MODULE *ZabWithQTest*

This is the test for formal specification for the *Zab* consensus algorithm,
which adds some restrictions like the number of rounds and
number of transactions broadcast based on *ZabWithQ*.

This work is driven by *Flavio P*. Junqueira,"*Zab*: High-performance broadcast for primary-backup systems"

EXTENDS *Integers*, *FiniteSets*, *Sequences*, *Naturals*, *TLC*

The set of server identifiers

CONSTANT *Server*

The set of requests that can go into history

CONSTANT *Value*

Server states

It is unnecessary to add state ELECTION, we can own it by setting *leaderOracle* to Null.

CONSTANTS *Follower*, *Leader*, *ProspectiveLeader*

Message types

CONSTANTS *CEPOCH*, *NEWEPOCH*, *ACKE*, *NEWLEADER*, *ACKLD*, *COMMITLD*, *PROPOSE*, *ACK*, *C*

Additional Message types used for recovery when restarting

CONSTANTS *RECOVERYREQUEST*, *RECOVERYRESPONSE*

the maximum round of epoch (initially {0, 1, 2}), currently not used

CONSTANT *Epoches*

Return the maximum value from the set *S*

$$Maximum(S) \triangleq \text{IF } S = \{\} \text{ THEN } -1 \text{ ELSE CHOOSE } n \in S : \forall m \in S : n \geq m$$

Return the minimum value from the set *S*

$$Minimum(S) \triangleq \text{IF } S = \{\} \text{ THEN } -1 \text{ ELSE CHOOSE } n \in S : \forall m \in S : n \leq m$$

$$Quorums \triangleq \{Q \in \text{SUBSET } Server : Cardinality(Q) * 2 > Cardinality(Server)\}$$

$$\text{ASSUME } QuorumsAssumption \triangleq \land \forall Q \in Quorums : Q \subseteq Server \quad \land \forall Q1, Q2 \in Quorums : Q1 \cap Q2 \neq \{\}$$

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

$$NullPoint \triangleq \text{CHOOSE } p : p \notin Server$$

The server's *state*(*Follower*, *Leader*, *ProspectiveLeader*).

VARIABLE *state*

The leader's epoch or the last new epoch proposal the follower acknowledged
(namely epoch of the last *NEWEPOCH* accepted, *f.p* in paper).

VARIABLE *currentEpoch*

The last new leader proposal the follower acknowledged
(namely epoch of the last *NEWLEADER* accepted, $f.a$ in paper).

VARIABLE *leaderEpoch*

The identifier of the leader for followers.

VARIABLE *leaderOracle*

The history of servers as the sequence of transactions.

VARIABLE *history*

The messages repersenting requests and responses sent from one server to another.
msgs[i][j] means the input buffer of server j from server i.

VARIABLE *msgs*

The set of servers which the leader think follow itself (Q in paper).

VARIABLE *cluster*

The set of followers who has successfully sent *CEPOCH* to pleader in pleader.

VARIABLE *cepochRecv*

The set of followers who has successfully sent *ACK*-E to pleader in pleader.

VARIABLE *ackeRecv*

The set of followers who has successfully sent *ACK*-LD to pleader in pleader.

VARIABLE *ackldRecv*

ackIndex[i][j] means leader i has received how many *ACK* messages from follower j.
So *ackIndex*[i][i] is not used.

VARIABLE *ackIndex*

currentCounter[i] means the count of transactions client requests leader.

VARIABLE *currentCounter*

sendCounter[i] means the count of transactions leader has broadcast.

VARIABLE *sendCounter*

initialHistory[i] means the initial history of leader i in epoch *currentEpoch*[i].

VARIABLE *initialHistory*

commitIndex[i] means leader/follower i should commit how many proposals and sent *COMMIT* messages.
It should be more formal to add variable *applyIndex*/*deliverIndex* to represent the prefix entries of the history that has applied to state machine, but we can tolerate that *applyIndex*(*deliverIndex here*) = *commitIndex*.
This does not violate correctness. (*commitIndex* increases monotonically before restarting)

VARIABLE *commitIndex*

commitIndex[i] means leader i has committed how many proposals and sent *COMMIT* messages.

VARIABLE *committedIndex*

Hepler matrix for follower to stop sending *CEPOCH* to pleader in followers.
Because *CEPOCH* is the sole message which follower actively sends to pleader.

VARIABLE *cepochSent*

the maximum epoch in *CEPOCH* pleader received from followers.

VARIABLE *tempMaxEpoch*

the maximum *leaderEpoch* and most up-to-date history in *ACKE* pleader received from followers.

VARIABLE *tempMaxLastEpoch*

Because pleader updates state and broadcasts *NEWLEADER* when it receives *ACKE* from a quorum of followers, and *initialHistory* is determined. But *tempInitialHistory* may change when receiving other *ACKEs* after entering into *phase*2. So it is necessary to split *initialHistory* with *tempInitialHistory*.

VARIABLE *tempInitialHistory*

the set of all broadcast messages whose tpye is proposal that any leader has sent, only used in verifying properties. So the variable will only be changed in transition *LeaderBroadcast*1.

VARIABLE *proposalMsgsLog*

Helper set for server who restarts to collect which servers has responded to it.

VARIABLE *recoveryRespRecv*

the maximum epoch and corresponding *leaderOracle* in *RECOVERYRESPONSE* from followers.

VARIABLE *recoveryMaxEpoch*

VARIABLE *recoveryMEOracle*

VARIABLE *recoverySent*

Persistent state of a server: history, *currentEpoch*, *leaderEpoch*

$serverVars \triangleq \langle state, currentEpoch, leaderEpoch, leaderOracle, history, commitIndex \rangle$
$leaderVars \triangleq \langle cluster, cepochRecv, ackeRecv, ackldRecv, ackIndex, currentCounter, sendCounter, initialHist$
$tempVars \triangleq \langle tempMaxEpoch, tempMaxLastEpoch, tempInitialHistory \rangle$
$recoveryVars \triangleq \langle recoveryRespRecv, recoveryMaxEpoch, recoveryMEOracle, recoverySent \rangle$

$vars \triangleq \langle serverVars, msgs, leaderVars, tempVars, recoveryVars, cepochSent, proposalMsgsLog \rangle$

$$LastZxid(his) \triangleq \text{IF } Len(his) > 0 \text{ THEN } \langle his[Len(his)].epoch, his[Len(his)].counter \rangle \text{ ELSE } \langle -1, -1 \rangle$$

Add a message to *msgs* – add a message *m* to *msgs*[*i*][*j*]

$$Send(i, j, m) \triangleq msgs' = [msgs \text{ EXCEPT } ![i][j] = Append(msgs[i][j], m)]$$

$$Send2(i, j, m1, m2) \triangleq msgs' = [msgs \text{ EXCEPT } ![i][j] = Append(Append(msgs[i][j], m1), m2)]$$

Remove a message from *msgs* – discard head of *msgs*[*i*][*j*]

$$Discard(i, j) \triangleq msgs' = \text{IF } msgs[i][j] \neq \langle\rangle \text{ THEN } [msgs \text{ EXCEPT } ![i][j] = Tail(msgs[i][j])] \text{ ELSE } msgs$$

Leader/Pleader broadcasts a message to all other servers in Q

$$\begin{aligned} Broadcast(i,\, m) \triangleq msgs' = [ii \in Server \mapsto [ij \in Server \mapsto \textsc{if}\ & \land ii = i \\ & \land ij \neq i \\ & \land ij \in cluster[i] \ \textsc{then}\ Append(msgs[ii][ij],\, m \\ & \qquad\qquad\quad\ \textsc{else}\ \ msgs[ii][ij]]] \end{aligned}$$

$$\begin{aligned} BroadcastToAll(i,\, m) \triangleq msgs' = [ii \in Server \mapsto [ij \in Server \mapsto \textsc{if}\ \land ii = i \land ij \neq i\ & \textsc{then}\ Append(msgs[ii][i \\ & \textsc{else}\ \ msgs[ii][ij]]] \end{aligned}$$

