MODULE *ZabWithQ*

This is the formal specification for the *Zab* consensus algorithm,
which means *Zookeeper* Atomic *Broadcast*.

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

$Quorums \triangleq \{Q \in \text{SUBSET } Server : Cardinality(Q) * 2 > Cardinality(Server)\}$
$\text{ASSUME } QuorumsAssumption \triangleq \land \forall\, Q \in Quorums : Q \subseteq Server$
$\land \forall\, Q1,\, Q2 \in Quorums : Q1 \cap Q2 \neq \{\}$

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

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

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*

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

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

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

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

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
$Broadcast(i,\ m) \triangleq msgs' = [ii \in Server \mapsto [ij \in Server \mapsto \text{IF } \wedge ii = i$
$\wedge ij \neq i$

$\land\ ij \in cluster[i]$ THEN $Append(msgs[ii][ij],\ m)$
ELSE $msgs[ii][ij]]]$

$BroadcastToAll(i,\ m) \triangleq msgs' = [ii \in Server \mapsto [ij \in Server \mapsto$ IF $\land\ ii = i \land ij \neq i$ THEN $Append(msgs[ii][i$
ELSE $msgs[ii][ij]]]$

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

$Reply(i,\ j,\ m) \triangleq msgs' = [msgs$ EXCEPT $![j][i] = Tail(msgs[j][i]),$
$![i][j] = Append(msgs[i][j],\ m)]$

$Reply2(i,\ j,\ m1,\ m2) \triangleq msgs' = [msgs$ EXCEPT $![j][i] = Tail(msgs[j][i]),$
$![i][j] = Append(Append(msgs[i][j],\ m1),\ m2)]$

$clean(i,\ j) \triangleq msgs' = [msgs$ EXCEPT $![i][j] = \langle\rangle,\ ![j][i] = \langle\rangle]$

Define initial values for all variables

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

A server becomes pleader and a quorum servers knows that.

$$
\begin{aligned}
Election(i,\ Q) \triangleq\ & \\
& \land i \in Q \\
& \land state' && = [s \in Server \mapsto \text{IF } s = i \text{ THEN } ProspectiveLeader
\end{aligned}
$$

```
                                                    ELSE IF s ∈ Q THEN Follower
                                                         ELSE state[s]]
  ∧ cluster'          = [cluster     EXCEPT ![i]  = Q]  cluster is first initialized in election, not phase1.
  ∧ cepochRecv'       = [cepochRecv  EXCEPT ![i] = {i}]
  ∧ ackeRecv'         = [ackeRecv    EXCEPT ![i] = {i}]
  ∧ ackldRecv'        = [ackldRecv   EXCEPT ![i] = {i}]
  ∧ ackIndex'         = [ii ∈ Server ↦ [ij ∈ Server ↦
                                IF ii = i THEN 0
                                     ELSE ackIndex[ii][ij]]]
  ∧ committedIndex'   = [committedIndex      EXCEPT ![i]  = 0]
  ∧ initialHistory'   = [initialHistory      EXCEPT ![i]  = ⟨⟩]
  ∧ tempMaxEpoch'     = [tempMaxEpoch        EXCEPT ![i] = currentEpoch[i]]
  ∧ tempMaxLastEpoch' = [tempMaxLastEpoch    EXCEPT ![i] = currentEpoch[i]]
  ∧ tempInitialHistory' = [tempInitialHistory EXCEPT ![i]  = history[i]]
  ∧ leaderOracle'     = [s ∈ Server ↦ IF s ∈ Q THEN i
                                     ELSE leaderOracle[s]]
  ∧ leaderEpoch'      = [s ∈ Server ↦ IF s ∈ Q THEN currentEpoch[s]
                                     ELSE leaderEpoch[s]]
  ∧ cepochSent'       = [s ∈ Server ↦ IF s ∈ Q THEN FALSE
                                     ELSE cepochSent[s]]
  ∧ msgs'             = [ii ∈ Server ↦ [ij  ∈ Server ↦
                                IF ii ∈ Q ∧ ij ∈ Q THEN ⟨⟩
                                     ELSE msgs[ii][ij]]]
  ∧ UNCHANGED ⟨currentEpoch, history, commitIndex, currentCounter, sendCounter, proposalMsgsLog⟩
```

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) ≜
    ∧ ∀ s ∈ Server : state[i] = Follower ∧ leaderOracle[i] = NullPoint
    ∧ Election(i, Q)
    ∧ UNCHANGED ⟨currentEpoch, history, commitIndex, currentCounter, sendCounter, recoveryVars, pr
```

The leader finds timeout with another follower.

