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*.

In this spec, we do model checking for *Zab* with the scale of 3 rounds of execution and 2 delivered value.

This work is driven by Junqueira *F P*, Reed *B C*, *Serafini M*. *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*

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

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

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

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

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

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

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

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

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

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

Define initial values for all variables

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

$\land\ proposalMsgsLog \quad = \{\}$

A server becomes pleader and a quorum servers knows that.

$Election(i,\ Q)\ \triangleq$

test restrictions

$\land \forall\ s \in Server : currentEpoch[s] \leq 2 \land Len(history[s]) \leq 2$

$\land\ i \in Q$

$\land\ state' = [s \in Server \mapsto \text{IF } s = i \text{ THEN } ProspectiveLeader$
$\text{ELSE IF } s \in Q \text{ THEN } Follower$
$\text{ELSE } state[s]]$

$\land\ cluster' = [cluster \text{ EXCEPT } ![i] = Q]$ cluster is first initialized in election, not $phase1$.

$\land\ cepochRecv' = [cepochRecv \text{ EXCEPT } ![i] = \{i\}]$

$\land\ ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \{i\}]$

$\land\ ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] = \{i\}]$

$\land\ ackIndex' = [ii \in Server \mapsto [ij \in Server \mapsto$
$\text{IF } ii = i \text{ THEN } 0$
$\text{ELSE } ackIndex[ii][ij]]]$

$\land\ committedIndex' = [committedIndex \text{ EXCEPT } ![i] = 0]$

$\land\ initialHistory' = [initialHistory \text{ EXCEPT } ![i] = \langle\rangle]$

$\land\ tempMaxEpoch' = [tempMaxEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$

$\land\ tempMaxLastEpoch' = [tempMaxLastEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$

$\land\ tempInitialHistory' = [tempInitialHistory \text{ EXCEPT } ![i] = history[i]]$

$\land\ leaderOracle' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } i$
$\text{ELSE } leaderOracle[s]]$

$\land\ leaderEpoch' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } currentEpoch[s]$
$\text{ELSE } leaderEpoch[s]]$

$\land\ cepochSent' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN FALSE}$
$\text{ELSE } cepochSent[s]]$

$\land\ msgs' = [ii \in Server \mapsto [ij \in Server \mapsto$
$\text{IF } ii \in Q \lor ij \in Q \text{ THEN } \langle\rangle$
$\text{ELSE } msgs[ii][ij]]]$

$\land$ 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

$\land\ currentEpoch[i] \leq 3$

$\land\ Len(history[i]) \leq 2$

$\land \forall\ s \in Server : state[s] = Follower \land leaderOracle[s] = NullPoint$

$\land\ Election(i,\ Q)$

$\land$ UNCHANGED $\langle currentEpoch, history, commitIndex, currentCounter, sendCounter, recoveryVars, pr$

The leader finds timeout with another follower.

$LeaderTimeout(i,\ j)\ \triangleq$

```
        test restrictions
    ∧ currentEpoch[i] ≤ 3
    ∧ 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, le
            ∨ ∧ 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, pr

A follower finds timeout with the leader.
FollowerTimeout(i) ≜
        test restrictions
    ∧ currentEpoch[i] ≤ 3
    ∧ 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, pr

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

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] ≤ 3
    ∧ 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]]]
```

$\land\ recoverySent' = [recoverySent \text{ EXCEPT } ![i] = \text{FALSE}]$
$\land \text{ UNCHANGED } \langle currentEpoch,\ leaderEpoch,\ history,\ leaderVars,\ tempVars,$
$recoveryRespRecv,\ recoveryMaxEpoch,\ recoveryMEOracle,\ proposalMsgsLog\rangle$

$RecoveryAfterRestart(i) \triangleq$
test restrictions
$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = Follower$
$\land\ leaderOracle[i] = NullPoint$
$\land\ \neg recoverySent[i]$
$\land\ recoveryRespRecv' = [recoveryRespRecv \text{ EXCEPT } ![i] = \{\}]$
$\land\ recoveryMaxEpoch' = [recoveryMaxEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$
$\land\ recoveryMEOracle' = [recoveryMEOracle \text{ EXCEPT } ![i] = NullPoint]$
$\land\ recoverySent' = [recoverySent \text{ EXCEPT } ![i] = \text{TRUE}]$
$\land\ BroadcastToAll(i, [mtype \mapsto RECOVERYREQUEST])$
$\land \text{ UNCHANGED } \langle serverVars,\ leaderVars,\ tempVars,\ cepochSent,\ proposalMsgsLog\rangle$

