MODULE *JustInTimePaxos*

EXTENDS *Naturals*, *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,
MClientResponse,
MRepairRequest,
MRepairResponse,
MAbortRequest,
MAbortResponse,
MViewChangeRequest,
MViewChangeResponse,
MStartViewRequest

Replica statuses
CONSTANTS
SNormal,
SAborting,
SViewChange

Entry types
CONSTANTS
TValue,
TNoOp

Message schemas

$ViewIDs \triangleq [viewID \mapsto n \in (1\ ..\ )]$

ClientRequest

$[src \mapsto c \in Clients,$
$\ \ dest \mapsto r \in Replicas,$
$\ \ type \mapsto MClientRequest,$
$\ \ viewID \mapsto i \in ViewIDs,$
$\ \ sessionID \mapsto c \in Clients,$
$\ \ value \mapsto v \in Values,$

```
  seqNum ↦ s ∈  (1 . . ),
  timestamp ↦ t ∈  (1 . . )]

ClientResponse
 [src       ↦ r ∈ Replicas,
  dest      ↦ c ∈ Clients,
  type      ↦ MClientResponse,
  viewID ↦ i ∈ ViewIDs,
  sessionID ↦ c ∈ Clients,
  value     ↦ v ∈ Values,
  seqNum ↦ s ∈  (1 . . ),
  index     ↦ i ∈  (1 . . ),
  succeeded ↦ TRUE ∨ FALSE]

RepairRequest
 [src       ↦ r ∈ Replicas,
  dest      ↦ r ∈ Replicas,
  type      ↦ MRepairRequest,
  viewID ↦ i ∈ ViewIDs,
  sessionID ↦ c ∈ Clients,
  seqNum ↦ s ∈  (1 . . ),
  timestamp ↦ t ∈  (1 . . )]

RepairResponse
 [src       ↦ r ∈ Replicas,
  dest      ↦ r ∈ Replicas,
  type      ↦ MRepairResponse,
  viewID ↦ i ∈ ViewIDs,
  sessionID ↦ c ∈ Clients,
  seqNum ↦ s ∈  (1 . . ),
  value     ↦ v ∈ Values,
  timestamp ↦ t ∈  (1 . . )]

AbortRequest
 [src       ↦ r ∈ Replicas,
  dest      ↦ r ∈ Replicas,
  type      ↦ MAbortRequest,
  viewID ↦ i ∈ ViewIDs,
  sessionID ↦ c ∈ Clients,
  seqNum ↦ s ∈  (1 . . ),
  timestamp ↦ t ∈  (1 . . )]

AbortResponse
 [src       ↦ r ∈ Replicas,
  dest      ↦ r ∈ Replicas,
  type      ↦ MAbortResponse,
  viewID ↦ i ∈ ViewIDs,
  sessionID ↦ c ∈ Clients,
  seqNum ↦ s ∈  (1 . . )]

ViewChangeRequest
 [src        ↦ r ∈ Replicas,
  dest       ↦ r ∈ Replicas,
```

$type \quad \mapsto MViewChangeRequest,$
$viewID \quad \mapsto i \in ViewIDs]$

$ViewChangeResponse$
$[src \quad \mapsto r \in Replicas,$
$dest \quad \mapsto r \in Replicas,$
$type \quad \mapsto MViewChangeResponse,$
$viewID \quad \mapsto i \in ViewIDs,$
$lastViewID \mapsto i \in \ (1 \, .. \,),$
$logs \quad \mapsto \ [c \in Clients \mapsto \langle\rangle]]$

$StartViewRequest$
$[src \quad \mapsto r \in Replicas,$
$dest \quad \mapsto r \in Replicas,$
$type \quad \mapsto MStartViewRequest,$
$viewID \mapsto i \in ViewIDs,$
$timestamp \mapsto t \in \ (1 \, .. \,),$
$log \quad \mapsto \ [c \in Clients \mapsto \langle\rangle]]$

A sequence of replicas used for deterministic primary election
VARIABLE $replicas$

$globalVars \triangleq \langle replicas \rangle$

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

$messageVars \triangleq \langle messages \rangle$

Local client state

Strictly increasing representation of synchronized time
VARIABLE $cTime$

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

The current sequence number for a client
VARIABLE $cSeqNum$

A client response buffer
VARIABLE $cResps$

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

$clientVars \triangleq \langle cTime,\ cViewID,\ cSeqNum,\ cResps,\ cCommits \rangle$