```
LeaderTimeout(i, j) ≜
    ∧ 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, le
            ∨ ∧ newCluster ∉ Quorums
              ∧  LET Q ≜ CHOOSE q ∈ Quorums: i ∈ q
```

$v \triangleq \text{CHOOSE } s \in Q\text{: TRUE}$
IN $Election(v, Q)$
$\exists\, Q \in Quorums : \land\, i \in Q$
$\land\, \exists\, v \in Q : Election(v, Q)$
$\land$ UNCHANGED $\langle currentEpoch, history, commitIndex, currentCounter, sendCounter, recoveryVars, pr$

A follower finds timeout with the leader.

$FollowerTimeout(i) \triangleq$
$\land\, state[i] = Follower$
$\land\, leaderOracle[i] \neq NullPoint$
$\land\, \exists\, Q \in Quorums : \land\, i \in Q$
$\land\, \exists\, v \in Q : Election(v, Q)$
$\land$ UNCHANGED $\langle currentEpoch, history, commitIndex, currentCounter, sendCounter, recoveryVars, pr$

A server halts and restarts.

$Restart(i) \triangleq$
$\land\, state'$ $= [state$ EXCEPT $![i]$ $= Follower]$
$\land\, leaderOracle'$ $= [leaderOracle$ EXCEPT $![i]$ $= NullPoint]$
$\land\, commitIndex'$ $= [commitIndex$ EXCEPT $![i] = 0]$
$\land\, cepochSent'$ $= [cepochSent$ EXCEPT $![i]$ $=$ FALSE$]$
$\land\, msgs'$ $= [ii \in Server \mapsto [ij \in Server \mapsto$ IF $ij = i$ THEN $\langle\rangle$
ELSE $msgs[ii][ij]]]$
$\land$ UNCHANGED $\langle currentEpoch, leaderEpoch, history, leaderVars, tempVars, recoveryVars, proposalMs$

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.

$RecoveryAfterRestart(i) \triangleq$
$\land\, state[i] = Follower$
$\land\, leaderOracle[i] = NullPoint$
$\land\, recoveryRespRecv'$ $= [recoveryRespRecv$ EXCEPT $![i]$ $= \{\}]$
$\land\, recoveryMaxEpoch'$ $= [recoveryMaxEpoch$ EXCEPT $![i] = currentEpoch[i]]$
$\land\, recoveryMEOracle'$ $= [recoveryMEOracle$ EXCEPT $![i] = NullPoint]$
$\land\, BroadcastToAll(i, [mtype \mapsto RECOVERYREQUEST])$
$\land$ UNCHANGED $\langle serverVars, leaderVars, tempVars, cepochSent, proposalMsgsLog\rangle$

$HandleRecoveryRequest(i, j) \triangleq$
$\land\, msgs[j][i] \neq \langle\rangle$
$\land\, msgs[j][i][1].mtype = RECOVERYREQUEST$
$\land\, Reply(i, j, [mtype$ $\mapsto RECOVERYRESPONSE,$
$moracle \mapsto leaderOracle[i],$
$mepoch \mapsto currentEpoch[i]])$
$\land$ UNCHANGED $\langle serverVars, leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog\rangle$

$HandleRecoveryResponse(i, j) \triangleq$
$\land\, msgs[j][i] \neq \langle\rangle$

$\land\ msgs[j][i][1].mtype = RECOVERYRESPONSE$
$\land$ LET $msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \land\ msg.mepoch \geq recoveryMaxEpoch[i]$
$\land\ msg.moracle \neq NullPoint$
IN $\lor \land\ infoOk$
$\land\ recoveryMaxEpoch' = [recoveryMaxEpoch$ EXCEPT $![i] = msg.mepoch]$
$\land\ recoveryMEOracle' = [recoveryMEOracle$ EXCEPT $![i] = msg.moracle]$
$\lor \land \neg infoOk$
$\land$ UNCHANGED $\langle recoveryMaxEpoch, recoveryMEOracle\rangle$
$\land\ Discard(j, i)$
$\land\ recoveryRespRecv' = [recoveryRespRecv$ EXCEPT $![i] =$ IF $j \in recoveryRespRecv[i]$ THEN $recoveryRe$
ELSE $recoveryRe$
$\land$ UNCHANGED $\langle serverVars, leaderVars, tempVars, cepochSent, proposalMsgsLog\rangle$

$FindCluster(i) \triangleq$
$\land\ state[i] = Follower$
$\land\ leaderOracle[i] = NullPoint$
$\land\ recoveryRespRecv[i] \in Quorums$
$\land$ LET $infoOk \triangleq \land\ recoveryMEOracle[i] \neq i$
$\land\ recoveryMEOracle[i] \neq NullPoint$
IN $\lor \land \neg infoOk$
$\land$ UNCHANGED $\langle currentEpoch, leaderOracle, msgs\rangle$
$\lor \land\ infoOk$
$\land\ currentEpoch' = [currentEpoch$ EXCEPT $![i] = recoveryMaxEpoch[i]]$
$\land\ leaderOracle' = [leaderOracle$ EXCEPT $![i] = recoveryMEOracle[i]]$
$\land\ Send(i, recoveryMEOracle[i], [mtype \mapsto CEPOCH,$
$mepoch \mapsto recoveryMaxEpoch[i]])$
$\land$ UNCHANGED $\langle state, leaderEpoch, history, commitIndex, leaderVars, tempVars, recoveryVars, cepo$

In phase $f11$, follower sends $f.p$ to pleader via $CEPOCH$.
$FollowerDiscovery1(i) \triangleq$
$\land\ state[i] = Follower$
$\land\ leaderOracle[i] \neq NullPoint$
$\land \neg cepochSent[i]$
$\land$ LET $leader \triangleq leaderOracle[i]$
IN $Send(i, leader, [mtype \mapsto CEPOCH,$
$mepoch \mapsto currentEpoch[i]])$
$\land\ cepochSent' = [cepochSent$ EXCEPT $![i] =$ TRUE$]$
$\land$ UNCHANGED $\langle serverVars, leaderVars, tempVars, recoveryVars, proposalMsgsLog\rangle$

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.
$LeaderHandleCEPOCH(i, j) \triangleq$
$\land\ state[i] = ProspectiveLeader$
$\land\ msgs[j][i] \neq \langle\rangle$

$\land\ msgs[j][i][1].mtype = CEPOCH$
$\land\ \lor$ new message - modify *tempMaxEpoch* and *cepochRecv*
