},\ b,\ v2)$
BY $\langle 2\rangle 2,\ \langle 3\rangle 1,\ \langle 3\rangle 2,\ \langle 4\rangle 1,\ \ \langle 1\rangle 4$
$\langle 4\rangle.$QED BY $\langle 3\rangle 1,\ \langle 4\rangle 2,\ \langle 2\rangle 2$ DEF $\mathit{VotedFor},\ \mathit{VoteFor},\ \mathit{VInv},\ \mathit{TypeOK}$
$\langle 3\rangle 3.$CASE $a2 \neq \mathit{self}$
BY $\langle 3\rangle 1,\ \langle 3\rangle 3,\ \langle 2\rangle 2$ DEF $\mathit{VotedFor},\ \mathit{VoteFor},\ \mathit{VInv},\ \mathit{TypeOK}$
$\langle 3\rangle 4.$ QED
BY $\langle 3\rangle 2,\ \langle 3\rangle 3$
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1,\ \langle 2\rangle 2$ DEF $\mathit{VInv}3$

$\langle 1\rangle 9.\ \mathit{VInv}4'$
$\langle 2\rangle 1.$ SUFFICES ASSUME NEW $a \in \mathit{Acceptor}$, NEW $c \in \mathit{Ballot}$,
$\mathit{maxBal}'[a] < c$,
$\neg \mathit{DidNotVoteIn}(a,\ c)'$
PROVE FALSE
BY DEF $\mathit{VInv}4$
$\langle 2\rangle 2.\ \mathit{maxBal}[a] < c$
BY $\langle 1\rangle 5,\ \langle 2\rangle 1$ DEF Ballot
$\langle 2\rangle 3.\ \mathit{DidNotVoteIn}(a,\ c)$
BY $\langle 2\rangle 2$ DEF $\mathit{VInv},\ \mathit{VInv}4$
$\langle 2\rangle 4.$ PICK $v \in \mathit{Value} : \mathit{VotedFor}(a,\ c,\ v)'$
BY $\langle 2\rangle 1$ DEF $\mathit{DidNotVoteIn}$
$\langle 2\rangle 5.\ (a = \mathit{self}) \land (c = b) \land \mathit{VoteFor}(\mathit{self},\ b,\ v)$
BY $\langle 1\rangle 4,\ \langle 2\rangle 1,\ \langle 2\rangle 3,\ \langle 2\rangle 4$ DEF $\mathit{DidNotVoteIn}$
$\langle 2\rangle 6.\ \mathit{maxBal}'[a] = c$
BY $\langle 2\rangle 5$ DEF $\mathit{VoteFor},\ \mathit{VInv},\ \mathit{TypeOK}$
$\langle 2\rangle 7.$ QED
BY $\langle 2\rangle 1,\ \langle 2\rangle 6$ DEF Ballot

$\langle 1\rangle 10.$ QED
BY $\langle 1\rangle 2,\ \langle 1\rangle 7,\ \langle 1\rangle 8,\ \langle 1\rangle 9$ DEF VInv

The invariance of *VInv* follows easily from theorem *InductiveInvariance* and the following result, which is easy to prove with *TLAPS*.

THEOREM $\mathit{InitImpliesInv} \triangleq \mathit{Init} \Rightarrow \mathit{VInv}$
BY DEF $\mathit{Init},\ \mathit{VInv},\ \mathit{TypeOK},\ \mathit{ProcSet},\ \mathit{VInv}2,\ \mathit{VInv}3,\ \mathit{VInv}4,\ \mathit{VotedFor},\ \mathit{DidNotVoteIn}$

The following theorem asserts that *VInv* is an invariant of *Spec*.

THEOREM $VT2 \triangleq \mathit{Spec} \Rightarrow \Box \mathit{VInv}$
BY $\mathit{InitImpliesInv},\ \mathit{InductiveInvariance},\ PTL$ DEF Spec

The following INSTANCE statement instantiates module *Consensus* with the following expressions substituted for the parameters (the CONSTANTS and VARIABLES) of that module:

Parameter of *Consensus*	Expression (of this module)
Value	Value
chosen	chosen

(Note that if no substitution is specified for a parameter, the default is to substitute the parameter or defined operator of the same name.) More precisely, for each defined identifier *id* of module *Consensus*, this statement defines $C!id$ to equal the value of *id* under these substitutions.

$C \triangleq$ INSTANCE *Consensus*

The following theorem asserts that the safety properties of the voting algorithm (specified by formula *Spec*) of this module implement the consensus safety specification *Spec* of module *Consensus* under the substitution (refinement mapping) of the INSTANCE statement.

THEOREM $VT3 \triangleq Spec \Rightarrow C!Spec$

$\langle 1 \rangle 1.$ $Init \Rightarrow C!Init$

  $\langle 2 \rangle$ SUFFICES ASSUME $Init$
             PROVE $C!Init$

    OBVIOUS

  $\langle 2 \rangle 1.$ SUFFICES ASSUME NEW $v \in chosen$
               PROVE FALSE

    BY DEF $C!Init$

  $\langle 2 \rangle 2.$ PICK $b \in Ballot,\ Q \in Quorum : \forall a \in Q : VotedFor(a, b, v)$

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

  $\langle 2 \rangle 3.$ PICK $a \in Q : \langle b, v \rangle \in votes[a]$

    BY $QuorumNonEmpty$, $\langle 2 \rangle 2$ DEF $VotedFor$

  $\langle 2 \rangle 4.$ QED

    BY $\langle 2 \rangle 3$, QA DEF $Init$

$\langle 1 \rangle 2.$ $VInv \land VInv' \land [Next]_{vars} \Rightarrow [C!Next]_{C!vars}$

  $\langle 2 \rangle$.SUFFICES ASSUME $VInv,\ VInv',\ [Next]_{vars}$
             PROVE $[C!Next]_{C!vars}$

