

MODULE *PaxosCommit*

This specification is discussed in “*Paxos Commit*”, Lecture 6 of the TLA+ Video Course.

This module specifies the *Paxos Commit* algorithm. We specify only safety properties, not liveness properties. We simplify the specification in the following ways.

- As in the specification of module *TwoPhase*, and for the same reasons, we let the variable *msgs* be the set of all messages that have ever been sent. If a message is sent to a set of recipients, only one copy of the message appears in *msgs*.
- We do not explicitly model the receipt of messages. If an operation can be performed when a process has received a certain set of messages, then the operation is represented by an action that is enabled when those messages are in the set *msgs* of sent messages. (We are specifying only safety properties, which assert what events can occur, and the operation can occur if the messages that enable it have been sent.)
- We do not model leader selection. We define actions that the current leader may perform, but do not specify who performs them.

As in the specification of Two-Phase commit in module *TwoPhase*, we have *RM*s spontaneously issue Prepared messages and we ignore *Prepare* messages.

EXTENDS *Integers*

$Maximum(S) \triangleq$

If *S* is a set of numbers, then this define $Maximum(S)$ to be the maximum of those numbers, or -1 if *S* is empty.

IF $S = \{\}$ THEN -1
 ELSE CHOOSE $n \in S : \forall m \in S : n \geq m$

CONSTANT <i>RM</i> ,	The set of resource managers.
<i>Acceptor</i> ,	The set of acceptors.
<i>Majority</i> ,	The set of majorities of acceptors
<i>Ballot</i>	The set of ballot numbers

We assume the following properties of the declared constants.

ASSUME

$\wedge Ballot \subseteq Nat$
 $\wedge 0 \in Ballot$
 $\wedge Majority \subseteq SUBSET Acceptor$
 $\wedge \forall MS1, MS2 \in Majority : MS1 \cap MS2 \neq \{\}$

All we assume about the set *Majority* of majorities is that any two majorities have non-empty intersection.

$Messages \triangleq$

The set of all possible messages. There are messages of type “Commit” and “Abort” to announce the decision, as well as messages for each phase of each instance of *ins* of the *Paxos* consensus algorithm. The *acc* field indicates the sender of a message from an acceptor to the leader; messages from a leader are broadcast to all acceptors.

$[type : \{\text{“phase1a”}\}, ins : RM, bal : Ballot \setminus \{0\}]$
 \cup
 $[type : \{\text{“phase1b”}\}, ins : RM, mbal : Ballot, bal : Ballot \cup \{-1\},$

