MODULE *OneVotePaxos*

This is a specification of the *Paxos* algorithm without explicit leaders or learners. It is adapted from *Paxos.tla*.

WARNING: It does not satisfy *Consistency*; see *OneVotePaxos*-NotOneValuePerBallot-*ErrorTrace.md*

In this version:

1. *Phase2a*(b, v): Delete the enabling condition "$\sim \exists\, m \in msgs : m.type =$ "2a" $\wedge\, m.bal = b$". Then, *OneValuePerBallot* (and hence, *OneVote*) does not hold anymore. Consistency is also broken. See the error trace file: *OneVotePaxos*-phase2a-error-*trace.md*

2. *Phase2b*(a): To fix (1), we change "$m.bal \geq maxBal[a]$" to "$m.bal > maxBal[a] \vee\ (m.bal = maxBal[a] \wedge\ (m.bal = maxVBal[a] \Rightarrow maxVal[a] = None))$" to restore *OneVote*.

Additionally,

Phase1b(a): it is safe to send "1b" messages unconditionally by merging "$\wedge\, m.bal > maxBal(a)$" and "$\wedge\, maxBal' = [maxBal$ EXCEPT $![a] = m.bal]$" into "$\wedge\, maxBal' = [maxBal$ EXCEPT $![a] = Max(m.bal, @)]$". However, this hurts performance significantly (therefore, we do not do this).

EXTENDS *Integers*, *FiniteSets*, *TLC*

$Max(m, n) \triangleq$ IF $m < n$ THEN n ELSE m

CONSTANT *Value*, *Acceptor*, *Quorum*

ASSUME *QuorumAssumption* $\triangleq$
$\wedge\quad \forall\, Q \in Quorum : Q \subseteq Acceptor$
$\wedge\quad \forall\, Q1,\, Q2 \in Quorum : Q1 \cap Q2 \neq \{\}$

$Ballot \triangleq Nat$
$None \triangleq$ CHOOSE $v : v \notin Ballot$

$Message \triangleq$
$\quad [type : \{\text{"1a"}\},\, bal : Ballot]$
$\cup\ [type : \{\text{"1b"}\},\, acc : Acceptor,\, bal : Ballot,$
$\quad mbal : Ballot \cup \{-1\},\, mval : Value \cup \{None\}]$
$\cup\ [type : \{\text{"2a"}\},\, bal : Ballot,\, val : Value]$
$\cup\ [type : \{\text{"2b"}\},\, acc : Acceptor,\, bal : Ballot,\, val : Value]$

VARIABLE *maxBal*, *maxVBal*, *maxVal*, *msgs*

maxBal$[a]$: the largest ballot number a has seen
$\langle$*maxVBal*$[a]$, *maxVal*$[a]\rangle$ is the vote with the largest ballot number cast by a; it is $\langle -1,\, None\rangle$ if a has not cast any vote.
The set of all messages that have been sent.

$Send(m) \triangleq msgs' = msgs \cup \{m\}$

$vars \triangleq \langle maxBal,\, maxVBal,\, maxVal,\, msgs\rangle$

$TypeOK \triangleq$
$\wedge\quad maxBal \in [Acceptor \to Ballot \cup \{-1\}]$

```
59      ∧   maxVBal ∈ [Acceptor → Ballot ∪ {−1}]
60      ∧   maxVal ∈ [Acceptor → Value ∪ {None}]
61      ∧   msgs ⊆ Message
62  ----------------------------------------------------------------
63  Init ≜
64      ∧ maxBal = [a ∈ Acceptor ↦ −1]
65      ∧ maxVBal = [a ∈ Acceptor ↦ −1]
66      ∧ maxVal = [a ∈ Acceptor ↦ None]
67      ∧ msgs = {}
```

In an implementation, there will be a leader process that orchestrates a ballot. The ballot b leader performs actions $Phase1a(b)$ and $Phase2a(b)$. The $Phase1a(b)$ action sends a phase $1a$ message that begins ballot b.

```
73  Phase1a(b) ≜
74      ∧   Send([type ↦ "1a", bal ↦ b])
75      ∧   UNCHANGED ⟨maxBal, maxVBal, maxVal⟩
```