    OBVIOUS

  $\langle 2 \rangle 1.$CASE $vars' = vars$

    BY $\langle 2 \rangle 1$ DEF $vars$, $C!vars$, $chosen$, $ChosenIn$, $VotedFor$

  $\langle 2 \rangle 2.$ SUFFICES ASSUME NEW $self \in Acceptor$,
                  NEW $b \in Ballot$,
                  $BallotAction(self, b)$
               PROVE $[C!Next]_{C!vars}$

    BY $\langle 2 \rangle 1$, $NextDef$ DEF $VInv$

  $\langle 2 \rangle 3.$ ASSUME $IncreaseMaxBal(self, b)$
      PROVE $C!vars' = C!vars$

    BY $\langle 2 \rangle 3$ DEF $IncreaseMaxBal$, $C!vars$, $chosen$, $ChosenIn$, $VotedFor$

  $\langle 2 \rangle 4.$ ASSUME NEW $v \in Value$,
          $VoteFor(self, b, v)$
      PROVE $[C!Next]_{C!vars}$

$\langle 3 \rangle 3.$ ASSUME NEW $w \in chosen$
PROVE $w \in chosen'$
$\langle 4 \rangle 1.$ PICK $c \in Ballot,\ Q \in Quorum : \forall a \in Q : \langle c,\ w \rangle \in votes[a]$
BY $\langle 3 \rangle 3$ DEF $chosen,\ ChosenIn,\ VotedFor$
$\langle 4 \rangle 2.$ SUFFICES ASSUME NEW $a \in Q$
PROVE $\langle c,\ w \rangle \in votes'[a]$
BY DEF $chosen,\ ChosenIn,\ VotedFor$
$\langle 4 \rangle 3.$CASE $a = self$
BY $\langle 2 \rangle 4,\ \langle 4 \rangle 1,\ \langle 4 \rangle 3$ DEF $VoteFor,\ VInv,\ TypeOK$
$\langle 4 \rangle 4.$CASE $a \neq self$
BY $\langle 2 \rangle 4,\ \langle 4 \rangle 1,\ \langle 4 \rangle 4,\ QA$ DEF $VoteFor,\ VInv,\ TypeOK$
$\langle 4 \rangle 5.$ QED
BY $\langle 4 \rangle 3,\ \langle 4 \rangle 4$
$\langle 3 \rangle 1.$ ASSUME NEW $w \in chosen,$
$v \in chosen'$
PROVE $w = v$
BY $\langle 3 \rangle 3,\ \langle 3 \rangle 1,\ VT1Prime$ DEF $VInv,\ VInv1,\ VInv3$
$\langle 3 \rangle 2.$ ASSUME NEW $w,\ w \notin chosen,\ w \in chosen'$
PROVE $w = v$
$\langle 4 \rangle 2.$ PICK $c \in Ballot,\ Q \in Quorum : \forall a \in Q : \langle c,\ w \rangle \in votes'[a]$
BY $\langle 3 \rangle 2$ DEF $chosen,\ ChosenIn,\ VotedFor$
$\langle 4 \rangle 3.$ PICK $a \in Q : \langle c,\ w \rangle \notin votes[a]$
BY $\langle 3 \rangle 2$ DEF $chosen,\ ChosenIn,\ VotedFor$
$\langle 4 \rangle 4.$CASE $a = self$
BY $\langle 2 \rangle 4,\ \langle 4 \rangle 4,\ \langle 4 \rangle 2,\ \langle 4 \rangle 3$ DEF $VoteFor,\ VInv,\ TypeOK$
$\langle 4 \rangle 5.$CASE $a \neq self$
BY $\langle 2 \rangle 4,\ \langle 4 \rangle 2,\ \langle 4 \rangle 3,\ \langle 4 \rangle 5,\ QA$ DEF $VoteFor,\ VInv,\ TypeOK$
$\langle 4 \rangle 6.$ QED
BY $\langle 4 \rangle 4,\ \langle 4 \rangle 5$
$\langle 3 \rangle$.QED
BY $\langle 3 \rangle 3,\ \langle 3 \rangle 1,\ \langle 3 \rangle 2$ DEF $C!Next,\ C!vars$
$\langle 2 \rangle 5.$ QED
BY $\langle 2 \rangle 2,\ \langle 2 \rangle 3,\ \langle 2 \rangle 4$ DEF $BallotAction$
$\langle 1 \rangle 3.$ QED
BY $\langle 1 \rangle 1,\ \langle 1 \rangle 2,\ VT2,\ PTL$ DEF $Spec,\ C!Spec$

Liveness

We now state the liveness property required of our voting algorithm and prove that it and the safety property imply specification *LiveSpec* of module *Consensus* under our refinement mapping.

We begin by stating two additional assumptions that are necessary for liveness. Liveness requires that some value eventually be chosen. This cannot hold with an infinite set of acceptors. More precisely, liveness requires the existence of a finite quorum. (Otherwise, it would be impossible for all acceptors of any quorum ever to have voted, so no value could ever be chosen.) Moreover, it is impossible to choose a value if there are no values. Hence, we make the following two assumptions.