Local replica state

The current status of a replica

```
VARIABLE rStatus

 A replica's commit log
VARIABLE rLog

 The current view ID for a replica
VARIABLE rViewID

 The current sequence number for each session
VARIABLE rSeqNum

 The highest known timestamp for all sessions
VARIABLE rTimestamp

 The last known normal view
VARIABLE rLastViewID

 The set of received view change responses
VARIABLE rViewChanges

 The point (client + sequence number) in the log currently being aborted
VARIABLE rAbortPoint

 The set of abort responses received
VARIABLE rAbortResps

replicaVars ≜ ⟨rStatus, rLog, rViewID, rSeqNum, rTimestamp,
               rLastViewID, rViewChanges, rAbortPoint, rAbortResps⟩

vars ≜ ⟨globalVars, messageVars, clientVars, replicaVars⟩

├──────────────────────────────────────────────────────────────────────┤

This section provides helpers for the spec.
RECURSIVE SeqFromSet(_)
SeqFromSet(S) ≜
    IF S = {} THEN
        ⟨⟩
     ELSE LET x ≜ CHOOSE x ∈ S : TRUE
           IN   ⟨x⟩ ∘ SeqFromSet(S \ {x})

Pick(S) ≜ CHOOSE s ∈ S : TRUE

RECURSIVE SetReduce(_, _, _)
SetReduce(Op(_, _), S, value) ≜
    IF S = {} THEN
        value
     ELSE
        LET s ≜ Pick(S)
```

$$\qquad \text{IN} \quad SetReduce(Op,\, S \setminus \{s\},\, Op(s,\, value))$$

$$Max(s) \triangleq \text{CHOOSE } x \in s : \forall\, y \in s : x \geq y$$

$$\begin{aligned} Sum(S) \triangleq\ & \text{LET } \_op(a,\, b) \triangleq a + b \\ & \text{IN} \quad SetReduce(\_op,\, S,\, 0) \end{aligned}$$

$$IsQuorum(s) \triangleq Cardinality(s) * 2 \geq Cardinality(Replicas)$$

$$Quorums \triangleq \{r \in \text{SUBSET } Replicas : IsQuorum(r)\}$$

$$Primary(v) \triangleq replicas[(v \% Len(replicas)) + (\text{IF } v \geq Len(replicas) \text{ THEN } 1 \text{ ELSE } 0)]$$

$$IsPrimary(r) \triangleq Primary(rViewID[r]) = r$$

This section models the network.

Send a set of messages

$$Sends(ms) \triangleq messages' = messages \cup ms$$

Send a message

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

Reply to a message with a set of responses

$$Replies(req,\, resps) \triangleq messages' = (messages \cup resps) \setminus \{req\}$$

Reply to a message

$$Reply(req,\, resp) \triangleq Replies(req,\, \{resp\})$$

Discard a message

$$Discard(m) \triangleq messages' = messages \setminus \{m\}$$

This section models client requests.

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

Client requests are ordered globally using physical timestamps and locally (within the client) using client sequence numbers. Sequence numbers are sequential and unique within each view.

When the client sends a request is generates a new timestamp. Physical timestamps are modeled here as a strictly increasing global clock simulating synchronized system clocks. The sequence number for the client is also incremented and sent with the request.

$$\begin{aligned} & ClientRequest(c,\, v) \triangleq \\ & \quad \land\ cTime' = cTime + 1 \\ & \quad \land\ cSeqNum' = [cSeqNum \text{ EXCEPT } ![c] = cSeqNum[c] + 1] \\ & \quad \land\ Sends(\{[src \quad\ \mapsto c, \\ & \qquad\qquad\quad\ dest \quad \mapsto r, \end{aligned}$$

```
                 type      ↦ MClientRequest,
                 viewID    ↦ cViewID[c],
                 seqNum    ↦ cSeqNum′[c],
                 value     ↦ v,
                 timestamp ↦ cTime′] : r ∈ Replicas})
    ∧ UNCHANGED ⟨globalVars, replicaVars, cViewID, cResps, cCommits⟩
```

