```
──────────────── MODULE MultiPaxos ────────────────
```

An abstract specification of the *MultiPaxos* algorithm. We do not model the network nor leaders explicitely. Instead, we keep the history of all votes cast and use this history to describe how new votes are cast. Note that, in some way, receiving a message corresponds to reading a past state of the sender. We produce the effect of having the leader by requiring that not two different values can be voted for in the same ballot.

This specification is inspired from the abstract specification of Generalized *Paxos* presented in the Generalized *Paxos* paper by *Lamport*.

```
EXTENDS MultiConsensus
```

The variable ballot maps an acceptor to its current ballot.

Given an acceptor a, an instance i, and a ballot b, $vote[a][i][b]$ records the vote that a casted in ballot b of instance i.

```
VARIABLES
    ballot, vote, propCmds

Init ≜
    ∧ ballot = [a ∈ Acceptors ↦ −1]
    ∧ vote = [a ∈ Acceptors ↦
                [i ∈ Instances ↦
                    [b ∈ Ballots ↦ None]]]
    ∧ propCmds = {}

TypeInv ≜
    ∧ ballot ∈ [Acceptors → {−1} ∪ Ballots]
    ∧ vote ∈ [Acceptors →
                [Instances →
                    [Ballots → {None} ∪ V]]]
    ∧ propCmds ∈ SUBSET V
```

Now starts the specification of the algorithm

A ballot is conservative when all acceptors which vote in the ballot vote for the same value. In *MultiPaxos*, the leader of a ballot ensures that the ballot is conservative.

```
Conservative(i, b) ≜
    ∀ a1, a2 ∈ Acceptors :
        LET v1 ≜ vote[a1][i][b]
            v2 ≜ vote[a2][i][b]
        IN  (v1 ≠ None ∧ v2 ≠ None) ⇒ v1 = v2

ConservativeVoteArray ≜
    ∀ i ∈ Instances : ∀ b ∈ Ballots :
        Conservative(i, b)
```

The maximal ballot smaller than *max* in which a has voted in instance i.

```
MaxVotedBallot(i, a, max) ≜
    Max({b ∈ Ballots : b ≤ max ∧ vote[a][i][b] ≠ None} ∪ {−1}, ≤ )

MaxVotedBallots(i, Q, max) ≜ {MaxVotedBallot(i, a, max) : a ∈ Q}
```

The vote casted in the maximal ballot smaller than max by an acceptor of the quorum Q.

```
HighestVote(i, max, Q) ≜
    IF ∃ a ∈ Q : MaxVotedBallot(i, a, max) ≠ −1
     THEN
        LET MaxVoter ≜ CHOOSE a ∈ Q :
                MaxVotedBallot(i, a, max) = Max(MaxVotedBallots(i, Q, max), ≤ )
        IN  vote[MaxVoter][i][MaxVotedBallot(i, MaxVoter, max)]
     ELSE
        None
```

Values that are safe to vote for in ballot b according to a quorum Q whose acceptors have all reached ballot b.

If there is an acceptor in Q that has voted in a ballot less than b, then the only safe value is the value voted for by an acceptor in Q in the highest ballot less than b.

Else, all values are safe.

In an implementation, the leader of a ballot b can compute $ProvedSafeAt(i, Q, b)$ when it receives $1b$ messages from the quorum Q.

```
ProvedSafeAt(i, Q, b) ≜
    IF HighestVote(i, b − 1, Q) ≠ None
     THEN {HighestVote(i, b − 1, Q)}
     ELSE V
```

The propose action:

```
Propose(v) ≜
    ∧ propCmds′ = propCmds ∪ {v}
    ∧ UNCHANGED ⟨ballot, vote⟩
```

The *JoinBallot* action: an acceptor can join a higher ballot at any time. In an implementation, the *JoinBallot* action is triggered by a $1a$ message from the leader of the new ballot.

```
JoinBallot(a, b) ≜
    ∧ ballot[a] < b
    ∧ ballot′ = [ballot EXCEPT ![a] = b]
    ∧ UNCHANGED ⟨vote, propCmds⟩
```

The *Vote* action: an acceptor casts a vote in instance i. This action is enabled when the acceptor has joined a ballot, has not voted in its current ballot, and can determine, by reading the last vote cast by each acceptor in a quorum, which value is safe to vote for. If multiple values are safe to vote for, we ensure that only one can be voted for by requiring that the ballot remain conservative.

In an implementation, the computation of safe values is done by the leader of the ballot when it receives $1b$ messages from a quorum of acceptors. The leader then picks a unique value among the safe values and suggests it to the acceptors.