ASSUME *AcceptorFinite* $\triangleq$ *IsFiniteSet*(*Acceptor*)

ASSUME *ValueNonempty* $\triangleq$ $Value \neq \{\}$

LEMMA *FiniteSetHasMax* $\triangleq$
  ASSUME NEW $S \in$ SUBSET *Int*, *IsFiniteSet*(S), $S \neq \{\}$
  PROVE $\exists\, max \in S : \forall\, x \in S : max \geq x$
⟨1⟩.DEFINE $P(T) \triangleq T \in$ SUBSET $Int \land T \neq \{\} \Rightarrow \exists\, max \in T : \forall\, x \in T : max \geq x$
⟨1⟩1. $P(\{\})$
  OBVIOUS
⟨1⟩2. ASSUME NEW T, NEW x, $P(T)$, $x \notin T$
    PROVE $P(T \cup \{x\})$
  BY ⟨1⟩2
⟨1⟩3. $\forall\, T : IsFiniteSet(T) \Rightarrow P(T)$
  ⟨2⟩.HIDE DEF P
  ⟨2⟩.QED BY ⟨1⟩1, ⟨1⟩2, *FS_Induction*, *IsaM*("blast")
⟨1⟩.QED BY ⟨1⟩3, *Zenon*

The following theorem implies that it is always possible to find a ballot number b and a value v safe at b by choosing b large enough and then having a quorum of acceptors perform *IncreaseMaxBal*(b) actions. It will be used in the liveness proof. Observe that it is for liveness, not safety, that invariant *VInv*3 is required.

THEOREM *VT*4 $\triangleq$ $TypeOK \land VInv2 \land VInv3 \Rightarrow$
  $\forall\, Q \in Quorum,\, b \in Ballot :$
  $(\forall\, a \in Q : (maxBal[a] \geq b)) \Rightarrow \exists\, v \in Value : SafeAt(b, v)$

Checked as an invariant by *TLC* with 3 acceptors, 3 ballots, 2 values

⟨1⟩.USE DEF *Ballot*
⟨1⟩1. SUFFICES ASSUME *TypeOK*, *VInv*2, *VInv*3,
    NEW $Q \in Quorum$, NEW $b \in Ballot$,
    $(\forall\, a \in Q : (maxBal[a] \geq b))$
  PROVE $\exists\, v \in Value : SafeAt(b, v)$
  OBVIOUS
⟨1⟩2.CASE $b = 0$
  BY *ValueNonempty*, ⟨1⟩1, *SafeAtProp*, ⟨1⟩2, *Zenon*
⟨1⟩4. SUFFICES ASSUME $b \neq 0$
  PROVE $\exists\, v \in Value :$
    $\exists\, c \in -1\,..\,(b-1) :$
      $\land\, (c \neq -1) \Rightarrow \land\, SafeAt(c, v)$
        $\land\, \forall\, a \in Q :$
          $\forall\, w \in Value :$
            $VotedFor(a, c, w) \Rightarrow (w = v)$
      $\land\, \forall\, d \in (c+1)\,..\,(b-1),\, a \in Q : DidNotVoteIn(a, d)$
  BY ⟨1⟩1, ⟨1⟩2, *SafeAtProp*
⟨1⟩5.CASE $\forall\, a \in Q,\, c \in 0\,..\,(b-1) : DidNotVoteIn(a, c)$

BY ⟨1⟩5, *ValueNonempty*

⟨1⟩6.CASE $\exists\, a \in Q,\ c \in 0 \,..\, (b-1) : \neg DidNotVoteIn(a,\ c)$

⟨2⟩1. PICK $c \in 0 \,..\, (b-1)$:
$\quad \land \exists\, a \in Q : \neg DidNotVoteIn(a,\ c)$
$\quad \land \forall\, d \in (c+1) \,..\, (b-1),\ a \in Q : DidNotVoteIn(a,\ d)$

⟨3⟩ DEFINE $S \triangleq \{c \in 0 \,..\, (b-1) : \exists\, a \in Q : \neg DidNotVoteIn(a,\ c)\}$

⟨3⟩1. $S \neq \{\}$
BY ⟨1⟩6

⟨3⟩2. PICK $c \in S : \forall\, d \in S : c \geq d$

⟨4⟩2. $IsFiniteSet(S)$
BY *FS_Interval*, *FS_Subset*, $0 \in Int$, $b - 1 \in Int$, *Zenon*

⟨4⟩3. QED
BY ⟨3⟩1, ⟨4⟩2, *FiniteSetHasMax*

⟨3⟩.QED
BY ⟨3⟩2 DEF *Ballot*

⟨2⟩4. PICK $a0 \in Q,\ v \in Value : VotedFor(a0,\ c,\ v)$
BY ⟨2⟩1 DEF *DidNotVoteIn*

⟨2⟩5. $\forall\, a \in Q : \forall\, w \in Value :$
$\quad VotedFor(a,\ c,\ w) \Rightarrow (w = v)$
BY ⟨2⟩4, *QA*, ⟨1⟩1 DEF *VInv*3

⟨2⟩6. $SafeAt(c,\ v)$
BY ⟨1⟩1, ⟨2⟩4, *QA* DEF *VInv*2

⟨2⟩7. QED
BY ⟨2⟩1, ⟨2⟩5, ⟨2⟩6

⟨1⟩7. QED
BY ⟨1⟩5, ⟨1⟩6

The progress property we require of the algorithm is that a quorum of acceptors, by themselves, can eventually choose a value v. This means that, for some quorum Q and ballot b, the acceptors a of Q must make $SafeAt(b,\ v)$ true by executing $IncreaseMaxBal(a,\ b)$ and then must execute $VoteFor(a,\ b,\ v)$ to choose v. In order to be able to execute $VoteFor(a,\ b,\ v)$, acceptor a must not execute a $Ballot(a,\ c)$ action for any $c > b$.

These considerations lead to the following liveness requirement *LiveAssumption*. The *WF* condition ensures that the acceptors a in Q eventually execute the necessary $BallotAction(a,\ b)$ actions if they are enabled, and the $\Box[\ldots]\_vars$ condition ensures that they never perform *BallotAction* actions for higher-numbered ballots, so the necessary $BallotAction(a,\ b)$ actions are enabled.

$$
\begin{aligned}
&LiveAssumption \triangleq \\
&\quad \exists\, Q \in Quorum,\ b \in Ballot : \\
&\qquad \land \forall\, self \in Q : \mathrm{WF}_{vars}(BallotAction(self,\ b)) \\
&\qquad \land \Box[\forall\, self \in Q : \forall\, c \in Ballot : \\
&\qquad\qquad (c > b) \Rightarrow \neg BallotAction(self,\ c)]_{vars}
\end{aligned}
$$

$$LiveSpec \triangleq Spec \land LiveAssumption$$

LiveAssumption is stronger than necessary. Instead of requiring that an acceptor in Q never executes an action of a higher-numbered ballot than b, it suffices that it doesn't execute such an action until unless it has voted in ballot b. However, the natural liveness requirement for a *Paxos* consensus algorithm implies condition *LiveAssumption*.