Combination of $Send$ and $Discard -$ discard head of $msgs[j][i]$ and add m into $msgs[i][j]$

$$\begin{aligned} Reply(i,\, j,\, m) \triangleq msgs' = [msgs\ \textsc{except}\ & ![j][i] = Tail(msgs[j][i]), \\ & ![i][j] = Append(msgs[i][j],\, m)] \end{aligned}$$

$$\begin{aligned} Reply2(i,\, j,\, m1,\, m2) \triangleq msgs' = [msgs\ \textsc{except}\ & ![j][i] = Tail(msgs[j][i]), \\ & ![i][j] = Append(Append(msgs[i][j],\, m1),\, m2)] \end{aligned}$$

$$clean(i,\, j) \triangleq msgs' = [msgs\ \textsc{except}\ ![i][j] = \langle\rangle,\ ![j][i] = \langle\rangle]$$

Define initial values for all variables

$$\begin{aligned} Init \triangleq\ & \land state &&= [s \in Server \mapsto Follower] \\ & \land currentEpoch &&= [s \in Server \mapsto 0] \\ & \land leaderEpoch &&= [s \in Server \mapsto 0] \\ & \land leaderOracle &&= [s \in Server \mapsto NullPoint] \\ & \land history &&= [s \in Server \mapsto \langle\rangle] \\ & \land msgs &&= [i \in Server \mapsto [j \in Server \mapsto \langle\rangle]] \\ & \land cluster &&= [i \in Server \mapsto \{\}] \\ & \land cepochRecv &&= [s \in Server \mapsto \{\}] \\ & \land ackeRecv &&= [s \in Server \mapsto \{\}] \\ & \land ackldRecv &&= [s \in Server \mapsto \{\}] \\ & \land ackIndex &&= [i \in Server \mapsto [j \in Server \mapsto 0]] \\ & \land currentCounter &&= [s \in Server \mapsto 0] \\ & \land sendCounter &&= [s \in Server \mapsto 0] \\ & \land commitIndex &&= [s \in Server \mapsto 0] \\ & \land committedIndex &&= [s \in Server \mapsto 0] \\ & \land initialHistory &&= [s \in Server \mapsto \langle\rangle] \\ & \land cepochSent &&= [s \in Server \mapsto \textsc{false}] \\ & \land tempMaxEpoch &&= [s \in Server \mapsto 0] \\ & \land tempMaxLastEpoch &&= [s \in Server \mapsto 0] \\ & \land tempInitialHistory &&= [s \in Server \mapsto \langle\rangle] \\ & \land recoveryRespRecv &&= [s \in Server \mapsto \{\}] \\ & \land recoveryMaxEpoch &&= [s \in Server \mapsto 0] \\ & \land recoveryMEOracle &&= [s \in Server \mapsto NullPoint] \\ & \land recoverySent &&= [s \in Server \mapsto \textsc{false}] \\ & \land proposalMsgsLog &&= \{\} \end{aligned}$$

```
A server becomes pleader and a quorum servers knows that.
$Election(i, Q) \triangleq$
        test restrictions
        $\land \forall s \in Server : currentEpoch[s] \leq 1 \land Len(history[s]) \leq 2$
        $\land i \in Q$
        $\land state' = [s \in Server \mapsto \text{IF } s = i \text{ THEN } ProspectiveLeader$
                                  $\text{ELSE IF } s \in Q \text{ THEN } Follower$
                                                  $\text{ELSE } state[s]]$
        $\land cluster' = [cluster \text{ EXCEPT } ![i] = Q]$   cluster is first initialized in election, not $phase1$.
        $\land cepochRecv' = [cepochRecv \text{ EXCEPT } ![i] = \{i\}]$
        $\land ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \{i\}]$
        $\land ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] = \{i\}]$
        $\land ackIndex' = [ii \in Server \mapsto [ij \in Server \mapsto$
                               $\text{IF } ii = i \text{ THEN } 0$
                                             $\text{ELSE } ackIndex[ii][ij]]]$
        $\land committedIndex' = [committedIndex \text{ EXCEPT } ![i] = 0]$
        $\land initialHistory' = [initialHistory \text{ EXCEPT } ![i] = \langle\rangle]$
        $\land tempMaxEpoch' = [tempMaxEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$
        $\land tempMaxLastEpoch' = [tempMaxLastEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$
        $\land tempInitialHistory' = [tempInitialHistory \text{ EXCEPT } ![i] = history[i]]$
        $\land leaderOracle' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } i$
                                                 $\text{ELSE } leaderOracle[s]]$
        $\land leaderEpoch' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } currentEpoch[s]$
                                                 $\text{ELSE } leaderEpoch[s]]$
        $\land cepochSent' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN FALSE}$
                                                 $\text{ELSE } cepochSent[s]]$
        $\land msgs' = [ii \in Server \mapsto [ij \in Server \mapsto$
                               $\text{IF } ii \in Q \lor ij \in Q \text{ THEN } \langle\rangle$
                                                      $\text{ELSE } msgs[ii][ij]]]$
        $\land \text{UNCHANGED } \langle currentEpoch, history, commitIndex, currentCounter, sendCounter, proposalMsgsLog\rangle$

The action should be triggered once at the beginning.
Because we abstract the part of leader election, we can use global variables in this action.
$InitialElection(i, Q) \triangleq$
        test restrictions
        $\land currentEpoch[i] \leq 2$
        $\land Len(history[i]) \leq 2$
        $\land \forall s \in Server : state[s] = Follower \land leaderOracle[s] = NullPoint$
        $\land Election(i, Q)$
        $\land \text{UNCHANGED } \langle currentEpoch, history, commitIndex, currentCounter, sendCounter, recoveryVars, pr$

The leader finds timeout with another follower.
$LeaderTimeout(i, j) \triangleq$
        test restrictions
        $\land currentEpoch[i] \leq 2$
```

```
∧ Len(history[i]) ≤ 2
∧ state[i] ≠ Follower
∧ j ≠ i
∧ j ∈ cluster[i]
∧ LET newCluster ≜ cluster[i] \ {j}
  IN   ∧ ∨ ∧ newCluster ∈ Quorums
           ∧ cluster' = [cluster EXCEPT ![i] = newCluster]
           ∧ clean(i, j)
           ∧ UNCHANGED ⟨state, cepochRecv, ackeRecv, ackldRecv, ackIndex, committedIndex, initia
                          tempMaxEpoch, tempMaxLastEpoch, tempInitialHistory, leaderOracle, lea
         ∨ ∧ newCluster ∉ Quorums
           ∧  LET Q ≜ CHOOSE q ∈ Quorums: i ∈ q
                  v ≜ CHOOSE s ∈ Q: TRUE
              IN  Election(v, Q)
             ∃ Q ∈ Quorums : ∧ i ∈ Q
                             ∧ ∃ v ∈ Q : Election(v, Q)
∧ UNCHANGED ⟨currentEpoch, history, commitIndex, currentCounter, sendCounter, recoveryVars, pro
```

A follower finds timeout with the leader.

```
FollowerTimeout(i) ≜
    test restrictions
    ∧ currentEpoch[i] ≤ 2
    ∧ Len(history[i]) ≤ 2
    ∧ state[i] = Follower
    ∧ leaderOracle[i] ≠ NullPoint
    ∧ ∃ Q ∈ Quorums : ∧ i ∈ Q
                      ∧ ∃ v ∈ Q : Election(v, Q)
    ∧ UNCHANGED ⟨currentEpoch, history, commitIndex, currentCounter, sendCounter, recoveryVars, pro
```

A server halts and restarts.

