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*, *FiniteSets*, *TLC*, *TLAPS*

THEOREM $SMT \triangleq \text{TRUE}$

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

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 specifies 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 ∈ Acceptor ↦ {}],
            maxBal = [a ∈ Acceptor ↦ -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) ≜
  LET SA[bb ∈ Ballot] ≜
```

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

```
                ∨ 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]
  }
```

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 ) {
  when ∧ maxBal[self] ≤ b
       ∧ DidNotVoteIn(self, b)
       ∧ ∀ p ∈ Acceptor \ {self} :
           ∀ w ∈ Value : VotedFor(p, b, w) ⇒ (w = v)
       ∧ SafeAt(b, v) ;
  votes[self]    := votes[self] ∪ {⟨b, v⟩} ;
  maxBal[self] := b
  }
```

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 ∈ Acceptor ) {
  acc :  while ( TRUE ) {
           with ( b ∈ 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 $Silly[42] \neq Silly[42]$.)

To prove the desired property of *SafeAt*, we use the following proof rule. It will eventually be in a standard module–probably in *TLAPS*. However, for now, we put it here.

THEOREM $RecursiveFcnOfNat \triangleq$
ASSUME NEW $Def(\_,\ \_)$,
$\forall\, n \in Nat :$
$\forall\, g,\ h : (\forall\, i \in 0\,..\,(n-1) : g[i] = h[i]) \Rightarrow (Def(g,\ n) = Def(h,\ n))$
PROVE LET $f[n \in Nat] \triangleq Def(f,\ n)$
IN $f = [n \in Nat \mapsto Def(f,\ n)]$
PROOF OMITTED

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

THEOREM $SafeAtProp \triangleq$
$\forall\, b \in Ballot,\ v \in Value :$
$SafeAt(b,\ v) =$
$\lor\ b = 0$
$\lor \exists\, Q \in Quorum :$
$\land \forall\, a \in Q \quad : maxBal[a] \geq b$
$\land \exists\, c \in -1\,..\,(b-1) :$
$\land\ (c \neq -1) \Rightarrow\ \land\ SafeAt(c,\ v)$
$\land \forall\, a \in Q :$
$\forall\, w \in Value :$
$VotedFor(a,\ c,\ w) \Rightarrow (w = v)$
$\land \forall\, d \in (c+1)\,..\,(b-1),\ a \in Q : DidNotVoteIn(a,\ d)$
$\langle 1\rangle 1$. SUFFICES ASSUME NEW $v \in Value$
PROVE $\forall\, b \in Ballot : SafeAtProp!(b,\ v)$

OBVIOUS

⟨1⟩ USE DEF $Ballot$

⟨1⟩ DEFINE $Def(SA,\ bb) \triangleq$

$\lor \quad bb = 0$

$\lor \quad \exists\, Q \in Quorum :$

$\land \forall\, a \in Q \quad : maxBal[a] \geq bb$

$\land \exists\, c \in -1 \,..\, (bb-1) :$

$\land (c \neq -1) \Rightarrow \land\, SA[c]$

$\land \forall\, a \in Q :$

$\forall\, w \in Value :$

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

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

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

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

BY DEF $SafeAt$

⟨1⟩3. $\forall\, n \in Nat :$

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

⟨2⟩1. SUFFICES ASSUME NEW $n \in Nat$, NEW g, NEW h,

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

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

OBVIOUS

⟨2⟩2. $Def(g,\ n)!2 = Def(h,\ n)!2$

⟨3⟩1. ASSUME NEW $Q \in Quorum$,

NEW $c \in -1 \,..\, (n-1)$,

$c \neq -1$

PROVE $Def(g,\ n)!2!(Q)!2!(c)!1!2 = Def(h,\ n)!2!(Q)!2!(c)!1!2$

⟨4⟩1. SUFFICES $g[c] = h[c]$

OBVIOUS

⟨4⟩2. $c \in 0 \,..\, (n-1)$

BY ⟨3⟩1, $SimpleArithmetic$

⟨4⟩3. QED

BY ⟨4⟩2, ⟨2⟩1

⟨3⟩2. QED

BY ⟨3⟩1

⟨2⟩3. QED

BY ⟨2⟩1, ⟨2⟩2

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

⟨2⟩ HIDE DEF Def

⟨2⟩ QED

BY ONLY ⟨1⟩3, $RecursiveFcnOfNat$

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

⟨2⟩ HIDE DEF Def

⟨2⟩ QED

BY ⟨1⟩4

⟨1⟩6. QED
  BY ⟨1⟩2, ⟨1⟩5 DEF *SafeAt*

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

$$\begin{aligned} \mathit{TypeOK} \triangleq\ & \land\ \mathit{votes} \in [\mathit{Acceptor} \to \text{SUBSET } (\mathit{Ballot} \times \mathit{Value})] \\ & \land\ \mathit{maxBal} \in [\mathit{Acceptor} \to \mathit{Ballot} \cup \{-1\}] \end{aligned}$$

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 $\mathit{ChosenIn}(b,\ v)$ to be true iff a quorum of acceptors have all voted for v in ballot b.

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

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

Mathematical Induction

The following axiom asserts the validity of a standard proof by mathematical induction. Some such axiom should be included in the standard *TLAPS* module. However, instead of a rule expressed it in terms of a function f, it would be more convenient to use one expressed as follows in terms of an operator f:

AXIOM ASSUME NEW $f(_),\ f(0),\ \forall\, n \in \mathit{Nat} : f(n) \Rightarrow f(n+1)$PROVE $\forall\, n \in \mathit{Nat} : f(n)$

However, the *TLAPS* proof system cannot yet handle proofs that use this rule. So, for now we use this axiom.

$$\begin{aligned} \text{AXIOM } \mathit{SimpleNatInduction} \triangleq\ & \forall\, f :\ \land\, f[0] \\ & \land\ \forall\, n\ \in \mathit{Nat} : f[n] \Rightarrow f[n+1] \\ & \Rightarrow \forall\, n \in \mathit{Nat} : f[n] \end{aligned}$$

We use the *SimpleNatInduction* rule to prove the following rule, which expresses the soundness of what I believe is sometimes called "General Induction" or "Strong Induction".

THEOREM *GeneralNatInduction* $\triangleq$

$$\begin{aligned} & \forall\, f :\ \land\, f[0] \\ & \qquad \land\ \forall\, n\ \in \mathit{Nat} : (\forall\, j \in 0\,..\,n : f[j]) \Rightarrow f[n+1] \\ & \qquad \Rightarrow \forall\, n \in \mathit{Nat} : f[n] \end{aligned}$$

⟨1⟩1. SUFFICES ASSUME NEW f,
  $f[0]$,
  $\forall\, m \in \mathit{Nat} : (\forall\, j \in 0\,..\,m : f[j]) \Rightarrow f[m+1]$,
  NEW $n \in \mathit{Nat}$
 PROVE $f[n]$
  OBVIOUS

⟨1⟩ DEFINE $g \triangleq [m \in \mathit{Nat} \mapsto \forall\, j \in 0\,..\,m : f[j]]$

⟨1⟩2. $g[0]$
  ⟨2⟩1. $\forall\, x \in 0\,..\,0 : x = 0$
    BY *SimpleArithmetic*

```
  ⟨2⟩ QED
    BY ⟨1⟩1, ⟨2⟩1
⟨1⟩3. ASSUME NEW k ∈ Nat,  g[k]
       PROVE  g[k + 1]
  ⟨2⟩1. ∧ k ∈ 0 .. k
        ∧ k + 1 ∈ Nat
        ∧ ∀ x ∈ 0 .. (k + 1) : (x ∈ 0 .. k) ∨ (x = k + 1)
    BY SimpleArithmetic
  ⟨2⟩2. ∀ j ∈ 0 .. k : f[j]
    BY ⟨1⟩3, ⟨2⟩1
  ⟨2⟩3. f[k + 1]
    BY ⟨1⟩1, ⟨2⟩1, ⟨2⟩2
  ⟨2⟩4. QED
    BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩3
⟨1⟩4. ∀ k ∈ Nat : g[k]
  BY ⟨1⟩2, ⟨1⟩3, SimpleNatInduction
⟨1⟩5. QED
  ⟨2⟩1. n ∈ 0 .. n
    BY SimpleArithmetic
  ⟨2⟩2. QED
    BY ⟨2⟩1, ⟨1⟩4
```