Condition *LiveAssumption* is a liveness property, constraining only what eventually happens. It is straightforward to replace "eventually happens" by "happens within some length of time" and convert *LiveAssumption* into a real-time condition. We have not done that for three reasons:

1. The real-time requirement and, we believe, the real-time reasoning will be more complicated, since temporal logic was developed to abstract away much of the complexity of reasoning about explicit times.
2. *TLAPS* does not yet support reasoning about real numbers.
3. Reasoning about real-time specifications consists entirely of safety reasoning, which is almost entirely action reasoning. We want to see how the TLA+ proof language and *TLAPS* do on temporal logic reasoning.

Here are two temporal-logic proof rules. Their validity is obvious when you understand what they mean.

THEOREM $AlwaysForall \triangleq$
ASSUME NEW CONSTANT S, NEW TEMPORAL $P(\_)$
PROVE $(\forall s \in S : \Box P(s)) \equiv \Box(\forall s \in S : P(s))$
OBVIOUS

LEMMA $EventuallyAlwaysForall \triangleq$
ASSUME NEW CONSTANT S, $IsFiniteSet(S)$,
NEW TEMPORAL $P(\_)$
PROVE $(\forall s \in S : \Diamond\Box P(s)) \Rightarrow \Diamond\Box(\forall s \in S : P(s))$
⟨1⟩.DEFINE $A(x) \triangleq \Diamond\Box P(x)$
$L(T) \triangleq \forall s \in T : A(s)$
$R(T) \triangleq \forall s \in T : P(s)$
$Q(T) \triangleq L(T) \Rightarrow \Diamond\Box R(T)$
⟨1⟩1. $Q(\{\})$
⟨2⟩1. $R(\{\})$ OBVIOUS
⟨2⟩2. $\Diamond\Box R(\{\})$ BY ⟨2⟩1, PTL
⟨2⟩.QED BY ⟨2⟩2
⟨1⟩2. ASSUME NEW T, NEW x
PROVE $Q(T) \Rightarrow Q(T \cup \{x\})$
⟨2⟩1. $L(T \cup \{x\}) \Rightarrow A(x)$
⟨3⟩.HIDE DEF A
⟨3⟩.QED OBVIOUS
⟨2⟩2. $L(T \cup \{x\}) \land Q(T) \Rightarrow \Diamond\Box R(T)$
OBVIOUS
⟨2⟩3. $\Diamond\Box R(T) \land A(x) \Rightarrow \Diamond\Box(R(T) \land P(x))$
BY PTL
⟨2⟩4. $R(T) \land P(x) \Rightarrow R(T \cup \{x\})$
OBVIOUS
⟨2⟩5. $\Diamond\Box(R(T) \land P(x)) \Rightarrow \Diamond\Box R(T \cup \{x\})$

BY $\langle 2\rangle 4$, PTL

$\langle 2\rangle$.QED

BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 5$

$\langle 1\rangle$.HIDE DEF Q

$\langle 1\rangle 3.\ \forall\, T : \mathit{IsFiniteSet}(T) \Rightarrow Q(T)$

BY $\langle 1\rangle 1$, $\langle 1\rangle 2$, $\mathit{FS\_Induction}$, IsaM(“blast”)

$\langle 1\rangle 4.\ Q(S)$

BY $\langle 1\rangle 3$

$\langle 1\rangle$.QED

BY $\langle 1\rangle 4$ DEF Q

Here is our proof that *LiveSpec* implements the specification *LiveSpec* of module *Consensus* under our refinement mapping.

THEOREM $\mathit{Liveness} \triangleq \mathit{LiveSpec} \Rightarrow C\,!\,\mathit{LiveSpec}$

$\langle 1\rangle$ SUFFICES ASSUME NEW $Q \in \mathit{Quorum}$, NEW $b \in \mathit{Ballot}$

PROVE $\mathit{Spec} \land \mathit{LiveAssumption}\,!(Q,\, b) \Rightarrow C\,!\,\mathit{LiveSpec}$

BY Isa DEF $\mathit{LiveSpec}$, $\mathit{LiveAssumption}$

$\langle 1\rangle$a. $\mathit{IsFiniteSet}(Q)$

BY QA, $\mathit{AcceptorFinite}$, $\mathit{FS\_Subset}$

$\langle 1\rangle 1.\ C\,!\,\mathit{LiveSpec} \equiv C\,!\,\mathit{Spec} \land (\Box\Diamond\langle C\,!\,\mathit{Next}\rangle_{C\,!\,vars} \lor \Box\Diamond(\mathit{chosen} \neq \{\}))$

BY $\mathit{ValueNonempty}$, $C\,!\,\mathit{LiveSpecEquals}$

$\langle 1\rangle$ DEFINE $\mathit{LNext} \triangleq \exists\, \mathit{self} \in \mathit{Acceptor},\ c \in \mathit{Ballot}$:

$\land\ \mathit{BallotAction}(\mathit{self},\, c)$

$\land\ (\mathit{self} \in Q) \Rightarrow (c \leq b)$

$\langle 1\rangle 2.\ \mathit{Spec} \land \mathit{LiveAssumption}\,!(Q,\, b) \Rightarrow \Box[\mathit{LNext}]_{vars}$