Like Recovery protocol in View-stamped Replication, we let a server join in cluster by broadcast recovery and wait until receiving responses from a quorum of servers.

```
Restart(i) ≜
     test restrictions
     ∧ currentEpoch[i] ≤ −1
     ∧ Len(history[i]) ≤ 2
     ∧ state'         = [state        EXCEPT ![i]   = Follower]
     ∧ leaderOracle'  = [leaderOracle EXCEPT ![i]   = NullPoint]
     ∧ commitIndex'   = [commitIndex   EXCEPT ![i] = 0]
     ∧ cepochSent'    = [cepochSent    EXCEPT ![i]  = FALSE]
     ∧ msgs'          = [ii ∈ Server ↦ [ij ∈ Server ↦ IF ij = i THEN ⟨⟩
                                                          ELSE  msgs[ii][ij]]]
     ∧ recoverySent'  = [recoverySent EXCEPT ![i] = FALSE]
     ∧ UNCHANGED ⟨currentEpoch, leaderEpoch, history, leaderVars, tempVars,
                    recoveryRespRecv, recoveryMaxEpoch, recoveryMEOracle, proposalMsgsLog⟩

RecoveryAfterRestart(i) ≜
    test restrictions
    ∧ currentEpoch[i] ≤ −1
    ∧ Len(history[i]) ≤ 2
    ∧ state[i] = Follower
    ∧ leaderOracle[i] = NullPoint
    ∧ ¬recoverySent[i]
    ∧ recoveryRespRecv′  = [recoveryRespRecv EXCEPT ![i]  = {}]
    ∧ recoveryMaxEpoch′ = [recoveryMaxEpoch EXCEPT ![i] = currentEpoch[i]]
    ∧ recoveryMEOracle′ = [recoveryMEOracle EXCEPT ![i] = NullPoint]
    ∧ recoverySent′       = [recoverySent      EXCEPT ![i]   = TRUE]
    ∧ BroadcastToAll(i, [mtype ↦ RECOVERYREQUEST])
    ∧ UNCHANGED ⟨serverVars, leaderVars, tempVars, cepochSent, proposalMsgsLog⟩

HandleRecoveryRequest(i, j) ≜
    test restrictions
    ∧ currentEpoch[i] ≤ 2
    ∧ Len(history[i]) ≤ 2
    ∧ msgs[j][i] ≠ ⟨⟩
    ∧ msgs[j][i][1].mtype = RECOVERYREQUEST
    ∧ Reply(i, j, [mtype    ↦ RECOVERYRESPONSE,
                   moracle ↦ leaderOracle[i],
                   mepoch  ↦ currentEpoch[i]])
    ∧ UNCHANGED ⟨serverVars, leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog⟩

HandleRecoveryResponse(i, j) ≜
    test restrictions
    ∧ currentEpoch[i] ≤ 2
    ∧ Len(history[i]) ≤ 2
    ∧ msgs[j][i] ≠ ⟨⟩
    ∧ msgs[j][i][1].mtype = RECOVERYRESPONSE
    ∧ LET msg     ≜ msgs[j][i][1]
          infoOk ≜ ∧ msg.mepoch ≥ recoveryMaxEpoch[i]
                   ∧ msg.moracle ≠ NullPoint
      IN   ∨ ∧ infoOk
             ∧ recoveryMaxEpoch′ = [recoveryMaxEpoch EXCEPT ![i] = msg.mepoch]
             ∧ recoveryMEOracle′ = [recoveryMEOracle EXCEPT ![i] = msg.moracle]
           ∨ ∧ ¬infoOk
             ∧ UNCHANGED ⟨recoveryMaxEpoch, recoveryMEOracle⟩
    ∧ Discard(j, i)
    ∧ recoveryRespRecv′ = [recoveryRespRecv EXCEPT ![i] = IF j ∈ recoveryRespRecv[i] THEN recoveryRe
                                                                                      ELSE  recoveryRe
    ∧ UNCHANGED ⟨serverVars, leaderVars, tempVars, cepochSent, recoverySent, proposalMsgsLog⟩
```

```
FindCluster(i) ≜
        test restrictions
        ∧ currentEpoch[i] ≤ 2
        ∧ Len(history[i]) ≤ 2
        ∧ state[i] = Follower
        ∧ leaderOracle[i] = NullPoint
        ∧ recoveryRespRecv[i] ∈ Quorums
        ∧ LET infoOk ≜ ∧ recoveryMEOracle[i] ≠ i
                       ∧ recoveryMEOracle[i] ≠ NullPoint
                       ∧ currentEpoch[i] ≤ recoveryMaxEpoch[i]
          IN ∨ ∧ ¬infoOk
                 ∧ recoverySent′ = [recoverySent EXCEPT ![i] = FALSE]
                 ∧ UNCHANGED ⟨currentEpoch, leaderOracle, msgs⟩
             ∨ ∧ infoOk
                 ∧ currentEpoch′ = [currentEpoch EXCEPT ![i] = recoveryMaxEpoch[i]]
                 ∧ leaderOracle′ = [leaderOracle EXCEPT ![i] = recoveryMEOracle[i]]
                 ∧ Send(i, recoveryMEOracle[i], [mtype ↦ CEPOCH,
                                                  mepoch ↦ recoveryMaxEpoch[i]])
                 ∧ UNCHANGED recoverySent
        ∧ UNCHANGED ⟨state, leaderEpoch, history, commitIndex, leaderVars, tempVars,
                     recoveryRespRecv, recoveryMaxEpoch, recoveryMEOracle, cepochSent, proposalMsgsL
```

```
In phase f11, follower sends f.p to pleader via CEPOCH.
FollowerDiscovery1(i) ≜
        test restrictions
        ∧ currentEpoch[i] ≤ 2
        ∧ Len(history[i]) ≤ 2
        ∧ state[i] = Follower
        ∧ leaderOracle[i] ≠ NullPoint
        ∧ ¬cepochSent[i]
        ∧ LET leader ≜ leaderOracle[i]
          IN  Send(i, leader, [mtype ↦ CEPOCH,
                               mepoch ↦ currentEpoch[i]])
        ∧ cepochSent′ = [cepochSent EXCEPT ![i] = TRUE]
        ∧ UNCHANGED ⟨serverVars, leaderVars, tempVars, recoveryVars, proposalMsgsLog⟩

In phase l11, pleader receives CEPOCH from a quorum, and choose a new epoch e′
as its own l.p and sends NEWEPOCH to followers.
LeaderHandleCEPOCH(i, j) ≜
        test restrictions
        ∧ tempMaxEpoch[i] ≤ 1
        ∧ Len(history[i])  ≤ 2
        ∧ state[i] = ProspectiveLeader
        ∧ msgs[j][i] ≠ ⟨⟩
```

```
$\land\ msgs[j][i][1].mtype = CEPOCH$
$\land\ \lor$  new message - modify tempMaxEpoch and cepochRecv
        $\land\ NullPoint \notin cepochRecv[i]$
        $\land\ \text{LET } newEpoch \triangleq Maximum(\{tempMaxEpoch[i],\ msgs[j][i][1].mepoch\})$
          $\text{IN}\ \ tempMaxEpoch' = [tempMaxEpoch \text{ EXCEPT } ![i] = newEpoch]$
        $\land\ cepochRecv' = [cepochRecv \text{ EXCEPT } ![i] = \text{IF } j \in cepochRecv[i] \text{ THEN } cepochRecv[i]$
                                                                      $\text{ELSE } cepochRecv[i] \cup \{j\}$
        $\land\ Discard(j,\ i)$
    $\lor$  new follower who joins in cluster / follower whose history and commitIndex do not match
        $\land\ NullPoint \in cepochRecv[i]$
        $\land\ \lor\ \land\ NullPoint \notin ackeRecv[i]$
                $\land\ Reply(i,\ j,\ [mtype \mapsto NEWEPOCH,$
                                  $mepoch \mapsto leaderEpoch[i]])$
            $\lor\ \land\ NullPoint \in ackeRecv[i]$
                $\land\ Reply2(i,\ j,\ [mtype \mapsto NEWEPOCH,$
                                   $mepoch \mapsto leaderEpoch[i]],$
                                  $[mtype \mapsto NEWLEADER,$
                                   $mepoch \mapsto currentEpoch[i],$
                                   $minitialHistory \mapsto initialHistory[i]])$
        $\land\ \text{UNCHANGED } \langle cepochRecv,\ tempMaxEpoch\rangle$
$\land\ cluster' = [cluster \text{ EXCEPT } ![i] = \text{IF } j \in cluster[i] \text{ THEN } cluster[i] \text{ ELSE } cluster[i] \cup \{j\}]$
$\land\ \text{UNCHANGED } \langle serverVars,\ ackeRecv,\ ackldRecv,\ ackIndex,\ currentCounter,\ sendCounter,\ initialHist$
                 $committedIndex,\ cepochSent,\ tempMaxLastEpoch,\ tempInitialHistory,\ recoveryVars,\ p$
```

Here I decide to change leader's epoch in $l12\&l21$, otherwise there may exist an old leader and a new leader who share the same expoch. So here I just change *leaderEpoch*, and use it in handling *ACK*-E.

```
$LeaderDiscovery1(i) \triangleq$
    test restrictions
    $\land\ tempMaxEpoch[i] \leq 1$
    $\land\ Len(history[i]) \leq 2$
    $\land\ state[i] = ProspectiveLeader$
    $\land\ cepochRecv[i] \in Quorums$
    $\land\ leaderEpoch' = [leaderEpoch \text{ EXCEPT } ![i] = tempMaxEpoch[i] + 1]$
    $\land\ cepochRecv' = [cepochRecv \text{ EXCEPT } ![i] = \{NullPoint\}]$
    $\land\ Broadcast(i,\ [mtype \mapsto NEWEPOCH,$
                      $mepoch \mapsto leaderEpoch'[i]])$
    $\land\ \text{UNCHANGED } \langle state,\ currentEpoch,\ leaderOracle,\ history,\ cluster,\ ackeRecv,\ ackldRecv,\ ackIndex,\ cu$
                     $initialHistory,\ commitIndex,\ committedIndex,\ cepochSent,\ tempVars,\ recoveryVars,\ p$
```