Client 'c' handles a response 'm' from replica 'r'
When a response is received by the client, if the client is still in the request view it can process the response. The client is responsible for determining commitment by counting responses for each sequence number. Once a response is received from a majority of the replicas including the primary replica, the response is committed. Committed responses are stored in a history variable for checking against invariants.

```
HandleClientResponse(c, r, m) ≜
    ∧ ∨ ∧ m.viewID = cViewID[c]
        ∧ cResps′ = [cResps EXCEPT ![c] = cResps[c] ∪ {m}]
        ∧ LET
              seqNumResps  ≜ {n ∈ cResps[c] : n.seqNum = m.seqNum}
              goodResps    ≜ {n ∈ seqNumResps : n.viewID = cViewID[c] ∧ n.succeeded}
              isCommitted  ≜ ∧ ∃ n ∈ goodResps : n.src       = Primary(n.viewID)
                             ∧ {n.src : n ∈ goodResps} ∈ Quorums
          IN
              ∧ ∨ ∧ isCommitted
                  ∧ cCommits′ = [cCommits EXCEPT ![c] = cCommits[c] ∪
                        {CHOOSE n ∈ goodResps : n.src = Primary(n.viewID)}]
                ∨ ∧ ¬isCommitted
                  ∧ UNCHANGED ⟨cCommits⟩
              ∧ UNCHANGED ⟨cViewID, cSeqNum⟩
      ∨ ∧ m.viewID > cViewID[c]
        ∧ cViewID′ = [cViewID EXCEPT ![c]  = m.viewID]
        ∧ cSeqNum′ = [cSeqNum EXCEPT ![c] = 0]
        ∧ cResps′    = [cResps  EXCEPT ![c]   = {}]
        ∧ UNCHANGED ⟨cCommits⟩
      ∨ ∧ m.viewID < cViewID[c]
        ∧ UNCHANGED ⟨cCommits⟩
    ∧ Discard(m)
    ∧ UNCHANGED ⟨globalVars, replicaVars, cTime, cSeqNum⟩
```

This section models the replica protocol.

Replica 'r' requests a repair of the client 'c' request 'm'

```
Repair(r, c, m) ≜
    ∧ Replies(m, {[src      ↦ r,
                   dest     ↦ d,
                   type     ↦ MRepairRequest,
                    $viewID \mapsto rViewID[r],$
                    $sessionID \mapsto c,$
                    $seqNum \mapsto m.seqNum,$
                    $timestamp \mapsto m.timestamp] : d \in Replicas\})$
```

Replica 'r' aborts the client 'c' request 'm'

```
$Abort(r, c, m) \triangleq$
    $\land IsPrimary(r)$
    $\land rStatus[r] = SNormal$
    $\land rStatus' = [rStatus \text{ EXCEPT } ![r] = SAborting]$
    $\land rAbortResps' = [rAbortResps \text{ EXCEPT } ![r] = \{\}]$
    $\land rAbortPoint' = [rAbortPoint \text{ EXCEPT } ![r] = [sessionID \mapsto c, seqNum \mapsto m.seqNum]]$
    $\land Replies(m, \{[src \mapsto r,$
                    $dest \mapsto d,$
                    $type \mapsto MAbortRequest,$
                    $viewID \mapsto rViewID[r],$
                    $sessionID \mapsto c,$
                    $seqNum \mapsto m.seqNum,$
                    $timestamp \mapsto m.timestamp] : d \in Replicas\})$
```

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

Client requests with a view *ID* not matching the replica's view are rejected.

Clients reset their sequence number of

For requests in the correct view, the request must be sequential and linear to be appended to the *log*. That is, the request must have a 'seqNum' that is $1 +$ the prior 'seqNum' for the client, and the 'timestamp' must be greater than all prior timestamps in the *log*. This is necessary to ensure the primary *log* does not change when requests are reordered. The client can retry requests that are reordered with a new sequence number and timestamp.

To maintain consistency within the *log*, a separate sequence is maintained for each session (client), and each sequence number is assigned to a unique position in the session *log*. Session logs are logically merged into a totally ordered *log* using the request timestamps.

When a sequence number is skipped, the primary must commit a *TNoOp* entry to the *log*. It does so by running the *AbortRequest* protocol.