Upon receipt of a ballot b phase $1a$ message, acceptor a can perform a $Phase1b(a)$ action only if $b > maxBal[a]$. The action sets $maxBal[a]$ to b and sends a phase $1b$ message to the leader containing the values of $maxVBal[a]$ and $maxVal[a]$.

```
81  Phase1b(a) ≜
82      ∧   ∃ m ∈ msgs :
83            ∧ m.type = "1a"
84            ∧ m.bal > maxBal[a]
85            ∧ maxBal' = [maxBal EXCEPT ![a] = m.bal]      make promise
86            ∧ maxBal' = [maxBal EXCEPT ![a] = Max(m.bal, @)]
87            ∧ Send([type ↦ "1b", acc ↦ a, bal ↦ m.bal,
88                    mbal ↦ maxVBal[a], mval ↦ maxVal[a]])
89      ∧   UNCHANGED ⟨maxVBal, maxVal⟩

91  NoBackInTime ≜
92      ∀ m ∈ msgs : m.type = "1b" ⇒ m.mbal < m.bal
```

The $Phase2a(b, v)$ action can be performed by the ballot b leader if two conditions are satisfied: (i) it has not already performed a phase $2a$ action for ballot b and (ii) it has received ballot b phase $1b$ messages from some quorum Q from which it can deduce that the value v is safe at ballot b. These enabling conditions are the first two conjuncts in the definition of $Phase2a(b, v)$. The second conjunct, expressing condition (ii), is the heart of the algorithm. To understand it, observe that the existence of a phase $1b$ message m in $msgs$ implies that $m.mbal$ is the highest ballot number less than $m.bal$ in which acceptor $m.acc$ has or ever will cast a vote, and that $m.mval$ is the value it voted for in that ballot if $m.mbal \neq -1$. It is not hard to deduce from this that the second conjunct implies that there exists a quorum Q such that $ShowsSafeAt(Q, b, v)$ (where $ShowsSafeAt$ is defined in module $Voting$).

The action sends a phase $2a$ message that tells any acceptor a that it can vote for v in ballot b, unless it has already set $maxBal[a]$ greater than b (thereby promising not to vote in ballot b).

```
112 P2C(b, v) ≜
113     ∃ Q ∈ Quorum :
114       LET Q2bv ≜ {m ∈ msgs : m.type = "2b" ∧ m.acc ∈ Q ∧ m.bal < b}
115       IN    ∨ Q2bv = {}
116             ∨ ∃ m    ∈ Q2bv :
```

```
                    ∧ m.val = v
                    ∧ ∀ mm ∈ Q2bv : m.bal ≥ mm.bal

Phase2a(b, v) ≜
  ∧ ∼ ∃ m ∈ msgs : m.type = "2a" ∧ m.bal = b \* allow different values for the same b
  ∧ ∃ Q ∈ Quorum :
      LET Q1b ≜ {m ∈ msgs : m.type = "1b" ∧ m.acc ∈ Q ∧ m.bal = b}
          Q1bv ≜ {m ∈ Q1b : m.mbal ≥ 0}
      IN  ∧ ∀ a ∈ Q : ∃ m ∈ Q1b : m.acc = a
          ∧ ∨ Q1bv = {}
            ∨ ∃ m ∈ Q1bv :
                ∧ m.mval = v
                ∧ ∀ mm ∈ Q1bv : m.mbal ≥ mm.mbal
  ∧ Send([type ↦ "2a", bal ↦ b, val ↦ v])
  ∧ Assert(P2C(b, v), "P2C Fails!")
  ∧ UNCHANGED ⟨maxBal, maxVBal, maxVal⟩
```

The $Phase2b(a)$ action is performed by acceptor a upon receipt of a phase $2a$ message. Acceptor a can perform this action only if the message is for a ballot number greater than or equal to $maxBal[a]$. In that case, the acceptor votes as directed by the phase $2a$ message, setting $maxBVal[a]$ and $maxVal[a]$ to record that vote and sending a phase $2b$ message announcing its vote.