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

```
LEMMA SafeLemma ≜
        TypeOK ⇒
          ∀ b ∈ Ballot :
            ∀ v ∈ Value :
              SafeAt(b, v) ⇒
                ∀ c ∈ 0 .. (b − 1) :
                  ∃ Q ∈ Quorum :
                    ∀ a ∈ Q : ∧ maxBal[a] ≥ c
                              ∧ ∨ DidNotVoteIn(a, c)
                                ∨ VotedFor(a, c, v)
⟨1⟩ SUFFICES ASSUME TypeOK
             PROVE  SafeLemma!2
  OBVIOUS
⟨1⟩ DEFINE P[b ∈ Ballot] ≜ ∀ c ∈ 0 .. b : SafeLemma!2!(c)
⟨1⟩1. P[0]
  ⟨2⟩1. SUFFICES ASSUME NEW c ∈ 0 .. 0
                 PROVE  SafeLemma!2!(c)
    BY DEF Ballot
  ⟨2⟩2. c = 0
    BY SimpleArithmetic DEF Ballot
  ⟨2⟩3. 0 .. (0 − 1) = {}
```

```
    ⟨3⟩1. ∀ x ∈ 0 .. (0 − 1) : FALSE
      BY SimpleArithmetic
    ⟨3⟩2. QED
      BY ⟨3⟩1
  ⟨2⟩4. QED
    BY ⟨2⟩2, ⟨2⟩3
⟨1⟩2. ASSUME NEW b ∈ Ballot, P[b]
       PROVE  P[b + 1]
  ⟨2⟩1. ∧ b + 1 ∈ Ballot
        ∧ (b + 1) − 1 = b
    BY SimpleArithmetic DEF Ballot
  ⟨2⟩2. 0 .. (b + 1) = (0 .. b) ∪ {b + 1}
    ⟨3⟩1. ∀ x ∈ 0 .. (b + 1) : x ∈ 0 .. b ∨ x = b + 1
      BY SimpleArithmetic DEF Ballot
    ⟨3⟩2. b + 1 ∈ 0 .. (b + 1) ∧ ∀ x ∈ 0 .. b : x ∈ 0 .. (b + 1)
      BY SimpleArithmetic DEF Ballot
    ⟨3⟩3. QED
      BY ⟨3⟩1, ⟨3⟩2
  ⟨2⟩3. SUFFICES ASSUME NEW v ∈ Value,
                        SafeAt(b + 1, v),
                        NEW c ∈ 0 .. b
                 PROVE  ∃ Q ∈ Quorum :
                          ∀ a ∈ Q : ∧ maxBal[a] ≥ c
                                    ∧ ∨ DidNotVoteIn(a, c)
                                      ∨ VotedFor(a, c, v)
    BY ⟨1⟩2, ⟨2⟩1, ⟨2⟩2
  ⟨2⟩4. PICK Q ∈ Quorum :
              ∧ ∀ a ∈ Q : maxBal[a] ≥ (b + 1)
              ∧ ∃ cc ∈ −1 .. b :
                  ∧ (cc ≠ −1) ⇒ ∧ SafeAt(cc, v)
                                ∧ ∀ a ∈ Q :
                                    ∀ w ∈ Value :
                                      VotedFor(a, cc, w) ⇒ (w = v)
                  ∧ ∀ d ∈ (cc + 1) .. b, a ∈ Q : DidNotVoteIn(a, d)
    ⟨3⟩1. b + 1 ≠ 0
      BY SimpleArithmetic DEF Ballot
    ⟨3⟩2. SafeAt(b + 1, v) = SafeAtProp!(b + 1, v)!2
      BY SafeAtProp, ⟨2⟩1
    ⟨3⟩3. @ = SafeAtProp!(b + 1, v)!2!2
      BY ⟨3⟩1
    ⟨3⟩4. @ = ∃ Q ∈ Quorum :
              ∧ ∀ a ∈ Q : maxBal[a] ≥ (b + 1)
              ∧ ∃ cc ∈ −1 .. b :
                  ∧ (cc ≠ −1) ⇒ ∧ SafeAt(cc, v)
                                ∧ ∀ a ∈ 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 $\langle 2\rangle 1$
$\langle 3\rangle 5$. QED
BY $\langle 3\rangle 2$, $\langle 3\rangle 3$, $\langle 3\rangle 4$, $\langle 2\rangle 3$
$\langle 2\rangle 5$. 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 $c > cc$
$\langle 3\rangle 1$. $c \in (cc+1) \,..\, b$
BY $\langle 2\rangle 6$, *SimpleArithmetic* DEF *Ballot*
$\langle 3\rangle 2$. $\forall\, a \in Q : \mathit{DidNotVoteIn}(a,\, c)$
BY $\langle 3\rangle 1$, $\langle 2\rangle 5$
$\langle 3\rangle 3$. $\forall\, a \in Q : \mathit{maxBal}[a] \geq c$
$\langle 4\rangle$ SUFFICES ASSUME NEW $a \in Q$
PROVE $\mathit{maxBal}[a] \geq c$
OBVIOUS
$\langle 4\rangle 1$. $\forall\, \mathit{mbal} \in \mathit{Ballot} \cup \{-1\} :$
$(\mathit{mbal} \geq b+1) \Rightarrow (\mathit{mbal} \geq c)$
BY $\langle 3\rangle 1$, *SimpleArithmetic* DEF *Ballot*
$\langle 4\rangle 2$. $\mathit{maxBal}[a] \geq b+1$
BY $\langle 2\rangle 4$
$\langle 4\rangle 3$. QED
BY QA, $a \in \mathit{Acceptor}$, $\langle 4\rangle 1$, $\langle 4\rangle 2$ DEF *TypeOK*
$\langle 3\rangle 4$. QED
BY $\langle 3\rangle 2$, $\langle 3\rangle 3$
$\langle 2\rangle 7$.CASE $c \leq cc$
$\langle 3\rangle 1$. $\land\, cc \in 0 \,..\, b$
$\land\, cc \neq -1$
BY $\langle 2\rangle 7$, *SimpleArithmetic* DEF *Ballot*
$\langle 3\rangle 2$. $\mathit{SafeLemma}!2!(cc)!(v)$
BY $\langle 1\rangle 2$, $\langle 3\rangle 1$
$\langle 3\rangle 3$. $\mathit{SafeAt}(cc,\, v)$
BY $\langle 2\rangle 5$, $\langle 3\rangle 1$
$\langle 3\rangle 4$.CASE $c = cc$
$\langle 4\rangle 1$. $\forall\, mb \in \mathit{Ballot} \cup \{-1\} : (mb \geq b+1) \Rightarrow (mb \geq c)$
BY $\langle 3\rangle 4$, $\langle 3\rangle 1$, *SimpleArithmetic* DEF *Ballot*
$\langle 4\rangle 2$. $\forall\, a \in Q : \mathit{maxBal}[a] \in \mathit{Ballot} \cup \{-1\}$
BY QA DEF *TypeOK*
$\langle 4\rangle 3$. $\forall\, a \in Q : \mathit{maxBal}[a] \geq c$

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

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

$\langle 5\rangle$ SUFFICES ASSUME NEW $a \in Q$,
$\neg DidNotVoteIn(a,\, c)$
PROVE $VotedFor(a,\, c,\, v)$

OBVIOUS

$\langle 5\rangle 1$. PICK $w \in Value : VotedFor(a,\, c,\, w)$

BY DEF $DidNotVoteIn$

$\langle 5\rangle 2$. $w = v$

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

$\langle 5\rangle 3$. QED

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

$\langle 4\rangle 5$. QED

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

$\langle 3\rangle 5$.CASE $c < cc$

$\langle 4\rangle 1$. $c \in 0 \,..\, (cc - 1)$

BY $\langle 3\rangle 1$, $\langle 3\rangle 5$, *SimpleArithmetic*

$\langle 4\rangle 2$. $SafeLemma!2!(cc)$

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

$\langle 4\rangle 3$. QED

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

$\langle 3\rangle 6$. QED

$\langle 4\rangle 1$. $(c < cc) \lor (c = cc)$

BY $\langle 2\rangle 7$, *SimpleArithmetic* DEF *Ballot*

$\langle 4\rangle 2$. QED

BY $\langle 3\rangle 4$, $\langle 3\rangle 5$, $\langle 4\rangle 1$

$\langle 2\rangle 8$. QED

$\langle 3\rangle 1$. $c \in Int \land cc \in Int$

BY *SimpleArithmetic* DEF *Ballot*

$\langle 3\rangle 2$. $(c > cc) \lor (c \leq cc)$

BY *SimpleArithmetic*

$\langle 3\rangle 3$. QED

BY $\langle 2\rangle 6$, $\langle 2\rangle 7$, $\langle 3\rangle 2$

$\langle 1\rangle 3$. $\forall\, b \in Ballot : P[b]$

BY $\langle 1\rangle 1$, $\langle 1\rangle 2$, *SimpleNatInduction* DEF *Ballot*

$\langle 1\rangle 4$. QED

$\langle 2\rangle 1$. $\forall\, b \in Ballot : b \in 0 \,..\, b$

BY *SimpleArithmetic* DEF *Ballot*

$\langle 2\rangle 2$. QED

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

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 ⟨1⟩6 of theorem *InductiveInvariance* below.

This lemma is used only in the proof of theorem $VT1$ below.

LEMMA $VT0 \triangleq \land\, TypeOK$
$\land\, VInv1$
$\land\, VInv2$
$\Rightarrow \forall\, v,\, w \in Value,\, b,\, c \in Ballot :$
$(b > c) \land SafeAt(b,\, v) \land ChosenIn(c,\, w) \Rightarrow (v = w)$

⟨1⟩ SUFFICES ASSUME $TypeOK,\, VInv1,\, VInv2,$
NEW $v \in Value$, NEW $w \in Value$
PROVE $\forall\, b,\, c \in Ballot :$
$(b > c) \land SafeAt(b,\, v) \land ChosenIn(c,\, w) \Rightarrow (v = w)$
OBVIOUS

⟨1⟩ $P \triangleq [b \in Ballot \mapsto$
$\forall\, c \in Ballot :$
$(b > c) \land SafeAt(b,\, v) \land ChosenIn(c,\, w) \Rightarrow (v = w)]$

⟨1⟩1. $P[0]$
⟨2⟩1. $\land\, 0 \in Ballot$
$\land \forall\, c \in Ballot : \neg(0 > c)$
BY *SimpleArithmetic* DEF *Ballot*
⟨2⟩2. QED

BY $\langle 2\rangle 1$
$\langle 1\rangle 2$. ASSUME NEW $b \in \mathit{Ballot}, \forall\, i \in 0 \,..\, b : P[i]$
PROVE $P[b+1]$
$\langle 2\rangle 1$. $b+1 \in \mathit{Ballot}$
BY *SimpleArithmetic* DEF *Ballot*
$\langle 2\rangle 2$. SUFFICES ASSUME NEW $c \in \mathit{Ballot}$, $b+1 > c$, $\mathit{SafeAt}(b+1,\, v)$, $\mathit{ChosenIn}(c,\, w)$
PROVE $v = w$
BY $\langle 2\rangle 1$
$\langle 2\rangle 3$. PICK $Q \in \mathit{Quorum} : \forall\, a \in Q : \mathit{VotedFor}(a,\, c,\, w)$
BY $\langle 2\rangle 2$ DEF *ChosenIn*
$\langle 2\rangle 4$. $b+1 \neq 0 \land ((b+1)-1 = b)$
BY *SimpleArithmetic* DEF *Ballot*
$\langle 2\rangle 5$. PICK $QQ \in \mathit{Quorum}$,
$d \in -1 \,..\, ((b+1)-1) :$
$\land\, (d \neq -1) \Rightarrow \land\, \mathit{SafeAt}(d,\, v)$
$\land\, \forall\, a \in QQ :$
$\forall\, x \in \mathit{Value} :$
$\mathit{VotedFor}(a,\, d,\, x) \Rightarrow (x = v)$
$\land\, \forall\, e \in (d+1) \,..\, ((b+1)-1),\, a \in QQ : \mathit{DidNotVoteIn}(a,\, e)$
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 4$, *SafeAtProp*
$\langle 2\rangle$ PICK $aa \in QQ \cap Q$: TRUE
BY *QA*
$\langle 2\rangle 6$. $c \leq d$
$\langle 3\rangle 1$. SUFFICES ASSUME $\neg(c \leq d)$
PROVE FALSE
OBVIOUS
$\langle 3\rangle 2$. $c \in (d+1) \,..\, ((b+1)-1)$
BY $\langle 2\rangle 2$, $\langle 3\rangle 1$, *SimpleArithmetic* DEF *Ballot*
$\langle 3\rangle 3$. $\mathit{VotedFor}(aa,\, c,\, w)$
BY $\langle 2\rangle 3$
$\langle 3\rangle 4$. $\mathit{DidNotVoteIn}(aa,\, c)$
BY $\langle 2\rangle 5$, $\langle 3\rangle 1$, $\langle 3\rangle 2$
$\langle 3\rangle 5$. QED
BY $\langle 3\rangle 3$, $\langle 3\rangle 4$ DEF *DidNotVoteIn*
$\langle 2\rangle 7$. $d \neq -1$
BY $\langle 2\rangle 6$, *SimpleArithmetic* DEF *Ballot*
$\langle 2\rangle 8$.CASE $c = d$
BY $\langle 2\rangle 3$, $\langle 2\rangle 5$, $\langle 2\rangle 7$, $\langle 2\rangle 8$
$\langle 2\rangle 9$.CASE $d > c$
$\langle 3\rangle 1$. $\mathit{SafeAt}(d,\, v)$
BY $\langle 2\rangle 5$, $\langle 2\rangle 7$
$\langle 3\rangle 2$. $d \in \mathit{Ballot} \land d \in 0 \,..\, b$
BY $\langle 2\rangle 6$, *SimpleArithmetic* DEF *Ballot*
$\langle 3\rangle 3$. $P[d]$
BY $\langle 1\rangle 2$, $\langle 3\rangle 2$

⟨3⟩4. QED
BY ⟨2⟩2, ⟨2⟩9, ⟨3⟩1, ⟨3⟩2, ⟨3⟩3
⟨2⟩10. QED
⟨3⟩1. $(c = d) \lor (d > c)$
BY ⟨2⟩6, *SimpleArithmetic* DEF *Ballot*
⟨3⟩2. QED
BY ⟨2⟩8, ⟨2⟩9, ⟨3⟩1
PICK $QQ \in Quorum : VotedFor(ac, c, w)$
BY *QuorumNonEmpty*, *QA* DEF *ChosenIn*
⟨1⟩3. $\forall\, b \in Ballot : P[b]$
BY ⟨1⟩1, ⟨1⟩2, *GeneralNatInduction* DEF *Ballot*

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

```
        ∧ VotedFor(a, c, w)
    BY ⟨1⟩4
  ⟨2⟩2. QED
    BY ⟨1⟩1, ⟨1⟩2, ⟨1⟩7, ⟨2⟩1, QA DEF VInv1
⟨1⟩8.CASE b > c
  BY ⟨1⟩1, ⟨1⟩6, ⟨1⟩3, ⟨1⟩8, VT0, ⟨1⟩2  ⟨2⟩1
⟨1⟩9.CASE c > b
    BY ⟨1⟩1, ⟨1⟩6, ⟨1⟩3, ⟨1⟩9, VT0, ⟨1⟩2  ⟨2⟩1
⟨1⟩10. QED
  ⟨2⟩1. (b = c) ∨ (b > c) ∨ (c > b)
    BY SimpleArithmetic DEF Ballot
  ⟨2⟩2. QED
    BY ⟨1⟩7, ⟨1⟩8, ⟨1⟩9, ⟨2⟩1
```

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). Since the proof of theorem $VT1$ shows that $VT1$ is true in any state, formula $VT1'$ is obviously true for any transition. However, proving this requires a kind of reasoning that distinguishes between inference and implication. If the difference between inference and implication means nothing to you, it is because that difference does not arise in ordinary logic; it becomes important only in modal logics. (Temporal logic is one example of modal logic.) Because *TLAPS* does not yet handle any modal-logic reasoning, it is yet able to deduce $VT1'$ from $VT1$. This ability will be added when *TLAPS* is enhanced to do temporal-logic reasoning. For now, we have write a separate theorem, whose proof is obtained from that of $VT1$ by priming everything. We also need the primed versions of *SafeAtProp* and $VT0$, which are used in its proof.

The proof of the primed version of a theorem is obtained by simply priming all the steps in the proof of the original theorem (replacing references to lemmas by references to their primed versions). Since we did not prove *SafeAtProp*, we cannot prove its primed version either.

```
THEOREM SafeAtPropPrime ≜
  ∀ 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)′
PROOF OMITTED

LEMMA VT0Prime ≜
              ∧ TypeOK′
              ∧ VInv1′
              ∧ VInv2′
              ⇒ ∀ v, w ∈ Value, b, c ∈ Ballot :
```

$(b > c) \land SafeAt(b,\, v)' \land ChosenIn(c,\, w)' \Rightarrow (v = w)$

⟨1⟩ SUFFICES ASSUME $TypeOK'$, $VInv1'$, $VInv2'$,
NEW $v \in Value$, NEW $w \in Value$
PROVE $\forall\, b,\, c \in Ballot :$
$(b > c) \land SafeAt(b,\, v)' \land ChosenIn(c,\, w)' \Rightarrow (v = w)$
OBVIOUS

⟨1⟩ $P \triangleq [b \in Ballot \mapsto$
$\forall\, c \in Ballot :$
$(b > c) \land SafeAt(b,\, v)' \land ChosenIn(c,\, w)' \Rightarrow (v = w)]$

⟨1⟩1. $P[0]$
⟨2⟩1. $\land\, 0 \in Ballot$
$\land\, \forall\, c \in Ballot : \neg(0 > c)$
BY *SimpleArithmetic* DEF *Ballot*
⟨2⟩2. QED
BY ⟨2⟩1
⟨1⟩2. ASSUME NEW $b \in Ballot$, $\forall\, i \in 0 \,..\, b : P[i]$
PROVE $P[b+1]$
⟨2⟩1. $b + 1 \in Ballot$
BY *SimpleArithmetic* DEF *Ballot*
⟨2⟩2. SUFFICES ASSUME NEW $c \in Ballot$, $b + 1 > c$, $SafeAt(b+1,\, v)'$, $ChosenIn(c,\, w)'$
PROVE $v = w$
BY ⟨2⟩1
⟨2⟩3. PICK $Q \in Quorum : \forall\, a \in Q : VotedFor(a,\, c,\, w)'$
BY ⟨2⟩2 DEF *ChosenIn*
⟨2⟩4. $b + 1 \neq 0 \land ((b+1) - 1 = b)$
BY *SimpleArithmetic* DEF *Ballot*
⟨2⟩5. PICK $QQ \in Quorum$,
$d \in -1 \,..\, ((b+1) - 1) :$
$\land\, (d \neq -1) \Rightarrow \land\, SafeAt(d,\, v)'$
$\land\, \forall\, a \in QQ :$
$\forall\, x \in Value :$
$VotedFor(a,\, d,\, x)' \Rightarrow (x = v)$
$\land\, \forall\, e \in (d+1) \,..\, ((b+1) - 1),\, a \in QQ : DidNotVoteIn(a,\, e)'$
BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩4, *SafeAtPropPrime*
⟨2⟩ PICK $aa \in QQ \cap Q$: TRUE
BY QA
⟨2⟩6. $c \leq d$
⟨3⟩1. SUFFICES ASSUME $\neg(c \leq d)$
PROVE FALSE
OBVIOUS
⟨3⟩2. $c \in (d+1) \,..\, ((b+1) - 1)$
BY ⟨2⟩2, ⟨3⟩1, *SimpleArithmetic* DEF *Ballot*
⟨3⟩3. $VotedFor(aa,\, c,\, w)'$
BY ⟨2⟩3

```
    ⟨3⟩4. DidNotVoteIn(aa, c)′
      BY ⟨2⟩5, ⟨3⟩1, ⟨3⟩2
    ⟨3⟩5. QED
      BY ⟨3⟩3, ⟨3⟩4 DEF DidNotVoteIn
  ⟨2⟩7. d ≠ −1
    BY ⟨2⟩6, SimpleArithmetic DEF Ballot
  ⟨2⟩8.CASE c = d
    BY ⟨2⟩3, ⟨2⟩5, ⟨2⟩7, ⟨2⟩8
  ⟨2⟩9.CASE d > c
    ⟨3⟩1. SafeAt(d, v)′
      BY ⟨2⟩5, ⟨2⟩7
    ⟨3⟩2. d ∈ Ballot ∧ d ∈ 0 .. b
      BY ⟨2⟩6, SimpleArithmetic DEF Ballot
    ⟨3⟩3. P[d]
      BY ⟨1⟩2, ⟨3⟩2
    ⟨3⟩4. QED
      BY ⟨2⟩2, ⟨2⟩9, ⟨3⟩1, ⟨3⟩2, ⟨3⟩3
  ⟨2⟩10. QED
    ⟨3⟩1. (c = d) ∨ (d > c)
      BY ⟨2⟩6, SimpleArithmetic DEF Ballot
    ⟨3⟩2. QED
      BY ⟨2⟩8, ⟨2⟩9, ⟨3⟩1
  PICK QQ ∈ Quorum : VotedFor(ac, c, w)
  BY QuorumNonEmpty, QA DEF ChosenIn
⟨1⟩3. ∀ b ∈ Ballot : P[b]
  BY ⟨1⟩1, ⟨1⟩2, GeneralNatInduction DEF Ballot

