MODULE *CASPaxos*

This is a high-level specification of the *CASPaxos* algorithm from the paper "" by.

TODO: It refines the spec in module Voting.

EXTENDS *Integers*

CONSTANTS
- *Value*, — the set of values to be proposed and chosen from
- *Acceptor*, — the set acceptors
- *Quorum* — the quorum system on acceptors

$None \triangleq$ CHOOSE $v : v \notin Value$

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

$Ballot \triangleq Nat$

We now define *Message* to be the set of all possible messages that can be sent in the algorithm. In TLA+, the expression

(1) $[type \mapsto \text{"1a"}, bal \mapsto b]$

is a record r with two components, a *type* component, $r.type$, that equals "1a" and whose *bal* component, $r.bal$, that equals b. The expression

(2) [type : $\{\text{"1a"}\}$, $bal : Ballot$]

is the set of all records r with a components *type* and *bal* such that $r.type$ is an element of $\{\text{"1a"}\}$ and $r.bal$ is an element of *Ballot*. Since "1a" is the only element of $\{\text{"1a"}\}$, formula (2) is the set of all elements (1) such that $b \in Ballot$.

The function of each type of message in the set *Message* is explained below with the action that can send it.

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

---

maxBal − Is the same as the variable of that name in the Voting algorithm.
maxVBal
maxVVal − As in the Voting algorithm, a vote is a $\langle ballot, value \rangle$ pair. The pair $\langle maxVBal[a], maxVVal[a] \rangle$ is the vote with the largest ballot number cast by acceptor a . It equals $\langle -1, None \rangle$ if a has not cast any vote.

VARIABLES
- *maxBal*,
- *maxVBal*,
- *maxVVal*,
- *msgs*, — the set of all messages that have been sent

cas $\quad$ $cas[b \in Ballot]$: $[cmpVal : Value \cup \{None\}, swapVal : Value \cup \{None\}\rangle$ * *TODO*

$$vars \triangleq \langle maxBal, maxVBal, maxVVal, msgs, cas \rangle$$

$$
\begin{aligned}
TypeOK \triangleq\ & \land\ maxBal \in [Acceptor \rightarrow Ballot \cup \{-1\}] \\
& \land\ maxVBal \in [Acceptor \rightarrow Ballot \cup \{-1\}] \\
& \land\ maxVVal \in [Acceptor \rightarrow Value \cup \{None\}] \\
& \land\ msgs \subseteq Message \\
& \land\ cas \in [Ballot \rightarrow [cmpVal : Value \cup \{None\}, \\
& \qquad\qquad\qquad\quad swapVal : Value \cup \{None\}]]
\end{aligned}
$$

$$
\begin{aligned}
Init \triangleq\ & \land\ maxBal = [a \in Acceptor \mapsto -1] \\
& \land\ maxVBal = [a \in Acceptor \mapsto -1] \\
& \land\ maxVVal = [a \in Acceptor \mapsto None] \\
& \land\ msgs = \{\} \\
& \land\ cas = [b \in Ballot \mapsto [cmpVal \mapsto None, swapVal \mapsto None]]
\end{aligned}
$$
TODO:

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

The leader of ballot $b \in Ballot$ issues an $CAS(cmpVal, swapVal)$ operation by sending a $Phase1a$ message.

$$
\begin{aligned}
& Phase1a(b, cmpVal, swapVal) \triangleq \\
& \quad \land\ Send([type \mapsto \text{“1a”}, bal \mapsto b]) \\
& \quad \land\ cas' = [cas \text{ EXCEPT } ![b] = [cmpVal \mapsto cmpVal, swapVal \mapsto swapVal]] \\
& \quad \land\ \text{UNCHANGED } \langle maxBal, maxVBal, maxVVal \rangle
\end{aligned}
$$

The acceptor $a \in Acceptor$ receives a $Phase1a$ message and sends back a $Phase1b$ message.

TODO: This action implements the $IncreaseMaxBal(a, b)$ action of the Voting algorithm for $b = m.bal$.

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

In the $Phase2a(b, v)$ action, the ballot b leader sends a type “2a” message asking the acceptors to vote for v in ballot number b. The enabling conditions of the action–its first two conjuncts–ensure that three of the four enabling conditions of action $VoteFor(a, b, v)$ in module Voting will be true when acceptor a receives that message. Those three enabling conditions are the second through fourth conjuncts of that action.

The first conjunct of $Phase2a(b, v)$ asserts that at most one phase $2a$ message is ever sent for ballot b. Since an acceptor will vote for a value in ballot b only when it receives the appropriate phase $2a$ message, the phase $2a$ message sent by this action this ensures that these two enabling conjuncts of $VoteFor(a, b, v)$ will be true forever:

$$\land \forall vt \in votes[a] : vt[1] \neq b$$
$$\land \forall c \in Acceptor \setminus \{a\} : \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v)$$

The second conjunct of the $Phase2a(b, v)$ action is the heart of the Paxos consensus algorithm. It's a bit complicated, but I've tried a number of times to write it in *English*, and it's much easier to understand when written in mathematics. The LET /IN construct locally defines $Q1$ to be the set of phase $1b$ messages sent in ballot number b by acceptors in quorum Q; and it defines $Q1bv$ to be the subset of those messages indicating that the sender had voted in some ballot (which must have been numbered less than b). You should study the IN clause to convince yourself that it equals $ShowsSafeAt(Q, b, v)$, defined in module Voting, using the values of $maxBal[a]$, $maxVBal[a]$, and $maxVVal[a]a$ sent in its phase $1b$ message to describe what votes it had cast when it sent that message. Moreover, since a will no longer cast any votes in ballots numbered less than b, the IN clause implies that $ShowsSafeAt(Q, b, v)$ is still true and will remain true forever. Hence, this conjunct of $Phase2a(b, v)$ checks the enabling condition

$$\land \exists Q \in Quorum : ShowsSafeAt(Q, b, v)$$

of module Voting's $VoteFor(a, b, v)$ action.

