MODULE *JustInTimePaxos*

EXTENDS *Naturals*, *Reals*, *Sequences*, *FiniteSets*, *TLC*

The set of *Paxos* replicas
CONSTANT *Replicas*

The set of *Paxos* clients
CONSTANT *Clients*

The set of possible values
CONSTANT *Values*

An empty value
CONSTANT *Nil*

Request/response types
CONSTANTS
MClientRequest,
MClientReply,
MReconcileRequest,
MReconcileReply,
MRepairRequest,
MRepairReply,
MViewChange,
MViewChangeReply,
MStartView

Replica statuses
CONSTANTS
SNormal,
SRepair,
SViewChange

The set of all messages on the network
VARIABLE *messages*

The total number of messages sent
VARIABLE *messageCount*

The total number of steps executed
VARIABLE *stepCount*

$messageVars \triangleq \langle messages, messageCount, stepCount \rangle$

Local client state

Strictly increasing representation of synchronized time
VARIABLE $cTime$

The highest known view *ID* for a client
VARIABLE $cViewID$

Client request *IDs*
VARIABLE $cReqID$

A client response buffer
VARIABLE $cReps$

A set of all commits - used for model checking
VARIABLE $cCommits$

$clientVars \triangleq \langle cTime,\ cViewID,\ cReqID,\ cReps,\ cCommits \rangle$

Local replica state

The current status of a replica
VARIABLE $rStatus$

The current view *ID* for a replica
VARIABLE $rViewID$

A replica's commit *log*
VARIABLE $rLog$

A replica's sync index
VARIABLE $rSyncIndex$

The view *ID* for the *log*
VARIABLE $rLogViewID$

The set of view change replies
VARIABLE $rViewChangeReps$

$replicaVars \triangleq \langle rStatus,\ rViewID,\ rLog,\ rSyncIndex,\ rLogViewID,\ rViewChangeReps \rangle$

$vars \triangleq \langle messageVars,\ clientVars,\ replicaVars \rangle$

This section provides utilities for implementing the spec.

Creates a sequence from set 'S'
RECURSIVE $SeqFromSet(\_)$
$SeqFromSet(S) \triangleq$
IF $S = \{\}$ THEN
$\langle\rangle$
ELSE LET $x \triangleq$ CHOOSE $x \in S$: TRUE

```
        IN    ⟨x⟩ ∘ SeqFromSet(S \ {x})

RECURSIVE SetReduce(_, _, _)
SetReduce(Op(_, _), S, value) ≜
    IF S = {} THEN
        value
     ELSE
        LET s ≜ CHOOSE s ∈ S : TRUE
        IN    SetReduce(Op, S \ {s}, Op(s, value))

 Computes the greatest vlue in set 'S'
Max(S) ≜ CHOOSE x ∈ S : ∀ y ∈ S : x ≥ y

 Computes the sum of numbers in set 'S'
Sum(S) ≜ LET _op(a, b) ≜ a + b
             IN    SetReduce(_op, S, 0)

 The values of a sequence
Range(s) ≜ {s[i] : i ∈ DOMAIN s}
```

```
This section provides helpers for the protocol.

 A sorted sequence of replicas
replicas ≜ SeqFromSet(Replicas)

 The primary index for view 'v'
PrimaryIndex(v) ≜ (v%Len(replicas)) + (IF v ≥ Len(replicas) THEN 1 ELSE 0)

 The primary for view 'v'
Primary(v) ≜ replicas[PrimaryIndex(v)]

 Quorum is the quorum for a given view
Quorum(v) ≜
    LET
        quorumSize ≜ Len(replicas) ÷ 2       + 1
        index(i)   ≜ PrimaryIndex(v) + (i − 1)
        member(i)  ≜ IF index(i) > Len(replicas) THEN replicas[index(i)%Len(replicas)] ELSE replicas[inde
    IN
        {member(i) : i ∈ 1 .. quorumSize}

 A boolean indicating whether the given set is a quorum
IsQuorum(S) ≜ Cardinality(S) ∗ 2 ≥ Cardinality(Replicas)

 A boolean indicating whether the given set is a quorum that includes the given replica
IsLocalQuorum(r, S) ≜ IsQuorum(S) ∧ r ∈ S
```

This section models the network.