$ \begin{aligned} & val : \{ \text{"prepared"}, \text{"aborted"}, \text{"none"} \}, acc : \text{Acceptor}] \\ & \cup \\ & [type : \{ \text{"phase2a"} \}, ins : RM, bal : Ballot, val : \{ \text{"prepared"}, \text{"aborted"} \}] \\ & \cup \\ & [type : \{ \text{"phase2b"} \}, acc : \text{Acceptor}, ins : RM, bal : Ballot, \\ & \quad val : \{ \text{"prepared"}, \text{"aborted"} \}] \\ & \cup \\ & [type : \{ \text{"Commit"}, \text{"Abort"} \}] \end{aligned} $	
<hr/>	
VARIABLES	
$rmState,$	$rmState[r]$ is the state of resource manager r .
$aState,$	$aState[ins][ac]$ is the state of acceptor ac for instance
	ins of the <i>Paxos</i> algorithm.
$msgs$	The set of all messages ever sent.
$PCTypeOK \triangleq$	
	The type-correctness invariant. Each acceptor maintains the values $mbal$, bal , and val for each instance of the <i>Paxos</i> consensus algorithm.
	$ \begin{aligned} & \wedge rmState \in [RM \rightarrow \{ \text{"working"}, \text{"prepared"}, \text{"committed"}, \text{"aborted"} \}] \\ & \wedge aState \in [RM \rightarrow [Acceptor \rightarrow [mbal : Ballot, \\ & \quad \quad \quad bal : Ballot \cup \{ -1 \}, \\ & \quad \quad \quad val : \{ \text{"prepared"}, \text{"aborted"}, \text{"none"} \}]]] \end{aligned} $
	$\wedge msgs \subseteq Messages$
$PCInit \triangleq$	The initial predicate.
	$ \begin{aligned} & \wedge rmState = [r \in RM \mapsto \text{"working"}] \\ & \wedge aState = [r \in RM \mapsto \\ & \quad [ac \in Acceptor \\ & \quad \quad \mapsto [mbal \mapsto 0, bal \mapsto -1, val \mapsto \text{"none"}]]] \end{aligned} $
	$\wedge msgs = \{ \}$
<hr/>	
THE ACTIONS	
$Send(m) \triangleq$	$msgs' = msgs \cup \{m\}$
	An action expression that describes the sending of message m .
<hr/>	
RM ACTIONS	
$RMPPrepare(r) \triangleq$	
	Resource manager r prepares by sending a phase 2a message for ballot number 0 with value "prepared".
	$ \begin{aligned} & \wedge rmState[r] = \text{"working"} \\ & \wedge rmState' = [rmState \text{ EXCEPT } ![r] = \text{"prepared"}] \\ & \wedge Send([type \mapsto \text{"phase2a"}, ins \mapsto r, bal \mapsto 0, val \mapsto \text{"prepared"}]) \\ & \wedge \text{UNCHANGED } aState \end{aligned} $
$RMChooseToAbort(r) \triangleq$	

Resource manager r spontaneously decides to abort. It may (but need not) send a phase $2a$ message for ballot number 0 with value “aborted”.

$\wedge rmState[r] = \text{“working”}$
 $\wedge rmState' = [rmState \text{ EXCEPT } ![r] = \text{“aborted”}]$
 $\wedge Send([type \mapsto \text{“phase2a”}, ins \mapsto r, bal \mapsto 0, val \mapsto \text{“aborted”}])$
 $\wedge \text{UNCHANGED } aState$

$RMRcvCommitMsg(r) \triangleq$

Resource manager r is told by the leader to commit. When this action is enabled, $rmState[r]$ must equal either “prepared” or “committed”. In the latter case, the action leaves the state unchanged (it is a “stuttering step”).

$\wedge [type \mapsto \text{“Commit”}] \in msgs$
 $\wedge rmState' = [rmState \text{ EXCEPT } ![r] = \text{“committed”}]$
 $\wedge \text{UNCHANGED } \langle aState, msgs \rangle$

$RMRcvAbortMsg(r) \triangleq$

Resource manager r is told by the leader to abort. It could be in any state except “committed”.

$\wedge [type \mapsto \text{“Abort”}] \in msgs$
 $\wedge rmState' = [rmState \text{ EXCEPT } ![r] = \text{“aborted”}]$
 $\wedge \text{UNCHANGED } \langle aState, msgs \rangle$

LEADER ACTIONS

The following actions are performed by any process that believes itself to be the current leader. Since leader selection is not assumed to be reliable, multiple processes could simultaneously consider themselves to be the leader.

$Phase1a(bal, r) \triangleq$

If the leader times out without learning that a decision has been reached on resource manager r ’s prepare/abort decision, it can perform this action to initiate a new ballot bal . (Sending duplicate phase $1a$ messages is harmless.)

$\wedge Send([type \mapsto \text{“phase1a”}, ins \mapsto r, bal \mapsto bal])$
 $\wedge \text{UNCHANGED } \langle rmState, aState \rangle$

$Phase2a(bal, r) \triangleq$

The action in which a leader sends a phase $2a$ message with ballot $bal > 0$ in instance r , if it has received phase $1b$ messages for ballot number bal from a majority of acceptors. If the leader received a phase $1b$ message from some acceptor that had sent a phase $2b$ message for this instance, then $maxbal \geq 0$ and the value val the leader sends is determined by the phase $1b$ messages. (If $val = \text{“prepared”}$, then r must have prepared.) Otherwise, $maxbal = -1$ and the leader sends the value “aborted”.