$HandleRecoveryRequest(i, j) \triangleq$
test restrictions
$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = RECOVERYREQUEST$
$\land\ Reply(i, j, [mtype \mapsto RECOVERYRESPONSE,$
$moracle \mapsto leaderOracle[i],$
$mepoch \mapsto currentEpoch[i]])$
$\land \text{ UNCHANGED } \langle serverVars,\ leaderVars,\ tempVars,\ cepochSent,\ recoveryVars,\ proposalMsgsLog\rangle$

$HandleRecoveryResponse(i, j) \triangleq$
test restrictions
$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = RECOVERYRESPONSE$
$\land \text{ LET } msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \land\ msg.mepoch \geq recoveryMaxEpoch[i]$
$\land\ msg.moracle \neq NullPoint$
IN $\lor \land\ infoOk$
$\land\ recoveryMaxEpoch' = [recoveryMaxEpoch \text{ EXCEPT } ![i] = msg.mepoch]$
$\land\ recoveryMEOracle' = [recoveryMEOracle \text{ EXCEPT } ![i] = msg.moracle]$
$\lor \land\ \neg infoOk$
$\land \text{ UNCHANGED } \langle recoveryMaxEpoch,\ recoveryMEOracle\rangle$
$\land\ Discard(j, i)$
$\land\ recoveryRespRecv' = [recoveryRespRecv \text{ EXCEPT } ![i] = \text{IF } j \in recoveryRespRecv[i] \text{ THEN } recoveryRe$
ELSE $recoveryRe$

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

$FindCluster(i) \triangleq$

test restrictions

$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = Follower$
$\land\ leaderOracle[i] = NullPoint$
$\land\ recoveryRespRecv[i] \in Quorums$
$\land$ LET $infoOk \triangleq \land\ recoveryMEOracle[i] \neq i$
$\land\ recoveryMEOracle[i] \neq NullPoint$
$\land\ currentEpoch[i] \leq recoveryMaxEpoch[i]$
IN $\lor \land \neg infoOk$
$\land\ recoverySent' = [recoverySent$ EXCEPT $![i] =$ FALSE$]$
$\land$ 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 $recoverySent$
$\land$ UNCHANGED $\langle state,\ leaderEpoch,\ history,\ commitIndex,\ leaderVars,\ tempVars,$
$recoveryRespRecv,\ recoveryMaxEpoch,\ recoveryMEOracle,\ cepochSent,\ proposalMsgsL$

In phase $f11$, follower sends $f.p$ to pleader via $CEPOCH$.

$FollowerDiscovery1(i) \triangleq$

test restrictions

$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = Follower$
$\land\ leaderOracle[i] \neq NullPoint$
$\land\ \neg cepochSent[i]$
$\land$ 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.

$LeaderHandleCEPOCH(i,\ j) \triangleq$

test restrictions

$\land\ tempMaxEpoch[i] \leq 2$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = ProspectiveLeader$

$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = CEPOCH$
$\land\ \lor$ 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.

$LeaderDiscovery1(i) \triangleq$
test restrictions
$\land\ tempMaxEpoch[i] \leq 2$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = ProspectiveLeader$
$\land\ cepochRecv[i] \in Quorums$
$\land\ leaderEpoch' = [leaderEpoch$ 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.

$FollowerDiscovery2(i,\ j) \triangleq$
test restrictions
$\land\ currentEpoch[i] \leq 3$