Send a set of messages

$$\begin{aligned}
&Sends(ms) \triangleq \\
&\quad \land\ messages' &&= messages \cup ms \\
&\quad \land\ messageCount' &&= messageCount + Cardinality(ms) \\
&\quad \land\ stepCount' &&= stepCount + 1
\end{aligned}$$

Send a message

$$Send(m) \triangleq Sends(\{m\})$$

Ack a message

$$\begin{aligned}
&Ack(m) \triangleq \\
&\quad \land\ messages' &&= messages \setminus \{m\} \\
&\quad \land\ messageCount' &&= messageCount + 1 \\
&\quad \land\ stepCount' &&= stepCount + 1
\end{aligned}$$

Ack a message and send a set of messages

$$\begin{aligned}
&AckAndSends(m,\ ms) \triangleq \\
&\quad \land\ messages' &&= (messages \cup ms) \setminus \{m\} \\
&\quad \land\ messageCount' &&= messageCount + Cardinality(ms) \\
&\quad \land\ stepCount' &&= stepCount + 1
\end{aligned}$$

Ack and send a message

$$AckAndSend(m,\ n) \triangleq AckAndSends(m,\ \{n\})$$

Reply to a message with a set of responses

$$Replies(req,\ reps) \triangleq AckAndSends(req,\ reps)$$

Reply to a message

$$Reply(req,\ resp) \triangleq AckAndSend(req,\ resp)$$

This section models client requests.

Client 'c' sends value 'v' to all replicas

$$\begin{aligned}
&ClientRequest(c,\ v) \triangleq \\
&\quad \land\ cTime' = cTime + 1 \\
&\quad \land\ cReqID' = [cReqID \text{ EXCEPT } ![c] = cReqID[c] + 1] \\
&\quad \land\ Sends(\{[src &&\mapsto c, \\
&\qquad\qquad dest &&\mapsto r, \\
&\qquad\qquad type &&\mapsto MClientRequest, \\
&\qquad\qquad viewID &&\mapsto cViewID[c], \\
&\qquad\qquad reqID &&\mapsto cReqID'[c], \\
&\qquad\qquad value &&\mapsto v, \\
&\qquad\qquad timestamp &&\mapsto cTime'] : r \in Quorum(cViewID[c])\}) \\
&\quad \land\ \text{UNCHANGED } \langle replicaVars,\ cViewID,\ cReps,\ cCommits\rangle
\end{aligned}$$

```
Client 'c' handles a response 'm' from replica 'r'
HandleClientReply(c, r, m) ≜
    If the reply view ID does not match the request view ID, update the client's view.
  ∧ ∨ ∧ m.viewID ≠ m.req.viewID
      ∧ ∨ ∧ cViewID[c] < m.viewID
            ∧ cViewID' = [cViewID EXCEPT ![c] = m.viewID]
        ∨ ∧ cViewID[c] ≥ m.viewID
          ∧ UNCHANGED ⟨cViewID⟩
      ∧ Ack(m)
      ∧ UNCHANGED ⟨cReps, cCommits⟩
    If the request and reply views match and the reply view matches the client's view,
    aggregate the replies for the associated client request.
    ∨ ∧ m.viewID = m.req.viewID
      ∧ m.viewID = cViewID[c]
      ∧ ∨ ∧ m.succeeded
            ∧ cReps' = [cReps EXCEPT ![c] =
                          (cReps[c] \ {n ∈ cReps[c] : ∧ n.src       = m.src
                                                      ∧ n.viewID    = cViewID[c]
                                                      ∧ n.req.reqID = m.req.reqID
                                                      ∧ ¬n.succeeded}) ∪ {m}]
        ∨ ∧ ¬m.succeeded
          ∧ ¬∃ n ∈ cReps[c] : ∧ n.src       = m.src
                              ∧ n.viewID    = cViewID[c]
                              ∧ n.req.reqID = m.req.reqID
                              ∧ n.succeeded
          ∧ cReps' = [cReps EXCEPT ![c] = cReps[c] ∪ {m}]
      ∧ LET reps       ≜ {n ∈ cReps'[c] : ∧ n.viewID    = cViewID[c]
                                          ∧ n.req.reqID = m.req.reqID}
            isQuorum   ≜ {n.src : n ∈ {n ∈ reps : n.succeeded}} = Quorum(cViewID[c])
            isCommitted ≜ ∧ ∀ n ∈ reps : n.succeeded
                          ∧ Cardinality({n.checksum : n ∈ reps}) = 1
            hasPrimary ≜ ∃ n ∈ reps : n.src = Primary(cViewID[c]) ∧ n.succeeded
        IN
          If a quorum of successful replies have been received and the checksums
          match, add the primary reply to commits.
          ∨ ∧ isQuorum
            ∧ isCommitted
            ∧ LET commit ≜ CHOOSE n ∈ reps : n.src = Primary(cViewID[c])
              IN  cCommits' = [cCommits EXCEPT ![c] = cCommits[c] ∪ {commit}]
            ∧ Ack(m)
          If some reply failed or was returned with an incorrect checksum,
          send a ReconcileRequest to the inconsistent node to force it to
          reconcile its log with the primary's log.
          ∨ ∧ ¬isCommitted
            ∧ ∨ ∧ hasPrimary
```