$\langle 2\rangle 1.\ \land\ \mathit{TypeOK}$

$\land\ [\mathit{Next}]_{vars}$

$\land\ [\forall\, \mathit{self} \in Q : \forall\, c \in \mathit{Ballot} : (c > b) \Rightarrow \neg \mathit{BallotAction}(\mathit{self},\, c)]_{vars}$

$\Rightarrow [\mathit{LNext}]_{vars}$

BY $\mathit{NextDef}$ DEF LNext, Ballot

$\langle 2\rangle 2.\ \land\ \Box\,\mathit{TypeOK}$

$\land\ \Box[\mathit{Next}]_{vars}$

$\land\ \Box[\forall\, \mathit{self} \in Q : \forall\, c \in \mathit{Ballot} : (c > b) \Rightarrow \neg \mathit{BallotAction}(\mathit{self},\, c)]_{vars}$

$\Rightarrow \Box[\mathit{LNext}]_{vars}$

BY $\langle 2\rangle 1$, PTL

$\langle 2\rangle 3$. QED

BY $\langle 2\rangle 2$, $VT2$, Isa DEF Spec, VInv

$\langle 1\rangle$ DEFINE $\mathit{LNInv1} \triangleq \forall\, a \in Q : \mathit{maxBal}[a] \leq b$

$\mathit{LInv1} \triangleq \mathit{VInv} \land \mathit{LNInv1}$

$\langle 1\rangle 3.\ \mathit{LInv1} \land [\mathit{LNext}]_{vars} \Rightarrow \mathit{LInv1}'$

$\langle 2\rangle 1$. SUFFICES ASSUME $\mathit{LInv1}$, $[\mathit{LNext}]_{vars}$

PROVE $LInv1'$

OBVIOUS

$\langle 2\rangle 2.$ $VInv'$

BY $\langle 2\rangle 1$, *NextDef*, *InductiveInvariance* DEF *LInv*1, *VInv*

$\langle 2\rangle 3.$ $LNInv1'$

BY $\langle 2\rangle 1$, *QA* DEF *BallotAction*, *IncreaseMaxBal*, *VoteFor*, *VInv*, *TypeOK*, *vars*

$\langle 2\rangle$.QED

BY $\langle 2\rangle 2$, $\langle 2\rangle 3$

$\langle 1\rangle 4.$ $\forall a \in Q :$

$$VInv \land (maxBal[a] = b) \land [LNext]_{vars} \Rightarrow VInv' \land (maxBal'[a] = b)$$

$\langle 2\rangle 1.$ SUFFICES ASSUME NEW $a \in Q$,

$VInv,\ maxBal[a] = b,\ [LNext]_{vars}$

PROVE $VInv' \land (maxBal'[a] = b)$

OBVIOUS

$\langle 2\rangle 2.$ $VInv'$

BY $\langle 2\rangle 1$, *NextDef*, *InductiveInvariance* DEF *VInv*

$\langle 2\rangle 3.$ $maxBal'[a] = b$

BY $\langle 2\rangle 1$, *QA* DEF *BallotAction*, *IncreaseMaxBal*, *VoteFor*, *VInv*, *TypeOK*, *Ballot*, *vars*

$\langle 2\rangle$.QED

BY $\langle 2\rangle 2$, $\langle 2\rangle 3$

$\langle 1\rangle 5.$ $Spec \land LiveAssumption!(Q, b) \Rightarrow$

$$\Diamond\Box(\forall self \in Q : maxBal[self] = b)$$

$\langle 2\rangle 1.$ SUFFICES ASSUME NEW $self \in Q$

PROVE $Spec \land LiveAssumption!(Q, b) \Rightarrow \Diamond\Box(maxBal[self] = b)$

BY $\langle 1\rangle$a, *EventuallyAlwaysForall* * doesn't check, even when introducing definitions

PROOF OMITTED

$\langle 2\rangle$ DEFINE $P \triangleq LInv1 \land \neg(maxBal[self] = b)$

$QQ \triangleq LInv1 \land (maxBal[self] = b)$

$A \triangleq BallotAction(self, b)$

$\langle 2\rangle 2.$ $\Box[LNext]_{vars} \land \mathrm{WF}_{vars}(A) \Rightarrow (LInv1 \leadsto QQ)$

$\langle 3\rangle 1.$ $P \land [LNext]_{vars} \Rightarrow (P' \lor QQ')$

BY $\langle 1\rangle 3$

$\langle 3\rangle 2.$ $P \land \langle LNext \land A\rangle_{vars} \Rightarrow QQ'$

$\langle 4\rangle 1.$ SUFFICES ASSUME $LInv1,\ LNext,\ A$

PROVE QQ'

OBVIOUS

$\langle 4\rangle 2.$ $LInv1'$

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

$\langle 4\rangle 3.$CASE $IncreaseMaxBal(self, b)$

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, *QA* DEF *IncreaseMaxBal*, *VInv*, *TypeOK*

$\langle 4\rangle 4.$CASE $\exists v \in Value : VoteFor(self, b, v)$

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 4$, *QA* DEF *VoteFor*, *VInv*, *TypeOK*

$\langle 4\rangle 5.$ QED

$\quad$ BY ⟨4⟩1, ⟨4⟩3, ⟨4⟩4 DEF *BallotAction*

⟨3⟩3. $P \Rightarrow$ ENABLED $\langle A \rangle_{vars}$

⟨4⟩1. (ENABLED $\langle A \rangle_{vars}$) $\equiv$

$$\begin{array}{l}
\exists\, votesp,\ maxBalp : \\
\quad \land \lor \land\, b > maxBal[self] \\
\qquad\quad \land\, maxBalp = [maxBal \text{ EXCEPT } ![self] = b] \\
\qquad\quad \land\, votesp = votes \\
\qquad \lor\, \exists\, v \in Value : \\
\qquad\qquad \land\, maxBal[self] \leq b \\
\qquad\qquad \land\, DidNotVoteIn(self,\ b) \\
\qquad\qquad \land\, \forall\, p \in Acceptor \setminus \{self\} : \\
\qquad\qquad\qquad \forall\, w \in Value : VotedFor(p,\ b,\ w) \Rightarrow (w = v) \\
\qquad\qquad \land\, SafeAt(b,\ v) \\
\qquad\qquad \land\, votesp = [votes \text{ EXCEPT } ![self] = votes[self] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \cup \{\langle b,\ v \rangle\}] \\
\qquad\qquad \land\, maxBalp = [maxBal \text{ EXCEPT } ![self] = b] \\
\quad \land\, \langle votesp,\ maxBalp \rangle \neq \langle votes,\ maxBal \rangle
\end{array}$$