$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = Follower$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = NEWEPOCH$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ new $NEWEPOCH$ − accept and reply
$\land\ currentEpoch[i] < msg.mepoch$
$\land\ \lor\ \land\ leaderOracle[i] = j$
$\land\ currentEpoch' = [currentEpoch$ EXCEPT $![i] = msg.mepoch]$
$\land\ Reply(i, j, [mtype \mapsto ACKE,$
$mepoch \mapsto msg.mepoch,$
$mlastEpoch \mapsto leaderEpoch[i],$
$mhf \mapsto history[i]])$
$\lor\ \land\ leaderOracle[i] \neq j$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $currentEpoch$
$\lor\ \land\ currentEpoch[i] = msg.mepoch$
$\land\ \lor\ \land\ leaderOracle[i] = j$
$\land\ Reply(i, j, [mtype \mapsto ACKE,$
$mepoch \mapsto msg.mepoch,$
$mlastEpoch \mapsto leaderEpoch[i],$
$mhf \mapsto history[i]])$
$\land$ UNCHANGED $currentEpoch$
$\lor$ It may happen when a leader do not update new epoch to all followers in Q, and a new election begi
$\land\ leaderOracle[i] \neq j$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $currentEpoch$
$\lor$ stale $NEWEPOCH$ − $diacard$
$\land\ currentEpoch[i] > msg.mepoch$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $currentEpoch$
$\land$ UNCHANGED $\langle state, leaderEpoch, leaderOracle, history, leaderVars,$
$commitIndex, cepochSent, tempVars, recoveryVars, proposalMsgsLog\rangle$

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.
$LeaderHandleACKE(i, j) \triangleq$
test restrictions
$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = ProspectiveLeader$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = ACKE$
$\land$ 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 $\lor \land leaderEpoch[i] = msg.mepoch$
$\land \lor \land infoOk$
$\land tempMaxLastEpoch' = [tempMaxLastEpoch \text{ EXCEPT } ![i] = msg.mlastEpoch]$
$\land tempInitialHistory' = [tempInitialHistory \text{ EXCEPT } ![i] = msg.mhf]$
$\lor \land \neg infoOk$
$\land$ UNCHANGED $\langle tempMaxLastEpoch, tempInitialHistory \rangle$

Followers not in Q will not receive $NEWEPOCH$, so leader will receive $ACKE$ only when the source is in Q

$\land ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \text{IF } j \notin ackeRecv[i] \text{ THEN } ackeRecv[i] \cup \{j\} \text{ ELSE } ackeRecv[i]]$

$\lor \land leaderEpoch[i] \neq msg.mepoch$
$\land$ UNCHANGED $\langle tempMaxLastEpoch, tempInitialHistory, ackeRecv \rangle$

$\land Discard(j, i)$
$\land$ UNCHANGED $\langle serverVars, cluster, cepochRecv, ackldRecv, ackIndex, currentCounter, sendCounter, initialHistory, committedIndex, cepochSent, tempMaxEpoch, recoveryVa$

$LeaderDiscovery2Sync1(i) \triangleq$

test restrictions

$\land currentEpoch[i] \leq 3$
$\land Len(history[i]) \leq 2$
$\land state[i] = ProspectiveLeader$
$\land ackeRecv[i] \in Quorums$
$\land currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = leaderEpoch[i]]$
$\land history' = [history \text{ EXCEPT } ![i] = tempInitialHistory[i]]$
$\land initialHistory' = [initialHistory \text{ EXCEPT } ![i] = tempInitialHistory[i]]$
$\land ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \{NullPoint\}]$
$\land ackIndex' = [ackIndex \text{ EXCEPT } ![i][i] = Len(tempInitialHistory[i])]$

until now, $phase1(Discovery)$ ends

$\land Broadcast(i, [mtype \mapsto NEWLEADER, mepoch \mapsto currentEpoch'[i], minitialHistory \mapsto history'[i]])$
$\land$ LET $m \triangleq [msource \mapsto i, mtype \mapsto NEWLEADER, mepoch \mapsto currentEpoch'[i], mproposals \mapsto histo$
IN $proposalMsgsLog' = \text{IF } m \in proposalMsgsLog \text{ THEN } proposalMsgsLog \text{ ELSE } proposalMsgsLog \cup \{m\}$
$\land$ UNCHANGED $\langle state, leaderEpoch, leaderOracle, commitIndex, cluster, cepochRecv, ackldRecv, currentCounter, sendCounter, committedIndex, cepochSent, tempVars, recoveryVars \rangle$

*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.