$\land$ LET $primaryRep \triangleq$ CHOOSE $n \in reps : \land n.src = Primary(cViewID[c])$
$\land n.succeeded$
$retryReps \triangleq \{n \in reps :$
$\land n.src \neq Primary(cViewID[c])$
$\land n.checksum \neq primaryRep.checksum\}$
IN $AckAndSends(m, \{[src \mapsto c,$
$dest \mapsto r,$
$type \mapsto MReconcileRequest,$
$viewID \mapsto cViewID[c],$
$reqID \mapsto m.req.reqID,$
$index \mapsto primaryRep.index] : n \in retryReps\})$

$\lor \land \neg hasPrimary$
$\land Ack(m)$
$\land$ UNCHANGED $\langle cCommits \rangle$

If a quorum has not yet been reached, wait for more replies.

$\lor \land \neg isQuorum$
$\land isCommitted$
$\land Ack(m)$
$\land$ UNCHANGED $\langle cCommits \rangle$

$\land$ UNCHANGED $\langle cViewID \rangle$

$\land$ UNCHANGED $\langle replicaVars, cTime, cReqID \rangle$

$HandleReconcileReply(c, r, m) \triangleq HandleClientReply(c, r, m)$

This section models the replica protocol.

Replica 'r' handles client 'c' request 'm'

$HandleClientRequest(r, c, m) \triangleq$

Client requests can only be handled while in the *SNormal* status.

$\land rStatus[r] = SNormal$

If the client's view matches the replica's view, process the client's request.

$\land \lor \land m.viewID = rViewID[r]$
$\land$ LET $lastTimestamp \triangleq Max(\{rLog[r][i].timestamp : i \in$ DOMAIN $rLog[r]\} \cup \{0\})$
IN

If the request timestamp is greater than the highest *log* timestamp, append the entry to the *log* and return a successful response with the appended entry index.

$\land \lor \land m.timestamp > lastTimestamp$
$\land rLog' = [rLog$ EXCEPT $![r] =$
$Append(rLog[r], [value \mapsto m.value,$
$timestamp \mapsto m.timestamp])]$
$\land Reply(m, [src \mapsto r,$
$dest \mapsto c,$
$req \mapsto m,$
$type \mapsto MClientReply,$

$viewID \mapsto rViewID[r],$
$index \mapsto Len(rLog'[r]),$
$checksum \mapsto rLog'[r],$
$value \mapsto m.value,$
$timestamp \mapsto m.timestamp,$
$succeeded \mapsto \text{TRUE}])$

If the request timestamp matches the highest *log* timestamp, treat the request as a duplicate. Return a successful response indicating the entry was appended.

$\lor \land m.timestamp = lastTimestamp$
$\land Reply(m, [src \mapsto r,$
$dest \mapsto c,$
$req \mapsto m,$
$type \mapsto MClientReply,$
$viewID \mapsto rViewID[r],$
$index \mapsto Len(rLog[r]),$
$checksum \mapsto rLog[r],$
$value \mapsto m.value,$
$timestamp \mapsto m.timestamp,$
$succeeded \mapsto \text{TRUE}])$
$\land \text{UNCHANGED } \langle rLog \rangle$

If the request timestamp is less than the highest *log* timestamp, reject the request.