$\land\ NullPoint \notin cepochRecv[i]$
$\land$ LET $newEpoch \triangleq Maximum(\{tempMaxEpoch[i],\ msgs[j][i][1].mepoch\})$
IN $tempMaxEpoch' = [tempMaxEpoch$ EXCEPT $![i] = newEpoch]$
$\land\ cepochRecv' = [cepochRecv$ EXCEPT $![i] =$ IF $j \in cepochRecv[i]$ THEN $cepochRecv[i]$
ELSE $cepochRecv[i] \cup \{j\}$
$\land\ Discard(j,\ i)$
$\lor$ new follower who joins in cluster / follower whose history and *commitIndex* do not match
$\land\ NullPoint \in cepochRecv[i]$
$\land\ \lor\ \land\ NullPoint \notin ackeRecv[i]$
$\land\ Reply(i,\ j,\ [mtype \mapsto NEWEPOCH,$
$mepoch \mapsto leaderEpoch[i]])$
$\lor\ \land\ NullPoint \in ackeRecv[i]$
$\land\ Reply2(i,\ j,\ [mtype \mapsto NEWEPOCH,$
$mepoch \mapsto leaderEpoch[i]],$
$[mtype \mapsto NEWLEADER,$
$mepoch \mapsto currentEpoch[i],$
$minitialHistory \mapsto initialHistory[i]])$
$\land$ UNCHANGED $\langle cepochRecv,\ tempMaxEpoch\rangle$
$\land\ cluster' = [cluster$ EXCEPT $![i] =$ IF $j \in cluster[i]$ THEN $cluster[i]$ ELSE $cluster[i] \cup \{j\}]$
$\land$ 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.
$LeaderDiscovery1(i) \triangleq$
$\land\ state[i] = ProspectiveLeader$
$\land\ cepochRecv[i] \in Quorums$
$\land\ leaderEpoch' = [leaderEpoch$ EXCEPT $![i] = tempMaxEpoch[i] + 1]$
$\land\ cepochRecv' = [cepochRecv$ EXCEPT $![i] = \{NullPoint\}]$
$\land\ Broadcast(i,\ [mtype \mapsto NEWEPOCH,$
$mepoch \mapsto leaderEpoch'[i]])$
$\land$ 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.
$FollowerDiscovery2(i,\ j) \triangleq$
$\land\ state[i] = Follower$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = NEWEPOCH$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ new *NEWEPOCH* – accept and reply
$\land\ currentEpoch[i] < msg.mepoch$

$\land\ currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch]$
$\land\ leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] = j]$
$\land\ Reply(i, j, [mtype \mapsto ACKE,$
$mepoch \mapsto msg.mepoch,$
$mlastEpoch \mapsto leaderEpoch[i],$
$mhf \mapsto history[i]])$
$\lor\ \land\ currentEpoch[i] = msg.mepoch$
$\land\ \lor\ \land\ leaderOracle[i] = j$
$\land\ Discard(j, i)$
$\land\ \text{UNCHANGED } \langle currentEpoch, leaderOracle\rangle$
$\lor$ It may happen when a leader do not update new epoch to all followers in Q, and a new election begi
$\land\ leaderOracle[i] \neq j$
$\land\ leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] = j]$
$\land\ Reply(i, j, [mtype \mapsto ACKE,$
$mepoch \mapsto msg.mepoch,$
$mlastEpoch \mapsto leaderEpoch[i],$
$mhf \mapsto history[i]])$
$\land\ \text{UNCHANGED } currentEpoch$
$\lor$ stale *NEWEPOCH − diacard*
$\land\ currentEpoch[i] > msg.mepoch$
$\land\ Discard(j, i)$
$\land\ \text{UNCHANGED } \langle currentEpoch, leaderOracle\rangle$
$\land\ \text{UNCHANGED } \langle state, leaderEpoch, history, leaderVars, commitIndex, cepochSent, tempVars, recovery$

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.

$LeaderHandleACKE(i, j) \triangleq$
$\land\ state[i] = ProspectiveLeader$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = ACKE$
$\land\ \text{LET } msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \lor\ msg.mlastEpoch > tempMaxLastEpoch[i]$
$\lor\ \land\ msg.mlastEpoch = tempMaxLastEpoch[i]$
$\land\ \lor\ LastZxid(msg.mhf)[1] > LastZxid(tempInitialHistory[i])[1]$
$\lor\ \land\ LastZxid(msg.mhf)[1] = LastZxid(tempInitialHistory[i])[1]$
$\land\ LastZxid(msg.mhf)[2] \geq LastZxid(tempInitialHistory[i])[2]$
IN $\lor\ \land\ leaderEpoch[i] = msg.mepoch$
$\land\ \lor\ \land\ infoOk$
$\land\ tempMaxLastEpoch' = [tempMaxLastEpoch \text{ EXCEPT } ![i] = msg.mlastEpoch]$
$\land\ tempInitialHistory' = [tempInitialHistory \text{ EXCEPT } ![i] = msg.mhf]$
$\lor\ \land\ \neg infoOk$
$\land\ \text{UNCHANGED } \langle tempMaxLastEpoch, tempInitialHistory\rangle$
Followers not in Q will not receive $NEWEPOCH$, so leader will receive $ACKE$ only when the source is in
$\land\ ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \text{IF } j \notin ackeRecv[i] \text{ THEN } ackeRecv[i] \cup \{j\}$
$\text{ELSE } ackeRecv[i]]$

$\lor \land leaderEpoch[i] \neq msg.mepoch$
$\land$ UNCHANGED $\langle tempMaxLastEpoch, tempInitialHistory, ackeRecv \rangle$
$\land Discard(j, i)$
$\land$ UNCHANGED $\langle serverVars, cluster, cepochRecv, ackldRecv, ackIndex, currentCounter,$
$sendCounter, initialHistory, committedIndex, cepochSent, tempMaxEpoch, recoveryVa$

$LeaderDiscovery2Sync1(i) \triangleq$
$\land state[i] = ProspectiveLeader$
$\land ackeRecv[i] \in Quorums$