⟨1⟩4. QED
  BY ⟨1⟩3

THEOREM VT1Prime ≜
                  ∧ TypeOK′
                  ∧ VInv1′
                  ∧ VInv2′
                  ⇒ ∀ v, w :
                        (v ∈ chosen′) ∧ (w ∈ chosen′) ⇒ (v = w)
⟨1⟩1. SUFFICES ASSUME TypeOK′, VInv1′, VInv2′,
                      NEW v, NEW w,
                      v ∈ chosen′, w ∈ chosen′
               PROVE  v = w
  OBVIOUS
⟨1⟩2. v ∈ Value ∧ w ∈ Value
  BY ⟨1⟩1 DEF chosen
⟨1⟩3. PICK b ∈ Ballot, c ∈ Ballot : ChosenIn(b, v)′ ∧ ChosenIn(c, w)′
  BY ⟨1⟩1 DEF chosen
```

⟨1⟩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 ⟨1⟩3 DEF *ChosenIn*
⟨1⟩5. PICK $av \in Q$, $aw \in R$: $\land\ VotedFor(av,\, b,\, v)'$
$\land\ VotedFor(aw,\, c,\, w)'$
BY ⟨1⟩4, *QuorumNonEmpty*
⟨1⟩6. $SafeAt(b,\, v)' \land SafeAt(c,\, w)'$
BY ⟨1⟩1, ⟨1⟩2, ⟨1⟩5, *QA* DEF *VInv*2
⟨1⟩7.CASE $b = c$
⟨2⟩ PICK $a \in Q \cap R$: TRUE
BY *QA*
⟨2⟩1. $\land\ VotedFor(a,\, b,\, v)'$
$\land\ VotedFor(a,\, c,\, w)'$
BY ⟨1⟩4
⟨2⟩2. QED
BY ⟨1⟩1, ⟨1⟩2, ⟨1⟩7, ⟨2⟩1, *QA* DEF *VInv*1
⟨1⟩8.CASE $b > c$
BY ⟨1⟩1, ⟨1⟩6, ⟨1⟩3, ⟨1⟩8, *VT0Prime*, ⟨1⟩2 ⟨2⟩1
⟨1⟩9.CASE $c > b$
BY ⟨1⟩1, ⟨1⟩6, ⟨1⟩3, ⟨1⟩9, *VT0Prime*, ⟨1⟩2 ⟨2⟩1
⟨1⟩10. QED
⟨2⟩1. $(b = c) \lor (b > c) \lor (c > b)$
BY *SimpleArithmetic* DEF *Ballot*
⟨2⟩2. QED
BY ⟨1⟩7, ⟨1⟩8, ⟨1⟩9, ⟨2⟩1

The invariance of *VInv*2 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.

$VInv4 \triangleq \forall\, a \in Acceptor,\, b \in Ballot :$
$maxBal[a] < b \Rightarrow DidNotVoteIn(a,\, b)$

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.

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

$VoteFor(self,\, b,\, v) \triangleq$
$\land\ maxBal[self] \leq b$

```
    ∧ DidNotVoteIn(self, b)
    ∧ ∀ p ∈ Acceptor \ {self} :
        ∀ w ∈ Value : VotedFor(p, b, w) ⇒ (w = v)
    ∧ SafeAt(b, v)
    ∧ votes' = [votes EXCEPT ![self] = votes[self] ∪ {⟨b, v⟩}]
    ∧ maxBal' = [maxBal EXCEPT ![self] = b]

BallotAction(self, b) ≜
    ∨ IncreaseMaxBal(self, b)
    ∨ ∃ v ∈ 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 ≜ Acceptor ≠ {}
```

The proof of the lemma itself is quite simple.