$\lor \land m.timestamp < lastTimestamp$
$\land Reply(m, [src \mapsto r,$
$dest \mapsto c,$
$req \mapsto m,$
$type \mapsto MClientReply,$
$viewID \mapsto rViewID[r],$
$index \mapsto Len(rLog[r]),$
$checksum \mapsto rLog[r],$
$value \mapsto m.value,$
$timestamp \mapsto m.timestamp,$
$succeeded \mapsto \text{FALSE}])$
$\land \text{UNCHANGED } \langle rLog \rangle$

$\land \text{UNCHANGED } \langle rViewID, rStatus, rViewChangeReps \rangle$

If the client's view is greater than the replica's view, reject the client's request with the outdated view *ID* and enter the view change protocol.

$\lor \land m.viewID > rViewID[r]$
$\land rViewID' = [rViewID \text{ EXCEPT } ![r] = m.viewID]$
$\land rStatus' = [rStatus \text{ EXCEPT } ![r] = SViewChange]$
$\land rViewChangeReps' = [rViewChangeReps \text{ EXCEPT } ![r] = \{\}]$
$\land Replies(m, \{[src \mapsto r,$
$dest \mapsto c,$
$req \mapsto m,$

$$
\begin{array}{ll}
 & type \mapsto MClientReply, \\
 & viewID \mapsto rViewID[r], \\
 & succeeded \mapsto \text{FALSE}], \\
[& src \mapsto r, \\
 & dest \mapsto Primary(m.viewID), \\
 & type \mapsto MViewChangeReply, \\
 & viewID \mapsto m.viewID, \\
 & logViewID \mapsto rLogViewID[r], \\
 & log \mapsto rLog[r]]\})
\end{array}
$$

$\land$ UNCHANGED $\langle rLog \rangle$

If the client's view is less than the replica's view, reject the client's request with the updated view *ID* to force the client to retry.

$$
\begin{array}{ll}
\lor & \land\ m.viewID < rViewID[r] \\
 & \land\ Reply(m, [src \mapsto r, \\
 & \qquad dest \mapsto c, \\
 & \qquad req \mapsto m, \\
 & \qquad type \mapsto MClientReply, \\
 & \qquad viewID \mapsto rViewID[r], \\
 & \qquad succeeded \mapsto \text{FALSE}]) \\
 & \land\ \text{UNCHANGED } \langle rViewID,\ rStatus,\ rLog,\ rViewChangeReps \rangle \\
\land\ \text{UNCHANGED } & \langle clientVars,\ rLogViewID,\ rSyncIndex \rangle
\end{array}
$$

$$
\begin{array}{ll}
HandleReconcileRequest(r,\ c,\ m) \triangleq & \\
\land\ rStatus[r] = SNormal & \\
\land\ rViewID[r] = m.viewID & \\
\land\ \lor & \land\ rSyncIndex[r] \geq m.index \\
 & \land\ Reply(m, [src \mapsto r, \\
 & \qquad dest \mapsto c, \\
 & \qquad req \mapsto m, \\
 & \qquad type \mapsto MReconcileReply, \\
 & \qquad viewID \mapsto rViewID[r], \\
 & \qquad index \mapsto m.index, \\
 & \qquad checksum \mapsto [i \in 1\ ..\ m.index \mapsto rLog[r][i]], \\
 & \qquad value \mapsto rLog[r][m.index].value, \\
 & \qquad timestamp \mapsto rLog[r][m.index].timestamp, \\
 & \qquad succeeded \mapsto \text{TRUE}]) \\
 & \land\ \text{UNCHANGED } \langle rStatus \rangle \\
\lor & \land\ rSyncIndex[r] < m.index \\
 & \land\ Primary(rViewID[r]) \neq r \\
 & \land\ rStatus' = [rStatus \text{ EXCEPT } ![r] = SRepair] \\
 & \land\ AckAndSend(m, [src \mapsto r, \\
 & \qquad dest \mapsto Primary(rViewID[r]), \\
 & \qquad req \mapsto m, \\
 & \qquad type \mapsto MRepairRequest, \\
 & \qquad viewID \mapsto rViewID[r],
\end{array}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad index \;\mapsto m.index])\\
\quad \land \text{UNCHANGED } \langle clientVars, rViewID, rLog, rLogViewID, rSyncIndex, rViewChangeReps\rangle
\end{array}
$$