In phase $f12$, follower receives *NEWEPOCH*. If $e' > f.p$ then sends back *ACKE*, and *ACKE* contains $f.a$ and hf to help pleader choose a newer history.

```
$FollowerDiscovery2(i,\ j) \triangleq$
    test restrictions
    $\land\ currentEpoch[i] \leq 2$
    $\land\ Len(history[i]) \leq 2$
```

```
        ∧ state[i] = Follower
        ∧ msgs[j][i] ≠ ⟨⟩
        ∧ msgs[j][i][1].mtype = NEWEPOCH
        ∧ LET msg ≜ msgs[j][i][1]
          IN  ∨  new NEWEPOCH − accept and reply
                 ∧ currentEpoch[i] < msg.mepoch
                 ∧ ∨ ∧ leaderOracle[i] = j
                     ∧ currentEpoch′ = [currentEpoch EXCEPT ![i] = msg.mepoch]
                     ∧ Reply(i, j, [mtype      ↦ ACKE,
                                    mepoch     ↦ msg.mepoch,
                                    mlastEpoch ↦ leaderEpoch[i],
                                    mhf        ↦ history[i]])
                   ∨ ∧ leaderOracle[i] ≠ j
                     ∧ Discard(j, i)
                     ∧ UNCHANGED currentEpoch
              ∨ ∧ currentEpoch[i] = msg.mepoch
                 ∧ ∨ ∧ leaderOracle[i] = j
                     ∧ Reply(i, j, [mtype      ↦ ACKE,
                                    mepoch     ↦ msg.mepoch,
                                    mlastEpoch ↦ leaderEpoch[i],
                                    mhf        ↦ history[i]])
                     ∧ UNCHANGED currentEpoch
                   ∨  It may happen when a leader do not update new epoch to all followers in Q, and a new election begi
                     ∧ leaderOracle[i] ≠ j
                     ∧ Discard(j, i)
                     ∧ UNCHANGED currentEpoch
              ∨  stale NEWEPOCH − diacard
                 ∧ currentEpoch[i] > msg.mepoch
                 ∧ Discard(j, i)
                 ∧ UNCHANGED currentEpoch
        ∧ UNCHANGED ⟨state, leaderEpoch, leaderOracle, history, leaderVars,
                     commitIndex, cepochSent, tempVars, recoveryVars, proposalMsgsLog⟩

  In phase l12, pleader receives ACKE from a quorum,
  and select the history of one most up-to-date follower to be the initial history.
LeaderHandleACKE(i, j) ≜
          test restrictions
        ∧ currentEpoch[i] ≤ 2
        ∧ Len(history[i]) ≤ 2
        ∧ state[i] = ProspectiveLeader
        ∧ msgs[j][i] ≠ ⟨⟩
        ∧ msgs[j][i][1].mtype = ACKE
        ∧ LET msg    ≜ msgs[j][i][1]
              infoOk ≜ ∨ msg.mlastEpoch > tempMaxLastEpoch[i]
                       ∨ ∧ msg.mlastEpoch = tempMaxLastEpoch[i]
```

```
                        $\land \lor LastZxid(msg.mhf)[1] > LastZxid(tempInitialHistory[i])[1]$
                           $\lor \land LastZxid(msg.mhf)[1] = LastZxid(tempInitialHistory[i])[1]$
                              $\land LastZxid(msg.mhf)[2] \geq LastZxid(tempInitialHistory[i])[2]$
        IN  $\lor \land leaderEpoch[i] = msg.mepoch$
               $\land \lor \land infoOk$
                        $\land tempMaxLastEpoch' = [tempMaxLastEpoch$ EXCEPT $![i] = msg.mlastEpoch]$
                        $\land tempInitialHistory' = [tempInitialHistory$ EXCEPT $![i] = msg.mhf]$
                  $\lor \land \lnot infoOk$
                        $\land$ UNCHANGED $\langle tempMaxLastEpoch, tempInitialHistory \rangle$
               Followers not in $Q$ will not receive $NEWEPOCH$, so leader will receive $ACKE$ only when the source is in 
               $\land ackeRecv' = [ackeRecv$ EXCEPT $![i] =$ IF $j \notin ackeRecv[i]$ THEN $ackeRecv[i] \cup \{j\}$
                                                                     ELSE $ackeRecv[i]]$
            $\lor \land leaderEpoch[i] \neq msg.mepoch$
               $\land$ UNCHANGED $\langle tempMaxLastEpoch, tempInitialHistory, ackeRecv \rangle$
    $\land Discard(j, i)$
    $\land$ UNCHANGED $\langle serverVars, cluster, cepochRecv, ackldRecv, ackIndex, currentCounter,$
                $sendCounter, initialHistory, committedIndex, cepochSent, tempMaxEpoch, recoveryV$

$LeaderDiscovery2Sync1(i) \triangleq$
    test restrictions
    $\land currentEpoch[i] \leq 2$
    $\land Len(history[i]) \leq 2$
    $\land state[i] = ProspectiveLeader$
    $\land ackeRecv[i] \in Quorums$
    $\land currentEpoch' = [currentEpoch$ EXCEPT $![i] = leaderEpoch[i]]$
    $\land history' = [history$ EXCEPT $![i] = tempInitialHistory[i]]$
    $\land initialHistory' = [initialHistory$ EXCEPT $![i] = tempInitialHistory[i]]$
    $\land ackeRecv' = [ackeRecv$ EXCEPT $![i] = \{NullPoint\}]$
    $\land ackIndex' = [ackIndex$ EXCEPT $![i][i] = Len(tempInitialHistory[i])]$
    until now, $phase1(Discovery)$ ends
    $\land Broadcast(i, [mtype \mapsto NEWLEADER,$
                      $mepoch \mapsto currentEpoch'[i],$
                      $minitialHistory \mapsto history'[i]])$
    $\land$ UNCHANGED $\langle state, leaderEpoch, leaderOracle, commitIndex, cluster, cepochRecv, ackldRecv,$
                $currentCounter, sendCounter, committedIndex, cepochSent, tempVars, recoveryVars,$
```

*Note*1: Delete the change of *commitIndex* in *LeaderDiscovery2Sync*1 and *FollowerSync*1, then we can promise that *commitIndex* of every server increases monotonically, except that some server halts and restarts.

*Note*2: Set *cepochRecv*, *ackeRecv*, *ackldRecv* to $\{NullPoint\}$ in corresponding three actions to make sure that the prospective leader will not broadcast $NEWEPOCH/NEWLEADER/COMMITLD$ twice.

In phase $f21$, follower receives $NEWLEADER$. The follower updates its epoch and history, and sends back ACK-LD to pleader.

```
$FollowerSync1(i, j) \triangleq$
    test restrictions
```

```
        $\land currentEpoch[i] \leq 2$
        $\land Len(history[i]) \leq 2$
        $\land state[i] = Follower$
        $\land msgs[j][i] \neq \langle\rangle$
        $\land msgs[j][i][1].mtype = NEWLEADER$
        $\land$ LET $msg \triangleq msgs[j][i][1]$
                $replyOk \triangleq \land currentEpoch[i] \leq msg.mepoch$
                            $\land leaderOracle[i] = j$
          IN  $\lor$ new $NEWLEADER -$ accept and reply
                $\land replyOk$
                $\land currentEpoch' = [currentEpoch$ EXCEPT $![i] = msg.mepoch]$
                $\land leaderEpoch' = [leaderEpoch$ EXCEPT $![i] = msg.mepoch]$
                $\land history' = [history$ EXCEPT $![i] = msg.minitialHistory]$
                $\land Reply(i, j, [mtype \mapsto ACKLD,$
                                $mepoch \mapsto msg.mepoch,$
                                $mhistory \mapsto msg.minitialHistory])$
              $\lor$ stale $NEWLEADER -$ discard
                $\land \neg replyOk$
                $\land Discard(j, i)$
                $\land$ UNCHANGED $\langle currentEpoch, leaderEpoch, history\rangle$
        $\land$ UNCHANGED $\langle state, commitIndex, leaderOracle, leaderVars, tempVars, cepochSent, recoveryVars, p$
```