$\land currentEpoch' = [currentEpoch$ 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 *LeaderDiscovery*2*Sync*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.

$FollowerSync1(i, j) \triangleq$
$\land state[i] = Follower$
$\land msgs[j][i] \neq \langle\rangle$
$\land msgs[j][i][1].mtype = NEWLEADER$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ new *NEWLEADER* − accept and reply
$\land currentEpoch[i] \leq msg.mepoch$
$\land currentEpoch' = [currentEpoch$ EXCEPT $![i] = msg.mepoch]$
$\land leaderEpoch' = [leaderEpoch$ EXCEPT $![i] = msg.mepoch]$
$\land leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$
$\land history' = [history$ EXCEPT $![i] = msg.minitialHistory]$
$\land Reply(i, j, [mtype \mapsto ACKLD,$
$mepoch \mapsto msg.mepoch,$
$mhistory \mapsto msg.minitialHistory])$
$\lor$ stale *NEWLEADER* − discard
$\land currentEpoch[i] > msg.mepoch$
$\land Discard(j, i)$

$\wedge$ UNCHANGED $\langle currentEpoch, leaderEpoch, leaderOracle, history\rangle$
$\wedge$ UNCHANGED $\langle state, commitIndex, leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLo$

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

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

$LeaderSync2(i) \triangleq$
$\wedge\quad state[i] = ProspectiveLeader$
$\wedge\quad ackldRecv[i] \in Quorums$
$\wedge\quad commitIndex' = [commitIndex$ EXCEPT $![i] = Len(history[i])]$
$\wedge\quad committedIndex' = [committedIndex$ EXCEPT $![i] = Len(history[i])]$
$\wedge\quad state' = [state$ EXCEPT $![i] = Leader]$
$\wedge\quad currentCounter' = [currentCounter$ EXCEPT $![i] = 0]$
$\wedge\quad sendCounter' = [sendCounter$ EXCEPT $![i] = 0]$
$\wedge\quad ackldRecv' = [ackldRecv$ EXCEPT $![i] = \{NullPoint\}]$
$\wedge\quad Broadcast(i, [mtype \mapsto COMMITLD,$
$mepoch \mapsto currentEpoch[i],$
$mlength \mapsto Len(history[i])])$
$\wedge\quad$ UNCHANGED $\langle currentEpoch, leaderEpoch, leaderOracle, history, cluster, cepochRecv,$
$ackeRecv, ackIndex, initialHistory, tempVars, cepochSent, recoveryVars, proposalM$

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

$FollowerSync2(i, j) \triangleq$
$\wedge\ state[i] = Follower$
$\wedge\ msgs[j][i] \neq \langle\rangle$
$\wedge\ msgs[j][i][1].mtype = COMMITLD$
$\wedge$ LET $msg \triangleq msgs[j][i][1]$
IN $\vee$ new $COMMIT$-LD - commit all transactions in initial history
Regradless of *Restart*, it must be true because one will receive $NEWLEADER$ before receiving $COMMIT$-L
$\wedge\ currentEpoch[i] = msg.mepoch$
$\wedge\ leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$ unnecessary

$\land \lor \land Len(history[i]) = msg.mlength$
$\land commitIndex' = [commitIndex \text{ EXCEPT } ![i] = Len(history[i])]$
$\land Discard(j, i)$
$\lor \land Len(history[i]) \neq msg.mlength$
$\land Reply(i, j, [mtype \mapsto CEPOCH,$
$mepoch \mapsto currentEpoch[i]])$
$\land$ UNCHANGED $commitIndex$
$\lor$ $>$: stale $COMMIT$-LD - discard
$<$: In our implementation, $' < '$ does not exist due to the guarantee of $Restart$
$<$: If $' < '$ exists, we can discard it and handle it in $phase3$
$\land currentEpoch[i] \neq msg.mepoch$
$\land Discard(j, i)$
$\land$ UNCHANGED $\langle commitIndex, leaderOracle \rangle$
$\land$ UNCHANGED $\langle state, currentEpoch, leaderOracle, history, leaderVars, tempVars, cepochSent, recover$

In phase $l31$, leader receives client request and broadcasts $PROPOSE$.

$ClientRequest(i, v) \triangleq$
$\land state[i] = Leader$
$\land currentCounter' = [currentCounter \text{ EXCEPT } ![i] = currentCounter[i] + 1]$
$\land$ LET $newTransaction \triangleq [epoch \mapsto currentEpoch[i],$
$counter \mapsto currentCounter'[i],$
$value \mapsto v]$
IN $\land history' = [history \text{ EXCEPT } ![i] = Append(history[i], newTransaction)]$
$\land ackIndex' = [ackIndex \text{ EXCEPT } ![i] = Len(history'[i])]$ necessary, to push $commitIndex$
$\land$ UNCHANGED $\langle msgs, state, currentEpoch, leaderEpoch, leaderOracle, commitIndex, cluster, cepochR$
$ackeRecv, ackldRecv, sendCounter, initialHistory, committedIndex, tempVars, cepoch$

$LeaderBroadcast1(i) \triangleq$
$\land state[i] = Leader$
$\land sendCounter[i] < currentCounter[i]$
$\land$ LET $toBeSentCounter \triangleq sendCounter[i] + 1$
$toBeSentIndex \triangleq Len(initialHistory[i]) + toBeSentCounter$
$toBeSentEntry \triangleq history[i][toBeSentIndex]$
IN $\land Broadcast(i, [mtype \mapsto PROPOSE,$
$mepoch \mapsto currentEpoch[i],$
$mproposal \mapsto toBeSentEntry])$
$\land sendCounter' = [sendCounter \text{ EXCEPT } ![i] = toBeSentCounter]$