$$
\begin{array}{l}
HandleRepairRequest(r, s, m) \triangleq\\
\quad \land rStatus[r] = SNormal\\
\quad \land rViewID[r] = m.viewID\\
\quad \land Primary(rViewID[r]) = r\\
\quad \land Reply(m, [src \mapsto r,\\
\qquad\qquad dest \mapsto s,\\
\qquad\qquad req \mapsto m.req,\\
\qquad\qquad type \mapsto MRepairReply,\\
\qquad\qquad viewID \mapsto rViewID[r],\\
\qquad\qquad index \mapsto m.index,\\
\qquad\qquad log \mapsto [i \in 1 \,..\, m.index \mapsto rLog[r][i]]])\\
\quad \land \text{UNCHANGED } \langle clientVars, replicaVars\rangle
\end{array}
$$

$$
\begin{array}{l}
HandleRepairReply(r, s, m) \triangleq\\
\quad \land rStatus[r] = SRepair\\
\quad \land rViewID[r] = m.viewID\\
\quad \land rStatus' = [rStatus \text{ EXCEPT } ![r] = SNormal]\\
\quad \land rLog' = [rLog \text{ EXCEPT } ![r] = m.log \circ SubSeq(rLog[r], Len(m.log), Len(rLog[r]))]\\
\quad \land rSyncIndex' = [rSyncIndex \text{ EXCEPT } ![r] = Len(rLog'[r])]\\
\quad \land Reply(m, [src \mapsto r,\\
\qquad\qquad dest \mapsto m.req.src,\\
\qquad\qquad req \mapsto m.req,\\
\qquad\qquad type \mapsto MReconcileReply,\\
\qquad\qquad viewID \mapsto rViewID[r],\\
\qquad\qquad index \mapsto m.index,\\
\qquad\qquad checksum \mapsto m.log,\\
\qquad\qquad value \mapsto m.log[m.index].value,\\
\qquad\qquad timestamp \mapsto m.log[m.index].timestamp,\\
\qquad\qquad succeeded \mapsto \text{TRUE}])\\
\quad \land \text{UNCHANGED } \langle clientVars, rViewID, rLogViewID, rViewChangeReps\rangle
\end{array}
$$

Replica 'r' requests a view change

$$
\begin{array}{l}
ChangeView(r) \triangleq\\
\quad \land Sends(\{[src \mapsto r,\\
\qquad\qquad dest \mapsto d,\\
\qquad\qquad type \mapsto MViewChange,\\
\qquad\qquad viewID \mapsto rViewID[r] + 1] : d \in Replicas\})\\
\quad \land \text{UNCHANGED } \langle clientVars, replicaVars\rangle
\end{array}
$$

Replica 'r' handles replica 's' view change request 'm'

$$
\begin{array}{l}
HandleViewChange(r, s, m) \triangleq\\
\quad \land \lor \land rViewID[r] < m.viewID\\
\qquad\quad \land rViewID' = [rViewID \text{ EXCEPT } ![r] = m.viewID]
\end{array}
$$

$\land\ rStatus' = [rStatus \text{ EXCEPT } ![r] = SViewChange]$
$\land\ rViewChangeReps' = [rViewChangeReps \text{ EXCEPT } ![r] = \{\}]$
$\land\ Reply(m, [src \mapsto r,$
$dest \mapsto Primary(m.viewID),$
$type \mapsto MViewChangeReply,$
$viewID \mapsto m.viewID,$
$logViewID \mapsto rLogViewID[r],$
$log \mapsto rLog[r]])$
$\lor\ \land\ rViewID[r] \geq m.viewID$
$\land\ Ack(m)$
$\land\ \text{UNCHANGED } \langle rViewID, rStatus, rViewChangeReps\rangle$
$\land\ \text{UNCHANGED } \langle clientVars, rLog, rLogViewID, rSyncIndex\rangle$