When a sequence number is skipped on a non-primary replica, the replica attempts to recover the request using the *RepairRequest* protocol.

```
$HandleClientRequest(r, c, m) \triangleq$
    $\land rStatus[r] = SNormal$
    $\land \lor \land m.viewID = rViewID[r]$
            $\land \text{LET}$
                $lastIndex \triangleq Sum(\{Len(rLog[r][i]) : i \in Clients\})$
                $index \triangleq lastIndex + 1$
                $lastTimestamp \triangleq rTimestamp[r]$
                $isSequential \triangleq m.seqNum = rSeqNum[r][c] + 1$
```

```
               isLinear        ≜ m.timestamp > lastTimestamp
               entry           ≜ [type      ↦ TValue,
                                  index     ↦ index,
                                  value     ↦ m.value,
                                  timestamp ↦ m.timestamp]
               append(e)       ≜ [rLog EXCEPT ![r] = [rLog[r] EXCEPT
                                                   ![c] = Append(rLog[r][c], e)]]
            IN
               ∨ ∧ isSequential
                 ∧ isLinear
                 ∧ rLog'        = append(entry)
                 ∧ rTimestamp' = [rTimestamp EXCEPT ![r] = m.timestamp]
                 ∧ Reply(m, [src       ↦ r,
                             dest      ↦ c,
                             type      ↦ MClientResponse,
                             viewID    ↦ rViewID[r],
                             seqNum    ↦ m.seqNum,
                             index     ↦ index,
                             value     ↦ m.value,
                             succeeded ↦ TRUE])
                 ∧ UNCHANGED ⟨rStatus, rAbortPoint, rAbortResps⟩
               ∨ ∧ ∨ ∧ ¬isSequential
                       ∧ m.seqNum > rSeqNum[r][c] + 1
                     ∨ ¬isLinear
                 ∧ ∨ ∧ IsPrimary(r)
                       ∧ Abort(r, c, m)
                     ∨ ∧ ¬IsPrimary(r)
                       ∧ Repair(r, c, m)
                       ∧ UNCHANGED ⟨rStatus, rAbortPoint, rAbortResps⟩
                 ∧ UNCHANGED ⟨rLog, rSeqNum, rTimestamp⟩
       ∨ ∧ m.viewID < rViewID[r]
         ∧ Reply(m, [src       ↦ r,
                     dest      ↦ c,
                     type      ↦ MClientResponse,
                     viewID    ↦ rViewID[r],
                     seqNum    ↦ m.seqNum,
                     succeeded ↦ FALSE])
         ∧ UNCHANGED ⟨rStatus, rLog, rSeqNum, rTimestamp, rAbortPoint, rAbortResps⟩
    ∧ UNCHANGED ⟨globalVars, clientVars, rViewID, rLastViewID, rViewChanges⟩
```

Replica 'r' handles replica 's' repair request 'm'

When a repair request is received, if the requested sequence number is in the session *log*, the entry is returned. Otherwise, the primary aborts the request.

```
HandleRepairRequest(r, s, m) ≜
    ∧ m.viewID = rViewID[r]
    $\land IsPrimary(r)$
    $\land rStatus[r] = SNormal$
    $\land$ LET $offset \triangleq Len(rLog[r][m.sessionID]) - (rSeqNum[r][m.sessionID] - m.seqNum)$
      IN
        $\lor \land offset \leq Len(rLog[r][m.sessionID])$
          $\land Reply(m, [src \mapsto r,$
                          $dest \mapsto s,$
                          $type \mapsto MRepairResponse,$
                          $viewID \mapsto rViewID[r],$
                          $sessionID \mapsto m.sessionID,$
                          $seqNum \mapsto m.seqNum,$
                          $value \mapsto rLog[r][m.sessionID][offset].value,$
                          $timestamp \mapsto rLog[r][m.sessionID][offset].timestamp])$
          $\land$ UNCHANGED $\langle rStatus, rAbortPoint, rAbortResps \rangle$
        $\lor \land offset = Len(rLog[r][m.sessionID]) + 1$
          $\land Abort(r, m.sessionID, m)$
    $\land$ UNCHANGED $\langle globalVars, clientVars, rLog, rSeqNum, rTimestamp, rViewID, rLastViewID, rViewChan$
```

Replica 'r' handles replica 's' repair response 'm'

Repair responses are handled like client requests.

```
$HandleRepairResponse(r, s, m) \triangleq$
    $HandleClientRequest(r, m.sessionID, [m$ EXCEPT $!.src = m.sessionID])$
```

Replica 'r' handles replica 's' abort request 'm'