In phase $l22$, pleader receives ACK-LD from a quorum of followers, and sends $COMMIT$-LD to followers.

```
$LeaderHandleACKLD(i, j) \triangleq$
        test restrictions
        $\land currentEpoch[i] \leq 2$
        $\land Len(history[i]) \leq 2$
        $\land state[i] = ProspectiveLeader$
        $\land msgs[j][i] \neq \langle\rangle$
        $\land msgs[j][i][1].mtype = ACKLD$
        $\land$ LET $msg \triangleq msgs[j][i][1]$
          IN  $\lor$ new $ACK$-LD - accept
                $\land currentEpoch[i] = msg.mepoch$
                $\land ackIndex' = [ackIndex$ EXCEPT $![i][j] = Len(initialHistory[i])]$
                $\land ackldRecv' = [ackldRecv$ EXCEPT $![i] =$ IF $j \notin ackldRecv[i]$ THEN $ackldRecv[i] \cup \{j\}$
                                                                        ELSE $ackldRecv[i]]$
              $\lor$ stale $ACK$-LD - discard
                $\land currentEpoch[i] \neq msg.mepoch$
                $\land$ UNCHANGED $\langle ackldRecv, ackIndex\rangle$
        $\land Discard(j, i)$
        $\land$ UNCHANGED $\langle serverVars, cluster, cepochRecv, ackeRecv, currentCounter,$
                        $sendCounter, initialHistory, committedIndex, tempVars, cepochSent, recoveryVars, pr$

$LeaderSync2(i) \triangleq$
        test restrictions
        ∧ currentEpoch[i] ≤ 2
        ∧ Len(history[i]) ≤ 2
        ∧ state[i] = ProspectiveLeader
        ∧ ackldRecv[i] ∈ Quorums
        ∧ commitIndex'    = [commitIndex    EXCEPT ![i] = Len(history[i])]
        ∧ committedIndex' = [committedIndex EXCEPT ![i] = Len(history[i])]
        ∧ state'          = [state          EXCEPT ![i] = Leader]
        ∧ currentCounter' = [currentCounter EXCEPT ![i] = 0]
        ∧ sendCounter'    = [sendCounter    EXCEPT ![i] = 0]
        ∧ ackldRecv'      = [ackldRecv      EXCEPT ![i] = {NullPoint}]
        ∧ Broadcast(i, [mtype   ↦ COMMITLD,
                        mepoch  ↦ currentEpoch[i],
                        mlength ↦ Len(history[i])])
        ∧ UNCHANGED ⟨currentEpoch, leaderEpoch, leaderOracle, history, cluster, cepochRecv,
                     ackeRecv, ackIndex, initialHistory, tempVars, cepochSent, recoveryVars, proposalMsg
```

In phase $f22$, follower receives *COMMIT*-LD and delivers all unprocessed transaction.

```
FollowerSync2(i, j) ≜
        test restrictions
        ∧ currentEpoch[i] ≤ 2
        ∧ Len(history[i]) ≤ 2
        ∧ state[i] = Follower
        ∧ msgs[j][i] ≠ ⟨⟩
        ∧ msgs[j][i][1].mtype = COMMITLD
        ∧ LET msg ≜ msgs[j][i][1]
              replyOk ≜ ∧ currentEpoch[i] = msg.mepoch
                        ∧ leaderOracle[i] = j
          IN  ∨ new COMMIT-LD - commit all transactions in initial history
                Regradless of Restart, it must be true because one will receive NEWLEADER before receiving COMMIT-
                ∧ replyOk
                ∧ ∨ ∧ Len(history[i]) = msg.mlength
                    ∧ commitIndex' = [commitIndex EXCEPT ![i] = Len(history[i])]
                    ∧ Discard(j, i)
                  ∨ ∧ Len(history[i]) ≠ msg.mlength
                    ∧ Reply(i, j, [mtype  ↦ CEPOCH,
                                   mepoch ↦ currentEpoch[i]])
                    ∧ UNCHANGED commitIndex
              ∨ > : stale COMMIT-LD - discard
                < : In our implementation, '<' does not exist due to the guarantee of Restart
                ∧ ¬replyOk
                ∧ Discard(j, i)
                ∧ UNCHANGED commitIndex
        ∧ UNCHANGED ⟨state, currentEpoch, leaderEpoch, leaderOracle, history,
                     leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog⟩
```

```
 In phase l31, leader receives client request and broadcasts PROPOSE.
ClientRequest(i, v) ≜
          test restrictions
          ∧ currentEpoch[i] ≤ 2
          ∧ Len(history[i])  ≤ 1
          ∧ state[i] = Leader
          ∧ currentCounter′ = [currentCounter EXCEPT ![i] = currentCounter[i] + 1]
          ∧ LET newTransaction ≜ [epoch    ↦ currentEpoch[i],
                                  counter ↦ currentCounter′[i],
                                  value    ↦ v]
             IN   ∧ history′    = [history  EXCEPT ![i] = Append(history[i], newTransaction)]
                  ∧ ackIndex′ = [ackIndex EXCEPT ![i][i] = Len(history′[i])]  necessary, to push commitIndex
          ∧ UNCHANGED ⟨msgs, state, currentEpoch, leaderEpoch, leaderOracle, commitIndex, cluster, cepochRe
                                 ackeRecv, ackldRecv, sendCounter, initialHistory, committedIndex, tempVars, cepoch

LeaderBroadcast1(i) ≜
          test restrictions
          ∧ currentEpoch[i] ≤ 2
          ∧ Len(history[i])  ≤ 2
          ∧ state[i] = Leader
          ∧ sendCounter[i] < currentCounter[i]
          ∧ LET toBeSentCounter  ≜ sendCounter[i] + 1
                toBeSentIndex     ≜ Len(initialHistory[i]) + toBeSentCounter
                toBeSentEntry     ≜ history[i][toBeSentIndex]
             IN   ∧ Broadcast(i, [mtype      ↦ PROPOSE,
                                  mepoch     ↦ currentEpoch[i],
                                  mproposal ↦ toBeSentEntry])
                  ∧ sendCounter′ = [sendCounter EXCEPT ![i] = toBeSentCounter]
                  ∧ LET m ≜ [msource ↦ i, mtype ↦ PROPOSE, mepoch ↦ currentEpoch[i], mproposal ↦ toB
                      IN   proposalMsgsLog′ = proposalMsgsLog ∪ {m}
          ∧ UNCHANGED ⟨serverVars, cepochRecv, cluster, ackeRecv, ackldRecv, ackIndex,
                                 currentCounter, initialHistory, committedIndex, tempVars, recoveryVars, cepochSent

 In phase f31, follower accepts proposal and append it to history.
FollowerBroadcast1(i, j) ≜
          test restrictions
          ∧ currentEpoch[i] ≤ 2
          ∧ Len(history[i])  ≤ 2
          ∧ state[i] = Follower
          ∧ msgs[j][i] ≠ ⟨⟩
          ∧ msgs[j][i][1].mtype = PROPOSE
          ∧ LET msg ≜ msgs[j][i][1]
                replyOk ≜ ∧ currentEpoch[i] = msg.mepoch
                           ∧ leaderOracle[i]  = j
```

```
        IN  ∨ It should be that ∨ msg.mproposal.counter = 1
                                ∨ msg.mrpoposal.counter = history[Len(history)].counter + 1
              ∧ replyOk
              ∧ history′ = [history EXCEPT ![i] = Append(history[i], msg.mproposal)]
              ∧ Reply(i, j, [mtype   ↦ ACK,
                             mepoch ↦ currentEpoch[i],
                             mindex ↦ Len(history′[i])])
            ∨ If happens, ≠ must be >, namely a stale leader sends it.
              ∧ ¬replyOk
              ∧ Discard(j, i)
              ∧ UNCHANGED history
     ∧ UNCHANGED ⟨state, currentEpoch, leaderEpoch, leaderOracle, commitIndex,
                   leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog⟩

In phase l32, leader receives ack from a quorum of followers to a certain proposal,
and commits the proposal.
LeaderHandleACK(i, j) ≜
     test restrictions
     ∧ currentEpoch[i] ≤ 2
     ∧ Len(history[i]) ≤ 2
     ∧ state[i] = Leader
     ∧ msgs[j][i] ≠ ⟨⟩
     ∧ msgs[j][i][1].mtype = ACK
     ∧ LET msg ≜ msgs[j][i][1]
        IN  ∨ It should be that ackIndex[i][j] + 1 ≜ msg.mindex
              ∧ currentEpoch[i] = msg.mepoch
              ∧ ackIndex′ = [ackIndex EXCEPT ![i][j] = Maximum({ackIndex[i][j], msg.mindex})]
            ∨ If happens, ≠ must be >, namely a stale follower sends it.
              ∧ currentEpoch[i] ≠ msg.mepoch
              ∧ UNCHANGED ackIndex
     ∧ Discard(j, i)
     ∧ UNCHANGED ⟨serverVars, cluster, cepochRecv, ackeRecv, ackldRecv, currentCounter,
                   sendCounter, initialHistory, committedIndex, tempVars, cepochSent, recoveryVars, pr

LeaderAdvanceCommit(i) ≜
     test restrictions
     ∧ currentEpoch[i] ≤ 2
     ∧ Len(history[i]) ≤ 2
     ∧ state[i] = Leader
     ∧ commitIndex[i] < Len(history[i])
     ∧ LET Agree(index)      ≜ {i} ∪ {k ∈ (Server \ {i}) : ackIndex[i][k] ≥ index}
           agreeIndexes      ≜ {index ∈ (commitIndex[i] + 1) .. Len(history[i]) : Agree(index) ∈ Quoru
           newCommitIndex    ≜ IF agreeIndexes ≠ {} THEN Maximum(agreeIndexes)
                                                    ELSE commitIndex[i]
        IN  commitIndex′ = [commitIndex EXCEPT ![i] = newCommitIndex]
```

```
          ∧ UNCHANGED ⟨state, currentEpoch, leaderEpoch, leaderOracle, history,
                        msgs, leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog⟩

LeaderBroadcast2(i) ≜
          test restrictions
          ∧ currentEpoch[i] ≤ 2
          ∧ Len(history[i])  ≤ 2
          ∧ state[i] = Leader
          ∧ committedIndex[i] < commitIndex[i]
          ∧ LET newCommittedIndex ≜ committedIndex[i] + 1
             IN   ∧ Broadcast(i, [mtype      ↦ COMMIT,
                                  mepoch     ↦ currentEpoch[i],
                                  mindex     ↦ newCommittedIndex,
                                  mcounter   ↦ history[i][newCommittedIndex].counter])
                  ∧ committedIndex′ = [committedIndex EXCEPT ![i] = committedIndex[i] + 1]
          ∧ UNCHANGED ⟨serverVars, cluster, cepochRecv, ackeRecv, ackldRecv, ackIndex, currentCounter,
                        sendCounter, initialHistory, tempVars, cepochSent, recoveryVars, proposalMsgsLog⟩

In phase f32, follower receives COMMIT and commits transaction.
FollowerBroadcast2(i, j) ≜
          test restrictions
          ∧ currentEpoch[i] ≤ 2
          ∧ Len(history[i])  ≤ 2
          ∧ state[i] = Follower
          ∧ msgs[j][i] ≠ ⟨⟩
          ∧ msgs[j][i][1].mtype = COMMIT
          ∧ msgs[j][i][1].mtype = COMMIT
          ∧ LET msg ≜ msgs[j][i][1]
                replyOk ≜ ∧ currentEpoch[i] = msg.mepoch
                          ∧ leaderOracle[i]  = j
             IN   ∨ ∧ replyOk
                    ∧ LET infoOk ≜ ∧ Len(history[i]) ≥ msg.mindex
                                   ∧ ∨ ∧ msg.mindex > 0
                                       ∧ history[i][msg.mindex].epoch = msg.mepoch
                                       ∧ history[i][msg.mindex].counter = msg.mcounter
                                     ∨ msg.mindex = 0
                       IN   ∨ new COMMIT − commit transaction in history
                              ∧ infoOk
                              ∧ commitIndex′ = [commitIndex EXCEPT ![i] = Maximum({commitIndex[i], msg.
                              ∧ Discard(j, i)
                            ∨ It may happen when the server is a new follower who joined in the cluster,
                              and it misses the corresponding PROPOSE.
                              ∧ ¬infoOk
                              ∧ Reply(i, j, [mtype   ↦ CEPOCH,
                                             mepoch ↦ currentEpoch[i]])
                    ∧ UNCHANGED commitIndex
             ∨  stale COMMIT −  discard
                ∧ ¬replyOk
                ∧ Discard(j, i)
                ∧ UNCHANGED commitIndex
          ∧ UNCHANGED ⟨state, currentEpoch, leaderEpoch, history, leaderOracle,
                       leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog⟩
```