BY DEF *BallotAction*, *IncreaseMaxBal*, *VoteFor*, *vars*, *SafeAt*, *DidNotVoteIn*, *VotedFor*

PROOF OMITTED

⟨4⟩.SUFFICES ASSUME P

PROVE $\exists\, votesp,\ maxBalp :$

$$\begin{array}{l}
\land\, b > maxBal[self] \\
\land\, maxBalp = [maxBal \text{ EXCEPT } ![self] = b] \\
\land\, votesp = votes \\
\land\, \langle votesp,\ maxBalp \rangle \neq \langle votes,\ maxBal \rangle
\end{array}$$

BY ⟨4⟩1

⟨4⟩ WITNESS *votes*, $[maxBal \text{ EXCEPT } ![self] = b]$

⟨4⟩.QED BY *QA* DEF *VInv*, *TypeOK*, *Ballot*

⟨3⟩.QED BY ⟨3⟩1, ⟨3⟩2, ⟨3⟩3, *PTL*

⟨2⟩3. $QQ \land \Box[LNext]_{vars} \Rightarrow \Box QQ$

⟨3⟩1. $QQ \land [LNext]_{vars} \Rightarrow QQ'$

BY ⟨1⟩3, ⟨1⟩4

⟨3⟩.QED BY ⟨3⟩1, *PTL*

⟨2⟩4.$\Box QQ \Rightarrow \Box(maxBal[self] = b)$

BY *PTL*

⟨2⟩5. $LiveAssumption!(Q,\ b) \Rightarrow \mathrm{WF}_{vars}(A)$

BY *Isa*

⟨2⟩6. $Spec \Rightarrow LInv1$

⟨3⟩1. $Init \Rightarrow VInv$

BY *InitImpliesInv*

⟨3⟩2. $Init \Rightarrow LNInv1$

BY *QA* DEF *Init*, *Ballot*

⟨3⟩.QED BY ⟨3⟩1, ⟨3⟩2 DEF *Spec*

⟨2⟩.QED

BY ⟨2⟩2, ⟨2⟩3, ⟨2⟩4, ⟨2⟩5, ⟨2⟩6, ⟨1⟩2, PTL

⟨1⟩ DEFINE $LNInv2 \triangleq \forall a \in Q : maxBal[a] = b$
$LInv2 \triangleq VInv \land LNInv2$

⟨1⟩6. $LInv2 \land [LNext]_{vars} \Rightarrow LInv2'$
BY ⟨1⟩4, $QuorumNonEmpty$

⟨1⟩7. $Spec \land LiveAssumption!(Q, b) \Rightarrow \Diamond\Box(chosen \neq \{\})$
⟨2⟩ DEFINE $Voted(a) \triangleq \exists v \in Value : VotedFor(a, b, v)$
⟨2⟩1. $Spec \land LiveAssumption!(Q, b) \Rightarrow \Diamond\Box LInv2$
⟨3⟩1. $Spec \land LiveAssumption!(Q, b) \Rightarrow \Diamond\Box LNInv2$
BY ⟨1⟩5 * doesn't check
PROOF OMITTED
⟨3⟩.QED BY ⟨3⟩1, $VT2$, PTL
⟨2⟩2. $LInv2 \land (\forall a \in Q : Voted(a)) \Rightarrow (chosen \neq \{\})$
⟨3⟩1. SUFFICES ASSUME $LInv2$,
$\forall a \in Q : Voted(a)$
PROVE $chosen \neq \{\}$
OBVIOUS
⟨3⟩2. $\exists v \in Value : \forall a \in Q : VotedFor(a, b, v)$
⟨4⟩2. PICK $a0 \in Q, v \in Value : VotedFor(a0, b, v)$
BY ⟨3⟩1, $QuorumNonEmpty$
⟨4⟩3. ASSUME NEW $a \in Q$
PROVE $VotedFor(a, b, v)$
BY ⟨3⟩1, ⟨4⟩2, QA DEF $VInv$, $VInv3$
⟨4⟩4. QED
BY ⟨4⟩3
⟨3⟩3. QED
BY ⟨3⟩2 DEF $chosen$, $ChosenIn$
⟨2⟩3. $Spec \land LiveAssumption!(Q, b) \Rightarrow (\forall a \in Q : \Diamond\Box Voted(a))$
⟨3⟩1. SUFFICES ASSUME NEW $self \in Q$
PROVE $Spec \land LiveAssumption!(Q, b) \Rightarrow \Diamond\Box Voted(self)$
OBVIOUS * doesn't check?!
PROOF OMITTED
⟨3⟩2. $Spec \land LiveAssumption!(Q, b) \Rightarrow \Diamond Voted(self)$
⟨4⟩2. $\Box[LNext]_{vars} \land WF_{vars}(BallotAction(self, b))$
$\Rightarrow ((LInv2 \land \neg Voted(self)) \leadsto LInv2 \land Voted(self))$
⟨5⟩ DEFINE $P \triangleq LInv2 \land \neg Voted(self)$
$QQ \triangleq LInv2 \land Voted(self)$
$A \triangleq BallotAction(self, b)$
⟨5⟩1. $P \land [LNext]_{vars} \Rightarrow (P' \lor QQ')$
BY ⟨1⟩6
⟨5⟩2. $P \land \langle LNext \land A \rangle_{vars} \Rightarrow QQ'$
⟨6⟩1. SUFFICES ASSUME P,
$LNext$,