If the aborted sequence number is in the session *log*, the entry is replaced with a no-op entry. If the sequence number can be appebded to the *log*, it is.

```
$HandleAbortRequest(r, s, m) \triangleq$
    $\land m.viewID = rViewID[r]$
    $\land rStatus[r] \in \{SNormal, SAborting\}$
    $\land$ LET
          $offset \triangleq Len(rLog[r][m.sessionID]) - (rSeqNum[r][m.sessionID] - m.seqNum)$
          $entry \triangleq [type \mapsto TNoOp, timestamp \mapsto m.timestamp]$
          $replace(l, i, e) \triangleq [j \in 1 \,..\, Max(\{Len(l), i\}) \mapsto$ IF $j = i$ THEN $e$ ELSE $l[j]]$
      IN
        $\land offset \leq Len(rLog[r][m.sessionID]) + 1$
        $\land rLog' = [rLog$ EXCEPT $![r] = [rLog[r]$ EXCEPT
                                 $![m.sessionID] = replace(rLog[r][m.sessionID], offset, entry)]]$
        $\land rTimestamp' = [rTimestamp$ EXCEPT $![r] = Max(\{rTimestamp[r], m.timestamp\})]$
        $\land rSeqNum' = [rSeqNum$ EXCEPT $![r] = [rSeqNum[r]$ EXCEPT
                                 $![m.sessionID] = Max(\{rSeqNum[r][m.sessionID], m.seqNum\})]]$
        $\land Replies(m, \{[src \mapsto r,$
                        $dest \mapsto Primary(rViewID[r]),$
                        $type \mapsto MAbortResponse,$
                        $viewID \mapsto rViewID[r],$
                        $sessionID \mapsto m.sessionID,$
```

```
                            seqNum    ↦ m.seqNum],
                           [src       ↦ r,
                            dest      ↦ m.sessionID,
                            type      ↦ MClientResponse,
                            viewID    ↦ rViewID[r],
                            seqNum    ↦ m.seqNum,
                            succeeded ↦ FALSE]})
    ∧ UNCHANGED ⟨globalVars, clientVars, rStatus, rAbortPoint,
                 rAbortResps, rViewID, rLastViewID, rViewChanges⟩
```

Replica 'r' handles replica 's' repair response 'm'

```
HandleAbortResponse(r, s, m) ≜
    ∧ rStatus[r] = SAborting
    ∧ m.viewID = rViewID[r]
    ∧ IsPrimary(r)
    ∧ m.seqNum = rAbortPoint[r].seqNum
    ∧ rAbortResps' = [rAbortResps EXCEPT ![r] = rAbortResps[r] ∪ {m}]
    ∧ LET resps ≜ {res.src : res ∈ {resp ∈ rAbortResps'[r] :
                        ∧ resp.viewID    = rViewID[r]
                        ∧ resp.sessionID = rAbortPoint[r].sessionID
                        ∧ resp.seqNum    = rAbortPoint[r].seqNum}}
          isQuorum ≜ r ∈ resps ∧ resps ∈ Quorums
      IN
          ∨ ∧ isQuorum
            ∧ rStatus' = [rStatus EXCEPT ![r] = SNormal]
          ∨ ∧ ¬isQuorum
            ∧ UNCHANGED ⟨rStatus⟩
    ∧ UNCHANGED ⟨globalVars, messageVars, clientVars, rLog, rSeqNum, rTimestamp,
                 rAbortPoint, rViewID, rViewChanges, rLastViewID⟩
```

Replica 'r' requests a view change

The view change is requested by sending a *ViewChangeRequest* to each replica.

```
ChangeView(r) ≜
    ∧ Sends({[src    ↦ r,
              dest   ↦ d,
              type   ↦ MViewChangeRequest,
              viewID ↦ rViewID[r] + 1] : d ∈ Replicas})
    ∧ UNCHANGED ⟨globalVars, clientVars, replicaVars⟩
```