The type "2a" message sent by this action therefore tells every acceptor a that, when it receives the message, all the enabling conditions of $VoteFor(a, b, v)$ but the first, $maxBal[a] \leq b$, are satisfied.

$$
\begin{array}{l}
Phase2a(b, v) \triangleq \\
\quad \land \neg \exists m \in msgs : m.type = \text{"2a"} \land m.bal = b \\
\quad \land \exists Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land m.type = \text{"1b"} \\
\qquad\qquad\qquad\qquad\qquad\qquad \land m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad \land m.bal = b\} \\
\qquad\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN } \quad \land \forall a \in Q : \exists m \in Q1b : m.acc = a \\
\qquad\qquad \land \lor Q1bv = \{\} \\
\qquad\qquad\quad \lor \exists m \in Q1bv : \\
\qquad\qquad\qquad \land m.mval = v \\
\qquad\qquad\qquad \land \forall mm \in Q1bv : m.mbal \geq mm.mbal \\
\quad \land Send([type \mapsto \text{"2a"}, bal \mapsto b, val \mapsto v]) \\
\quad \land \text{UNCHANGED } \langle maxBal, maxVBal, maxVVal \rangle
\end{array}
$$

The $Phase2b(a)$ action describes what acceptor a does when it receives a phase $2a$ message m, which is sent by the leader of ballot $m.bal$ asking acceptors to vote for $m.val$ in that ballot. Acceptor a acts on that request, voting for $m.val$ in ballot number $m.bal$, iff $m.bal \geq maxBal[a]$, which means that a has not participated in any ballot numbered greater than $m.bal$. Thus, this enabling condition of the $Phase2b(a)$ action together with the receipt of the phase $2a$ message m implies that the $VoteFor(a, m.bal, m.val)$ action of module Voting is enabled and can be executed. The $Phase2b(a)$ message updates $maxBal[a]$, $maxVBal[a]$, and $maxVVal[a]$ so their values mean what they were claimed to mean in the comments preceding the variable declarations.

$$
\begin{array}{l}
Phase2b(a) \triangleq \\
\quad \exists m \in msgs : \\
\qquad \land m.type = \text{"2a"} \\
\qquad \land m.bal \geq maxBal[a] \\
\qquad \land maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land maxVBal' = [maxVBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land maxVVal' = [maxVVal \text{ EXCEPT } ![a] = m.val]
\end{array}
$$

$$\land\ Send([type \mapsto \text{"2b"},\ acc \mapsto a,\ bal \mapsto m.bal,\ val \mapsto m.val])$$

The definitions of *Next* and *Spec* are what we expect them to be.

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

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

This current module is distributed with two models, *TinyModel* and *SmallModel*. *SmallModel* is the same as the model by that name for the Voting specification. *TinyModel* is the same except it defines *Ballot* to contain only two elements. Run *TLC* on these models. You should find that it takes a couple of seconds to run *TinyModel* and two or three minutes to run *SmallModel*.

Next, try the same thing you did with the Voting algorithm: Modify the models so the assumption that any pair of quorums has an element in common is no longer true. (Again, it's best to modify clones of the models.) This time, running *TLC* will not find an error. The correctness of theorems Invariance and Implementation does not depend on that assumption. The *Paxos* consensus algorithm still correctly implements the Voting algorithm; but the Voting algorithm is incorrect if the assumption does not hold.

Now go back to the original *SmallModel*, in which the quorum assumption holds. The sets *Acceptor* and *Value* are symmetry sets for the spec. (See the "Model *Values* and Symmetry" help page to find out what that means.) Try editing the values substituted for *Acceptor* and/or *Value* by selecting the "Symmetry set" option and comparing the number of reachable states *TLC* found and the time it took. (Remember to use cloned models.)

When you have other things to do while *TLC* is running, try increasing the size of the model a very little bit at a time and see how the running time increases. You'll find that it increases exponentially with the numbers of acceptors, values, and ballots.

Fortunately, exhaustively checking a small model is very effective at finding errors. Since the *Paxos* consensus algorithm has been proved correct, and that proof has been read by many people, I'm sure that the basic algorithm is correct. Checking this spec on *SmallModel* makes me quite confident that there are no "coding errors" in this TLA+ specification of the algorithm.

For checking safety properties, *TLC* can obtain close to linear speedup using dozens of processors. After designing a new distributed algorithm, you will have plenty of time to run *TLC* while the algorithm is being implemented and the implementation tested. Use that time to run it for as long as you can on the largest $machine(s)$ that you can. Testing the implementation is unlikely to find subtle errors in the algorithm.