```
LEMMA NextDef ≜
  TypeOK ⇒
   (Next = ∃ self ∈ Acceptor :
                ∃ b ∈ Ballot   : BallotAction(self, b))
⟨1⟩ HAVE TypeOK
⟨1⟩2. Next = ∃ self ∈ Acceptor : acceptor(self)
 BY AcceptorNonempty DEF Next, ProcSet
⟨1⟩3. @ = NextDef!2!2
  BY  DEF Next, BallotAction, IncreaseMaxBal, VoteFor, ProcSet, acceptor
⟨1⟩4. QED
  BY ⟨1⟩2, ⟨1⟩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 ≜ VInv ∧ [Next]_vars ⇒ VInv'
⟨1⟩1. VInv ∧ (vars' = vars) ⇒ VInv'
  ⟨2⟩ SUFFICES ASSUME VInv, vars' = vars
               PROVE  VInv'
    OBVIOUS
  ⟨2⟩ USE  DEF vars, VInv
  ⟨2⟩1. TypeOK'
    BY  DEF TypeOK
  ⟨2⟩2. VInv2'
    BY  DEF VInv2, VotedFor, SafeAt, DidNotVoteIn
  ⟨2⟩3. VInv3'
    BY  DEF VInv3, VotedFor
```

1003 ⟨2⟩4. $VInv4'$
1004 BY DEF $VInv4$, $DidNotVoteIn$, $VotedFor$
1005 ⟨2⟩5. QED
1006 BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩3, ⟨2⟩4

1008 ⟨1⟩ SUFFICES ASSUME $VInv$,
1009 NEW $self \in Acceptor$,
1010 NEW $b \in Ballot$,
1011 $BallotAction(self,\ b)$
1012 PROVE $VInv'$
1013 BY ⟨1⟩1, $NextDef$ DEF $VInv$

1015 ⟨1⟩2. $TypeOK'$
1016 ⟨2⟩1.CASE $IncreaseMaxBal(self,\ b)$
1017 BY ⟨2⟩1 DEF $IncreaseMaxBal$, $VInv$, $TypeOK$
1018 ⟨2⟩2.CASE $\exists\, v \in Value : VoteFor(self,\ b,\ v)$
1019 BY ⟨2⟩2 DEF $VInv$, $TypeOK$, $VoteFor$
1020 ⟨2⟩3. QED
1021 BY ⟨2⟩1, ⟨2⟩2 DEF $BallotAction$

1023 ⟨1⟩3. ASSUME NEW $a \in Acceptor$, NEW $c \in Ballot$, NEW $w \in Value$,
1024 $VotedFor(a,\ c,\ w)$
1025 PROVE $VotedFor(a,\ c,\ w)'$
1026 ⟨2⟩1.CASE $IncreaseMaxBal(self,\ b)$
1027 BY ⟨2⟩1, ⟨1⟩3 DEF $IncreaseMaxBal$, $VotedFor$
1028 ⟨2⟩2.CASE $\exists\, v \in Value : VoteFor(self,\ b,\ v)$
1029 ⟨3⟩1. PICK $v \in Value : VoteFor(self,\ b,\ v)$
1030 BY ⟨2⟩2
1031 ⟨3⟩2.CASE $a = self$
1032 ⟨4⟩1. $votes'[a] = votes[a] \cup \{\langle b,\ v\rangle\}$
1033 BY ⟨3⟩1, ⟨3⟩2 DEF $VoteFor$, $VInv$, $TypeOK$
1034 ⟨4⟩2. QED
1035 BY ⟨1⟩3, ⟨4⟩1 DEF $VotedFor$
1036 ⟨3⟩3.CASE $a \neq self$
1037 ⟨4⟩1. $votes[a] = votes'[a]$
1038 BY ⟨3⟩1, ⟨3⟩3 DEF $VoteFor$, $VInv$, $TypeOK$
1039 ⟨4⟩2. QED
1040 BY ⟨1⟩3, ⟨4⟩1 DEF $VotedFor$
1041 ⟨3⟩4. QED
1042 BY ⟨3⟩2, ⟨3⟩3 DEF $VoteFor$
1043 ⟨2⟩3. QED
1044 BY ⟨2⟩1, ⟨2⟩2 DEF $BallotAction$

1046 ⟨1⟩4. ASSUME NEW $a \in Acceptor$, NEW $c \in Ballot$, NEW $w \in Value$,
1047 $\neg VotedFor(a,\ c,\ w),\ VotedFor(a,\ c,\ w)'$
1048 PROVE $(a = self) \land (c = b) \land VoteFor(self,\ b,\ w)$

⟨2⟩1.CASE $IncreaseMaxBal(self,\ b)$
BY ⟨2⟩1, ⟨1⟩4 DEF $IncreaseMaxBal,\ VInv,\ TypeOK,\ VotedFor$
⟨2⟩2.CASE $\exists\, v \in Value : VoteFor(self,\ b,\ v)$
⟨3⟩1. PICK $v \in Value : VoteFor(self,\ b,\ v)$
BY ⟨2⟩2
⟨3⟩2. $a = self$
⟨4⟩ SUFFICES ASSUME $a \neq self$
PROVE FALSE
OBVIOUS
⟨4⟩1. $votes'[a] = votes[a]$
BY ⟨3⟩1 DEF $VoteFor,\ VInv,\ TypeOK$
⟨4⟩2. QED
BY ⟨4⟩1, ⟨1⟩4 DEF $VotedFor$
⟨3⟩3. $votes'[a] = votes[a] \cup \{\langle b,\ v\rangle\}$
BY ⟨3⟩1, ⟨3⟩2 DEF $VoteFor,\ VInv,\ TypeOK$
⟨3⟩4. $c = b \land v = w$
BY ⟨1⟩4, ⟨3⟩3 DEF $VotedFor$
⟨3⟩5. QED
BY ⟨3⟩1, ⟨3⟩2, ⟨3⟩4
⟨2⟩3. QED
BY ⟨2⟩1, ⟨2⟩2 DEF $BallotAction$

⟨1⟩5. ASSUME NEW $a \in Acceptor$
PROVE $\land\, maxBal[a] \in Ballot \cup \{-1\}$
$\land\, maxBal'[a] \in Ballot \cup \{-1\}$
$\land\, maxBal'[a] \geq maxBal[a]$
⟨2⟩1. $\land\, maxBal[a] \in Ballot \cup \{-1\}$
$\land\, maxBal'[a] \in Ballot \cup \{-1\}$
BY ⟨1⟩2 DEF $VInv,\ TypeOK$
⟨2⟩2. $\land\, (a = self) \Rightarrow \land\, maxBal'[a] = b$
$\land\ \lor\, b > maxBal[a]$
$\lor\, maxBal[a] \leq b$
$\land\, (a \neq self) \Rightarrow (maxBal'[a] = maxBal[a])$
⟨3⟩1.CASE $IncreaseMaxBal(self,\ b)$
BY ⟨3⟩1 DEF $IncreaseMaxBal,\ VInv,\ TypeOK$
⟨3⟩2.CASE $\exists\, v \in Value : VoteFor(self,\ b,\ v)$
BY ⟨3⟩2 DEF $VoteFor,\ VInv,\ TypeOK$
⟨3⟩3. QED
BY ⟨3⟩1, ⟨3⟩2 DEF $BallotAction$
⟨2⟩3. $\forall\, mb \in Ballot \cup \{-1\} : \land\, (b\ \ > mb) \Rightarrow (b \geq mb)$
$\land\, mb \geq mb$
BY $SimpleArithmetic$ DEF $Ballot$
⟨2⟩4. QED
BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩3

1095 $\langle 1\rangle 6$. ASSUME NEW $c \in \mathit{Ballot}$, NEW $w \in \mathit{Value}$,
1096 $\mathit{SafeAt}(c,\, w)$
1097 PROVE $\mathit{SafeAt}(c,\, w)'$
1098 $\langle 2\rangle$ DEFINE $P[i \in \mathit{Ballot}] \triangleq \forall\, j \in 0\,..\,i : \mathit{SafeAt}(j,\, w) \Rightarrow \mathit{SafeAt}(j,\, w)'$
1099 $\langle 2\rangle 1$. $P[0]$
1100 $\langle 3\rangle 1$. $0 \in \mathit{Ballot} \land \forall\, i \in 0\,..\,0 : i = 0$
1101 BY *SimpleArithmetic* DEF *Ballot*
1102 $\langle 3\rangle 2$. QED
1103 BY *SafeAtPropPrime*, $\langle 3\rangle 1$
1104 $\langle 2\rangle 2$. ASSUME NEW $d \in \mathit{Ballot}$, $P[d]$
1105 PROVE $P[d+1]$
1106 $\langle 3\rangle 1$. $d+1 \in \mathit{Ballot} \land d+1 \neq 0$
1107 BY *SimpleArithmetic* DEF *Ballot*
1108 $\langle 3\rangle 2$. SUFFICES ASSUME NEW $e \in 0\,..\,(d+1)$, $\mathit{SafeAt}(e,\, w)$
1109 PROVE $\mathit{SafeAt}(e,\, w)'$
1110 BY $\langle 3\rangle 1$
1111 $\langle 3\rangle 3$. $e \in 0\,..\,d \lor e = d+1$
1112 BY *SimpleArithmetic* DEF *Ballot*
1113 $\langle 3\rangle 4$.CASE $e \in 0\,..\,d$
1114 BY $\langle 2\rangle 2$, $\langle 3\rangle 2$, $\langle 3\rangle 4$
1115 $\langle 3\rangle 5$.CASE $e = d+1$
1116 $\langle 4\rangle 1$. PICK $Q \in \mathit{Quorum} : \mathit{SafeAtProp}!(e,\, w)!2!2!(Q)$
1117 BY $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 5$, *SafeAtProp*
1118 $\langle 4\rangle 2$. $\forall\, aa \in Q : \mathit{maxBal}'[aa] \geq e$
1119 $\langle 5\rangle 1$. SUFFICES ASSUME NEW $aa \in Q$
1120 PROVE $\mathit{maxBal}'[aa] \geq e$
1121 OBVIOUS
1122 $\langle 5\rangle 2$. $\forall\, \mathit{mbp},\, \mathit{mb} \in \mathit{Ballot} \cup \{-1\} :$
1123 $\mathit{mbp} \geq \mathit{mb} \land \mathit{mb} \geq e \Rightarrow \mathit{mbp} \geq e$
1124 BY *SimpleArithmetic* DEF *Ballot*
1125 $\langle 5\rangle 3$. QED
1126 BY $\langle 5\rangle 2$, $\langle 1\rangle 5$, $\langle 4\rangle 1$, *QA*
1127 $\langle 4\rangle 3$. $\exists\, cc \in -1\,..\,(e-1) :$
1128 $\land\, (cc \neq -1) \Rightarrow \land\, \mathit{SafeAt}(cc,\, w)'$
1129 $\land\, \forall\, ax \in Q :$
1130 $\forall\, z \in \mathit{Value} :$
1131 $\mathit{VotedFor}(ax,\, cc,\, z)' \Rightarrow (z = w)$
1132 $\land\, \forall\, dd \in (cc+1)\,..\,(e-1),\, ax \in Q : \mathit{DidNotVoteIn}(ax,\, dd)'$
1133 $\langle 5\rangle 1$. ASSUME NEW $cc \in 0\,..\,(e-1)$,
1134 NEW $ax \in Q$, NEW $z \in \mathit{Value}$,
1135 $\mathit{VotedFor}(ax,\, cc,\, z)'$, $\neg\, \mathit{VotedFor}(ax,\, cc,\, z)$
1136 PROVE FALSE
1137 $\langle 6\rangle 1$. $cc \in \mathit{Ballot}$
1138 BY $\langle 5\rangle 1$, $\langle 3\rangle 5$, *SimpleArithmetic* DEF *Ballot*
1139 $\langle 6\rangle 2$. $(ax = \mathit{self}) \land (cc = b) \land \mathit{VoteFor}(\mathit{self},\, b,\, z)$

BY $\langle 5\rangle 1$, $\langle 6\rangle 1$, $\langle 1\rangle 4$, QA

$\langle 6\rangle 3$. $maxBal[ax] \geq e$

BY $\langle 4\rangle 1$

$\langle 6\rangle 4$. $maxBal[self] \leq b$

BY $\langle 6\rangle 2$ DEF $VoteFor$

$\langle 6\rangle 5$. $\forall\, mb \in Ballot \cup \{-1\} : \neg(mb \geq e \wedge mb \leq b)$

BY $\langle 3\rangle 5$, $\langle 6\rangle 2$, $SimpleArithmetic$ DEF $Ballot$

$\langle 6\rangle 6$. QED

BY $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $\langle 6\rangle 5$ DEF $VInv$, $TypeOK$

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

BY $\langle 4\rangle 1$

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

$\langle 6\rangle 1$. SUFFICES ASSUME $cc \neq -1$
PROVE $\wedge\, SafeAt(cc,\, w)'$
$\wedge\, \forall\, ax \in Q :$
$\forall\, z \in Value :$
$VotedFor(ax,\, cc,\, z)' \Rightarrow (z = w)$

OBVIOUS

$\langle 6\rangle 2$. $\wedge\, SafeAt(cc,\, w)$
$\wedge\, \forall\, ax \in Q :$
$\forall\, z \in Value : VotedFor(ax,\, cc,\, z) \Rightarrow (z = w)$

BY $\langle 5\rangle 2$, $\langle 6\rangle 1$

$\langle 6\rangle 3$. $SafeAt(cc,\, w)'$

$\langle 7\rangle 1$. $cc \in 0 \,..\, d \wedge cc \in Ballot$

BY $\langle 6\rangle 1$, $\langle 3\rangle 5$, $SimpleArithmetic$ DEF $Ballot$

$\langle 7\rangle 2$. QED

BY $\langle 2\rangle 2$, $\langle 6\rangle 2$, $\langle 7\rangle 1$

$\langle 6\rangle 4$. 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 $\langle 6\rangle 2$, $\langle 7\rangle 1$

$\langle 7\rangle 2$.CASE $\neg\, VotedFor(ax,\, cc,\, z)$

$\langle 8\rangle 1$. $cc \in 0 \,..\, (e-1)$

BY $\langle 6\rangle 1$, $SimpleArithmetic$ DEF $Ballot$

$\langle 8\rangle 2$ QED

BY $\langle 8\rangle 1$, $\langle 7\rangle 2$, $\langle 6\rangle 4$, $\langle 5\rangle 1$

$\langle 7\rangle 3$. QED

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

$\langle 6\rangle 5$. QED

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

$\langle 5\rangle 4$. $\forall\, dd \in (cc+1) \,..\, (e-1),\, ax \in Q : DidNotVoteIn(ax,\, dd)'$

$\langle 6\rangle 1$. SUFFICES ASSUME NEW $dd \in (cc+1) \,..\, (e-1)$, NEW $ax \in Q$,

$\neg DidNotVoteIn(ax,\ dd)'$

PROVE FALSE

OBVIOUS

$\langle 6\rangle 2.\ DidNotVoteIn(ax,\ dd)$

BY $\langle 5\rangle 2$

$\langle 6\rangle 3.$ PICK $v \in Value : VotedFor(ax,\ dd,\ v)'$

BY $\langle 6\rangle 1$ DEF $DidNotVoteIn$

$\langle 6\rangle 4.\ dd \in 0\,..\,(e-1)$

BY $SimpleArithmetic$ DEF $Ballot$

$\langle 6\rangle 5.$ QED

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

$\langle 5\rangle 5.$ QED

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

$\langle 4\rangle 4.\ \lor\ e = 0$

$\lor\ \exists\, Q\_1 \in Quorum :$

$\land\ \forall\, aa \in Q\_1 : maxBal'[aa] \geq e$

$\land\ \exists\, c\_1 \in -1\,..\,e-1 :$

$\land\ c\_1 \neq -1$

$\Rightarrow\ (\ \land\ SafeAt(c\_1,\ w)'$

$\land\ \forall\, aa \in Q\_1 :$

$\forall\, w\_1 \in Value :$

$VotedFor(aa,\ c\_1,\ w\_1)' \Rightarrow w\_1 = w)$

$\land\ \forall\, d\_1 \in c\_1 + 1\,..\,e-1,\ aa \in Q\_1 :$

$DidNotVoteIn(aa,\ d\_1)'$

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

$\langle 4\rangle 5.\ e \in Ballot$

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

$\langle 4\rangle 6.\ SafeAt(e,\ w)' = \langle 4\rangle 4$

BY $SafeAtPropPrime, \langle 4\rangle 5$

$\langle 4\rangle 7.$ QED

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

$\langle 3\rangle 6.$ QED

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

$\langle 2\rangle 3.\ \forall\, d \in Ballot : P[d]$

BY $\langle 2\rangle 1, \langle 2\rangle 2, SimpleNatInduction$ DEF $Ballot$

$\langle 2\rangle 4.$ QED

$\langle 3\rangle 1.\ c \in 0\,..\,c$

BY $SimpleArithmetic$ DEF $Ballot$

$\langle 3\rangle 2.$ QED

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

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

$\langle 2\rangle 1.$ 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 $\neg VotedFor(a,\ c,\ v)$

$\langle 3\rangle 1.\ (a = self) \land (c = b) \land VoteFor(self,\ b,\ v)$

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

$\langle 3\rangle 2.\ SafeAt(c,\ v)$

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

$\langle 3\rangle 3$. QED

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

$\langle 2\rangle 4$. 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 $v1 = v2$

$\langle 3\rangle 1.\ (a1 = self) \land (c = b) \land VoteFor(self,\ b,\ v1)$

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

$\langle 3\rangle 2$.CASE $a2 = self$

$\langle 4\rangle 1.\ \neg VotedFor(self,\ b,\ v2)$

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

$\langle 4\rangle 2.\ VoteFor(self,\ b,\ v2)$

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

$\langle 4\rangle 3.\ votes'[self] = votes[self] \cup \{\langle b,\ v1\rangle\}$

BY $\langle 3\rangle 1$ DEF $VoteFor$, $VInv$, $TypeOK$

$\langle 4\rangle 4.\ votes'[self] = votes[self] \cup \{\langle b,\ v2\rangle\}$

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

$\langle 4\rangle 5.\ \langle b,\ v1\rangle \notin votes[self]$

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

$\langle 4\rangle 6$. QED

$\langle 5\rangle 1.\ \langle b,\ v1\rangle \in votes[self] \cup \{\langle b,\ v2\rangle\}$

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

$\langle 5\rangle 2$. QED

BY $\langle 5\rangle 1$, $\langle 4\rangle 5$

$\langle 3\rangle 3$.CASE $a2 \neq self$

$\langle 4\rangle 1$. $votes'[a2] = votes[a2]$

BY $\langle 3\rangle 1$, $\langle 3\rangle 3$ DEF $VoteFor$, $VInv$, $TypeOK$

$\langle 4\rangle 2$. $VotedFor(a2, b, v2)$

BY $\langle 2\rangle 2$, $\langle 3\rangle 1$, $\langle 4\rangle 1$ DEF $VotedFor$

$\langle 4\rangle 3$. QED

BY $\langle 3\rangle 1$, $\langle 3\rangle 3$, $\langle 4\rangle 2$ DEF $VoteFor$ *SMT* fails

$\langle 3\rangle 4$. 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 $VInv3$

$\langle 1\rangle 9$. $VInv4'$

$\langle 2\rangle 1$. SUFFICES ASSUME NEW $a \in Acceptor$, NEW $c \in Ballot$,
$maxBal'[a] < c$,
$\neg DidNotVoteIn(a, c)'$
PROVE FALSE

BY DEF $VInv4$

$\langle 2\rangle 2$. $maxBal[a] < c$

$\langle 3\rangle 1$. $\forall\, x,\ xp \in Ballot \cup \{-1\} : xp < c \land xp \geq x \Rightarrow x < c$

BY *SimpleArithmetic* DEF $Ballot$ *SMT* fails

$\langle 3\rangle 2$. QED

BY $\langle 1\rangle 5$, $\langle 2\rangle 1$, $\langle 3\rangle 1$ *SMT* fails

$\langle 2\rangle 3$. $DidNotVoteIn(a, c)$

BY $\langle 2\rangle 2$ DEF $VInv$, $VInv4$

$\langle 2\rangle 4$. PICK $v \in Value : VotedFor(a, c, v)'$

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

$\langle 2\rangle 5$. $(a = self) \land (c = b) \land VoteFor(self, b, v)$

BY $\langle 1\rangle 4$, $\langle 2\rangle 1$, $\langle 2\rangle 3$, $\langle 2\rangle 4$ DEF $DidNotVoteIn$

$\langle 2\rangle 6$. $maxBal'[a] = c$

BY $\langle 2\rangle 5$ DEF $VoteFor$, $VInv$, $TypeOK$

$\langle 2\rangle 7$. QED

$\langle 3\rangle 1$. $\neg(c < c)$

BY *SimpleArithmetic* DEF $Ballot$ *SMT* fails

$\langle 3\rangle 2$. QED

BY $\langle 2\rangle 1$, $\langle 2\rangle 6$, $\langle 3\rangle 1$ *SMT* fails

$\langle 1\rangle 10$. 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 $InitImpliesInv \triangleq Init \Rightarrow VInv$

$\langle 1\rangle$ SUFFICES ASSUME $Init$PROVE $VInv$

OBVIOUS

⟨1⟩ USE DEF *Init*
⟨1⟩1. *TypeOK*
BY DEF *TypeOK*, *ProcSet*
⟨1⟩2. *VInv*2
BY DEF *VInv*2, *VotedFor*
⟨1⟩3. *VInv*3
BY DEF *VInv*3, *VotedFor*
⟨1⟩4. *VInv*4
BY DEF *VInv*4, *DidNotVoteIn*, *VotedFor*
⟨1⟩5. QED
BY ⟨1⟩1, ⟨1⟩2, ⟨1⟩3, ⟨1⟩4 DEF *VInv*

The following theorem asserts that *VInv* is an invariant of *Spec*. Because *TLAPS* does not yet do any temporal reasoning, we have to omit the proof of all steps that assert temporal logic formulas. Both the steps of this trivial proof are therefore omitted. However, for all such omitted proofs, we give the proof that we expect to be approximately a proof that *TLAPS* will accept once it does handle temporal logic reasoning.

THEOREM $VT2 \triangleq Spec \Rightarrow \Box VInv$
⟨1⟩1. $VInv \wedge \Box[Next]_{vars} \Rightarrow \Box VInv$
BY *InductiveInvariance*, *RuleINV*1
PROOF OMITTED
⟨1⟩2. QED
BY *InitImpliesInv*, ⟨1⟩1
PROOF OMITTED

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 $VT3 \triangleq Spec \Rightarrow C!Spec$
⟨1⟩1. $Init \Rightarrow C!Init$
⟨2⟩ SUFFICES ASSUME *Init*
PROVE $C!Init$
OBVIOUS
⟨2⟩1. SUFFICES ASSUME $chosen \neq \{\}$
PROVE FALSE
BY DEF $C!Init$

```
⟨2⟩2. PICK v ∈ chosen : TRUE
  BY ⟨2⟩1
⟨2⟩3. PICK b ∈ Ballot : ChosenIn(b, v)
  BY ⟨2⟩2 DEF chosen
⟨2⟩4. PICK Q ∈ Quorum : ∀ a ∈ Q : VotedFor(a, b, v)
  BY ⟨2⟩3 DEF ChosenIn
⟨2⟩5. PICK a ∈ Q : ⟨b, v⟩ ∈ votes[a]  VotedFor(a, b, v)
  BY QuorumNonEmpty, ⟨2⟩4 DEF VotedFor
⟨2⟩6. QED
  BY ⟨2⟩5, QA DEF Init

