MODULE *PaxosCommit*

This module specifies the *Paxos Commit* algorithm. We specify only safety properties, not liveness properties. We simplify the specification in the following ways. \begin{*itemize*} \item 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 $*msgs*$.

\item 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.)

\item We do not model leader selection. We define actions that the current leader may perform, but do not specify who performs them. \end{*itemize*}

As in the specification of Two-Phase commit in module $*TwoPhase*$, we have *RMs* 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.

$$
\begin{array}{l}
\text{LET } Max[T \in \text{SUBSET } S] \triangleq \\
\quad\quad \text{IF } T = \{\} \text{ THEN } -1 \\
\quad\quad\quad\quad \text{ELSE LET } n \triangleq \text{CHOOSE } n \in T : \text{TRUE} \\
\quad\quad\quad\quad\quad\quad\quad\quad rmax \triangleq Max[T \setminus \{n\}] \\
\quad\quad\quad\quad\quad\quad \text{IN IF } n \geq rmax \text{ THEN } n \text{ ELSE } rmax \\
\text{IN } Max[S]
\end{array}
$$

CONSTANT
- RM, — The set of resource managers.
- $Acceptor$, — The set of acceptors.
- $Majority$, — The set of majorities of acceptors
- $Ballot$ — The set of ballot numbers

ASSUME — We assume these properties of the declared constants.

$$
\begin{array}{l}
\land\ Ballot \subseteq Nat \\
\land\ 0 \in Ballot \\
\land\ Majority \subseteq \text{SUBSET } Acceptor \\
\land\ \forall MS1, MS2 \in Majority : MS1 \cap MS2 \neq \{\}
\end{array}
$$

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

$Message \triangleq$

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.

$$[type : \{\text{“phase1a”}\}, ins : RM, bal : Ballot \setminus \{0\}]$$

$\cup$

$[type : \{\text{"phase1b"}\},\ ins : RM,\ mbal : Ballot,\ bal : Ballot \cup \{-1\},$
$\ val : \{\text{"prepared"}, \text{"aborted"}, \text{"none"}\},\ acc : Acceptor]$
$\cup$
$[type : \{\text{"phase2a"}\},\ ins : RM,\ bal : Ballot,\ val : \{\text{"prepared"}, \text{"aborted"}\}]$
$\cup$
$[type : \{\text{"phase2b"}\},\ acc : Acceptor,\ ins : RM,\ bal : Ballot,$
$\ val : \{\text{"prepared"}, \text{"aborted"}\}]$
$\cup$
$[type : \{\text{"Commit"}, \text{"Abort"}\}]$

VARIABLES

$rmState,$ \$$rmState[rm]$\$ is the state of resource manager \$$rm$\$.

$aState,$ \$$aState[ins][ac]$\$ is the state of acceptor \$$ac$\$ for instance \$$ins$\$ of the *Paxos* 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 *Paxos* consensus algorithm.

$$\begin{aligned}
&\land\ rmState \in [RM \rightarrow \{\text{"working"}, \text{"prepared"}, \text{"committed"}, \text{"aborted"}\}]\\
&\land\ aState \in [RM \rightarrow [Acceptor \rightarrow [mbal : Ballot,\\
&\qquad\qquad bal : Ballot \cup \{-1\},\\
&\qquad\qquad val : \{\text{"prepared"}, \text{"aborted"}, \text{"none"}\}]]]\\
&\land\ msgs \in \text{SUBSET } Message
\end{aligned}$$

$PCInit \triangleq$ The initial predicate.

$$\begin{aligned}
&\land\ rmState = [rm \in RM \mapsto \text{"working"}]\\
&\land\ aState = [ins \in RM \mapsto\\
&\qquad [ac \in Acceptor\\
&\qquad\quad \mapsto [mbal \mapsto 0,\ bal \mapsto -1,\ val \mapsto \text{"none"}]]]\\
&\land\ msgs = \{\}
\end{aligned}$$

\mbox{}\hfill{\large \textbf{The Actions}}\hfill\mbox{}

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

An action expression that describes the sending of message \$$m$\$.

{\large \textbf{RM Actions}}

$RMPrepare(rm) \triangleq$

Resource manager \$$rm$\$ prepares by sending a phase $2a$ message for ballot number 0 with value \$"prepared".\$

$$\begin{aligned}
&\land\ rmState[rm] = \text{"working"}\\
&\land\ rmState' = [rmState \text{ EXCEPT } ![rm] = \text{"prepared"}]\\
&\land\ Send([type \mapsto \text{"phase2a"},\ ins \mapsto rm,\ bal \mapsto 0,\ val \mapsto \text{"prepared"}])\\
&\land\ \text{UNCHANGED } aState
\end{aligned}$$

$RMChooseToAbort(rm) \triangleq$

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

$\land\ rmState[rm] = \text{"working"}$
$\land\ rmState' = [rmState\ \text{EXCEPT}\ ![rm] = \text{"aborted"}]$
$\land\ Send([type \mapsto \text{"phase2a"},\ ins \mapsto rm,\ bal \mapsto 0,\ val \mapsto \text{"aborted"}])$
$\land\ \text{UNCHANGED}\ aState$

$RMRcvCommitMsg(rm) \triangleq$

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

$\land\ [type \mapsto \text{"Commit"}] \in msgs$
$\land\ rmState' = [rmState\ \text{EXCEPT}\ ![rm] = \text{"committed"}]$
$\land\ \text{UNCHANGED}\ \langle aState,\ msgs\rangle$

$RMRcvAbortMsg(rm) \triangleq$

Resource manager \$$rm$\$ is told by the leader to abort. It could be in any state except \$"committed"\$.