There may be two ways to make sure all followers as up-to-date as the leader.
*way*1: choose *Send* not *Broadcast* when leader is going to send *PROPOSE* and *COMMIT*.
*way*2: When one follower receives *PROPOSE* or *COMMIT* which misses some entries between
  its history and the newest entry, the follower send *CEPOCH* to catch pace.
Here I choose *way*2, which I need not to rewrite *PROPOSE* and *COMMIT*, but need to
modify the code when follower receives *COMMIT*-LD and *COMMIT*.

In phase *l*33, upon receiving *CEPOCH*, leader l proposes back *NEWEPOCH* and *NEWLEADER*.

```
LeaderHandleCEPOCHinPhase3(i, j) ≜
          test restrictions
          ∧ currentEpoch[i] ≤ 2
          ∧ Len(history[i]) ≤ 2
          ∧ state[i] = Leader
          ∧ msgs[j][i] ≠ ⟨⟩
          ∧ msgs[j][i][1].mtype = CEPOCH
          ∧ LET msg ≜ msgs[j][i][1]
            IN  ∨ ∧ currentEpoch[i] ≥ msg.mepoch
                  ∧ Reply2(i, j, [mtype  ↦ NEWEPOCH,
                                  mepoch ↦ currentEpoch[i]],
                                 [mtype          ↦ NEWLEADER,
                                  mepoch         ↦ currentEpoch[i],
                                  minitialHistory ↦ history[i]])
                ∨ ∧ currentEpoch[i] < msg.mepoch
                  ∧ UNCHANGED msgs
          ∧ UNCHANGED ⟨serverVars, leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog⟩
```

In phase *l*34, upon receiving ack from f of the *NEWLEADER*, it sends a commit message to f.
Leader l also makes $Q := Q \cup \{f\}$.

```
LeaderHandleACKLDinPhase3(i, j) ≜
          test restrictions
          ∧ currentEpoch[i] ≤ 2
          ∧ Len(history[i]) ≤ 2
          ∧ state[i] = Leader
          ∧ msgs[j][i] ≠ ⟨⟩
          ∧ msgs[j][i][1].mtype = ACKLD
          ∧ LET msg ≜ msgs[j][i][1]
                aimCommitIndex ≜ Minimum({commitIndex[i], Len(msg.mhistory)})
```

```
            $aimCommitCounter \triangleq$ IF $aimCommitIndex = 0$ THEN $0$ ELSE $history[i][aimCommitIndex].co$
        IN  $\lor \land currentEpoch[i] = msg.mepoch$
              $\land ackIndex' = [ackIndex$ EXCEPT $![i][j] = Len(msg.mhistory)]$
              $\land Reply(i, j, [mtype \mapsto COMMIT,$
                                $mepoch \mapsto currentEpoch[i],$
                                $mindex \mapsto aimCommitIndex,$
                                $mcounter \mapsto aimCommitCounter])$
            $\lor \land currentEpoch[i] \neq msg.mepoch$
              $\land Discard(j, i)$
              $\land$ UNCHANGED $ackIndex$
    $\land cluster' = [cluster$ EXCEPT $![i] =$ IF $j \in cluster[i]$ THEN $cluster[i]$
                                                          ELSE $cluster[i] \cup \{j\}]$
    $\land$ UNCHANGED $\langle serverVars, cepochRecv, ackeRecv, ackldRecv, currentCounter, sendCounter,$
                   $initialHistory, committedIndex, tempVars, cepochSent, recoveryVars, proposalMsgsLo$
```

To ensure any follower can find the correct leader, the follower should modify *leaderOracle* anytime when it receive messages from leader, because a server may restart and join the cluster Q halfway and receive the first message which is not *NEWEPOCH*. But we can delete this restriction when we ensure *Broadcast* function acts on the followers in the cluster not any servers in the whole system, then one server must has correct *leaderOracle* before it receives messages.