$\land$ LET $m \triangleq [msource \mapsto i, mtype \mapsto PROPOSE, mepoch \mapsto currentEpoch[i], mproposal \mapsto toI$
IN $proposalMsgsLog' = proposalMsgsLog \cup \{m\}$
$\land$ UNCHANGED $\langle serverVars, cepochRecv, cluster, ackeRecv, ackldRecv, ackIndex,$
$currentCounter, initialHistory, committedIndex, tempVars, recoveryVars, cepochSent$

In phase $f31$, follower accepts proposal and append it to history.

$FollowerBroadcast1(i, j) \triangleq$
$\land state[i] = Follower$

$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = PROPOSE$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ It should be that $\lor\ msg.mproposal.counter = 1$
$\lor\ msg.mrpoposal.counter = history[Len(history)].counter + 1$
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ history' = [history$ EXCEPT $![i] = Append(history[i],\ msg.mproposal)]$
$\land\ leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$
$\land\ Reply(i,\ j,\ [mtype \mapsto ACK,$
$mepoch \mapsto currentEpoch[i],$
$mindex \mapsto Len(history'[i])])$
$\lor$ If happens, $\neq$ must be $>$, namely a stale leader sends it.
$\land\ currentEpoch[i] \neq msg.mepoch$
$\land\ Discard(j,\ i)$
$\land$ UNCHANGED $\langle history,\ leaderOracle\rangle$
$\land$ UNCHANGED $\langle state,\ currentEpoch,\ leaderEpoch,\ commitIndex,\ leaderVars,\ tempVars,\ cepochSent,\ re$

In phase $l32$, leader receives ack from a quorum of followers to a certain proposal,
and commits the proposal.
$LeaderHandleACK(i,\ j) \triangleq$
$\land\ state[i] = Leader$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = ACK$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ It should be that $ackIndex[i][j] + 1 \triangleq msg.mindex$
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ ackIndex' = [ackIndex$ EXCEPT $![i][j] = Maximum(\{ackIndex[i][j],\ msg.mindex\})]$
$\lor$ If happens, $\neq$ must be $>$, namely a stale follower sends it.
$\land\ currentEpoch[i] \neq msg.mepoch$
$\land$ UNCHANGED $ackIndex$
$\land\ Discard(j,\ i)$
$\land$ UNCHANGED $\langle serverVars,\ cluster,\ cepochRecv,\ ackeRecv,\ ackldRecv,\ currentCounter,$
$sendCounter,\ initialHistory,\ committedIndex,\ tempVars,\ cepochSent,\ recoveryVars,\ p$

$LeaderAdvanceCommit(i) \triangleq$
$\land\ state[i] = Leader$
$\land\ commitIndex[i] < Len(history[i])$
$\land$ LET $Agree(index) \triangleq \{i\} \cup \{k \in (Server \setminus \{i\}) : ackIndex[i][k] \geq index\}$
$agreeIndexes \triangleq \{index \in (commitIndex[i] + 1)\ ..\ Len(history[i]) : Agree(index) \in Quorum$
$newCommitIndex \triangleq$ IF $agreeIndexes \neq \{\}$ THEN $Maximum(agreeIndexes)$
ELSE $commitIndex[i]$
IN $commitIndex' = [commitIndex$ EXCEPT $![i] = newCommitIndex]$
$\land$ UNCHANGED $\langle state,\ currentEpoch,\ leaderEpoch,\ leaderOracle,\ history,$
$msgs,\ leaderVars,\ tempVars,\ cepochSent,\ recoveryVars,\ proposalMsgsLog\rangle$

$LeaderBroadcast2(i) \triangleq$

$\land\ state[i] = Leader$
$\land\ committedIndex[i] < commitIndex[i]$
$\land$ LET $newCommittedIndex \triangleq committedIndex[i] + 1$
IN $\land\ Broadcast(i, [mtype \mapsto COMMIT,$
$mepoch \mapsto currentEpoch[i],$
$mindex \mapsto newCommittedIndex,$
$mcounter \mapsto history[newCommittedIndex].counter])$
$\land\ committedIndex' = [committedIndex$ EXCEPT $![i] = committedIndex[i] + 1]$
$\land$ UNCHANGED $\langle serverVars, cluster, cepochRecv, ackeRecv, ackldRecv, ackIndex, currentCounter,$
$sendCounter, initialHistory, tempVars, cepochSent, recoveryVars, proposalMsgsLog\rangle$

In phase $f32$, follower receives *COMMIT* and commits transaction.
$FollowerBroadcast2(i, j) \triangleq$
$\land\ state[i] = Follower$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = COMMIT$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor\ \land\ currentEpoch[i] = msg.mepoch$
$\land\ leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$
$\land$ LET $infoOk \triangleq \land\ Len(history[i]) \geq msg.mindex$
$\land\ history[i][msg.mindex].epoch = msg.mepoch$
$\land\ history[i][msg.mindex].counter = msg.mcounter$
IN $\lor$ new *COMMIT* – commit transaction in history
$\land\ infoOk$
$\land\ commitIndex' = [commitIndex$ EXCEPT $![i] = Maximum(\{commitIndex[i], msg.$
$\land\ Discard(j, i)$
$\lor$ It may happen when the server is a new follower who joined in the cluster,
and it misses the corresponding *PROPOSE*.
$\land\ \neg infoOk$
$\land\ Reply(i, j, [mtype \mapsto CEPOCH,$
$mepoch \mapsto currentEpoch[i]])$
$\land$ UNCHANGED $commitIndex$
$\lor$ stale *COMMIT* – discard
$\land\ currentEpoch[i] \neq msg.mepoch$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $\langle commitIndex, leaderOracle\rangle$