$FollowerSync1(i,\ j) \triangleq$

test restrictions

$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = Follower$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = NEWLEADER$
$\land$ 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

$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = ProspectiveLeader$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = ACKLD$
$\land$ 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 $\langle serverVars,\ cluster,\ cepochRecv,\ ackeRecv,\ currentCounter,$

$sendCounter$, $initialHistory$, $committedIndex$, $tempVars$, $cepochSent$, $recoveryVars$, p

$LeaderSync2(i) \triangleq$

test restrictions

$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = ProspectiveLeader$
$\land\ ackldRecv[i] \in Quorums$
$\land\ commitIndex' = [commitIndex \text{ EXCEPT } ![i] = Len(history[i])]$
$\land\ committedIndex' = [committedIndex \text{ EXCEPT } ![i] = Len(history[i])]$
$\land\ state' = [state \text{ EXCEPT } ![i] = Leader]$
$\land\ currentCounter' = [currentCounter \text{ EXCEPT } ![i] = 0]$
$\land\ sendCounter' = [sendCounter \text{ EXCEPT } ![i] = 0]$
$\land\ ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] = \{NullPoint\}]$
$\land\ Broadcast(i, [mtype \mapsto COMMITLD,$
$mepoch \mapsto currentEpoch[i],$
$mlength \mapsto Len(history[i])])$
$\land$ UNCHANGED $\langle currentEpoch, leaderEpoch, leaderOracle, history, cluster, cepochRecv,$
$ackeRecv, ackIndex, initialHistory, tempVars, cepochSent, recoveryVars, proposalMsg$

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

$FollowerSync2(i, j) \triangleq$

test restrictions

$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = Follower$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = COMMITLD$
$\land$ LET $msg \triangleq msgs[j][i][1]$
$replyOk \triangleq \land currentEpoch[i] = msg.mepoch$
$\land leaderOracle[i] = j$
IN $\lor$ new $COMMIT$-LD - commit all transactions in initial history
Regradless of $Restart$, it must be true because one will receive $NEWLEADER$ before receiving $COMMIT$-
$\land\ replyOk$
$\land\ \lor\ \land Len(history[i]) = msg.mlength$
$\land commitIndex' = [commitIndex \text{ EXCEPT } ![i] = Len(history[i])]$
$\land Discard(j, i)$
$\lor\ \land Len(history[i]) \neq msg.mlength$
$\land Reply(i, j, [mtype \mapsto CEPOCH,$
$mepoch \mapsto currentEpoch[i]])$
$\land$ UNCHANGED $commitIndex$
$\lor$ $>$: stale $COMMIT$-LD - discard
$<$: In our implementation, $' < '$ does not exist due to the guarantee of $Restart$
$\land\ \neg replyOk$
$\land\ Discard(j, i)$

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

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

$ClientRequest(i, v) \triangleq$

test restrictions

$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 1$
$\land\ state[i] = Leader$
$\land\ currentCounter' = [currentCounter \text{ EXCEPT } ![i] = currentCounter[i] + 1]$
$\land$ LET $newTransaction \triangleq [epoch \mapsto currentEpoch[i],$
$counter \mapsto currentCounter'[i],$
$value \mapsto v]$
IN $\land\ history' = [history \text{ EXCEPT } ![i] = Append(history[i], newTransaction)]$
$\land\ ackIndex' = [ackIndex \text{ EXCEPT } ![i][i] = Len(history'[i])]$ necessary, to push $commitIndex$
$\land$ UNCHANGED $\langle msgs, state, currentEpoch, leaderEpoch, leaderOracle, commitIndex, cluster, cepochR$
$ackeRecv, ackldRecv, sendCounter, initialHistory, committedIndex, tempVars, cepoch$

$LeaderBroadcast1(i) \triangleq$

test restrictions

$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = Leader$
$\land\ sendCounter[i] < currentCounter[i]$
$\land$ 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 $m \triangleq [msource \mapsto i, mtype \mapsto PROPOSE, mepoch \mapsto currentEpoch[i], mproposal \mapsto toE$
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.

$FollowerBroadcast1(i, j) \triangleq$

test restrictions

$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i]) \leq 2$
$\land\ state[i] = Follower$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = PROPOSE$

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

In phase $l32$, leader receives ack from a quorum of followers to a certain proposal, and commits the proposal.

$LeaderHandleACK(i, j) \triangleq$
test restrictions
$\land currentEpoch[i] \leq 3$
$\land Len(history[i]) \leq 2$
$\land state[i] = Leader$
$\land msgs[j][i] \neq \langle\rangle$
$\land msgs[j][i][1].mtype = ACK$
$\land$ 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 $\langle serverVars, cluster, cepochRecv, ackeRecv, ackldRecv, currentCounter,$
$sendCounter, initialHistory, committedIndex, tempVars, cepochSent, recoveryVars, p$