Let me suppose two conditions when one follower sends *CEPOCH* to leader:
0. Usually, the server becomes follower in election and sends *CEPOCH* before receiving *NEWEPOCH*.
1. The follower wants to join the cluster halfway and get the newest history.
2. The follower has received *COMMIT*, but there exists the gap between its own history and *mindex*,
   which means there are some transactions before *mindex* miss. Here we choose to send *CEPOCH*
   again, to receive the newest history from leader.

```
$BecomeFollower(i) \triangleq$
    test restrictions
    $\land currentEpoch[i] \leq 2$
    $\land Len(history[i]) \leq 2$
    $\land state[i] \neq Follower$
    $\land \exists j \in Server \setminus \{i\} : \land msgs[j][i] \neq \langle\rangle$
                              $\land msgs[j][i][1].mtype \neq RECOVERYREQUEST$
                              $\land msgs[j][i][1].mtype \neq RECOVERYRESPONSE$
                              $\land$ LET $msg \triangleq msgs[j][i][1]$
                                IN  $\land NullPoint \in cepochRecv[i]$
                                    $\land Maximum(\{currentEpoch[i], leaderEpoch[i]\}) < msg.mepoch$
                                    $\land \lor msg.mtype = NEWEPOCH$
                                      $\lor msg.mtype = NEWLEADER$
                                      $\lor msg.mtype = COMMITLD$
                                      $\lor msg.mtype = PROPOSE$
                                      $\lor msg.mtype = COMMIT$
                                    $\land state' = [state$ EXCEPT $![i] = Follower]$
                                    $\land currentEpoch' = [currentEpoch$ EXCEPT $![i] = msg.mepoch]$
```

```
                        ∧ leaderOracle′  = [leaderOracle EXCEPT ![i]  = j]
                        ∧ Reply(i, j, [mtype   ↦ CEPOCH,
                                       mepoch ↦ currentEpoch[i]])
                        Here we should not use Discard.
      ∧ UNCHANGED ⟨leaderEpoch, history, commitIndex, leaderVars, tempVars, cepochSent, recoveryVars,
```

```
DiscardStaleMessage(i) ≜
      test restrictions
      ∧ currentEpoch[i] ≤ 2
      ∧ Len(history[i])  ≤ 2
      ∧ ∃ j ∈ Server \ {i} : ∧ msgs[j][i] ≠ ⟨⟩
                             ∧ msgs[j][i][1].mtype ≠ RECOVERYREQUEST
                             ∧ msgs[j][i][1].mtype ≠ RECOVERYRESPONSE
                             ∧ LET msg ≜ msgs[j][i][1]
                               IN   ∨ ∧ state[i] = Follower
                                      ∧  ∨ msg.mepoch < currentEpoch[i] \ * Discussed before.
                                         ∨ msg.mtype = CEPOCH
                                         ∨ msg.mtype = ACKE
                                         ∨ msg.mtype = ACKLD
                                         ∨ msg.mtype = ACK
                                    ∨ ∧ state[i] ≠ Follower
                                      ∧ msg.mtype ≠ CEPOCH
                                      ∧ ∨ ∧ state[i] = ProspectiveLeader
                                          ∧ ∨ msg.mtype = ACK
                                            ∨ ∧ msg.mepoch ≤ Maximum({currentEpoch[i], leaderEpoch[i
                                              ∧ ∨ msg.mtype = NEWEPOCH
                                                ∨ msg.mtype = NEWLEADER
                                                ∨ msg.mtype = COMMITLD
                                                ∨ msg.mtype = PROPOSE
                                                ∨ msg.mtype = COMMIT
                                        ∨ ∧ state[i] = Leader
                                          ∧ ∨ msg.mtype = ACKE
                                            ∨ ∧ msg.mepoch ≤ currentEpoch[i]
                                              ∧ ∨ msg.mtype = NEWEPOCH
                                                ∨ msg.mtype = NEWLEADER
                                                ∨ msg.mtype = COMMITLD
                                                ∨ msg.mtype = PROPOSE
                                                ∨ msg.mtype = COMMIT
                             ∧ Discard(j, i)
      ∧ UNCHANGED ⟨serverVars, leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog⟩
```

```
 Defines how the variables may transition.
Next ≜
```

$$\begin{array}{ll}
\lor \exists i \in \mathit{Server},\ Q \in \mathit{Quorums} : \mathit{InitialElection}(i,\ Q) & \\
\lor \exists i \in \mathit{Server} : & \mathit{Restart}(i) \\
\lor \exists i \in \mathit{Server} : & \mathit{RecoveryAfterRestart}(i) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{HandleRecoveryRequest}(i,\ j) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{HandleRecoveryResponse}(i,\ j) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{FindCluster}(i) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{LeaderTimeout}(i,\ j) \\
\lor \exists i \in \mathit{Server} : & \mathit{FollowerTimeout}(i) \\
\lor \exists i \in \mathit{Server} : & \mathit{FollowerDiscovery}1(i) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{LeaderHandleCEPOCH}(i,\ j) \\
\lor \exists i \in \mathit{Server} : & \mathit{LeaderDiscovery}1(i) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{FollowerDiscovery}2(i,\ j) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{LeaderHandleACKE}(i,\ j) \\
\lor \exists i \in \mathit{Server} : & \mathit{LeaderDiscovery}2\mathit{Sync}1(i) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{FollowerSync}1(i,\ j) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{LeaderHandleACKLD}(i,\ j) \\
\lor \exists i \in \mathit{Server} : & \mathit{LeaderSync}2(i) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{FollowerSync}2(i,\ j) \\
\lor \exists i \in \mathit{Server},\ v \in \mathit{Value} : \mathit{ClientRequest}(i,\ v) & \\
\lor \exists i \in \mathit{Server} : & \mathit{LeaderBroadcast}1(i) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{FollowerBroadcast}1(i,\ j) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{LeaderHandleACK}(i,\ j) \\
\lor \exists i \in \mathit{Server} : & \mathit{LeaderAdvanceCommit}(i) \\
\lor \exists i \in \mathit{Server} : & \mathit{LeaderBroadcast}2(i) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{FollowerBroadcast}2(i,\ j) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{LeaderHandleCEPOCHinPhase}3(i,\ j) \\
\lor \exists i, j \in \mathit{Server} : & \mathit{LeaderHandleACKLDinPhase}3(i,\ j) \\
\lor \exists i \in \mathit{Server} : & \mathit{DiscardStaleMessage}(i) \\
\lor \exists i \in \mathit{Server} : & \mathit{BecomeFollower}(i)
\end{array}$$

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

Define some variants, safety propoties, and liveness propoties of *Zab* consensus algorithm.

Safety properties

There is most one leader/prospective leader in a certain epoch.

$$\begin{array}{l}
\mathit{Leadership} \triangleq \forall i, j \in \mathit{Server} : \\
\quad \land \lor \mathit{state}[i] = \mathit{Leader} \\
\quad \lor \land \mathit{state}[i] = \mathit{ProspectiveLeader} \\
\quad \land \mathit{NullPoint} \in \mathit{ackeRecv}[i] \quad \text{prospective leader determines its epoch after broadcasting } \mathit{NEWLE} \\
\quad \land \lor \mathit{state}[j] = \mathit{Leader} \\
\quad \lor \land \mathit{state}[j] = \mathit{ProspectiveLeader} \\
\quad \land \mathit{NullPoint} \in \mathit{ackeRecv}[j] \\
\quad \land \mathit{currentEpoch}[i] = \mathit{currentEpoch}[j]
\end{array}$$

$$\Rightarrow i = j$$

Here, delivering means deliver some transaction from history to replica. We can assume $\mathit{deliverIndex} = \mathit{commitIndex}$.
So we can assume the set of delivered transactions is the prefix of history with index from 1 to *commitIndex*.
We can express a transaction by two-tuple $< \mathit{epoch},\mathit{counter} >$ according to its uniqueness.