$$
\begin{array}{l}
\mathit{Vote}(a,\ i) \ \triangleq \\
\quad \land\ \mathit{ballot}[a] \neq -1 \\
\quad \land\ \mathit{vote}[a][i][\mathit{ballot}[a]] = \mathit{None} \\
\quad \land\ \exists\, Q \in \mathit{Quorums} : \\
\qquad\quad \land\ \forall\, q \in Q : \mathit{ballot}[q] \geq \mathit{ballot}[a] \\
\qquad\quad \land\ \exists\, v \in \mathit{ProvedSafeAt}(i,\ Q,\ \mathit{ballot}[a]) \cap \mathit{propCmds} : \\
\qquad\qquad\quad \mathit{vote}' = [\mathit{vote}\ \text{EXCEPT}\ ![a] = \\
\qquad\qquad\qquad\quad [@\ \text{EXCEPT}\ ![i] = [@\ \text{EXCEPT}\ ![\mathit{ballot}[a]] = v]]] \\
\quad \land\ \text{UNCHANGED}\ \langle \mathit{ballot},\ \mathit{propCmds} \rangle \\
\quad \land\ \mathit{Conservative}(i,\ \mathit{ballot}[a])'
\end{array}
$$

$$
\begin{array}{l}
\mathit{Next} \ \triangleq \\
\quad \lor\ \exists\, v \in V : \mathit{Propose}(v) \\
\quad \lor\ \exists\, a \in \mathit{Acceptors} : \exists\, b \in \mathit{Ballots} : \mathit{JoinBallot}(a,\ b) \\
\quad \lor\ \exists\, a \in \mathit{Acceptors} : \exists\, i \in \mathit{Instances} : \mathit{Vote}(a,\ i)
\end{array}
$$

$$
\mathit{Spec} \ \triangleq\ \mathit{Init} \land \Box[\mathit{Next}]_{\langle \mathit{ballot},\ \mathit{vote},\ \mathit{propCmds} \rangle}
$$

Some properties and invariants that help understanding the algo and would probably be needed in a proof.

The maximal ballot in which an acceptor a voted is always less than or equal to its current ballot.

$$
\begin{array}{l}
\mathit{WellFormed} \ \triangleq\ \forall\, a \in \mathit{Acceptors} : \forall\, i \in \mathit{Instances} : \forall\, b \in \mathit{Ballots} : \\
\quad b > \mathit{ballot}[a] \Rightarrow \mathit{vote}[a][i][b] = \mathit{None}
\end{array}
$$

$$
\text{THEOREM}\ \mathit{Spec} \Rightarrow \Box\, \mathit{WellFormed}
$$

$$
\begin{array}{l}
\mathit{ChosenAt}(i,\ b,\ v) \ \triangleq \\
\quad \exists\, Q \in \mathit{Quorums} : \forall\, a \in Q : \mathit{vote}[a][i][b] = v
\end{array}
$$

$$
\begin{array}{l}
\mathit{Chosen}(i,\ v) \ \triangleq \\
\quad \exists\, b \in \mathit{Ballots} : \mathit{ChosenAt}(i,\ b,\ v)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Choosable}(v,\ i,\ b) \ \triangleq \\
\quad \exists\, Q \in \mathit{Quorums} : \forall\, a \in Q : \mathit{ballot}[a] > b \Rightarrow \mathit{vote}[a][i][b] = v
\end{array}
$$

$$
\begin{array}{l}
\mathit{SafeAt}(v,\ i,\ b) \ \triangleq \\
\quad \forall\, b2 \in \mathit{Ballots} : \forall\, v2 \in V : \\
\qquad (b2 < b \land \mathit{Choosable}(v2,\ i,\ b2)) \\
\qquad\ \Rightarrow v = v2
\end{array}
$$

$$
\begin{array}{l}
\mathit{SafeInstanceVoteArray}(i) \ \triangleq\ \forall\, b \in \mathit{Ballots} : \forall\, a \in \mathit{Acceptors} : \\
\quad \text{LET}\ v \ \triangleq\ \mathit{vote}[a][i][b] \\
\quad \text{IN}\quad v \neq \mathit{None} \Rightarrow \mathit{SafeAt}(v,\ i,\ b)
\end{array}
$$

$SafeVoteArray \triangleq \forall\, i \in Instances : SafeInstanceVoteArray(i)$

$\text{THEOREM } Spec \Rightarrow \Box SafeVoteArray$

If the vote array is well formed and the vote array is safe, then for each instance only a unique value can be chosen.

$\text{THEOREM } TypeInv \land WellFormed \land SafeVoteArray \Rightarrow \forall\, i \in Instances :$
$\quad \forall\, v1,\ v2 \in V : Chosen(i,\ v1) \land Chosen(i,\ v2) \Rightarrow v1 = v2$

In a well-formed, safe, and conservative vote array, all values that are proved safe are safe.

$\text{THEOREM } TypeInv \land WellFormed \land SafeVoteArray \land ConservativeVoteArray$
$\quad \Rightarrow \quad \forall\, v \in V : \forall\, i \in Instances :$
$\qquad\qquad \forall\, Q \in Quorums : \forall\, b \in Ballots :$
$\qquad\qquad\quad \land\ \forall\, a \in Q : ballot[a] \geq b$
$\qquad\qquad\quad \land\ v \in ProvedSafeAt(i,\ Q,\ b)$
$\qquad\qquad\quad \Rightarrow SafeAt(v,\ i,\ b)$

$Correctness \triangleq$
$\quad \forall\, i \in Instances : \forall\, v1,\ v2 \in V :$
$\qquad Chosen(i,\ v1) \land Chosen(i,\ v2) \Rightarrow v1 = v2$

$\text{THEOREM } Spec \Rightarrow \Box Correctness$

* Modification History
* Last modified *Thu Jan* 21 01:21:39 *EST* 2016 by *nano*
* Created *Mon Nov* 02 09:08:37 *EST* 2015 by *nano*