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

Replicas respond to *ViewChangeRequests* with the contents of their logs for reconciliation. When a new view change is requested, the replica updates its *ViewID* and transitions to the *ViewChange* status to block writes during the transition.

```
HandleViewChangeRequest(r, s, m) ≜
    ∧ rViewID[r] < m.viewID
    ∧ rViewID'      = [rViewID EXCEPT ![r] = m.viewID]
    ∧ rStatus'      = [rStatus EXCEPT ![r]  = SViewChange]
    ∧ rViewChanges' = [rViewChanges EXCEPT ![r] = {}]
    ∧ Reply(m, [src         ↦ r,
                dest        ↦ Primary(m.viewID),
                type        ↦ MViewChangeResponse,
                viewID      ↦ m.viewID,
                lastViewID  ↦ rLastViewID[r],
                logs        ↦ rLog[r]])
    ∧ UNCHANGED ⟨globalVars, clientVars, rLog, rSeqNum, rTimestamp,
                 rAbortPoint, rAbortResps, rLastViewID⟩
```

Replica ’r’ handles replica ’s’ view change response ’m’
ViewChangeResponses are handled by the primary for the new view. Once responses are received from a majority of the replicas including the new primary, the logs received from each replica are merged together to form the *log* for the new view. For each known session, the logs from each replica are merged by comparing each entry and keeping all non-empty sequential entries in the quorum. An updated timestamp is calculated from the reconciled *log*, and a *StartViewRequest* containing the new logs is sent to each replica.

```
HandleViewChangeResponse(r, s, m) ≜
    ∧ IsPrimary(r)
    ∧ rViewID[r]    = m.viewID
    ∧ rStatus[r]    = SViewChange
    ∧ rViewChanges' = [rViewChanges EXCEPT ![r] = rViewChanges[r] ∪ {m}]
    ∧ LET viewChanges      ≜ {v ∈ rViewChanges'[r] : v.viewID = rViewID[r]}
          viewSources      ≜ {v.src : v ∈ viewChanges}
          isQuorum         ≜ r ∈ viewSources ∧ viewSources ∈ Quorums
          lastViewIDs      ≜ {v.lastViewID : v ∈ viewChanges}
          lastViewID       ≜ (CHOOSE v1 ∈ lastViewIDs : ∀ v2 ∈ lastViewIDs : v2 ≤ v1)
          lastViewChanges  ≜ {v2 ∈ viewChanges : v2.lastViewID = lastViewID}
          viewLogs         ≜ [c ∈ Clients ↦ {v1.logs[c] : v1      ∈ lastViewChanges}]
          mergeEnts(es)    ≜
              IF es = {} ∨ ∃ e ∈ es : e.type = TNoOp THEN
                  [type ↦ TNoOp]
               ELSE
                  CHOOSE e ∈ es : e.type ≠ TNoOp
          range(ls)        ≜ Max({Len(l) : l ∈ ls})
          entries(ls, i)   ≜ {l[i] : l ∈ {k ∈ ls : i ≤ Len(k)}}
          mergeLogs(ls)    ≜ [i ∈ 1 .. range(ls) ↦ mergeEnts(entries(ls, i))]
          viewLog          ≜ [c ∈ Clients ↦ mergeLogs(viewLogs[c])]
          viewRange        ≜ Max({Len(viewLog[c]) : c ∈ Clients})
          viewTimestamp    ≜ IF viewRange > 0 THEN
                                  Max(UNION {{l[i].timestamp : i ∈ DOMAIN l} :
                                                  l ∈ {viewLog[c] : c ∈ Clients}})
```

```
                           ELSE  0
      IN
         ∨ ∧ isQuorum
           ∧ Replies(m, {[src       ↦ r,
                          dest      ↦ d,
                          type      ↦ MStartViewRequest,
                          viewID    ↦ rViewID[r],
                          timestamp ↦ viewTimestamp,
                          log       ↦ viewLog] : d ∈ Replicas})
         ∨ ∧ ¬isQuorum
           ∧ Discard(m)
      ∧ UNCHANGED ⟨globalVars, clientVars, rStatus, rViewID, rLog, rSeqNum,
                    rTimestamp, rAbortPoint, rAbortResps, rLastViewID⟩
```