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad A \\
\qquad\qquad\quad \textsc{prove}\ \ QQ' \\
\quad \textsc{obvious} \\
\langle 6\rangle 2.\textsc{case}\ \exists\, v \in Value : VoteFor(self,\, b,\, v) \\
\quad \textsc{by}\ \langle 6\rangle 1,\ \langle 6\rangle 2,\ \langle 5\rangle 1,\ QA,\ Zenon\ \textsc{def}\ VoteFor,\ Voted,\ VotedFor,\ LInv2,\ VInv,\ TypeOK \\
\langle 6\rangle 3.\textsc{case}\ IncreaseMaxBal(self,\, b) \\
\quad \textsc{by}\ \langle 6\rangle 1,\ \langle 6\rangle 3\ \ \textsc{def}\ IncreaseMaxBal,\ Ballot \\
\langle 6\rangle 4.\ \textsc{qed} \\
\quad \textsc{by}\ \langle 6\rangle 1,\ \langle 6\rangle 2,\ \langle 6\rangle 3\ \ \textsc{def}\ BallotAction
\end{array}
$$

$$
\begin{array}{l}
\langle 5\rangle 3.\ P \Rightarrow \textsc{enabled}\ \langle A\rangle_{vars} \\
\quad \langle 6\rangle 1.\ \textsc{suffices}\ \textsc{assume}\ P \\
\qquad\qquad\qquad \textsc{prove}\ \ \textsc{enabled}\ \langle A\rangle_{vars} \\
\qquad \textsc{obvious} \\
\quad \langle 6\rangle 2.\ (\textsc{enabled}\ \langle A\rangle_{vars}) \equiv \\
\qquad\qquad \exists\, votesp,\, maxBalp : \\
\qquad\qquad\quad \land\ \lor\ \land\ b > maxBal[self] \\
\qquad\qquad\qquad\quad\ \land\ maxBalp = [maxBal\ \textsc{except}\ ![self] = b] \\
\qquad\qquad\qquad\quad\ \land\ votesp = votes \\
\qquad\qquad\qquad \lor\ \exists\, v \in Value : \\
\qquad\qquad\qquad\qquad \land\ maxBal[self] \le b \\
\qquad\qquad\qquad\qquad \land\ DidNotVoteIn(self,\, b) \\
\qquad\qquad\qquad\qquad \land\ \forall\, p \in Acceptor \setminus \{self\} : \\
\qquad\qquad\qquad\qquad\quad \forall\, w \in Value : VotedFor(p,\, b,\, w) \Rightarrow (w = v) \\
\qquad\qquad\qquad\qquad \land\ SafeAt(b,\, v) \\
\qquad\qquad\qquad\qquad \land\ votesp = [votes\ \textsc{except}\ ![self] = votes[self] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \cup \{\langle b,\, v\rangle\}] \\
\qquad\qquad\qquad\qquad \land\ maxBalp = [maxBal\ \textsc{except}\ ![self] = b] \\
\qquad\qquad\quad \land\ \langle votesp,\, maxBalp\rangle \neq \langle votes,\, maxBal\rangle \\
\textsc{by}\ \ \textsc{def}\ BallotAction,\ IncreaseMaxBal,\ VoteFor,\ vars,\ SafeAt, \\
\qquad DidNotVoteIn,\ VotedFor \\
\quad \textsc{proof omitted} \\
\quad \langle 6\rangle\ \textsc{suffices} \\
\qquad\qquad \exists\, votesp,\, maxBalp : \\
\qquad\qquad\quad \land\ \exists\, v \in Value : \\
\qquad\qquad\qquad\quad \land\ maxBal[self] \le b \\
\qquad\qquad\qquad\quad \land\ DidNotVoteIn(self,\, b) \\
\qquad\qquad\qquad\quad \land\ \forall\, p \in Acceptor \setminus \{self\} : \\
\qquad\qquad\qquad\qquad\quad \forall\, w \in Value : VotedFor(p,\, b,\, w) \Rightarrow (w = v) \\
\qquad\qquad\qquad\quad \land\ SafeAt(b,\, v) \\
\qquad\qquad\qquad\quad \land\ votesp = [votes\ \textsc{except}\ ![self] = votes[self] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \cup \{\langle b,\, v\rangle\}] \\
\qquad\qquad\qquad\quad \land\ maxBalp = [maxBal\ \textsc{except}\ ![self] = b] \\
\qquad\qquad\quad \land\ \langle votesp,\, maxBalp\rangle \neq \langle votes,\, maxBal\rangle \\
\qquad \textsc{by}\ \langle 6\rangle 2 \\
\quad \langle 6\rangle\ \textsc{define}\ someVoted\ \triangleq\ \exists\, p \in Acceptor \setminus \{self\} :
\end{array}
$$

```
                                ∃ w ∈ Value : VotedFor(p, b, w)
              vp ≜ CHOOSE p ∈ Acceptor \ {self} :
                                ∃ w ∈ Value : VotedFor(p, b, w)
              vpval ≜ CHOOSE w ∈ Value : VotedFor(vp, b, w)
      ⟨6⟩3. someVoted ⇒ ∧ vp ∈ Acceptor
                        ∧ vpval ∈ Value
                        ∧ VotedFor(vp, b, vpval)
        BY Zenon
      ⟨6⟩ DEFINE v ≜ IF someVoted THEN vpval
                                  ELSE CHOOSE v ∈ Value : SafeAt(b, v)
      ⟨6⟩4. (v ∈ Value) ∧ SafeAt(b, v)
        BY ⟨6⟩1, ⟨6⟩3, VT4 DEF VInv, VInv2, Ballot
      ⟨6⟩ DEFINE votesp ≜ [votes EXCEPT ![self] = votes[self] ∪ {⟨b, v⟩}]
                 maxBalp ≜ [maxBal EXCEPT ![self] = b]
      ⟨6⟩ WITNESS votesp, maxBalp
      ⟨6⟩ SUFFICES ∧ maxBal[self] ≤ b
                   ∧ DidNotVoteIn(self, b)
                   ∧ ∀ p ∈ Acceptor \ {self} :
                        ∀ w ∈ Value : VotedFor(p, b, w) ⇒ (w = v)
                   ∧ votesp ≠ votes
        BY ⟨6⟩4, Zenon
      ⟨6⟩5. maxBal[self] ≤ b
        BY ⟨6⟩1 DEF Ballot
      ⟨6⟩6. DidNotVoteIn(self, b)
        BY ⟨6⟩1 DEF Voted, DidNotVoteIn
      ⟨6⟩7. ASSUME NEW p ∈ Acceptor \ {self},
                   NEW w ∈ Value,
                   VotedFor(p, b, w)
            PROVE  w = v
        BY ⟨6⟩7, ⟨6⟩3, ⟨6⟩1 DEF VInv, VInv3
      ⟨6⟩8. votesp ≠ votes
        ⟨7⟩1. votesp[self] = votes[self] ∪ {⟨b, v⟩}
          BY ⟨6⟩1, QA DEF LInv2, VInv, TypeOK
        ⟨7⟩2. ∀ w ∈ Value : ⟨b, w⟩ ∉ votes[self]
          BY ⟨6⟩6 DEF DidNotVoteIn, VotedFor
        ⟨7⟩3. QED
          BY ⟨7⟩1, ⟨7⟩2, ⟨6⟩4, Zenon
      ⟨6⟩9. QED
        BY ⟨6⟩5, ⟨6⟩6, ⟨6⟩7, ⟨6⟩8, Zenon
    ⟨5⟩4. QED
      BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, PTL
  ⟨4⟩3.□LInv2 ∧ ((LInv2 ∧ ¬Voted(self)) ⇝ LInv2 ∧ Voted(self))
         ⇒ ◇Voted(self)
    BY PTL
  ⟨4⟩4. LiveAssumption!(Q, b) ⇒ WF_vars(BallotAction(self, b))
```