$LeaderAdvanceCommit(i) \triangleq$
test restrictions
$\land currentEpoch[i] \leq 3$
$\land Len(history[i]) \leq 2$
$\land state[i] = Leader$
$\land commitIndex[i] < Len(history[i])$
$\land$ LET $Agree(index) \triangleq \{i\} \cup \{k \in (Server \setminus \{i\}) : ackIndex[i][k] \geq index\}$
$agreeIndexes \triangleq \{index \in (commitIndex[i] + 1) \,..\, Len(history[i]) : Agree(index) \in Quoru$

```
                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] ≤ 3
        ∧ 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.
```

$\land \neg infoOk$
$\land Reply(i,\ j,\ [mtype \mapsto CEPOCH,$
$mepoch \mapsto currentEpoch[i]])$
$\land$ UNCHANGED $commitIndex$
$\lor$ stale $COMMIT$ − discard
$\land \neg replyOk$
$\land Discard(j,\ i)$
$\land$ UNCHANGED $commitIndex$
$\land$ UNCHANGED $\langle state,\ currentEpoch,\ leaderEpoch,\ history,\ leaderOracle,$
$leaderVars,\ tempVars,\ cepochSent,\ recoveryVars,\ proposalMsgsLog\rangle$

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 *l*33, upon receiving *CEPOCH*, leader l proposes back *NEWEPOCH* and *NEWLEADER*.
$LeaderHandleCEPOCHinPhase3(i,\ j) \triangleq$
test restrictions
$\land currentEpoch[i] \leq 3$
$\land Len(history[i]) \leq 2$
$\land state[i] = Leader$
$\land msgs[j][i] \neq \langle\rangle$
$\land msgs[j][i][1].mtype = CEPOCH$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor \land currentEpoch[i] \geq msg.mepoch$
$\land Reply2(i,\ j,\ [mtype \mapsto NEWEPOCH,$
$mepoch \mapsto currentEpoch[i]],$
$[mtype \mapsto NEWLEADER,$
$mepoch \mapsto currentEpoch[i],$
$minitialHistory \mapsto history[i]])$
$\lor \land currentEpoch[i] < msg.mepoch$
$\land$ UNCHANGED $msgs$
$\land$ UNCHANGED $\langle serverVars,\ leaderVars,\ tempVars,\ cepochSent,\ recoveryVars,\ proposalMsgsLog\rangle$

In phase *l*34, upon receiving ack from f of the *NEWLEADER*, it sends a commit message to f.
Leader l also makes $Q := Q \cup \{f\}$.
$LeaderHandleACKLDinPhase3(i,\ j) \triangleq$
test restrictions
$\land currentEpoch[i] \leq 3$
$\land Len(history[i]) \leq 2$
$\land state[i] = Leader$
$\land msgs[j][i] \neq \langle\rangle$

$\land\ msgs[j][i][1].mtype = ACKLD$
$\land$ LET $msg \triangleq msgs[j][i][1]$
$aimCommitIndex \triangleq Minimum(\{commitIndex[i],\ Len(msg.mhistory)\})$
$aimCommitCounter \triangleq$ 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
$\land currentEpoch[i] \leq 3$
$\land Len(history[i]) \leq 2$
$\land state[i] \neq Follower$
$\land \exists j \in Server \setminus \{i\} : \land msgs[j][i] \neq \langle\rangle$
$\land msgs[j][i][1].mtype \neq RECOVERYREQUEST$
$\land msgs[j][i][1].mtype \neq RECOVERYRESPONSE$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\land NullPoint \in cepochRecv[i]$
$\land Maximum(\{currentEpoch[i],\ leaderEpoch[i]\}) < msg.mepoch$
$\land \lor msg.mtype = NEWEPOCH$
$\lor msg.mtype = NEWLEADER$
$\lor msg.mtype = COMMITLD$
$\lor msg.mtype = PROPOSE$

$\lor\ msg.mtype = COMMIT$
$\land\ state'\ = [state\ \text{EXCEPT}\ ![i]\ = Follower]$
$\land\ currentEpoch' = [currentEpoch\ \text{EXCEPT}\ ![i] = msg.mepoch]$
$\land\ leaderOracle'\ = [leaderOracle\ \text{EXCEPT}\ ![i]\ = j]$
$\land\ Reply(i, j, [mtype\ \mapsto CEPOCH,$
$mepoch \mapsto currentEpoch[i]])$
Here we should not use *Discard*.
$\land$ UNCHANGED $\langle leaderEpoch, history, commitIndex, leaderVars, tempVars, cepochSent, recoveryVars,$