⟨1⟩2. ASSUME VInv, VInv', [Next]_vars
      PROVE  [C!Next]_C!vars
 ⟨2⟩ USE VInv
 ⟨2⟩1.CASE vars' = vars
  BY ⟨2⟩1 DEF vars, C!vars, chosen, ChosenIn, VotedFor
 ⟨2⟩2. SUFFICES ASSUME NEW self ∈ Acceptor,
                        NEW b ∈ Ballot,
                        BallotAction(self, b)
                PROVE  [C!Next]_C!vars
  BY ⟨1⟩2, ⟨2⟩1, NextDef DEF VInv
 ⟨2⟩3. ASSUME IncreaseMaxBal(self, b)
       PROVE  C!vars' = C!vars
  BY ⟨2⟩3 DEF IncreaseMaxBal, C!vars, chosen, ChosenIn, VotedFor
 ⟨2⟩4. ASSUME NEW v ∈ Value,
              VoteFor(self, b, v)
       PROVE  [C!Next]_C!vars
   ⟨3⟩3. ASSUME NEW w ∈ chosen
         PROVE  w ∈ chosen'
    ⟨4⟩1. PICK c  ∈ Ballot : ChosenIn(c, w)
      BY ⟨3⟩3 DEF chosen
    ⟨4⟩2. PICK Q ∈ Quorum : ∀ a ∈ Q : ⟨c, w⟩ ∈ votes[a]
      BY ⟨4⟩1 DEF ChosenIn, VotedFor
    ⟨4⟩3. SUFFICES ASSUME NEW a ∈ Q
                   PROVE  ⟨c, w⟩ ∈ votes'[a]
      BY  DEF chosen, ChosenIn, VotedFor
    ⟨4⟩4.CASE a = self
      ⟨6⟩1. votes'[self] = votes[self] ∪ {⟨b, v⟩}
        BY ⟨2⟩4, ⟨1⟩2 DEF VoteFor, VInv, TypeOK
      ⟨6⟩2. QED
        BY ⟨4⟩2, ⟨4⟩4, ⟨6⟩1
    ⟨4⟩5.CASE a ≠ self
        BY ⟨2⟩4, ⟨1⟩2, ⟨4⟩2, ⟨4⟩5, QA DEF VoteFor, VInv, TypeOK
    ⟨4⟩6. QED
      BY ⟨4⟩4, ⟨4⟩5
```

1429 ⟨3⟩1. ASSUME NEW $w \in chosen$,
1430 $v \in chosen'$
1431 PROVE $w = v$
1432 ⟨4⟩1. $w \in chosen'$
1433 BY ⟨3⟩3
1434 ⟨4⟩2. $VInv1'$
1435 BY ⟨1⟩2 DEF $VInv$, $VInv1$, $VInv3$
1436 ⟨4⟩3. QED
1437 BY ⟨1⟩2, ⟨3⟩1, ⟨4⟩1, ⟨4⟩2, $VT1Prime$ DEF $VInv$
1438 ⟨3⟩2. ASSUME NEW w, $w \notin chosen$, $w \in chosen'$
1439 PROVE $w = v$
1440 ⟨4⟩1. PICK $c \in Ballot : ChosenIn(c, w)'$
1441 BY ⟨3⟩2 DEF $chosen$
1442 ⟨4⟩2. PICK $Q \in Quorum : \forall\, a \in Q : \langle c, w\rangle \in votes'[a]$
1443 BY ⟨4⟩1 DEF $ChosenIn$, $VotedFor$
1444 ⟨4⟩3. PICK $a \in Q : \langle c, w\rangle \notin votes[a]$
1445 BY ⟨3⟩2 DEF $chosen$, $ChosenIn$, $VotedFor$
1446 ⟨4⟩4.CASE $a = self$
1447 ⟨5⟩1. $votes'[self] = votes[self] \cup \{\langle b, v\rangle\}$
1448 BY ⟨2⟩4, ⟨1⟩2 DEF $VoteFor$, $VInv$, $TypeOK$
1449 ⟨5⟩2. QED
1450 BY ⟨4⟩4, ⟨5⟩1, ⟨4⟩2, ⟨4⟩3
1451 ⟨4⟩5.CASE $a \neq self$
1452 BY ⟨2⟩4, ⟨1⟩2, ⟨4⟩2, ⟨4⟩3, ⟨4⟩5, QA DEF $VoteFor$, $VInv$, $TypeOK$
1453 ⟨4⟩6. QED
1454 BY ⟨4⟩4, ⟨4⟩5
1455 ⟨3⟩4.CASE $chosen' = chosen$
1456 BY ⟨3⟩4 DEF $C!vars$
1457 ⟨3⟩5.CASE $chosen' \neq chosen$
1458 ⟨4⟩1. $chosen \subseteq chosen'$
1459 BY ⟨3⟩3
1460 ⟨4⟩2. PICK $w \in chosen' : w \notin chosen$
1461 BY ⟨3⟩5, ⟨4⟩1
1462 ⟨4⟩3. $w = v$
1463 BY ⟨4⟩2, ⟨3⟩2
1464 ⟨4⟩4. $chosen' = chosen \cup \{v\}$
1465 BY ⟨3⟩2, ⟨4⟩1, ⟨4⟩3
1466 ⟨4⟩5. $chosen = \{\}$
1467 BY ⟨4⟩4, ⟨3⟩1, ⟨3⟩5
1468 ⟨4⟩6. QED
1469 BY ⟨4⟩4, ⟨4⟩5 DEF $C!Next$
1470 ⟨3⟩6. QED
1471 BY ⟨3⟩4, ⟨3⟩5
1472 ⟨2⟩5. QED
1473 BY ⟨2⟩2, ⟨2⟩3, ⟨2⟩4 DEF $BallotAction$

1474 $\langle 1\rangle 3$. QED
1475 $\langle 2\rangle 1. \Box\, VInv \wedge \Box[Next]_{vars} \Rightarrow \Box[C!Next]_{C}!vars$
1476 BY $\langle 1\rangle 2$, *RuleTLA2*
1477 PROOF OMITTED
1478 $\langle 2\rangle 2$. QED
1479 BY $\langle 1\rangle 1$, $\langle 2\rangle 1$, $VT2$ DEF *Spec*, $C!Spec$
1480 PROOF OMITTED
1481 ├──┤

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.

1498 ASSUME $AcceptorFinite \triangleq IsFiniteSet(Acceptor)$

1500 ASSUME $ValueNonempty \triangleq Value \neq \{\}$

1501 ├──┤

We need the following simple results about sets and sets of numbers. The first belongs in a library of theorems about finite sets and cardinality. Perhaps such a library will eventually be added to the *FiniteSets* module.

1508 AXIOM $SubsetOfFiniteSetFinite \triangleq$
1509 $\forall\, S,\ T : IsFiniteSet(T) \wedge (S \subseteq T) \Rightarrow IsFiniteSet(S)$

The next result can be proved from simple facts about finite sets and cardinality by induction on the cardinality of S.

1515 AXIOM $FiniteSetHasMax \triangleq$
1516 $\forall\, S \in$ SUBSET $Int :$
1517 $IsFiniteSet(S) \wedge (S \neq \{\}) \Rightarrow \exists\, max \in S : \forall\, x \in S : max \geq x$

The next result can be proved from the following facts about sets

- The empty set is finite.
- A singleton set is finite.
- The union of two finite sets is finite

by induction on $j - i$.

1528 AXIOM $IntervalFinite \triangleq \forall\, i, j \in Int : IsFiniteSet(i\,..\,j)$

1529 ├──┤

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 $VInv3$ is required.

1537 THEOREM $VT4 \triangleq TypeOK \wedge VInv2 \wedge VInv3 \Rightarrow$
1538 $\forall\, Q \in Quorum,\ b \in Ballot :$
1539 $(\forall\, a \in Q : (maxBal[a] \geq b)) \Rightarrow \exists\, v \in Value : SafeAt(b,\ v)$

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

$\langle 1\rangle 1$. SUFFICES ASSUME $TypeOK$, $VInv2$, $VInv3$,
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

$\langle 1\rangle 2$.CASE $b = 0$
BY $ValueNonempty$, $\langle 1\rangle 1$, $SafeAtProp$, $\langle 1\rangle 2$

$\langle 1\rangle 3$. SUFFICES ASSUME $b \neq 0$
PROVE $\exists\, v \in Value : SafeAt(b, v)$
BY $\langle 1\rangle 2$

$\langle 1\rangle 4$. SUFFICES $\exists\, v \in Value :$
$\exists\, c \in -1 \,..\, (b-1) :$
$\land\, (c \neq -1) \Rightarrow \land\, SafeAt(c, v)$
$\land\, \forall\, a \in Q :$
$\forall\, w \in Value :$
$VotedFor(a, c, w) \Rightarrow (w = v)$
$\land\, \forall\, d \in (c+1) \,..\, (b-1),\ a \in Q : DidNotVoteIn(a, d)$

$\langle 2\rangle 1$. SUFFICES ASSUME NEW $v \in Value$,
$\langle 1\rangle 4!1!(v)$
PROVE $SafeAt(b, v)$
OBVIOUS

$\langle 2\rangle 2$. $SafeAtProp!(b, v)$
BY $SafeAtProp$

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

$\langle 1\rangle 5$.CASE $\forall\, a \in Q,\ c \in 0 \,..\, (b-1) : DidNotVoteIn(a, c)$

$\langle 2\rangle 1$. PICK $v \in Value$: TRUE
BY $ValueNonempty$

$\langle 2\rangle -1 \in -1 \,..\, (b-1)$
BY $SimpleArithmetic$ DEF $Ballot$

$\langle 2\rangle 2$. WITNESS $v \in Value$

$\langle 2\rangle 3$. WITNESS $-1 \in -1 \,..\, (b-1)$

$\langle 2\rangle 4$. QED
BY $\langle 1\rangle 5$

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

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

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

$\langle 3\rangle 1$. $S \neq \{\}$
BY $\langle 1\rangle 6$

$\langle 3\rangle 2$. PICK $c \in S : \forall\, d \in S : c \geq d$

$\langle 4\rangle 2$. $(0 \in Int) \land (b-1 \in Int) \ \land (\forall\, x \in 0 \,..\, (b-1) : x \in Int)$
BY $\langle 1\rangle 3$, $SimpleArithmetic$ DEF $Ballot$

1585 ⟨4⟩3. $(S \in \text{SUBSET } Int)$
1586 BY ⟨4⟩2
1587 ⟨4⟩4. $IsFiniteSet(S)$
1588 BY ⟨4⟩2, *IntervalFinite*, *SubsetOfFiniteSetFinite*
1589 ⟨4⟩5. QED
1590 BY ⟨3⟩1, ⟨4⟩3, ⟨4⟩4, *FiniteSetHasMax*
1591 ⟨3⟩3. $c \in 0 \,..\, (b-1)$
1592 OBVIOUS
1593 ⟨3⟩4. $\forall\, d \in (c+1) \,..\, (b-1) : d \in 0 \,..\, (b-1) \land \neg(c \geq d)$
1594 BY ⟨3⟩3, *SimpleArithmetic*
1595 ⟨3⟩5. $\forall\, d \in (c+1) \,..\, (b-1),\ a \in Q : DidNotVoteIn(a,\ d)$
1596 BY ⟨3⟩2, ⟨3⟩4
1597 ⟨3⟩6. $\exists\, a \in Q : \neg DidNotVoteIn(a,\ c)$
1598 BY ⟨3⟩1
1599 ⟨3⟩7. QED
1600 BY ⟨3⟩3, ⟨3⟩5, ⟨3⟩6
1601 ⟨2⟩2. $(c \in -1 \,..\, (b-1)) \land (c \neq -1) \land (c \in Ballot)$
1602 BY *SimpleArithmetic* DEF *Ballot*
1603 ⟨2⟩3. PICK $a0 \in Q : \neg DidNotVoteIn(a0,\ c)$
1604 BY ⟨2⟩1
1605 ⟨2⟩4. PICK $v \in Value : VotedFor(a0,\ c,\ v)$
1606 BY ⟨2⟩3 DEF *DidNotVoteIn*
1607 ⟨2⟩5. $\forall\, a \in Q : \forall\, w \in Value :$
1608 $VotedFor(a,\ c,\ w) \Rightarrow (w = v)$
1609 BY ⟨2⟩2, ⟨2⟩4, *QA*, ⟨1⟩1 DEF *VInv*3
1610 ⟨2⟩6. $SafeAt(c,\ v)$
1611 BY ⟨1⟩1, ⟨2⟩4, *QA*, ⟨2⟩2 DEF *VInv*2
1612 ⟨2⟩7. QED
1613 BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩5, ⟨2⟩6

1615 ⟨1⟩7. QED
1616 BY ⟨1⟩5, ⟨1⟩6
1617 ├──┤

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.

1634 $LiveAssumption \triangleq$
1635 $\exists\, Q \in Quorum,\ b \in Ballot :$
1636 $\forall\, self \in Q :$

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

$LiveSpec \triangleq Spec \land LiveAssumption$

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.

Some Temporal Logic Proof Rules

We now state some temporal logic proof rules that are used in the liveness proof. Some version of these rules will eventually be added to the *TLAPS* module.

The first rule is the lattice rule. To state it, we define *WellFounded*$(S,\ LT)$ to assert that the relation LT is a well-founded "less-than" relation on the set S. This means that there is no infinite sequence of elements of S, each of which is less than the previous one. We represent a relation the way mathematicians generally do, as a set of ordered pairs. In this case $\langle s,\ t\rangle \in LT$ means that s is less than t.

$WellFounded(S,\ LT) \triangleq \neg\exists\, f \in [Nat \to S] :$
$\qquad\qquad \forall\, i \in Nat : \langle f[i+1],\ f[i]\rangle \in LT$

We now define *ProperSubsetRel*(S) to be the relation on a set S such that $\langle U,\ V\rangle \in S$ if and only if U and V are subsets of S with U a proper subset of V. We then state without proof the result that, if S is a finite set, then *ProperSubsetRel*(S) is a well-founded relation on S.

$ProperSubsetRel(S) \triangleq$
$\{r \in (\text{SUBSET } S) \times (\text{SUBSET } S) : \land\, r[1] \subseteq r[2]$
$\qquad\qquad\qquad\qquad\qquad\qquad \land\, r[1] \neq r[2]\}$

THEOREM $SubsetWellFounded \triangleq$
$\qquad \forall\, S : IsFiniteSet(S) \Rightarrow WellFounded(\text{SUBSET } S,\ ProperSubsetRel(S))$
PROOF OMITTED

Here is our statement of the Lattice Rule, which is discussed in

AUTHOR = "Leslie *Lamport*", *TITLE* = "The Temporal Logic of Actions", *JOURNAL* = *toplas*,
volume = 16,
number = 3, *YEAR* = 1994,
month = may,
PAGES = "872–923"

```
THEOREM LatticeRule ≜  ASSUME NEW S, NEW LT, WellFounded(S, LT),
                               NEW TEMPORAL P(_), NEW TEMPORAL Q
                       PROVE   ∧ Q ∨ (∃ i ∈ S : P(i))
                               ∧ ∀ i ∈ S :
                                    P(i) ⇝ (Q ∨ ∃ j ∈ S : (⟨j, i⟩ ∈ LT) ∧ P(j))
                               ⇒ ((∃ i ∈ S : P(i)) ⇝ Q)
PROOF OMITTED
```

Here are two more temporal-logic proof rules. Their validity is obvious when you understand what they mean. We present a proof of the second, mostly to show how temporal logic proofs look in TLA+. Since almost all the steps are temporal formulas, we don't bother trying to check any of them.