$$\begin{aligned}
&\mathit{equal}(\mathit{entry}1,\ \mathit{entry}2) \triangleq\ \land\ \mathit{entry}1.\mathit{epoch} \quad\ = \mathit{entry}2.\mathit{epoch}\\
&\qquad\qquad\qquad\qquad\qquad\quad \land\ \mathit{entry}1.\mathit{counter} = \mathit{entry}2.\mathit{counter}
\end{aligned}$$

$$\begin{aligned}
&\mathit{precede}(\mathit{entry}1,\ \mathit{entry}2) \triangleq\ \lor\ \mathit{entry}1.\mathit{epoch} < \mathit{entry}2.\mathit{epoch}\\
&\qquad\qquad\qquad\qquad\qquad\quad \lor\ \land\ \mathit{entry}1.\mathit{epoch} \quad\ = \mathit{entry}2.\mathit{epoch}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ \mathit{entry}1.\mathit{counter} < \mathit{entry}2.\mathit{counter}
\end{aligned}$$

PrefixConsistency: The prefix that have been delivered in history in any process is the same.

$$\begin{aligned}
&\mathit{PrefixConsistency} \triangleq\ \forall\, i,\ j \in \mathit{Server} :\\
&\qquad\qquad \text{LET } \mathit{smaller} \triangleq \mathit{Minimum}(\{\mathit{commitIndex}[i],\ \mathit{commitIndex}[j]\})\\
&\qquad\qquad \text{IN} \quad \lor\ \mathit{smaller} \quad = 0\\
&\qquad\qquad\qquad\ \lor\ \land\ \mathit{smaller} > 0\\
&\qquad\qquad\qquad\quad\ \land\ \forall\, \mathit{index} \in 1 \,..\, \mathit{smaller} : \mathit{equal}(\mathit{history}[i][\mathit{index}],\ \mathit{history}[j][\mathit{index}])
\end{aligned}$$

Integrity: If some follower delivers one transaction, then some primary has broadcast it.

$$\begin{aligned}
&\mathit{Integrity} \triangleq\ \forall\, i \in \mathit{Server} :\\
&\qquad \mathit{state}[i] = \mathit{Follower} \land \mathit{commitIndex}[i] > 0\\
&\qquad \Rightarrow \forall\, \mathit{index} \in 1 \,..\, \mathit{commitIndex}[i] : \exists\, \mathit{msg} \in \mathit{proposalMsgsLog} :\\
&\qquad\qquad \mathit{equal}(\mathit{msg}.\mathit{mproposal},\ \mathit{history}[i][\mathit{index}])
\end{aligned}$$

Agreement: If some follower f delivers transaction a and some follower f' delivers transaction b,
then f' delivers a or f delivers b.

$$\begin{aligned}
&\mathit{Agreement} \triangleq\ \forall\, i,\ j \in \mathit{Server} :\\
&\qquad \land\ \mathit{state}[i] = \mathit{Follower} \land \mathit{commitIndex}[i] > 0\\
&\qquad \land\ \mathit{state}[j] = \mathit{Follower} \land \mathit{commitIndex}[j] > 0\\
&\qquad \Rightarrow\\
&\qquad \forall\, \mathit{index}1 \in 1 \,..\, \mathit{commitIndex}[i],\ \mathit{index}2 \in 1 \,..\, \mathit{commitIndex}[j] :\\
&\qquad\quad \lor\ \exists\, \mathit{indexj} \in 1 \,..\, \mathit{commitIndex}[j] :\\
&\qquad\qquad \mathit{equal}(\mathit{history}[j][\mathit{indexj}],\ \mathit{history}[i][\mathit{index}1])\\
&\qquad\quad \lor\ \exists\, \mathit{indexi} \in 1 \,..\, \mathit{commitIndex}[i] :\\
&\qquad\qquad \mathit{equal}(\mathit{history}[i][\mathit{indexi}],\ \mathit{history}[j][\mathit{index}2])
\end{aligned}$$

Total order: If some follower delivers a before b, then any process that delivers b
must also deliver a and deliver a before b.

$$\begin{aligned}
&\mathit{TotalOrder} \triangleq\ \forall\, i,\ j \in \mathit{Server} : \mathit{commitIndex}[i] \geq 2 \land \mathit{commitIndex}[j] \geq 2\\
&\qquad \Rightarrow \forall\, \mathit{indexi}1 \in 1 \,..\, (\mathit{commitIndex}[i] - 1) : \forall\, \mathit{indexi}2 \in (\mathit{indexi}1 + 1) \,..\, \mathit{commitIndex}[i] :\\
&\qquad\quad \text{LET } \mathit{logOk} \triangleq \exists\, \mathit{index} \in 1 \,..\, \mathit{commitIndex}[j] : \mathit{equal}(\mathit{history}[i][\mathit{indexi}2],\ \mathit{history}[j][\mathit{index}]\\
&\qquad\quad \text{IN} \quad \lor\ \neg \mathit{logOk}\\
&\qquad\qquad\quad\ \lor\ \land\ \mathit{logOk}\\
&\qquad\qquad\qquad \land\ \exists\, \mathit{indexj}2 \in 1 \,..\, \mathit{commitIndex}[j] :\\
&\qquad\qquad\qquad\qquad\qquad\qquad \land\ \mathit{equal}(\mathit{history}[i][\mathit{indexi}2],\ \mathit{history}[j][\mathit{indexj}2])\\
&\qquad\qquad\qquad\qquad\qquad\qquad \land\ \exists\, \mathit{indexj}1 \in 1 \,..\, (\mathit{indexj}2 - 1) : \mathit{equal}(\mathit{history}[i][\mathit{indexi}1],\ \mathit{histo}
\end{aligned}$$

Local primary order: If a primary broadcasts a before it broadcasts b, then a follower that delivers b must also deliver a before b.

```
LocalPrimaryOrder ≜ LET mset(i, e) ≜ {msg ∈ proposalMsgsLog : msg.msource = i ∧ msg.mproposal.epoch
                        mentries(i, e) ≜ {msg.mproposal : msg ∈ mset(i, e)}
                    IN  ∀ i ∈ Server : ∀ e ∈ 1 . . currentEpoch[i] :
                          ∧ Cardinality(mentries(i, e)) ≥ 2
                          ∧ ∃ tsc1 ∈ mentries(i, e) : ∃ tsc2 ∈ mentries(i, e) :
                               ∧ ¬equal(tsc2, tsc1)
                               ∧ LET tscPre  ≜ IF precede(tsc1, tsc2) THEN tsc1 ELSE tsc2
                                     tscNext ≜ IF precede(tsc1, tsc2) THEN tsc2 ELSE tsc1
                                 IN  ∀ j ∈ Server : ∧ commitIndex[j] ≥ 2
                                                    ∧ ∃ index ∈ 1 . . commitIndex[j] : equal(history[j][inde
                                   ⇒
                                    ∃ index2 ∈ 1 . . commitIndex[j] :
                                        ∧ equal(history[j][index2], tscNext)
                                        ∧ index2 > 1
                                        ∧ ∃ index1 ∈ 1 . . (index2 − 1) : equal(history[j][index1], tscPre)
```

Global primary order: A follower f delivers both a with epoch e and b with epoch e', and $e < e'$, then f must deliver a before b.

```
GlobalPrimaryOrder ≜ ∀ i ∈ Server : commitIndex[i] ≥ 2
                        ⇒ ∀ idx1, idx2 ∈ 1 . . commitIndex[i] : ∨ history[i][idx1].epoch ≥ history[i][idx2].ep
                                                                ∨ ∧ history[i][idx1].epoch < history[i][idx2]
                                                                  ∧ idx1 < idx2
```

Primary integrity: If primary p broadcasts a and some follower f delivers b such that b has epoch smaller than epoch of p, then p must deliver b before it broadcasts a.

```
PrimaryIntegrity ≜ ∀ i, j ∈ Server : ∧ state[i] = Leader
                                     ∧ state[j] = Follower ∧ commitIndex[j] ≥ 1
                      ⇒ ∀ index ∈ 1 . . commitIndex[j] : ∨ history[j][index].epoch ≥ currentEpoch[i]
                                                         ∨ ∧ history[j][index].epoch < currentEpoch[i]
                                                           ∧ ∃ idx ∈ 1 . . commitIndex[i] : equal(history[i][i
```

Liveness property

Suppose that :

– A quorum Q of followers are up.

– The followers in Q elect the same process l and l is up.

– Messages between a follower in Q and l are received in a timely fashion.

If l proposes a transaction a, then a is eventually committed.

\ ∗ Modification History

\ ∗ Last modified *Thu Apr* 29 22:54:49 *CST* 2021 by Dell

\ ∗ Created Sat *Dec* 05 13:32:08 *CST* 2020 by Dell