$\land$ UNCHANGED $\langle state, currentEpoch, leaderEpoch, history,$
$leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog\rangle$

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 $l33$, upon receiving *CEPOCH*, leader l proposes back *NEWEPOCH* and *NEWLEADER*.

$$
\begin{aligned}
&\mathit{LeaderHandleCEPOCHinPhase3}(i,\, j) \triangleq \\
&\quad \land\ \mathit{state}[i] = \mathit{Leader} \\
&\quad \land\ \mathit{msgs}[j][i] \neq \langle\rangle \\
&\quad \land\ \mathit{msgs}[j][i][1].\mathit{mtype} = \mathit{CEPOCH} \\
&\quad \land\ \text{LET}\ \mathit{msg} \triangleq \mathit{msgs}[j][i][1] \\
&\qquad \text{IN}\quad \lor\ \land\ \mathit{currentEpoch}[i] \geq \mathit{msg}.\mathit{mepoch} \\
&\qquad\qquad\ \ \land\ \mathit{Reply2}(i,\, j,\, [\mathit{mtype} \mapsto \mathit{NEWEPOCH}, \\
&\qquad\qquad\qquad\qquad\qquad\ \ \mathit{mepoch} \mapsto \mathit{currentEpoch}[i]], \\
&\qquad\qquad\qquad\qquad\qquad [\mathit{mtype} \mapsto \mathit{NEWLEADER}, \\
&\qquad\qquad\qquad\qquad\qquad\ \ \mathit{mepoch} \mapsto \mathit{currentEpoch}[i], \\
&\qquad\qquad\qquad\qquad\qquad\ \ \mathit{minitialHistory} \mapsto \mathit{history}[i]]) \\
&\qquad\qquad \lor\ \land\ \mathit{currentEpoch}[i] < \mathit{msg}.\mathit{mepoch} \\
&\qquad\qquad\ \ \land\ \text{UNCHANGED}\ \mathit{msgs} \\
&\quad \land\ \text{UNCHANGED}\ \langle \mathit{serverVars},\, \mathit{leaderVars},\, \mathit{tempVars},\, \mathit{cepochSent},\, \mathit{recoveryVars},\, \mathit{proposalMsgsLog}\rangle
\end{aligned}
$$

In phase $l34$, upon receiving ack from f of the *NEWLEADER*, it sends a commit message to f. Leader l also makes $Q := Q \cup \{f\}$.

$$
\begin{aligned}
&\mathit{LeaderHandleACKLDinPhase3}(i,\, j) \triangleq \\
&\quad \land\ \mathit{state}[i] = \mathit{Leader} \\
&\quad \land\ \mathit{msgs}[j][i] \neq \langle\rangle \\
&\quad \land\ \mathit{msgs}[j][i][1].\mathit{mtype} = \mathit{ACKLD} \\
&\quad \land\ \text{LET}\ \mathit{msg} \triangleq \mathit{msgs}[j][i][1] \\
&\qquad\ \ \mathit{aimCommitIndex} \triangleq \mathit{Minimum}(\{\mathit{commitIndex}[i],\, \mathit{Len}(\mathit{msg}.\mathit{mhistory})\}) \\
&\qquad \text{IN}\quad \lor\ \land\ \mathit{currentEpoch}[i] = \mathit{msg}.\mathit{mepoch} \\
&\qquad\qquad\ \ \land\ \mathit{ackIndex}' = [\mathit{ackIndex}\ \text{EXCEPT}\ ![i][j] = \mathit{Len}(\mathit{msg}.\mathit{mhistory})] \\
&\qquad\qquad\ \ \land\ \mathit{Reply}(i,\, j,\, [\mathit{mtype} \mapsto \mathit{COMMIT}, \\
&\qquad\qquad\qquad\qquad\qquad\ \ \mathit{mepoch} \mapsto \mathit{currentEpoch}[i], \\
&\qquad\qquad\qquad\qquad\qquad\ \ \mathit{mindex} \mapsto \mathit{aimCommitIndex}, \\
&\qquad\qquad\qquad\qquad\qquad\ \ \mathit{mcounter} \mapsto \mathit{history}[\mathit{aimCommitIndex}].\mathit{counter}]) \\
&\qquad\qquad \lor\ \land\ \mathit{currentEpoch}[i] \neq \mathit{msg}.\mathit{mepoch} \\
&\qquad\qquad\ \ \land\ \mathit{Discard}(j,\, i) \\
&\qquad\qquad\ \ \land\ \text{UNCHANGED}\ \mathit{ackIndex} \\
&\quad \land\ \mathit{cluster}' = [\mathit{cluster}\ \text{EXCEPT}\ ![i] = \text{IF}\ j \in \mathit{cluster}[i]\ \text{THEN}\ \mathit{cluster}[i] \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \ \text{ELSE}\ \ \mathit{cluster}[i] \cup \{j\}] \\
&\quad \land\ \text{UNCHANGED}\ \langle \mathit{serverVars},\, \mathit{cepochRecv},\, \mathit{ackeRecv},\, \mathit{ackldRecv},\, \mathit{currentCounter},\, \mathit{sendCounter}, \\
&\qquad\qquad\qquad \mathit{initialHistory},\, \mathit{committedIndex},\, \mathit{tempVars},\, \mathit{cepochSent},\, \mathit{recoveryVars},\, \mathit{proposalMsgsLo}
\end{aligned}
$$

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.

$$
\begin{array}{l}
BecomeFollower(i) \triangleq \\
\quad \land\ state[i] \neq Follower \\
\quad \land\ \exists j \in Server \setminus \{i\} : \land\ msgs[j][i] \neq \langle\rangle \\
\qquad\qquad\qquad\qquad\ \land\ msgs[j][i][1].mtype \neq RECOVERYREQUEST \\
\qquad\qquad\qquad\qquad\ \land\ msgs[j][i][1].mtype \neq RECOVERYRESPONSE \\
\qquad\qquad\qquad\qquad\ \land\ \text{LET}\ msg \triangleq msgs[j][i][1] \\
\qquad\qquad\qquad\qquad\quad \text{IN}\ \ \land\ currentEpoch[i] < msg.mepoch \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ \lor\ msg.mtype = NEWEPOCH \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ msg.mtype = NEWLEADER \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ msg.mtype = COMMITLD \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ msg.mtype = PROPOSE \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ msg.mtype = COMMIT \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ state' = [state \ \text{EXCEPT}\ ![i] = Follower] \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ currentEpoch' = [currentEpoch \ \text{EXCEPT}\ ![i] = msg.mepoch] \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ leaderOracle' = [leaderOracle \ \text{EXCEPT}\ ![i] = j] \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{Here we should not use } Discard. \\