```
THEOREM AlwaysForall ≜
            ASSUME NEW CONSTANT S, NEW TEMPORAL P(_)
            PROVE  (∀ s ∈ S : □P(s)) ≡ □(∀ s ∈ S : P(s))

LEMMA EventuallyAlwaysForall ≜
          ASSUME NEW CONSTANT S, IsFiniteSet(S),
                 NEW TEMPORAL P(_)
          PROVE  (∀ s ∈ S : ◇□P(s)) ⇒ ◇□(∀ s ∈ S : P(s))
 ******
⟨1⟩ DEFINE Hyp ≜ ∀ s ∈ S : ◇□P(s)Q(T) ≜ ∀ s ∈ S \ T : □P(s)
           LT ≜ ProperSubsetRel(S)
⟨1⟩1. Hyp ⇒ ∃ T ∈ SUBSET S : Q(T)
 ⟨2⟩1. Hyp ⇒ Q(S)
 ⟨2⟩2. QED
  BY ⟨2⟩1
⟨1⟩2. Hyp ⇒ ∀ T ∈ SUBSET S : Q(T) ⇝ (Q({}) ∨ ∃ R ∈ SUBSET S : ⟨R, T⟩ ∈ LT ∧ Q(R))
 ⟨2⟩1 SUFFICES ASSUME NEW T ∈ SUBSET S
          PROVE Hyp ⇒
                (Q(T) ⇝ (Q({}) ∨ ∃ R ∈ SUBSET S : ⟨R, T⟩ ∈ LT ∧ Q(R)))
  OBVIOUS
 ⟨2⟩2.CASE T ≠ {}
  ⟨3⟩1. PICK s ∈ T : TRUE
   BY ⟨2⟩2
  ⟨3⟩1a. s ∈ S
   OBVIOUS
  ⟨3⟩ DEFINE R ≜ T \ {s}
  ⟨3⟩2. (⟨R, T⟩ ∈ LT) ∧ (S \ R = (S \ T) ∪ {s})
   BY DEF ProperSubsetRel
  ⟨3⟩3. Hyp ⇒ ◇□P(s)
   BY s ∈ S
  ⟨3⟩4. Q(T) ≡ □Q(T)
   BY AlwaysForall
  ⟨3⟩5. ◇□P(s) ⇒ □(Q(T) ⇒ ◇(Q(T) ∧ □P(s)))
   BY ⟨3⟩4
  ⟨3⟩6. Q(T) ∧ □P(s) ≡ Q(R)
   BY ⟨3⟩2
  ⟨3⟩7. Hyp ⇒ (Q(T) ⇝ Q(R)) BY ⟨3⟩3, ⟨3⟩5, ⟨3⟩6
```

⟨3⟩8. QED
BY ⟨3⟩7, ⟨3⟩2
⟨2⟩3.CASE $T = \{\}$
OBVIOUS
⟨2⟩4. QED
BY ⟨2⟩2, ⟨2⟩3
⟨1⟩3. $(\forall\, s \in S : \Diamond\Box P(s)) \Rightarrow (\exists\, T \in \text{SUBSET } S : Q(T)) \rightsquigarrow Q(\{\})$BY ⟨1⟩2, *SubsetWellFounded*, *LatticeRule*
⟨1⟩4. $\Diamond Q(\{\})$
BY ⟨1⟩1, ⟨1⟩3
⟨1⟩5. QED
⟨2⟩1. $Q(\{\}) \equiv \Box(\forall\, s \in S : P(s))$
BY *AlwaysForall*
⟨2⟩2. QED
BY ⟨1⟩4, ⟨2⟩1
****

PROOF OMITTED

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

THEOREM $Liveness \triangleq LiveSpec \Rightarrow C!LiveSpec$
⟨1⟩ SUFFICES ASSUME NEW $Q \in Quorum$, NEW $b \in Ballot$
PROVE $Spec \land LiveAssumption!(Q,\, b) \Rightarrow C!LiveSpec$
BY DEF *LiveSpec*, *LiveAssumption*
PROOF OMITTED

⟨1⟩1. $C!LiveSpec \equiv C!Spec \land (\Box\Diamond\langle C!Next\rangle_{C!vars} \lor \Box\Diamond(chosen \neq \{\}))$
BY *ValueNonempty*, *C!LiveSpecEquals*
PROOF OMITTED

⟨1⟩ DEFINE $LNext \triangleq \exists\, self \in Acceptor,\, c \in Ballot :$
$\land\, BallotAction(self,\, c)$
$\land\, (self \in Q) \Rightarrow (c \leq b)$

⟨1⟩2. $Spec \land LiveAssumption!(Q,\, b) \Rightarrow \Box[LNext]_{vars}$
⟨2⟩1. $\land\, TypeOK$
$\land\, [Next]_{vars}$
$\land\, \forall\, self \in Q :$
$[\forall\, c \in Ballot : (c > b) \Rightarrow \neg BallotAction(self,\, c)]_{vars}$
$\Rightarrow [LNext]_{vars}$
⟨3⟩ SUFFICES ASSUME ⟨2⟩1!1
PROVE $[LNext]_{vars}$
OBVIOUS
⟨3⟩1. $[\land \exists\, self \in Acceptor :$
$\exists\, c \in Ballot : BallotAction(self,\, c)]_{vars}$
BY *NextDef*
⟨3⟩2. $[\forall\, self \in Q,\ \ c \in Ballot :$

```
                (c > b) ⇒ ¬BallotAction(self, c)]_vars
      OBVIOUS
    ⟨3⟩3. [ ∧ ∃ self ∈ Acceptor :
                ∃ c ∈ Ballot : BallotAction(self, c)
            ∧ ∀ self ∈ Q,  c ∈ Ballot :
                (c > b) ⇒ ¬BallotAction(self, c)]_vars
      BY ⟨3⟩1, ⟨3⟩2
    ⟨3⟩4. ∧ ∃ self ∈ Acceptor :
                ∃ c ∈ Ballot : BallotAction(self, c)
          ∧ ∀ self  ∈ Q,  c ∈ Ballot :
                (c  > b) ⇒ ¬BallotAction(self, c)
          ⇒ ∃ self ∈ Acceptor, c ∈ Ballot :
                ∧ BallotAction(self, c)
                ∧ (self ∈ Q) ⇒ (c ≤ b)
      ⟨4⟩ SUFFICES ASSUME ⟨3⟩4!1
                  PROVE  ⟨3⟩4!2
        OBVIOUS
      ⟨4⟩1. PICK self ∈ Acceptor : ∃ c ∈ Ballot : BallotAction(self, c)
        OBVIOUS
      ⟨4⟩2. PICK c ∈ Ballot : BallotAction(self, c)
        BY ⟨4⟩1
      ⟨4⟩3. (c > b) ≡ ¬(c ≤ b)
        BY SimpleArithmetic DEF Ballot
      ⟨4⟩99. QED
        BY ⟨4⟩2, ⟨4⟩3
    ⟨3⟩5. QED
      BY ⟨3⟩3, ⟨3⟩4  DEF LNext
  ⟨2⟩2. ∧ □TypeOK
        ∧ □[Next]_vars
        ∧ ∀ self ∈ Q :
            □[∀ c ∈ Ballot : (c > b) ⇒ ¬BallotAction(self, c)]_vars
        ⇒ □[LNext]_vars
    BY ⟨2⟩1
  PROOF OMITTED
  ⟨2⟩3. LiveAssumption!(Q, b) ⇒
          ∀ self ∈ Q :
            □[∀ c ∈ Ballot : (c > b) ⇒ ¬BallotAction(self, c)]_vars
    OBVIOUS
    PROOF OMITTED
  ⟨2⟩4. QED
    BY ⟨2⟩2, ⟨2⟩3, VT2 DEF Spec
    PROOF OMITTED