$\land\ [type \mapsto \text{"Abort"}] \in msgs$
$\land\ rmState' = [rmState\ \text{EXCEPT}\ ![rm] = \text{"aborted"}]$
$\land\ \text{UNCHANGED}\ \langle aState,\ msgs\rangle$

{\large \textbf{Leader Actions}} \vspace{.25\baselineskip}

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.

$Phase1a(bal,\ rm) \triangleq$

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

$\land\ Send([type \mapsto \text{"phase1a"},\ ins \mapsto rm,\ bal \mapsto bal])$
$\land\ \text{UNCHANGED}\ \langle rmState,\ aState\rangle$

$Phase2a(bal,\ rm) \triangleq$

The action in which a leader sends a phase $2a$ message with ballot \$$bal > 0$\$ in instance \$$rm$\$, 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 =$ "prepared"\$, then \$$rm$\$ must have prepared.) Otherwise,
\$$maxbal = -1$\$ and the leader sends the value \$"aborted"\$. \vspace{.5\baselineskip}

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.

$\land\ \neg\exists\, m \in msgs : \land\ m.type = \text{"phase2a"}$

```
                    ∧ m.bal = bal
                    ∧ m.ins = rm
  ∧ ∃ MS ∈ Majority :
       LET mset ≜ {m ∈ msgs : ∧ m.type  = "phase1b"
                              ∧ m.ins   = rm
                              ∧ m.mbal  = bal
                              ∧ m.acc   ∈ MS}
           maxbal ≜ Maximum({m.bal : m ∈ mset})
           val ≜ IF maxbal = −1
                   THEN "aborted"
                   ELSE (CHOOSE m ∈ mset : m.bal = maxbal).val
       IN  ∧ ∀ ac ∈ MS : ∃ m ∈ mset : m.acc = ac
           ∧ Send([type ↦ "phase2a", ins ↦ rm, bal ↦ bal, val ↦ val])
  ∧ UNCHANGED ⟨rmState, aState⟩

Decide ≜
```

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.

```
  ∧ LET Decided(rm, v) ≜
```

True iff instance \$*rm*\$ of the *Paxos* consensus algorithm has chosen the value \$*v*\$.

```
          ∃ b ∈ Ballot, MS ∈ Majority :
            ∀ ac ∈ MS : [type ↦ "phase2b", ins ↦ rm,
                         bal ↦ b, val ↦ v, acc ↦ ac] ∈ msgs
    IN  ∨ ∧ ∀ rm ∈ RM : Decided(rm, "prepared")
          ∧ Send([type ↦ "Commit"])
        ∨ ∧ ∃ rm ∈ RM : Decided(rm, "aborted")
          ∧ Send([type ↦ "Abort"])
  ∧ UNCHANGED ⟨rmState, aState⟩
```

{\large \textbf{Acceptor Actions}}

```
Phase1b(acc) ≜
  ∃ m ∈ msgs :
    ∧ m.type = "phase1a"
    ∧ aState[m.ins][acc].mbal < m.bal
    ∧ aState' = [aState EXCEPT ![m.ins][acc].mbal = m.bal]
    ∧ Send([type  ↦ "phase1b",
            ins   ↦ m.ins,
            mbal  ↦ m.bal,
            bal   ↦ aState[m.ins][acc].bal,
            val   ↦ aState[m.ins][acc].val,
            acc   ↦ acc])
    ∧ UNCHANGED rmState

Phase2b(acc) ≜
  ∧ ∃ m ∈ msgs :
```

$$
\begin{aligned}
&\quad\quad \land\ m.type = \text{"phase2a"} \\
&\quad\quad \land\ aState[m.ins][acc].mbal \leq m.bal \\
&\quad\quad \land\ aState' = [aState\ \text{EXCEPT}\ ![m.ins][acc].mbal = m.bal, \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad ![m.ins][acc].bal \ \ = m.bal, \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad ![m.ins][acc].val \ \ = m.val] \\
&\quad\quad \land\ Send([type \mapsto \text{"phase2b"},\ ins \mapsto m.ins,\ bal \mapsto m.bal, \\
&\quad\quad\quad\quad\quad val \mapsto m.val,\ acc \mapsto acc]) \\
&\quad \land\ \text{UNCHANGED}\ rmState
\end{aligned}
$$

$PCNext \triangleq$ The next-state action

$$
\begin{aligned}
&\lor\ \exists\, rm \in RM : \lor\ RMPrepare(rm) \\
&\quad\quad\quad\quad\quad\quad\ \lor\ RMChooseToAbort(rm) \\
&\quad\quad\quad\quad\quad\quad\ \lor\ RMRcvCommitMsg(rm) \\
&\quad\quad\quad\quad\quad\quad\ \lor\ RMRcvAbortMsg(rm) \\
&\lor\ \exists\, bal \in Ballot \setminus \{0\},\ rm \in RM : Phase1a(bal, rm) \lor Phase2a(bal, rm) \\
&\lor\ Decide \\
&\lor\ \exists\, acc \in Acceptor : Phase1b(acc) \lor Phase2b(acc)
\end{aligned}
$$

$PCSpec \triangleq PCInit \land \Box[PCNext]_{\langle rmState,\ aState,\ msgs \rangle}$

The complete spec of the *Paxos Commit* protocol.

THEOREM $PCSpec \Rightarrow PCTypeOK$

We now assert that the two-phase commit protocol implements the transaction commit protocol of module *TCommit*. The following statement defines \$*TC* ! *TCSpec*\$ to be the formula \$*TCSpec*\$ of module \$*TCommit*\$. (The TLA\$$^+$\$ \textsc{*instance*} statement must is used to rename the operators defined in module \$*TCommit*\$ to avoid possible name conflicts with operators in the current module having the same name.)

$TC \triangleq$ INSTANCE $TCommit$

THEOREM $PCSpec \Rightarrow TC!TCSpec$