$DiscardStaleMessage(i) \triangleq$
test restrictions
$\land\ currentEpoch[i] \leq 3$
$\land\ Len(history[i])\ \leq 2$
$\land \exists j \in Server \setminus \{i\} : \land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype \neq RECOVERYREQUEST$
$\land\ msgs[j][i][1].mtype \neq RECOVERYRESPONSE$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor \land state[i] = Follower$
$\land$ $\lor msg.mepoch < currentEpoch[i] \setminus *$ Discussed before.
$\lor\ msg.mtype = CEPOCH$
$\lor\ msg.mtype = ACKE$
$\lor\ msg.mtype = ACKLD$
$\lor\ msg.mtype = ACK$
$\lor \land state[i] \neq Follower$
$\land\ msg.mtype \neq CEPOCH$
$\land \lor \land state[i] = ProspectiveLeader$
$\land \lor msg.mtype = ACK$
$\lor \land msg.mepoch \leq Maximum(\{currentEpoch[i], leaderEpoch[i]$
$\land \lor msg.mtype = NEWEPOCH$
$\lor\ msg.mtype = NEWLEADER$
$\lor\ msg.mtype = COMMITLD$
$\lor\ msg.mtype = PROPOSE$
$\lor\ msg.mtype = COMMIT$
$\lor \land state[i] = Leader$
$\land \lor msg.mtype = ACKE$
$\lor \land msg.mepoch \leq currentEpoch[i]$
$\land \lor msg.mtype = NEWEPOCH$
$\lor\ msg.mtype = NEWLEADER$
$\lor\ msg.mtype = COMMITLD$
$\lor\ msg.mtype = PROPOSE$
$\lor\ msg.mtype = COMMIT$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $\langle serverVars, leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog\rangle$

Defines how the variables may transition.

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

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

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.

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

$$
\begin{array}{l}
\quad \lor \ \land \ state[j] = ProspectiveLeader \\
\qquad \land \ NullPoint \ \in ackeRecv[j] \\
\land \ currentEpoch[i] = currentEpoch[j] \\
\Rightarrow \ i = j
\end{array}
$$

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.

$$
\begin{array}{l}
equal(entry1,\ entry2) \ \triangleq \ \land \ entry1.epoch \quad = entry2.epoch \\
\qquad\qquad\qquad\qquad\qquad\quad \land \ entry1.counter = entry2.counter
\end{array}
$$

$$
\begin{array}{l}
precede(entry1,\ entry2) \ \triangleq \ \lor \ entry1.epoch < entry2.epoch \\
\qquad\qquad\qquad\qquad\qquad\quad \lor \ \land \ entry1.epoch \quad = entry2.epoch \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \ entry1.counter < entry2.counter
\end{array}
$$

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

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

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

$$
\begin{array}{l}
Integrity \ \triangleq \ \forall\, i \in Server : \\
\qquad state[i] = Follower \land commitIndex[i] > 0 \\
\qquad \Rightarrow \forall\, index \in 1\ldots commitIndex[i] : \exists\, msg \in proposalMsgsLog : \\
\qquad\qquad \lor \ \land \ msg.mtype = PROPOSE \\
\qquad\qquad\quad \land \ equal(msg.mproposal,\ history[i][index]) \\
\qquad\qquad \lor \ \land \ msg.mtype = NEWLEADER \\
\qquad\qquad\quad \land \exists\, pindex \quad \in 1\ldots Len(msg.mproposals) : equal(msg.mproposals[pindex],\ history[i][ind
\end{array}
$$

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

$$
\begin{array}{l}
Agreement \ \triangleq \ \forall\, i,\, j \in Server : \\
\qquad \land \ state[i] = Follower \land commitIndex[i] > 0 \\
\qquad \land \ state[j] = Follower \land commitIndex[j] > 0 \\
\qquad \Rightarrow \\
\qquad \forall\, index1 \in 1\ldots commitIndex[i],\ index2 \in 1\ldots commitIndex[j] : \\
\qquad\quad \lor \exists\, indexj \in 1\ldots commitIndex[j] : \\
\qquad\qquad equal(history[j][indexj],\ history[i][index1]) \\
\qquad\quad \lor \exists\, indexi \in 1\ldots commitIndex[i] : \\
\qquad\qquad equal(history[i][indexi],\ history[j][index2])
\end{array}
$$