Replica 'r' handles replica 's' view change reply 'm'
$HandleViewChangeReply(r, s, m) \triangleq$
The view change protocol is run by the primary for the view.
$\land\ Primary(m.viewID) = r$
$\land\ rViewID[r] = m.viewID$
$\land\ rStatus[r] = SViewChange$
$\land\ rViewChangeReps' = [rViewChangeReps \text{ EXCEPT } ![r] = rViewChangeReps[r] \cup \{m\}]$
$\land\ \text{LET } viewChanges \triangleq \{v \in rViewChangeReps'[r] : v.viewID = rViewID[r]\}$
IN
In order to ensure the new view is initialized with the latest view,
a quorum of view change replies must be received to guarantee the last
activated view is present in the set of replies.
If view change replies have been received from a majority of the replicas,
initialize the view using the *log* from the highest activated view.
$\lor\ \land\ IsLocalQuorum(r, \{v.src : v \in viewChanges\})$
$\land\ \text{LET } latestViewID \triangleq Max(\{v.logViewID : v \in viewChanges\})$
$latestChange \triangleq \text{CHOOSE } v \in viewChanges :$
$\land\ v.logViewID = latestViewID$
$\land\ v.src \in Quorum(latestViewID)$
IN $AckAndSends(m, \{[src \mapsto r,$
$dest \mapsto d,$
$type \mapsto MStartView,$
$viewID \mapsto rViewID[r],$
$log \mapsto latestChange.log] : d \in Replicas\})$
If view change replies have not yet been received from a quorum, record
the view change reply and discard the message.
$\lor\ \land\ \neg IsLocalQuorum(r, \{v.src : v \in viewChanges\})$
$\land\ Ack(m)$
$\land\ \text{UNCHANGED } \langle clientVars, rStatus, rViewID, rLog, rLogViewID, rSyncIndex\rangle$

Replica 'r' handles replica 's' start view request 'm'
$HandleStartView(r, s, m) \triangleq$

```
    To activate a view, the replica must either not know of the view or already
    be participating in the view change protocol for the view.
  ∧ ∨ rViewID[r] < m.viewID
    ∨ ∧ rViewID[r] = m.viewID
      ∧ rStatus[r]  = SViewChange
    If the replica is part of the quorum for the activated view, update the log
    and record the activated view for use in the view change protocol.
  ∧ ∨ ∧ r ∈ Quorum(m.viewID)
      ∧ rLog'          = [rLog          EXCEPT ![r]  = m.log]
      ∧ rLogViewID'    = [rLogViewID    EXCEPT ![r] = m.viewID]
      ∧ rSyncIndex'    = [rSyncIndex    EXCEPT ![r]  = Len(m.log)]
    ∨ ∧ r ∉ Quorum(m.viewID)
      ∧ UNCHANGED ⟨rLog, rLogViewID, rSyncIndex⟩
    Update the replica's view ID and status and clean up view change state.
  ∧ rViewID' = [rViewID       EXCEPT ![r] = m.viewID]
  ∧ rStatus'  = [rStatus      EXCEPT ![r]  = SNormal]
  ∧ LET viewChanges ≜ {v ∈ rViewChangeReps[r] : v.viewID = rViewID[r]}
    IN  rViewChangeReps' = [rViewChangeReps EXCEPT ![r] = rViewChangeReps[r] \ viewChanges]
  ∧ Ack(m)
  ∧ UNCHANGED ⟨clientVars⟩

──────────────────────────────────────────────────────────────────────────────

InitMessageVars ≜
    ∧ messages     = {}
    ∧ messageCount = 0
    ∧ stepCount    = 0

InitClientVars ≜
    ∧ cTime     = 0
    ∧ cViewID   = [c ∈ Clients ↦ 1]
    ∧ cReqID    = [c ∈ Clients ↦ 0]
    ∧ cReps     = [c ∈ Clients ↦ {}]
    ∧ cCommits  = [c ∈ Clients ↦ {}]

InitReplicaVars ≜
    ∧ rStatus       = [r ∈ Replicas ↦ SNormal]
    ∧ rViewID       = [r ∈ Replicas ↦ 1]
    ∧ rLog          = [r ∈ Replicas ↦ ⟨⟩]
    ∧ rSyncIndex    = [r ∈ Replicas ↦ 0]
    ∧ rLogViewID    = [r ∈ Replicas ↦ 1]
    ∧ rViewChangeReps = [r ∈ Replicas ↦ {}]

Init ≜
    ∧ InitMessageVars
    ∧ InitClientVars
```

$\land$ *InitReplicaVars*

This section specifies the invariants for the protocol.