\quad \land\ \text{UNCHANGED}\ \langle leaderEpoch, history, commitIndex, msgs, leaderVars, tempVars, cepochSent, recover
\end{array}
$$

$$
\begin{array}{l}
DiscardStaleMessage(i) \triangleq \\
\quad \land\ \exists j \in Server \setminus \{i\} : \land\ msgs[j][i] \neq \langle\rangle \\
\qquad\qquad\qquad\qquad\ \land\ msgs[j][i][1].mtype \neq RECOVERYREQUEST \\
\qquad\qquad\qquad\qquad\ \land\ msgs[j][i][1].mtype \neq RECOVERYRESPONSE \\
\qquad\qquad\qquad\qquad\ \land\ \text{LET}\ msg \triangleq msgs[j][i][1] \\
\qquad\qquad\qquad\qquad\quad \text{IN}\ \ \lor\ \land\ state[i] = Follower \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ \ \lor\ msg.mepoch < currentEpoch[i] \setminus * \text{ Discussed before.} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ msg.mtype = CEPOCH \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ msg.mtype = ACKE \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ msg.mtype = ACKLD \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ msg.mtype = ACK \\
\qquad\qquad\qquad\qquad\qquad\ \lor\ \land\ state[i] = Leader \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ msg.mtype \neq CEPOCH \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ \lor\ msg.mepoch \leq currentEpoch[i] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ msg.mtype = ACKE \ \text{response of } NEWEPOCH \\
\qquad\qquad\qquad\qquad\qquad\ \lor\ \land\ state[i] = ProspectiveLeader \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ msg.mtype \neq CEPOCH \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ \lor\ msg.mepoch \leq currentEpoch[i] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ msg.mtype = ACK \\
\qquad\qquad\qquad\qquad\ \land\ Discard(j, i)
\end{array}
$$

$\land$ UNCHANGED $\langle serverVars,\ leaderVars,\ tempVars,\ cepochSent,\ recoveryVars,\ proposalMsgsLog\rangle$

Defines how the variables may transition.

$Next \triangleq$

$\lor \exists\, i \in Server,\ Q \in Quorums : InitialElection(i,\ Q)$
$\lor \exists\, i \in Server : \quad Restart(i)$
$\lor \exists\, i \in Server : \quad RecoveryAfterRestart(i)$
$\lor \exists\, i, j \in Server : \quad HandleRecoveryRequest(i,\ j)$
$\lor \exists\, i, j \in Server : \quad HandleRecoveryResponse(i,\ j)$
$\lor \exists\, i, j \in Server : \quad FindCluster(i)$
$\lor \exists\, i, j \in Server : \quad LeaderTimeout(i,\ j)$
$\lor \exists\, i \in Server : \quad FollowerTimeout(i)$
$\lor \exists\, i \in Server : \quad FollowerDiscovery1(i)$
$\lor \exists\, i, j \in Server : \quad LeaderHandleCEPOCH(i,\ j)$
$\lor \exists\, i \in Server : \quad LeaderDiscovery1(i)$
$\lor \exists\, i, j \in Server : \quad FollowerDiscovery2(i,\ j)$
$\lor \exists\, i, j \in Server : \quad LeaderHandleACKE(i,\ j)$
$\lor \exists\, i \in Server : \quad LeaderDiscovery2Sync1(i)$
$\lor \exists\, i, j \in Server : \quad FollowerSync1(i,\ j)$
$\lor \exists\, i, j \in Server : \quad LeaderHandleACKLD(i,\ j)$
$\lor \exists\, i \in Server : \quad LeaderSync2(i)$
$\lor \exists\, i, j \in Server : \quad FollowerSync2(i,\ j)$
$\lor \exists\, i \in Server,\ v \in Value : ClientRequest(i,\ v)$
$\lor \exists\, i \in Server : \quad LeaderBroadcast1(i)$
$\lor \exists\, i, j \in Server : \quad FollowerBroadcast1(i,\ j)$
$\lor \exists\, i, j \in Server : \quad LeaderHandleACK(i,\ j)$
$\lor \exists\, i \in Server : \quad LeaderAdvanceCommit(i)$
$\lor \exists\, i \in Server : \quad LeaderBroadcast2(i)$
$\lor \exists\, i, j \in Server : \quad FollowerBroadcast2(i,\ j)$
$\lor \exists\, i, j \in Server : \quad LeaderHandleCEPOCHinPhase3(i,\ j)$
$\lor \exists\, i, j \in Server : \quad LeaderHandleACKLDinPhase3(i,\ j)$
$\lor \exists\, i \in Server : \quad DiscardStaleMessage(i)$
$\lor \exists\, i \in Server : \quad BecomeFollower(i)$

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

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.

$Leadership \triangleq \forall\, i, j \in Server :$
$\land\ state[i] = Leader \lor state[i] = ProspectiveLeader$
$\land\ state[j] = Leader \lor state[j] = ProspectiveLeader$

$\land\ currentEpoch[i] = currentEpoch[j]$
$\Rightarrow i = j$