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

$$
\begin{array}{l}
TotalOrder \ \triangleq \ \forall\, i,\, j \in Server : commitIndex[i] \geq 2 \land commitIndex[j] \geq 2 \\
\qquad\qquad \Rightarrow \forall\, indexi1 \in 1\ldots(commitIndex[i] - 1) : \forall\, indexi2 \in (indexi1 + 1)\ldots commitIndex[i] :
\end{array}
$$

$$
\begin{array}{l}
\text{LET } logOk \triangleq \exists\, index \in 1\,..\,commitIndex[j] : equal(history[i][indexi2],\, history[j][index] \\
\text{IN} \quad \lor \neg logOk \\
\qquad \lor \land logOk \\
\qquad\quad \land \exists\, indexj2 \in 1\,..\,commitIndex[j] : \\
\qquad\qquad\qquad \land equal(history[i][indexi2],\, history[j][indexj2]) \\
\qquad\qquad\qquad \land \exists\, indexj1 \in 1\,..\,(indexj2-1) : equal(history[i][indexi1],\, histo
\end{array}
$$

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

$$
\begin{array}{l}
LocalPrimaryOrder \triangleq \text{LET } mset(i,\, e) \triangleq \{msg \in proposalMsgsLog : \land msg.mtype = PROPOSE \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land msg.msource = i \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land msg.mepoch = e\} \\
\qquad\qquad mentries(i,\, e) \triangleq \{msg.mproposal : msg \in mset(i,\, e)\} \\
\qquad \text{IN} \quad \forall\, i \in Server : \forall\, e \in 1\,..\,currentEpoch[i] : \\
\qquad\qquad \lor Cardinality(mentries(i,\, e)) < 2 \\
\qquad\qquad \lor \land Cardinality(mentries(i,\, e)) \geq 2 \\
\qquad\qquad\quad \land \exists\, tsc1,\, tsc2 \in mentries(i,\, e) : \\
\qquad\qquad\qquad \lor equal(tsc1,\, tsc2) \\
\qquad\qquad\qquad \lor \land \neg equal(tsc1,\, tsc2) \\
\qquad\qquad\qquad\quad \land \text{LET } tscPre \triangleq \text{IF } precede(tsc1,\, tsc2) \text{ THEN } tsc1 \text{ ELSE } tsc2 \\
\qquad\qquad\qquad\qquad\qquad tscNext \triangleq \text{IF } precede(tsc1,\, tsc2) \text{ THEN } tsc2 \text{ ELSE } tsc1 \\
\qquad\qquad\qquad\qquad\quad \text{IN} \quad \forall\, j \in Server : \land commitIndex[j] \geq 2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \exists\, index \in 1\,..\,commitIndex[j] : equal(history[j][ind \\
\qquad\qquad\qquad\qquad\qquad\quad \Rightarrow \exists\, index2 \in 1\,..\,commitIndex[j] : \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land equal(history[j][index2],\, tscNext) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land index2 > 1 \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \exists\, index1 \in 1\,..\,(index2-1) : equal(history[j][index1],\, tscPre
\end{array}
$$

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.

$$
\begin{array}{l}
GlobalPrimaryOrder \triangleq \forall\, i \in Server : commitIndex[i] \geq 2 \\
\qquad\qquad\qquad\qquad \Rightarrow \forall\, idx1,\, idx2 \in 1\,..\,commitIndex[i] : \lor history[i][idx1].epoch \geq history[i][idx2].ep \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \lor \land history[i][idx1].epoch < history[i][idx2] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land idx1 < idx2
\end{array}
$$

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.

$$
\begin{array}{l}
PrimaryIntegrity \triangleq \forall\, i,\, j \in Server : \land state[i] = Leader \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land state[j] = Follower \land commitIndex[j] \geq 1 \\
\qquad\qquad\qquad\qquad\qquad\quad \Rightarrow \forall\, index \in 1\,..\,commitIndex[j] : \lor history[j][index].epoch \geq currentEpoch[i] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \lor \land history[j][index].epoch < currentEpoch[i] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \exists\, idx \in 1\,..\,commitIndex[i] : equal(history[i][i
\end{array}
$$

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* May 03 21:57:22 *CST* 2021 by Dell
\ ∗ Created Sat *Dec* 05 13:32:08 *CST* 2020 by Dell