MODULE *Paxos*

This is a specification of the *Paxos* algorithm without explicit leaders or learners. It refines the spec in *Voting*

EXTENDS *Integers*

The constant parameters and the set Ballots are the same as in *Voting*.

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

$$\text{ASSUME } QuorumAssumption \triangleq \begin{array}{l} \land \forall Q \in Quorum : Q \subseteq Acceptor \\ \land \forall Q1, Q2 \in Quorum : Q1 \cap Q2 \neq \{\} \end{array}$$

$$Ballot \triangleq Nat$$

$$None \triangleq \text{CHOOSE } v : v \notin Ballot$$

An unspecified value that is not a ballot number.

This is a message-passing algorithm, so we begin by defining the set Message of all possible messages. The messages are explained below with the actions that send them.

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

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

NOTE: The algorithm is easier to understand in terms of the set *msgs* of all messages that have ever been sent. A more accurate model would use one or more variables to represent the messages actually in transit, and it would include actions representing message loss and duplication as well as message receipt.

In the current spec, there is no need to model message loss because we are mainly concerned with the algorithm's safety property. The safety part of the spec says only what messages may be received and does not assert that any message actually is received. Thus, there is no difference between a lost message and one that is never received. The liveness property of the spec that we check makes it clear what what messages must be received (and hence either not lost or successfully retransmitted if lost) to guarantee progress.

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

It is convenient to define some identifier to be the tuple of all variables. I like to use the identifier *vars* .

The type invariant and initial predicate.

$$TypeOK \triangleq \land maxBal \in [Acceptor \rightarrow Ballot \cup \{-1\}]$$

$$\begin{aligned}
&\land\ maxVBal \in [Acceptor \rightarrow Ballot \cup \{-1\}]\\
&\land\ maxVal \in [Acceptor \rightarrow Value \cup \{None\}]\\
&\land\ msgs \subseteq Message
\end{aligned}$$

$$\begin{aligned}
Init \triangleq\ &\land\ maxBal = [a \in Acceptor \mapsto -1]\\
&\land\ maxVBal = [a \in Acceptor \mapsto -1]\\
&\land\ maxVal = [a \in Acceptor \mapsto None]\\
&\land\ msgs = \{\}
\end{aligned}$$

The actions. We begin with the subaction (an action that will be used to define the actions that make up the next-state action.

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

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 (a message m with $m.type =$ "1a") that begins ballot b.

$$\begin{aligned}
Phase1a(b) \triangleq\ &\land\ Send([type \mapsto \text{"1a"},\ bal \mapsto b])\\
&\land\ \text{UNCHANGED}\ \langle maxBal,\ maxVBal,\ maxVal\rangle
\end{aligned}$$

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 $maxVal[a]$.

$$\begin{aligned}
Phase1b(a) \triangleq\ &\land\ \exists\, m \in msgs :\\
&\qquad \land\ m.type = \text{"1a"}\\
&\qquad \land\ m.bal > maxBal[a]\\
&\qquad \land\ maxBal' = [maxBal\ \text{EXCEPT}\ ![a] = m.bal]\\
&\qquad \land\ Send([type \mapsto \text{"1b"},\ acc \mapsto a,\ bal \mapsto m.bal,\\
&\qquad\qquad\qquad mbal \mapsto maxVBal[a],\ mval \mapsto maxVal[a]])\\
&\land\ \text{UNCHANGED}\ \langle maxVBal,\ maxVal\rangle
\end{aligned}$$

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 $Phase2a(b, v)$. This 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).

$$\begin{aligned}
&Phase2a(b,\ v) \triangleq\\
&\quad \land\ \neg\exists\, m \in msgs : m.type = \text{"2a"} \land m.bal = b\\
&\quad \land\ \exists\, Q \in Quorum :\\
&\qquad \text{LET}\ Q1b \triangleq \{m \in msgs : \land\ m.type = \text{"1b"}
\end{aligned}$$

$$
\begin{aligned}
&\qquad\qquad\qquad\qquad\qquad\qquad \land\ m.acc \in Q \\
&\qquad\qquad\qquad\qquad\qquad\qquad \land\ m.bal = b\} \\
&\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
&\textsc{in}\quad \land\ \forall a \in Q : \exists m \in Q1b : m.acc = a \\
&\qquad \land\ \lor\ Q1bv = \{\} \\
&\qquad\quad \lor\ \exists m \in Q1bv : \\
&\qquad\qquad\quad \land\ m.mval = v \\
&\qquad\qquad\quad \land\ \forall mm \in Q1bv : m.mbal \geq mm.mbal \\
&\land\ Send([type \mapsto \text{“2a”},\ bal \mapsto b,\ val \mapsto v]) \\
&\land\ \textsc{unchanged}\ \langle maxBal,\ maxVBal,\ maxVal \rangle
\end{aligned}
$$

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. It also sets $maxBal[a]$ to the message's. ballot number

$$
\begin{aligned}
Phase2b(a) \triangleq \exists m \in msgs :\ & \land\ m.type = \text{“2a”} \\
& \land\ m.bal \geq maxBal[a] \\
& \land\ maxBal' = [maxBal\ \textsc{except}\ ![a] = m.bal] \\
& \land\ maxVBal' = [maxVBal\ \textsc{except}\ ![a] = m.bal] \\
& \land\ maxVal' = [maxVal\ \textsc{except}\ ![a] = m.val] \\
& \land\ Send([type \mapsto \text{“2b”},\ acc \mapsto a, \\
& \qquad\qquad bal \mapsto m.bal,\ val \mapsto m.val])
\end{aligned}
$$

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.

Below are defined the next-state action and the complete spec.

$$
\begin{aligned}
Next \triangleq\ & \lor\ \exists b \in Ballot :\ \lor\ Phase1a(b) \\
& \qquad\qquad\qquad\quad \lor\ \exists v \in Value : Phase2a(b, v) \\
& \lor\ \exists a \in Acceptor : Phase1b(a) \lor Phase2b(a)
\end{aligned}
$$

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

We now define the refinement mapping under which this algorithm implements the specification in module *Voting*.

As we observed, votes are registered by sending phase $2b$ messages. So the array *votes* describing the votes cast by the acceptors is defined as follows.

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

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*.

$$
V \triangleq \textsc{instance}\ Voting
$$

THEOREM $Spec \Rightarrow V!Spec$ chosen*Voting*chosen

Here is a first attempt at an inductive invariant used to prove this theorem.

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