Here, delivering means deliver some transaction from history to replica. We can assume *deliverIndex* = *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 < *epoch*,*counter* > according to its uniqueness.

$equal(entry1,\ entry2) \triangleq \land\ entry1.epoch = entry2.epoch$
$\land\ entry1.counter = entry2.counter$

$precede(entry1,\ entry2) \triangleq \lor\ entry1.epoch < entry2.epoch$
$\lor\ \land\ entry1.epoch = entry2.epoch$
$\land\ entry1.counter < entry2.counter$

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

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

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

$Integrity \triangleq \forall\, i \in Server :$
$state[i] = Follower \land commitIndex[i] > 0$
$\Rightarrow \forall\, index \in 1\,..\,commitIndex[i] : \exists\, msg \in proposalMsgsLog :$
$equal(msg.mproposal,\ history[i][index])$

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

$Agreement \triangleq \forall\, i, j \in Server :$
$\land\ state[i] = Follower \land commitIndex[i] > 0$
$\land\ state[j] = Follower \land commitIndex[j] > 0$
$\Rightarrow$
$\forall\, index1 \in 1\,..\,commitIndex[i],\ index2 \in 1\,..\,commitIndex[j] :$
$\lor \exists\, indexj \in 1\,..\,commitIndex[j] :$
$equal(history[j][indexj],\ history[i][index1])$
$\lor \exists\, indexi \in 1\,..\,commitIndex[i] :$
$equal(history[i][indexi],\ history[j][index2])$

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

$TotalOrder \triangleq \forall\, i, j \in Server : commitIndex[i] \geq 2 \land commitIndex[j] \geq 2$
$\Rightarrow \forall\, indexi1 \in 1\,..\,(commitIndex[i] - 1) : \forall\, indexi2 \in (indexi1 + 1)\,..\,commitIndex[i] :$
LET $logOk \triangleq \exists\, index \in 1\,..\,commitIndex[j] : equal(history[i][indexi2],\ history[j][index]$
IN $\lor \neg logOk$
$\lor\ \land\ logOk$
$\land$ LET $indexj2 \triangleq$ CHOOSE $idx \in 1\,..\,commitIndex[j]$: $equal(history[i][indexi2],\ history[j][idx$
IN $\exists\, indexj1 \in 1\,..\,(indexj2 - 1)$: $equal(history[i][indexi1],\ history[j][indexj1])$

$\exists\, indexj2 \in 1\,..\,commitIndex[j]:$
$\wedge\, equal(history[i][indexi2],\, history[j][indexj2])$
$\wedge\, \exists\, indexj1 \in 1\,..\,(indexj2-1) : equal(history[i][indexi1],\, histo$

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

$LocalPrimaryOrder \triangleq$ LET $mset(i,\, e) \triangleq \{msg \in proposalMsgsLog : msg.msource = i \wedge msg.mproposal.epoch$
$mentries(i,\, e) \triangleq \{msg.mproposal : msg \in mset(i,\, e)\}$
IN $\forall\, i \in Server : \forall\, e \in 1\,..\,currentEpoch[i]:$
$\wedge\, Cardinality(mentries(i,\, e)) \geq 2$
$\wedge$ LET $tsc1 \triangleq$ CHOOSE $p \in mentries(i,\, e)$: TRUE
$tsc2 \triangleq$ CHOOSE $p \in mentries(i,\, e)$: $\neg equal(p,\, tsc1)$
$\exists\, tsc1 \in mentries(i,\, e) : \exists\, tsc2 \in mentries(i,\, e):$
$\wedge\, \neg equal(tsc2,\, tsc1)$
$\wedge$ LET $tscPre \triangleq$ IF $precede(tsc1,\, tsc2)$ THEN $tsc1$ ELSE $tsc2$
$tscNext \triangleq$ IF $precede(tsc1,\, tsc2)$ THEN $tsc2$ ELSE $tsc1$
IN $\forall\, j \in Server :\ \wedge\, commitIndex[j] \geq 2$
$\wedge\, \exists\, index \in 1\,..\,commitIndex[j] : equal(history[j][inde$
$\Rightarrow$ LET $index2 \triangleq$ CHOOSE $idx \in 1\,..\,commitIndex[j]$: $equal(history[j][idx],\, tscNex$
IN $\wedge\, index2 > 1$
$\wedge\, \exists\, index1 \in 1\,..\,(index2-1)$: $equal(history[j][index1],\, tscPre)$
$\exists\, index2 \in 1\,..\,commitIndex[j]:$
$\wedge\, equal(history[j][index2],\, tscNext)$
$\wedge\, index2 > 1$
$\wedge\, \exists\, index1 \in 1\,..\,(index2-1) : equal(history[j][index1],\, tscPre)$

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.

$GlobalPrimaryOrder \triangleq \forall\, i \in Server : commitIndex[i] \geq 2$
$\Rightarrow \forall\, idx1,\, idx2 \in 1\,..\,commitIndex[i] :\ \vee\, history[i][idx1].epoch \geq history[i][idx2].ep$
$\vee\ \wedge\, history[i][idx1].epoch < history[i][idx2]$
$\wedge\, idx1 < idx2$

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.

$PrimaryIntegrity \triangleq \forall\, i,\, j \in Server :\ \wedge\, state[i] = Leader$
$\wedge\, state[j] = Follower \wedge commitIndex[j] \geq 1$
$\Rightarrow \forall\, index \in 1\,..\,commitIndex[j] :\ \vee\, history[j][index].epoch \geq currentEpoch[i]$
$\vee\ \wedge\, history[j][index].epoch < currentEpoch[i]$
$\wedge\, \exists\, idx \in 1\,..\,commitIndex[i] : equal(history[i][i$

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 *Mon Apr* 26 21:48:14 *CST* 2021 by Dell
\ ∗ Created Sat *Dec* 05 13:32:08 *CST* 2020 by Dell