The first conjunct asserts that the action is disabled if any commit leader has already sent a phase $2a$ message with ballot number bal . In practice, this is implemented by having ballot numbers partitioned among potential leaders, and having a leader record in stable storage the largest ballot number for which it sent a phase $2a$ message.

$\wedge \neg \exists m \in msgs : \wedge m.type = \text{“phase2a”}$
 $\wedge m.bal = bal$
 $\wedge m.ins = r$
 $\wedge \exists MS \in Majority :$

$\text{LET } mset \triangleq \{m \in msgs : \wedge m.type = \text{"phase1b"} \\
\wedge m.ins = r \\
\wedge m.mbal = bal \\
\wedge m.acc \in MS\}$
 $maxbal \triangleq \text{Maximum}(\{m.bal : m \in mset\})$
 $val \triangleq \text{IF } maxbal = -1$
 $\quad \text{THEN "aborted"}$
 $\quad \text{ELSE } (\text{CHOOSE } m \in mset : m.bal = maxbal).val$
 $\text{IN } \wedge \forall ac \in MS : \exists m \in mset : m.acc = ac$
 $\quad \wedge \text{Send}([type \mapsto \text{"phase2a"}, ins \mapsto r, bal \mapsto bal, val \mapsto val])$
 $\wedge \text{UNCHANGED } \langle rmState, aState \rangle$

$PCDecide \triangleq$

A leader can decide that *Paxos Commit* has reached a result and send a message announcing the result if it has received the necessary phase 2b messages.

$\wedge \text{LET } Decided(r, v) \triangleq$
 $\quad \text{True iff instance } r \text{ of the Paxos consensus algorithm has chosen the value } v.$
 $\quad \exists b \in \text{Ballot}, MS \in \text{Majority} :$
 $\quad \quad \forall ac \in MS : [type \mapsto \text{"phase2b"}, ins \mapsto r,$
 $\quad \quad \quad bal \mapsto b, val \mapsto v, acc \mapsto ac] \in msgs$
 $\text{IN } \vee \wedge \forall r \in RM : Decided(r, \text{"prepared"})$
 $\quad \wedge \text{Send}([type \mapsto \text{"Commit"}])$
 $\vee \wedge \exists r \in RM : Decided(r, \text{"aborted"})$
 $\quad \wedge \text{Send}([type \mapsto \text{"Abort"}])$
 $\wedge \text{UNCHANGED } \langle rmState, aState \rangle$

ACCEPTOR ACTIONS

$Phase1b(acc) \triangleq$

$\exists m \in msgs :$
 $\quad \wedge m.type = \text{"phase1a"}$
 $\quad \wedge aState[m.ins][acc].mbal < m.bal$
 $\quad \wedge aState' = [aState \text{ EXCEPT } ![m.ins][acc].mbal = m.bal]$
 $\quad \wedge \text{Send}([type \mapsto \text{"phase1b"},$
 $\quad \quad ins \mapsto m.ins,$
 $\quad \quad mbal \mapsto m.bal,$
 $\quad \quad bal \mapsto aState[m.ins][acc].bal,$
 $\quad \quad val \mapsto aState[m.ins][acc].val,$
 $\quad \quad acc \mapsto acc])$
 $\wedge \text{UNCHANGED } rmState$

$Phase2b(acc) \triangleq$

$\wedge \exists m \in msgs :$
 $\quad \wedge m.type = \text{"phase2a"}$
 $\quad \wedge aState[m.ins][acc].mbal \leq m.bal$
 $\quad \wedge aState' = [aState \text{ EXCEPT } ![m.ins][acc].mbal = m.bal,$