⟨1⟩ DEFINE LNInv1 ≜ ∀ a ∈ Q : maxBal[a] ≤ b
           LInv1  ≜ VInv ∧ LNInv1
```

1865 ⟨1⟩3. $LInv1 \land [LNext]_{vars} \Rightarrow LInv1'$
1866 ⟨2⟩1. SUFFICES ASSUME $LInv1$, $[LNext]_{vars}$
1867 PROVE $LInv1'$
1868 OBVIOUS
1869 ⟨2⟩2. $VInv'$
1870 ⟨3⟩1. $[LNext]_{vars} \Rightarrow$
1871 $[\land \exists\, self \in Acceptor :$
1872 $\exists\, c \in Ballot : BallotAction(self,\, c)]_{vars}$
1873 OBVIOUS
1874 ⟨3⟩2. $[Next]_{vars} =$
1875 $[\exists\, self \in Acceptor :$
1876 $\exists\, c \in Ballot : BallotAction(self,\, c)]_{vars}$
1877 BY ⟨2⟩1, *NextDef* DEF $LInv1$, $VInv$
1878 ⟨3⟩3. QED
1879 BY ⟨2⟩1, ⟨3⟩1, ⟨3⟩2, *InductiveInvariance* DEF $LInv1$
1880 ⟨2⟩3. $LNInv1'$
1881 ⟨3⟩1.CASE $vars' = vars$
1882 BY ⟨2⟩1, ⟨3⟩1 DEF $vars$, $LNInv1$
1883 ⟨3⟩2. SUFFICES ASSUME NEW $self \in Acceptor$,
1884 NEW $c \in Ballot$,
1885 $BallotAction(self,\, c)$,
1886 $(self \in Q) \Rightarrow (c \leq b)$,
1887 NEW $a \in Q$
1888 PROVE $maxBal'[a] \leq b$
1889 BY ⟨2⟩1, ⟨3⟩1 DEF $LNInv1$
1890 ⟨3⟩ $a \in Acceptor$
1891 BY QA
1892 ⟨3⟩3.CASE $self = a$
1893 ⟨4⟩1. $c \leq b$
1894 BY ⟨3⟩2, ⟨3⟩3
1895 ⟨4⟩2. $maxBal'[self] = c$
1896 BY ⟨3⟩2, ⟨3⟩3, ⟨2⟩1
1897 DEF $BallotAction$, $IncreaseMaxBal$, $VoteFor$, $VInv$, $TypeOK$
1898 ⟨4⟩3. QED
1899 BY ⟨4⟩1, ⟨4⟩2, ⟨3⟩3
1900 ⟨3⟩4.CASE $self \neq a$ $\notin Q$
1901 ⟨4⟩1.CASE $IncreaseMaxBal(self,\, c)$
1902 BY ⟨4⟩1, ⟨3⟩4, ⟨2⟩1 DEF $LInv1$, $LNInv1$, $IncreaseMaxBal$, $VInv$, $TypeOK$
1903 ⟨4⟩2.CASE $\exists\, v \in Value : VoteFor(self,\, c,\, v)$
1904 BY ⟨4⟩2, ⟨3⟩4, ⟨2⟩1 DEF $LInv1$, $LNInv1$, $VoteFor$, $VInv$, $TypeOK$
1905 ⟨4⟩3. QED
1906 BY ⟨3⟩2, ⟨4⟩1, ⟨4⟩2 DEF $BallotAction$
1907 ⟨3⟩5. QED
1908 BY ⟨3⟩3, ⟨3⟩4
1909 ⟨2⟩4. QED

BY $\langle 2\rangle 2$, $\langle 2\rangle 3$ DEF $LInv1$

$\langle 1\rangle 4.\ \forall\, a \in Q :$
$VInv \wedge (maxBal[a] = b) \wedge [LNext]_{vars} \Rightarrow VInv' \wedge (maxBal'[a] = b)$
Much of this proof copied without from proof for $LInv1$, without checking how much of it was needed.
$\langle 2\rangle 1.$ SUFFICES ASSUME NEW $a \in Q$,
$VInv$, $maxBal[a] = b$, $[LNext]_{vars}$
PROVE $VInv' \wedge (maxBal'[a] = b)$
OBVIOUS
$\langle 2\rangle 2.\ VInv'$
$\langle 3\rangle 1.\ [LNext]_{vars} \Rightarrow$
$[\wedge \exists\, self \in Acceptor :$
$\exists\, c \in Ballot : BallotAction(self,\ c)]_{vars}$
OBVIOUS
$\langle 3\rangle 2.\ [Next]_{vars} =$
$[\exists\, self \in Acceptor :$
$\exists\, c \in Ballot : BallotAction(self,\ c)]_{vars}$
BY $\langle 2\rangle 1$, $NextDef$ DEF $VInv$
$\langle 3\rangle 3.$ QED
BY $\langle 2\rangle 1$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $InductiveInvariance$
$\langle 2\rangle 3.\ maxBal'[a] = b$
$\langle 3\rangle 1.$CASE $vars' = vars$
BY $\langle 2\rangle 1$, $\langle 3\rangle 1$ DEF $vars$
$\langle 3\rangle 2.$ SUFFICES ASSUME NEW $self \in Acceptor$,
NEW $c \in Ballot$,
$BallotAction(self,\ c)$,
$(self \in Q) \Rightarrow (c \leq b)$
PROVE $maxBal'[a] = b$
BY $\langle 2\rangle 1$, $\langle 3\rangle 1$
$\langle 3\rangle\ a \in Acceptor$
BY QA
$\langle 3\rangle 3.$CASE $self = a$
$\langle 4\rangle 1.\ c \leq b$
BY $\langle 3\rangle 2$, $\langle 3\rangle 3$
$\langle 4\rangle 2.$CASE $IncreaseMaxBal(self,\ c)$
$\langle 5\rangle 1.\ c > b$
BY $\langle 4\rangle 2$, $\langle 3\rangle 3$, $\langle 2\rangle 1$ DEF $IncreaseMaxBal$
$\langle 5\rangle 2.\ \neg(c > b)$
BY $\langle 4\rangle 1$, $SimpleArithmetic$ DEF $Ballot$
$\langle 5\rangle 3.$ QED BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 3.$CASE $\exists\, v \in Value : VoteFor(self,\ c,\ v)$
$\langle 5\rangle 1.$ PICK $v \in Value : VoteFor(self,\ c,\ v)$
BY $\langle 4\rangle 3$

⟨5⟩2. $b \leq c$
BY ⟨3⟩3, ⟨5⟩1, ⟨2⟩1 DEF *VoteFor*
⟨5⟩3. $b = c$
BY ⟨4⟩1, ⟨5⟩2, *SimpleArithmetic* DEF *Ballot*
⟨5⟩4. QED
BY ⟨2⟩1, ⟨5⟩1, ⟨5⟩3 DEF *VoteFor*, *VInv*, *TypeOK*
⟨4⟩4. QED
BY ⟨4⟩2, ⟨4⟩3, ⟨3⟩2 DEF *BallotAction*
⟨3⟩4.CASE $self \neq a$ $\notin Q$
⟨4⟩1.CASE *IncreaseMaxBal*(*self*, c)
BY ⟨4⟩1, ⟨3⟩4, ⟨2⟩1 DEF *IncreaseMaxBal*, *VInv*, *TypeOK*
⟨4⟩2.CASE $\exists\, v \in Value : VoteFor(self, c, v)$
BY ⟨4⟩2, ⟨3⟩4, ⟨2⟩1 DEF *VoteFor*, *VInv*, *TypeOK*
⟨4⟩3. QED
BY ⟨3⟩2, ⟨4⟩1, ⟨4⟩2 DEF *BallotAction*
⟨3⟩5. QED
BY ⟨3⟩3, ⟨3⟩4
⟨2⟩99. QED
BY ⟨2⟩2, ⟨2⟩3

⟨1⟩5. $Spec \wedge LiveAssumption!(Q, b) \Rightarrow$
$\Diamond\Box(\forall\, self \in Q : maxBal[self] = b)$
⟨2⟩1. SUFFICES ASSUME NEW $self \in Q$
PROVE $Spec \wedge LiveAssumption!(Q, b) \Rightarrow \Diamond\Box(maxBal[self] = b)$
BY *EventuallyAlwaysForall*
PROOF OMITTED
⟨2⟩ DEFINE $P \triangleq LInv1 \wedge (maxBal[self] \neq b)$
$QQ \triangleq LInv1 \wedge (maxBal[self] = b)$
$A \triangleq BallotAction(self, b)$
⟨2⟩2.$\Box[LNext]_{vars} \wedge \mathrm{WF}_{vars}(A) \Rightarrow (LInv1 \leadsto QQ)$
⟨3⟩1. $P \wedge [LNext]_{vars} \Rightarrow (P' \vee QQ')$
BY ⟨1⟩3
⟨3⟩2. $P \wedge \langle LNext \wedge A\rangle_{vars} \Rightarrow QQ'$
⟨4⟩1. SUFFICES ASSUME $LInv1$, $LNext$, A
PROVE QQ'
OBVIOUS
⟨4⟩2. $LInv1'$
BY ⟨4⟩1, ⟨1⟩3
⟨4⟩3.CASE *IncreaseMaxBal*(*self*, b)
⟨5⟩1. $maxBal'[self] = b$
BY ⟨4⟩1, ⟨4⟩3, QA DEF *IncreaseMaxBal*, *VInv*, *TypeOK*
⟨5⟩2. QED
BY ⟨4⟩2, ⟨5⟩1
⟨4⟩4.CASE $\exists\, v \in Value : VoteFor(self, b, v)$
⟨5⟩1. PICK $v \in Value : VoteFor(self, b, v)$

BY $\langle 4\rangle 4$
$\langle 5\rangle 2.\ maxBal'[self] = b$
BY $\langle 4\rangle 1$, $\langle 5\rangle 1$, QA DEF $VoteFor$, $VInv$, $TypeOK$
$\langle 5\rangle 3.$ QED
BY $\langle 4\rangle 2$, $\langle 5\rangle 2$
$\langle 4\rangle 5.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 3$, $\langle 4\rangle 4$ DEF $BallotAction$
$\langle 3\rangle 3.\ P \Rightarrow$ ENABLED $\langle A\rangle_{vars}$
$\langle 4\rangle 1.$ (ENABLED $\langle A\rangle_{vars}$) =
$\exists\, votesp,\ maxBalp :$
$\land\ \lor\ \land\ b > maxBal[self]$
$\land\ maxBalp = [maxBal$ EXCEPT $![self] = b]$
$\land\ votesp = votes$
$\lor \exists\, v \in Value :$
$\land\ maxBal[self] \leq b$
$\land\ DidNotVoteIn(self,\ b)$
$\land \forall\, p \in Acceptor \setminus \{self\} :$
$\forall\, w \in Value : VotedFor(p,\ b,\ w) \Rightarrow (w = v)$
$\land\ SafeAt(b,\ v)$
$\land\ votesp = [votes$ EXCEPT $![self] = votes[self]$
$\cup\, \{\langle b,\ v\rangle\}]$
$\land\ maxBalp = [maxBal$ EXCEPT $![self] = b]$
$\land\ \langle votesp,\ maxBalp\rangle \neq\ \langle votes,\ maxBal\rangle$
BY DEF $BallotAction$, $IncreaseMaxBal$, $VoteFor$, $vars$, $SafeAt$, $DidNotVoteIn$, $VotedFor$
PROOF OMITTED
$\langle 4\rangle 2.$ SUFFICES ASSUME P
PROVE $\langle 4\rangle 1!2$
BY $\langle 4\rangle 1$
PROOF OMITTED
$\langle 4\rangle 3.$ SUFFICES $\exists\, votesp,\ maxBalp :$
$\land\ \land\ b > maxBal[self]$
$\land\ maxBalp = [maxBal$ EXCEPT $![self] = b]$
$\land\ votesp = votes$
$\land\ \langle votesp,\ maxBalp\rangle \neq\ \langle votes,\ maxBal\rangle$
OBVIOUS
$\langle 4\rangle$ WITNESS $votes$, $[maxBal$ EXCEPT $![self] = b]$
$\langle 4\rangle 4.\ b > maxBal[self]$
$\langle 5\rangle 1. \forall\, mbs \in Ballot \cup \{-1\} : mbs \leq b \land mbs \neq b \Rightarrow b > mbs$
BY $SimpleArithmetic$ DEF $Ballot$
$\langle 5\rangle 2.\ maxBal[self] \in Ballot \cup \{-1\}$
BY $\langle 4\rangle 2$, QA DEF $LInv1$, $VInv$, $TypeOK$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 4\rangle 2$ DEF $LInv1$, $LNInv1$
$\langle 4\rangle 5.\ [maxBal$ EXCEPT $![self] = b][self] = b$

BY $\langle 4\rangle 2$, QA DEF $LInv1$, $VInv$, $TypeOK$
$\langle 4\rangle 6.\ b \neq maxBal[self]$
$\langle 5\rangle 1.\ \forall\, mbs \in Ballot \cup \{-1\} : b > mbs \Rightarrow b \neq mbs$
BY $SimpleArithmetic$ DEF $Ballot$
$\langle 5\rangle 2.$ QED
BY $\langle 4\rangle 4$, $\langle 4\rangle 2$, QA DEF $LInv1$, $VInv$, $TypeOK$
$\langle 4\rangle 7.$ QED
BY $\langle 4\rangle 4$, $\langle 4\rangle 5$, $\langle 4\rangle 6$
$\langle 3\rangle 4.$ QED
BY $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$, $RuleWF1$
PROOF OMITTED
$\langle 2\rangle 3.\ QQ \wedge \Box[LNext]_{vars} \Rightarrow \Box QQ$
$\langle 3\rangle 1.\ QQ \wedge [LNext]_{vars} \Rightarrow QQ'$
BY $\langle 1\rangle 3$, $\langle 1\rangle 4$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$, $RuleINV1$
PROOF OMITTED
$\langle 2\rangle 4.\Box QQ \Rightarrow \Box(maxBal[self] = b)$
$\langle 3\rangle 1.\ QQ \Rightarrow (maxBal[self] = b)$
OBVIOUS
$\langle 3\rangle 2.$ QED
OBVIOUS
PROOF OMITTED
$\langle 2\rangle 5.$ QED
BY $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$, $\langle 1\rangle 2$
PROOF OMITTED

$\langle 1\rangle$ DEFINE $LNInv2 \triangleq \forall\, a \in Q : maxBal[a] = b$
$LInv2 \triangleq VInv \wedge LNInv2$

$\langle 1\rangle 6.\ LInv2 \wedge [LNext]_{vars} \Rightarrow LInv2'$
$\langle 2\rangle 1.$ SUFFICES ASSUME $VInv$, $\forall\, a \in Q : maxBal[a] = b$, $[LNext]_{vars}$
PROVE $VInv' \wedge (\forall\, a \in Q : maxBal'[a] = b)$
OBVIOUS
$\langle 2\rangle 2.\ VInv'$
$\langle 3\rangle 1.$ PICK $a \in Q : maxBal[a] = b$
BY $\langle 2\rangle 1$, $QuorumNonEmpty$
$\langle 3\rangle 2.$ QED
BY $\langle 2\rangle 1$, $\langle 3\rangle 1$, $\langle 1\rangle 4$
$\langle 2\rangle 3.$ ASSUME NEW $a \in Q$
PROVE $maxBal'[a] = b$
BY $\langle 2\rangle 1$, $\langle 2\rangle 3$, $\langle 1\rangle 4$
$\langle 2\rangle 4.$ QED
BY $\langle 2\rangle 2$, $\langle 2\rangle 3$

2091 $\langle 1\rangle 7.\ Spec \land LiveAssumption!(Q,\ b) \Rightarrow \Diamond(chosen \neq \{\})$
2092 $\langle 2\rangle$ DEFINE $Voted(a) \triangleq \exists\, v \in Value : VotedFor(a,\ b,\ v)$
2093 $\langle 2\rangle 1.\ LInv2 \land (\forall\, a \in Q : Voted(a)) \Rightarrow (chosen \neq \{\})$
2094 $\langle 3\rangle 1.$ SUFFICES ASSUME $LInv2$,
2095 $\forall\, a \in Q : Voted(a)$
2096 PROVE $chosen \neq \{\}$
2097 OBVIOUS
2098 $\langle 3\rangle 2.\ \exists\, v \in Value : \forall\, a \in Q :\ VotedFor(a,\ b,\ v)$
2099 $\langle 4\rangle 1.$ PICK $a0 \in Q : Voted(a0)$
2100 BY $\langle 3\rangle 1,\ QuorumNonEmpty$
2101 $\langle 4\rangle 2.$ PICK $v \in Value : VotedFor(a0,\ b,\ v)$
2102 BY $\langle 4\rangle 1$
2103 $\langle 4\rangle 3.$ ASSUME NEW $a \in Q$
2104 PROVE $VotedFor(a,\ b,\ v)$
2105 $\langle 5\rangle 1.$ PICK $w \in Value : VotedFor(a,\ b,\ w)$
2106 BY $\langle 3\rangle 1$
2107 $\langle 5\rangle 2.\ a0 \in Acceptor \land a \in Acceptor$
2108 BY QA
2109 $\langle 5\rangle 3.\ w = v$
2110 BY $\langle 4\rangle 2,\ \langle 5\rangle 1,\ \langle 5\rangle 2,\ \langle 3\rangle 1$ DEF $LInv2,\ VInv,\ VInv3$
2111 $\langle 5\rangle 4.$ QED
2112 BY $\langle 5\rangle 1,\ \langle 5\rangle 3$
2113 $\langle 4\rangle 4.$ QED
2114 BY $\langle 4\rangle 3$
2115 $\langle 3\rangle 3.$ QED
2116 BY $\langle 3\rangle 2$ DEF $chosen,\ ChosenIn$
2117 $\langle 2\rangle 2.\ Spec \land LiveAssumption!(Q,\ b) \Rightarrow (\forall\, a \in Q : \Diamond\Box\, Voted(a))$
2118 $\langle 3\rangle 1.$ SUFFICES ASSUME NEW $self \in Q$
2119 PROVE $Spec \land LiveAssumption!(Q,\ b) \Rightarrow \Diamond\Box\, Voted(self)$
2120 OBVIOUS
2121 PROOF OMITTED
2122 $\langle 3\rangle 2.\ Spec \land LiveAssumption!(Q,\ b) \Rightarrow \Diamond\, Voted(self)$
2123 $\langle 4\rangle 1.\ Spec \land LiveAssumption!(Q,\ b) \Rightarrow \Diamond\Box LInv2$
2124 $\langle 5\rangle 1.\ Spec \Rightarrow \Box\, VInv$
2125 BY $VT2$
2126 PROOF OMITTED
2127 $\langle 5\rangle 2.\ Spec \land LiveAssumption!(Q,\ b) \Rightarrow \Diamond\Box LNInv2$
2128 BY $\langle 1\rangle 5$
2129 PROOF OMITTED
2130 $\langle 5\rangle 3.$ QED
2131 BY $\langle 5\rangle 1,\ \langle 5\rangle 2$
2132 PROOF OMITTED
2133 $\langle 4\rangle 2.\Box[LNext]_{vars} \land \mathrm{WF}_{vars}(BallotAction(self,\ b))$
2134 $\Rightarrow ((LInv2 \land \neg Voted(self)) \rightsquigarrow LInv2 \land Voted(self))$
2135 $\langle 5\rangle$ DEFINE $P \triangleq LInv2 \land \neg Voted(self)$

$QQ \triangleq LInv2 \land Voted(self)$
$A \triangleq BallotAction(self,\ b)$
$\langle 5\rangle 1.\ P \land [LNext]_{vars} \Rightarrow (P' \lor QQ')$
BY $\langle 1\rangle 6$
$\langle 5\rangle 2.\ P \land \langle LNext \land A\rangle_{vars} \Rightarrow QQ'$
$\langle 6\rangle 1.$ SUFFICES ASSUME P, $LNext$, A PROVE QQ'
OBVIOUS
$\langle 6\rangle 2.$CASE $\exists\, v \in Value : VoteFor(self,\ b,\ v)$
$\langle 7\rangle 2.$ PICK $v \in Value : VoteFor(self,\ b,\ v)$
BY $\langle 6\rangle 2$
$\langle 7\rangle 3.\ LInv2'$
BY $\langle 5\rangle 1$, $\langle 6\rangle 1$
$\langle 7\rangle 4.\ Voted(self)'$
BY $\langle 6\rangle 1$, $\langle 7\rangle 2$, QA DEF $VoteFor$, $Voted$, $VotedFor$, $LInv2$, $VInv$, $TypeOK$
$\langle 7\rangle 5.$ QED
BY $\langle 7\rangle 3$, $\langle 7\rangle 4$
$\langle 6\rangle 3.$CASE $IncreaseMaxBal(self,\ b)$
$\langle 7\rangle 1.\ \neg(b > b)$
BY $SimpleArithmetic$ DEF $Ballot$
$\langle 7\rangle 2.$ QED
BY $\langle 7\rangle 1$, $\langle 6\rangle 1$, $\langle 6\rangle 3$ DEF $IncreaseMaxBal$
$\langle 6\rangle 4.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$ DEF $BallotAction$
$\langle 5\rangle 3.\ P \Rightarrow$ ENABLED $\langle A\rangle_{vars}$
$\langle 6\rangle 1.$ SUFFICES ASSUME P PROVE ENABLED $\langle A\rangle_{vars}$
OBVIOUS
PROOF OMITTED
$\langle 6\rangle 2.$ (ENABLED $\langle A\rangle_{vars}$) =

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