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

If the view is new, the replica updates its logs and session state from the request.

```
HandleStartViewRequest(r, s, m) ≜
    ∧ ∨ rViewID[r] < m.viewID
      ∨ ∧ rViewID[r] = m.viewID
        ∧ rStatus[r]  = SViewChange
    ∧ rLog'        = [rLog         EXCEPT ![r]  = m.log]
    ∧ rSeqNum'     = [rSeqNum      EXCEPT ![r] = [c ∈ Clients ↦ 0]]
    ∧ rTimestamp'  = [rTimestamp   EXCEPT ![r] = m.timestamp]
    ∧ rStatus'     = [rStatus      EXCEPT ![r]  = SNormal]
    ∧ rViewID'     = [rViewID      EXCEPT ![r] = m.viewID]
    ∧ rLastViewID' = [rLastViewID  EXCEPT ![r] = m.viewID]
    ∧ Discard(m)
    ∧ UNCHANGED ⟨globalVars, clientVars, rAbortPoint, rAbortResps, rViewChanges⟩
```

```
InitMessageVars ≜
    ∧ messages = {}

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

InitReplicaVars ≜
    ∧ replicas      = SeqFromSet(Replicas)
    ∧ rStatus       = [r ∈ Replicas ↦ SNormal]
    ∧ rLog          = [r ∈ Replicas ↦ [c ∈ Clients ↦ ⟨⟩]]
    ∧ rSeqNum       = [r ∈ Replicas ↦ [c ∈ Clients ↦ 0]]
    $\land\ rTimestamp \quad\ = [r \in Replicas \mapsto 0]$
    $\land\ rAbortPoint \quad = [r \in Replicas \mapsto [client \mapsto Nil,\ seqNum \mapsto 0]]$
    $\land\ rAbortResps \quad = [r \in Replicas \mapsto \{\}]$
    $\land\ rViewID \qquad\ \ = [r \in Replicas \mapsto 1]$
    $\land\ rLastViewID \quad = [r \in Replicas \mapsto 1]$
    $\land\ rViewChanges = [r \in Replicas \mapsto \{\}]$

$Init\ \triangleq$
    $\land\ InitMessageVars$
    $\land\ InitClientVars$
    $\land\ InitReplicaVars$
```

The type invariant verifies that clients do not receive two commits at the same index with different values.

```
$TypeOK\ \triangleq$
   $\forall\, c1,\ c2 \in Clients :$
     $\forall\, e1 \in cCommits[c1] :$
       $\neg\exists\, e2 \in cCommits[c2] :$
           $\land\ e1.index = e2.index$
           $\land\ e1.value \neq e2.value$

$Next\ \triangleq$
    $\lor\ \exists\, c \in Clients :$
         $\exists\, v \in Values :$
           $\land\ ClientRequest(c,\ v)$
    $\lor\ \exists\, r \in Replicas :$
         $\land\ ChangeView(r)$
    $\lor\ \exists\, m \in messages :$
         $\land\ m.type = MClientRequest$
         $\land\ HandleClientRequest(m.dest,\ m.src,\ m)$
    $\lor\ \exists\, m \in messages :$
         $\land\ m.type = MClientResponse$
         $\land\ HandleClientResponse(m.dest,\ m.src,\ m)$
    $\lor\ \exists\, m \in messages :$
         $\land\ m.type = MRepairRequest$
         $\land\ HandleRepairRequest(m.dest,\ m.src,\ m)$
    $\lor\ \exists\, m \in messages :$
         $\land\ m.type = MRepairResponse$
         $\land\ HandleRepairResponse(m.dest,\ m.src,\ m)$
    $\lor\ \exists\, m \in messages :$
         $\land\ m.type = MAbortRequest$
         $\land\ HandleAbortRequest(m.dest,\ m.src,\ m)$
    $\lor\ \exists\, m \in messages :$
         $\land\ m.type = MAbortResponse$
```

```
        $\land HandleAbortResponse(m.dest, m.src, m)$
    $\lor \exists m \in messages :$
        $\land m.type = MViewChangeRequest$
        $\land HandleViewChangeRequest(m.dest, m.src, m)$
    $\lor \exists m \in messages :$
        $\land m.type = MViewChangeResponse$
        $\land HandleViewChangeResponse(m.dest, m.src, m)$
    $\lor \exists m \in messages :$
        $\land m.type = MStartViewRequest$
        $\land HandleStartViewRequest(m.dest, m.src, m)$
    $\lor \exists m \in messages :$
        $\land Discard(m)$
        $\land$ UNCHANGED $\langle globalVars, clientVars, replicaVars \rangle$

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

```
\* Modification History
\* Last modified Tue Sep 22 15:08:43 PDT 2020 by jordanhalterman
\* Created Fri Sep 18 22:45:21 PDT 2020 by jordanhalterman
```