
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*

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

Ballot $\triangleq \text{Nat}$

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

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.

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

VARIABLE *maxBal*,
maxVBal, $\langle \text{maxVBal}[a], \text{maxVal}[a] \rangle$ is the vote with the largest
maxVal, ballot number cast by a; it equals $\langle -1, \text{None} \rangle$ 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 messages must be received (and hence either not lost or successfully retransmitted if lost) to guarantee progress.

vars $\triangleq \langle \text{maxBal}, \text{maxVBal}, \text{maxVal}, \text{msgs} \rangle$

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 $\triangleq \wedge \text{maxBal} \in [\text{Acceptor} \rightarrow \text{Ballot} \cup \{ -1 \}]$

$$\begin{aligned}
& \wedge \max VBal \in [Acceptor \rightarrow Ballot \cup \{-1\}] \\
& \wedge \max Val \in [Acceptor \rightarrow Value \cup \{None\}] \\
& \wedge msgs \subseteq Message
\end{aligned}$$

$$\begin{aligned}
Init & \triangleq \wedge \max Bal = [a \in Acceptor \mapsto -1] \\
& \wedge \max VBal = [a \in Acceptor \mapsto -1] \\
& \wedge \max Val = [a \in Acceptor \mapsto None] \\
& \wedge msgs = \{\}
\end{aligned}$$

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 .

$$\begin{aligned}
Phase1a(b) & \triangleq \wedge Send([type \mapsto "1a", bal \mapsto b]) \\
& \wedge UNCHANGED \langle \max Bal, \max VBal, \max Val \rangle
\end{aligned}$$

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

$$\begin{aligned}
Phase1b(a) & \triangleq \wedge \exists m \in msgs : \\
& \quad \wedge m.type = "1a" \\
& \quad \wedge m.bal > \max Bal[a] \\
& \quad \wedge \max Bal' = [\max Bal \text{ EXCEPT } ![a] = m.bal] \\
& \quad \wedge Send([type \mapsto "1b", acc \mapsto a, bal \mapsto m.bal, \\
& \quad \quad \quad m.bal \mapsto \max VBal[a], m.val \mapsto \max Val[a]]) \\
& \quad \wedge UNCHANGED \langle \max VBal, \max Val \rangle
\end{aligned}$$

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 $\max Bal[a]$ greater than b (thereby promising not to vote in ballot b).

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

$$\begin{aligned}
& \wedge m.acc \in Q \\
& \wedge m.bal = b\} \\
Q1bv & \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
IN & \wedge \forall a \in Q : \exists m \in Q1b : m.acc = a \\
& \wedge \vee Q1bv = \{\} \\
& \vee \exists m \in Q1bv : \\
& \quad \wedge m.mval = v \\
& \quad \wedge \forall mm \in Q1bv : m.mbal \geq mm.mbal \\
& \wedge Send([type \mapsto "2a", bal \mapsto b, val \mapsto v]) \\
& \wedge UNCHANGED \langle maxBal, maxVbal, maxVal \rangle
\end{aligned}$$

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 $maxVbal[a]$ 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

$$\begin{aligned}
Phase2b(a) & \triangleq \exists m \in msgs : \wedge m.type = "2a" \\
& \wedge m.bal \geq maxBal[a] \\
& \wedge maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
& \wedge maxVbal' = [maxVbal \text{ EXCEPT } ![a] = m.bal] \\
& \wedge maxVal' = [maxVal \text{ EXCEPT } ![a] = m.val] \\
& \wedge Send([type \mapsto "2b", acc \mapsto a, \\
& \quad bal \mapsto m.bal, val \mapsto m.val])
\end{aligned}$$

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.

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

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

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.

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

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*.

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

THEOREM $Spec \Rightarrow V!Spec$

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

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