```
                    ∧ ⟨votesp, maxBalp⟩ ≠ ⟨votes, maxBal⟩
    BY DEF BallotAction, IncreaseMaxBal, VoteFor, vars, SafeAt,
       DidNotVoteIn, VotedFor
      PROOF OMITTED
      ⟨6⟩ SUFFICES ⟨6⟩2!2
      BY ⟨6⟩2
        PROOF OMITTED
      ⟨6⟩ SUFFICES
            ∃ votesp, maxBalp :
              ∧ ∃ v ∈ Value :
                  ∧ maxBal[self] ≤ b
                  ∧ DidNotVoteIn(self, b)
                  ∧ ∀ p ∈ Acceptor \ {self} :
                      ∀ w ∈ Value : VotedFor(p, b, w) ⇒ (w = v)
                  ∧ SafeAt(b, v)
                  ∧ votesp = [votes EXCEPT ![self] = votes[self]
                                           ∪ {⟨b, v⟩}]
                  ∧ maxBalp = [maxBal EXCEPT ![self] = b]
              ∧ ⟨votesp, maxBalp⟩ ≠ ⟨votes, maxBal⟩
        OBVIOUS
      ⟨6⟩ DEFINE someVoted ≜ ∃ p ∈ Acceptor \ {self} :
                               ∃ 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)
        OBVIOUS
      ⟨6⟩ DEFINE v ≜ IF someVoted THEN vpval
                                  ELSE CHOOSE v ∈ Value : SafeAt(b, v)
      ⟨6⟩4. (v ∈ Value) ∧ SafeAt(b, v)
        ⟨7⟩1.CASE someVoted
          BY ⟨6⟩3, ⟨6⟩1, ⟨7⟩1 DEF LInv2, VInv, VInv2
        ⟨7⟩2.CASE ¬someVoted
          ⟨8⟩1. b ≥ b
            BY SimpleArithmetic DEF Ballot
          ⟨8⟩2. QED
            BY ⟨6⟩1, ⟨7⟩2, ⟨8⟩1, VT4 DEF LInv2, VInv
        ⟨7⟩3. QED
          BY ⟨7⟩1, ⟨7⟩2
      ⟨6⟩ DEFINE votesp ≜ [votes EXCEPT ![self] = votes[self] ∪ {⟨b, v⟩}]
                 maxBalp ≜ [maxBal EXCEPT ![self] = b]
      ⟨6⟩ WITNESS votesp, maxBalp
      ⟨6⟩ SUFFICES ∧ maxBal[self] ≤ b
```

$\land$ *DidNotVoteIn*(*self*, b)
$\land \forall\, p \in Acceptor \setminus \{self\} :$
$\forall\, w \in Value : VotedFor(p,\, b,\, w) \Rightarrow (w = v)$
$\land\ votesp \neq votes$
BY $\langle 6\rangle 4$
$\langle 6\rangle 5.\ maxBal[self] \leq b$
$\langle 7\rangle 1.\ b \leq b$
BY *SimpleArithmetic* DEF *Ballot*
$\langle 7\rangle 2.$ QED
BY $\langle 7\rangle 1, \langle 6\rangle 1$
$\langle 6\rangle 6.$ *DidNotVoteIn*(*self*, b)
BY $\langle 6\rangle 1$ DEF *Voted*, *DidNotVoteIn*
$\langle 6\rangle 7.$ ASSUME NEW $p \in Acceptor \setminus \{self\}$,
NEW $w \in Value$,
$VotedFor(p,\, b,\, w)$
PROVE $w = v$
BY $\langle 6\rangle 7, \langle 6\rangle 3, \langle 6\rangle 1$ DEF *LInv*2, *VInv*, *VInv*3
$\langle 6\rangle 8.\ votesp \neq votes$
$\langle 7\rangle 1.\ votesp[self] = votes[self] \cup \{\langle b,\, v\rangle\}$
BY $\langle 6\rangle 1$, *QA* DEF *LInv*2, *VInv*, *TypeOK*
$\langle 7\rangle 2.\ \langle b,\, v\rangle \in votesp[self]$
BY $\langle 7\rangle 1$
$\langle 7\rangle 3.\ \forall\, w \in Value : \langle b,\, w\rangle \notin votes[self]$
BY $\langle 6\rangle 6$ DEF *DidNotVoteIn*, *VotedFor*
$\langle 7\rangle 4.$ QED
BY $\langle 7\rangle 2, \langle 7\rangle 3, \langle 6\rangle 4$
$\langle 6\rangle 9.$ QED
BY $\langle 6\rangle 5, \langle 6\rangle 6, \langle 6\rangle 7, \langle 6\rangle 8$
$\langle 5\rangle 4.$ QED
BY $\langle 5\rangle 1, \langle 5\rangle 2, \langle 5\rangle 3$, *RuleWF*1
PROOF OMITTED
$\langle 4\rangle 3.\Box\, LInv2 \land ((LInv2 \land \neg\, Voted(self)) \rightsquigarrow LInv2 \land Voted(self))$
$\Rightarrow \Diamond\, Voted(self)$
OBVIOUS
PROOF OMITTED
$\langle 4\rangle 4.$ QED
BY $\langle 1\rangle 2, \langle 4\rangle 1, \langle 4\rangle 2, \langle 4\rangle 3$
PROOF OMITTED
$\langle 3\rangle 3.\ Spec \Rightarrow \Box(\, Voted(self) \Rightarrow \Box\, Voted(self))$
$\langle 4\rangle 1.\ (\, VInv \land Voted(self)) \land [Next]_{vars} \Rightarrow (\, VInv \land Voted(self))'$
$\langle 5\rangle$ SUFFICES ASSUME *VInv*, *Voted*(*self*), $[Next]_{vars}$
PROVE $VInv' \land Voted(self)'$
OBVIOUS
$\langle 5\rangle 1.\ VInv'$
BY *InductiveInvariance*

2271 ⟨5⟩2. $Voted(self)'$
2272 ⟨6⟩CASE $vars' = vars$
2273 BY DEF $vars$, $Voted$, $VotedFor$
2274 ⟨6⟩CASE $Next$
2275 ⟨7⟩1. $\exists\, a \in Acceptor :$
2276 $\exists\, c \in Ballot : BallotAction(a, c)$
2277 BY $NextDef$ DEF $VInv$
2278 ⟨7⟩2. PICK $a \in Acceptor,\ c \in Ballot : BallotAction(a, c)$
2279 BY ⟨7⟩1
2280 ⟨7⟩3.CASE $IncreaseMaxBal(a, c)$
2281 BY ⟨7⟩3 DEF $IncreaseMaxBal$, $Voted$, $VotedFor$
2282 ⟨7⟩4.CASE $\exists\, v \in Value : VoteFor(a, c, v)$
2283 ⟨8⟩1. PICK $v \in Value : VoteFor(a, c, v)$
2284 BY ⟨7⟩4
2285 ⟨8⟩2. $votes' = [votes \text{ EXCEPT } ![a] = votes[a] \cup \{\langle c, v\rangle\}]$
2286 BY ⟨8⟩1 DEF $VoteFor$
2287 ⟨8⟩3.CASE $a = self$
2288 ⟨9⟩1. $votes'[self] = votes[self] \cup \{\langle c, v\rangle\}$
2289 BY ⟨8⟩2, ⟨8⟩3 DEF $VInv$, $TypeOK$
2290 ⟨9⟩2. PICK $w \in Value : \langle b, w\rangle \in votes[self]$
2291 BY DEF $Voted$, $VotedFor$
2292 ⟨9⟩3. $\langle b, w\rangle \in votes'[self]$
2293 BY ⟨9⟩1, ⟨9⟩2
2294 ⟨9⟩4. QED
2295 BY ⟨9⟩3 DEF $Voted$, $VotedFor$
2296 ⟨8⟩4.CASE $a \neq self$
2297 ⟨9⟩1. $votes'[self] = votes[self]$
2298 BY ⟨8⟩2, ⟨8⟩4, QA DEF $VInv$, $TypeOK$
2299 ⟨9⟩2. QED
2300 BY ⟨9⟩1 DEF $Voted$, $VotedFor$
2301 ⟨8⟩5. QED
2302 BY ⟨8⟩3, ⟨8⟩4
2303 ⟨7⟩5. QED
2304 BY ⟨7⟩2, ⟨7⟩3, ⟨7⟩4 DEF $BallotAction$
2305 ⟨6⟩ QED
2306 OBVIOUS
2307 ⟨5⟩3. QED
2308 BY ⟨5⟩1, ⟨5⟩2

2310 ⟨4⟩2. $(VInv \wedge Voted(self)) \wedge \Box[Next]_{vars} \Rightarrow \Box(VInv \wedge Voted(self))$
2311 BY ⟨4⟩1, $RuleINV1$
2312 PROOF OMITTED
2313 ⟨4⟩3. QED
2314 ⟨5⟩1. $VInv \wedge \Box[Next]_{vars} \Rightarrow (Voted(self) \Rightarrow \Box\, Voted(self))$
2315 BY ⟨4⟩2

PROOF OMITTED
⟨5⟩2. $\Box\, VInv \land \Box[Next]_{vars} \Rightarrow \Box(Voted(self) \Rightarrow \Box\, Voted(self))$
BY ⟨5⟩1
PROOF OMITTED
⟨5⟩3. QED
BY ⟨5⟩2, *VT2* DEF *Spec*
PROOF OMITTED
⟨3⟩4. QED
BY ⟨3⟩2, ⟨3⟩3
PROOF OMITTED
⟨2⟩3. $(\forall\, a \in Q : \Diamond\Box\, Voted(a)) \Rightarrow \Diamond\Box(\forall\, a \in Q : Voted(a))$
⟨3⟩1. *IsFiniteSet*(Q)
BY *AcceptorFinite*, *QA*, *SubsetOfFiniteSetFinite*
⟨3⟩2. QED
BY ⟨3⟩1, *EventuallyAlwaysForall*
PROOF OMITTED
⟨2⟩4. $\Diamond\Box(\forall\, a \in Q : Voted(a)) \Rightarrow \Diamond(\forall\, a \in Q : Voted(a))$
OBVIOUS
PROOF OMITTED
⟨2⟩5. $\Diamond(\forall\, a \in Q : Voted(a)) \Rightarrow \Diamond(chosen \neq \{\})$
BY ⟨2⟩1
PROOF OMITTED
⟨2⟩6. QED
BY ⟨2⟩2, ⟨2⟩3, ⟨2⟩4, ⟨2⟩5
PROOF OMITTED

⟨1⟩8. $\Box\, VInv \land \Box[Next]_{vars} \land \Diamond(chosen \neq \{\}) \Rightarrow \Diamond\Box(chosen \neq \{\})$
⟨2⟩1. $\Box\, VInv \land (chosen \neq \{\}) \land\ \Box[Next]_{vars} \Rightarrow \Box(chosen \neq \{\})$
⟨3⟩1. $(chosen \neq \{\}) \land [Next \land VInv \land VInv']_{vars} \Rightarrow (chosen' \neq \{\})$
⟨4⟩ SUFFICES ASSUME $chosen \neq \{\}$, $[Next \land VInv \land VInv']_{vars}$
PROVE $chosen' \neq \{\}$
OBVIOUS
⟨4⟩CASE $vars' = vars$
BY DEF *vars*, *chosen*, *ChosenIn*, *VotedFor*
⟨4⟩CASE $Next \land VInv \land VInv'$
⟨5⟩1. PICK $self \in Acceptor$, $c \in Ballot : BallotAction(self, c)$
BY *NextDef* DEF *VInv*
⟨5⟩2.CASE $IncreaseMaxBal(self, c)$
BY ⟨5⟩2 DEF *IncreaseMaxBal*, *chosen*, *ChosenIn*, *VotedFor*
⟨5⟩3.CASE $\exists\, v \in Value : VoteFor(self, c, v)$
⟨6⟩1. PICK $v \in Value : VoteFor(self, c, v)$
BY ⟨5⟩3
⟨6⟩2. $votes' = [votes$ EXCEPT $![self] = votes[self] \cup \{\langle c, v\rangle\}]$
BY ⟨6⟩1 DEF *VoteFor*, *VInv*, *TypeOK*
⟨6⟩3. $\forall\, a \in Acceptor : votes[a] \subseteq votes'[a]$

```
2361              ⟨7⟩ SUFFICES ASSUME NEW a ∈ Acceptor
2362                           PROVE  votes[a] ⊆ votes′[a]
2363                OBVIOUS
2364              ⟨7⟩ QED
2365                BY ⟨6⟩2 DEF VInv, TypeOK
2366            ⟨6⟩4. ∀ a ∈ Acceptor, d ∈ Ballot, w ∈ Value :
2367                    VotedFor(a, d, w) ⇒ VotedFor(a, d, w)′
2368              BY ⟨6⟩3 DEF VotedFor
2369            ⟨6⟩5. ASSUME NEW d ∈ Ballot, NEW w ∈ Value, ChosenIn(d, w)
2370                  PROVE  ChosenIn(d, w)′
2371              ⟨7⟩1. PICK QQ ∈ Quorum : ∀ a ∈ QQ : VotedFor(a, d, w)
2372                BY ⟨6⟩5 DEF ChosenIn
2373              ⟨7⟩2. ∀ a ∈ QQ : VotedFor(a, d, w)′
2374                BY QA, ⟨6⟩4, ⟨7⟩1
2375              ⟨7⟩3. QED
2376                BY ⟨7⟩2 DEF ChosenIn
2377            ⟨6⟩6. QED
2378              BY ⟨6⟩5 DEF chosen
2379          ⟨5⟩4. QED
2380            BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3 DEF BallotAction
2381        ⟨4⟩ QED
2382          OBVIOUS
2383      ⟨3⟩2. (chosen ≠ {}) ∧ □[Next ∧ VInv ∧ VInv′]_vars ⇒ □(chosen ≠ {})
2384        BY ⟨3⟩1, RuleINV1
2385          PROOF OMITTED
2386      ⟨3⟩3.□ VInv ∧ □[Next]_vars ⇒ □[Next ∧ VInv ∧ VInv′]_vars
2387        BY RuleINV2
2388          PROOF OMITTED
2389      ⟨3⟩4. QED
2390        BY ⟨3⟩2, ⟨3⟩3
2391        PROOF OMITTED
2392    ⟨2⟩2. QED
2393      BY ⟨2⟩1 \* and PTL
2394        PROOF OMITTED

2396  ⟨1⟩9 Spec ∧ LiveAssumption!(Q, b) ⇒ ◇□(chosen ≠ {})
2397    ⟨2⟩1. Spec ⇒ □ VInv ∧ □[Next]_vars
2398      BY VT2 DEF Spec
2399        PROOF OMITTED
2400    ⟨2⟩2. Spec ∧ LiveAssumption!(Q, b) ⇒
2401              □ VInv ∧ □[Next]_vars ∧ ◇(chosen ≠ {})
2402      BY ⟨2⟩1, ⟨1⟩7
2403        PROOF OMITTED
2404    ⟨2⟩3. QED
2405      BY ⟨1⟩8, ⟨2⟩2
```

PROOF OMITTED

⟨1⟩10. QED
⟨2⟩1. $Spec \land LiveAssumption!(Q,\ b) \Rightarrow C!Spec \land \Diamond\Box(chosen \neq \{\})$
BY $VT3$, ⟨1⟩9
PROOF OMITTED
⟨2⟩2. $Spec \land LiveAssumption!(Q,\ b) \Rightarrow C!Spec \land \Box\Diamond(chosen \neq \{\})$
BY ⟨2⟩1
PROOF OMITTED
⟨2⟩3. QED
BY ⟨2⟩2, ⟨1⟩1
PROOF OMITTED

* Modification History
* Last modified Sat *Nov* 16 22:18:41 *CST* 2019 by *hengxin*
* Last modified *Wed Feb* 16 15:20:49 *PST* 2011 by lamport