BY *Isa*

$\langle 4 \rangle$.QED

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

$\langle 3 \rangle 3$. $Spec \Rightarrow \Box(Voted(self) \Rightarrow \Box Voted(self))$

$\langle 4 \rangle 1$. $(VInv \land Voted(self)) \land [Next]_{vars} \Rightarrow (VInv \land Voted(self))'$

$\langle 5 \rangle$ SUFFICES ASSUME $VInv$, $Voted(self)$, $[Next]_{vars}$

PROVE $VInv' \land Voted(self)'$

OBVIOUS

$\langle 5 \rangle 1$. $VInv'$

BY *InductiveInvariance*

$\langle 5 \rangle 2$. $Voted(self)'$

$\langle 6 \rangle$CASE $vars' = vars$

BY DEF $vars$, $Voted$, $VotedFor$

$\langle 6 \rangle$CASE $Next$

$\langle 7 \rangle 2$. PICK $a \in Acceptor, c \in Ballot : BallotAction(a, c)$

BY *NextDef* DEF $VInv$

$\langle 7 \rangle 3$.CASE $IncreaseMaxBal(a, c)$

BY $\langle 7 \rangle 3$ DEF $IncreaseMaxBal$, $Voted$, $VotedFor$

$\langle 7 \rangle 4$.CASE $\exists v \in Value : VoteFor(a, c, v)$

BY $\langle 7 \rangle 4$, QA DEF $VInv$, $TypeOK$, $VoteFor$, $Voted$, $VotedFor$

$\langle 7 \rangle 5$. QED

BY $\langle 7 \rangle 2$, $\langle 7 \rangle 3$, $\langle 7 \rangle 4$ DEF *BallotAction*

$\langle 6 \rangle$ QED

OBVIOUS

$\langle 5 \rangle 3$. QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

$\langle 4 \rangle 3$. QED

BY $\langle 4 \rangle 1$, $VT2$, PTL DEF $Spec$

$\langle 3 \rangle 4$. QED

BY $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, PTL

$\langle 2 \rangle 4$. $(\forall a \in Q : \Diamond\Box Voted(a)) \Rightarrow \Diamond\Box(\forall a \in Q : Voted(a))$

BY $\langle 1 \rangle$a, *EventuallyAlwaysForall* * doesn't check

PROOF OMITTED

$\langle 2 \rangle$.QED

BY $\langle 2 \rangle 1$, $VT2$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, $\langle 2 \rangle 4$, PTL

$\langle 1 \rangle$.QED

$\langle 2 \rangle 1$. $Spec \land LiveAssumption!(Q, b) \Rightarrow C!Spec \land \Diamond\Box(chosen \neq \{\})$

BY $VT3$, $\langle 1 \rangle 7$, *Isa*

$\langle 2 \rangle 2$. $Spec \land LiveAssumption!(Q, b) \Rightarrow C!Spec \land \Box\Diamond(chosen \neq \{\})$

BY $\langle 2 \rangle 1$, PTL

$\langle 2 \rangle$.QED

BY $\langle 2 \rangle 2$, $\langle 1 \rangle 1$, *Isa*

\ * Modification History
\ * Last modified *Fri Jul* 24 18:20:31 *CEST* 2020 by *merz*
\ * Last modified *Wed Apr* 29 12:24:23 *CEST* 2020 by *merz*
\ * Last modified *Mon* May 28 08:53:38 *PDT* 2012 by lamport