The type invariant asserts that the leader's *log* will never contain a different value at the same index as a client commit.

$$
\begin{aligned}
&Inv \triangleq \\
&\quad \forall\, c \in Clients : \\
&\qquad \forall\, e \in cCommits[c] : \\
&\qquad\quad \neg\exists\, r \in Replicas : \\
&\qquad\qquad \land\, rStatus[r] \;\; = SNormal \\
&\qquad\qquad \land\, rViewID[r] \geq e.viewID \\
&\qquad\qquad \land\, r \in Quorum(rViewID[r]) \\
&\qquad\qquad \land\, rLog[r][e.index].value \neq e.value
\end{aligned}
$$

$$
\begin{aligned}
&NextClientRequest \triangleq \\
&\quad \exists\, c \in Clients : \\
&\qquad \exists\, v \in Values : \\
&\qquad\quad ClientRequest(c,\, v)
\end{aligned}
$$

$$
\begin{aligned}
&NextChangeView \triangleq \\
&\quad \exists\, r \in Replicas : \\
&\qquad ChangeView(r)
\end{aligned}
$$

$$
\begin{aligned}
&NextHandleClientRequest \triangleq \\
&\quad \exists\, m \in messages : \\
&\qquad \land\, m.type = MClientRequest \\
&\qquad \land\, HandleClientRequest(m.dest,\, m.src,\, m)
\end{aligned}
$$

$$
\begin{aligned}
&NextHandleClientReply \triangleq \\
&\quad \exists\, m \in messages : \\
&\qquad \land\, m.type = MClientReply \\
&\qquad \land\, HandleClientReply(m.dest,\, m.src,\, m)
\end{aligned}
$$

$$
\begin{aligned}
&NextHandleReconcileRequest \triangleq \\
&\quad \exists\, m \in messages : \\
&\qquad \land\, m.type = MReconcileRequest \\
&\qquad \land\, HandleReconcileRequest(m.dest,\, m.src,\, m)
\end{aligned}
$$

$$
\begin{aligned}
&NextHandleReconcileReply \triangleq \\
&\quad \exists\, m \in messages : \\
&\qquad \land\, m.type = MReconcileReply \\
&\qquad \land\, HandleReconcileReply(m.dest,\, m.src,\, m)
\end{aligned}
$$

$$
NextHandleRepairRequest \triangleq
$$

$\exists\, m \in messages :$
$\land\, m.type = MRepairRequest$
$\land\, HandleRepairRequest(m.dest,\ m.src,\ m)$

$NextHandleRepairReply \triangleq$
$\exists\, m \in messages :$
$\land\, m.type = MRepairReply$
$\land\, HandleRepairReply(m.dest,\ m.src,\ m)$

$NextHandleViewChange \triangleq$
$\exists\, m \in messages :$
$\land\, m.type = MViewChange$
$\land\, HandleViewChange(m.dest,\ m.src,\ m)$

$NextHandleViewChangeReply \triangleq$
$\exists\, m \in messages :$
$\land\, m.type = MViewChangeReply$
$\land\, HandleViewChangeReply(m.dest,\ m.src,\ m)$

$NextHandleStartView \triangleq$
$\exists\, m \in messages :$
$\land\, m.type = MStartView$
$\land\, HandleStartView(m.dest,\ m.src,\ m)$

$NextDropMessage \triangleq$
$\exists\, m \in messages :$
$\land\, Ack(m)$
$\land$ UNCHANGED $\langle clientVars,\ replicaVars\rangle$

$Next \triangleq$
$\lor\, NextClientRequest$
$\lor\, NextChangeView$
$\lor\, NextHandleClientRequest$
$\lor\, NextHandleClientReply$
$\lor\, NextHandleReconcileRequest$
$\lor\, NextHandleReconcileReply$
$\lor\, NextHandleRepairRequest$
$\lor\, NextHandleRepairReply$
$\lor\, NextHandleViewChange$
$\lor\, NextHandleViewChangeReply$
$\lor\, NextHandleStartView$
$\lor\, NextDropMessage$

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

\ * Modification History
\ * Last modified *Wed Sep* 30 12:21:00 *PDT* 2020 by *jordanhalterman*

\ ∗ Created *Fri Sep* 18 22:45:21 *PDT* 2020 by *jordanhalterman*