Note: It also sets $maxBal[a]$ to the message's ballot number. Otherwise,
(1) $NoBackInTime$ for $Phase1b$ does not hold.
(2) "Non-Increasing Error" assertion in $Phase2b(a)$ fails.
(3) $P2C$ assertion for $Phase2a$ does not hold.

```
Phase2b(a) ≜
  ∃ m ∈ msgs :
    ∧ m.type = "2a"
    ∧ m.bal ≥ maxBal[a]
    ∧ ∨ m.bal > maxBal[a]
      ∨ m.bal = maxBal[a] ∧ (m.bal = maxVBal[a] ⇒ maxVal[a] = None)   write-once
    ∧ maxBal' = [maxBal EXCEPT ![a] = m.bal]
    ∧ maxVBal' = [maxVBal EXCEPT ![a] = m.bal]
    ∧ Assert(maxVBal'[a] ≥ maxVBal[a], "Non-Increasing Error!")
    ∧ maxVal' = [maxVal EXCEPT ![a] = m.val]
    ∧ Send([type ↦ "2b", acc ↦ a, bal ↦ m.bal, val ↦ m.val])
```

In an implementation, there will be learner processes that learn from the phase $2b$ messages if a value has been chosen. The learners are omitted from this abstract specification of the algorithm.

```
Next ≜
  ∨ ∃ b ∈ Ballot :
      ∨ Phase1a(b)
      ∨ ∃ v ∈ Value : Phase2a(b, v)
  ∨ ∃ a ∈ Acceptor : Phase1b(a) ∨ Phase2b(a)

Spec ≜ Init ∧ □[Next]_vars
```

We now instantiate module *Voting*, substituting the constants *Value*, *Acceptor*, and *Quorum* declared in this module for the corresponding constants of that module *Voting*, and substituting the variable *maxBal* and the defined state function *votes* for the correspondingly-named variables of module *Voting*.

$$votes \triangleq [a \in Acceptor \mapsto \{\langle m.bal,\ m.val\rangle : m \in \{mm \in msgs : \land\ mm.type = \text{“2b”} \land\ mm.acc = a\}\}]$$

$$V \triangleq \text{INSTANCE } Voting$$

$$Consistency \triangleq V!C!Inv$$ Only about “chosen”: $TypeOK \land Cardinality(chosen) \leq 1$

$$StrongConsistency \triangleq V!Inv$$ $TypeOK \land VotesSafe \land OneValuePerBallot$

A first attempt at an inductive invariant used to prove this theorem.

$$\text{THEOREM } Spec \Rightarrow V!Spec$$

$$\begin{array}{l}
Inv \triangleq \land\ TypeOK \\
\qquad \land \forall\, a \in Acceptor : \text{IF } maxVBal[a] = -1 \\
\qquad\qquad\qquad\qquad\qquad \text{THEN } maxVal[a] = None \\
\qquad\qquad\qquad\qquad\qquad \text{ELSE } \langle maxVBal[a],\ maxVal[a]\rangle \in votes[a] \\
\qquad \land \forall\, m \in msgs : \\
\qquad\qquad \land\ (m.type = \text{“1b”}) \Rightarrow\ \land\ maxBal[m.acc] \geq m.bal \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ (m.mbal \geq 0) \Rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \langle m.mbal,\ m.mval\rangle \in votes[m.acc] \\
\qquad\qquad \land\ (m.type = \text{“2a”}) \Rightarrow\ \land \exists\, Q \in Quorum : \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad V!ShowsSafeAt(Q,\ m.bal,\ m.val) \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land \forall\, mm \in msgs :\ \land\ mm.type = \text{“2a”} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ mm.bal\ \ = m.bal \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \Rightarrow\ mm.val = m.val \\
\qquad \land\ V!Inv
\end{array}$$