MODULE *Zab*

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, 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*

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

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

A server becomes pleader and a quorum servers knows that.

$Election(i,\ Q)\ \triangleq$
$\quad \wedge\ i \in Q$
$\quad \wedge\ state' = [s \in Server \mapsto \text{IF } s = i \text{ THEN } ProspectiveLeader \text{ ELSE IF } s \in Q \text{ THEN } Follower \text{ ELSE } state[s]]$
$\quad \wedge\ cluster' = [cluster \text{ EXCEPT } ![i] = Q]$ cluster is first initialized in election, not *phase*1.
$\quad \wedge\ cepochRecv' = [cepochRecv \text{ EXCEPT } ![i] = \{i\}]$
$\quad \wedge\ ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \{i\}]$
$\quad \wedge\ ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] = \{i\}]$
$\quad \wedge\ ackIndex' = [ii \in Server \mapsto [ij \in Server \mapsto \text{IF } ii = i \text{ THEN } 0 \text{ ELSE } ackIndex[ii][ij]]]$
$\quad \wedge\ committedIndex' = [committedIndex \text{ EXCEPT } ![i] = 0]$
$\quad \wedge\ initialHistory' = [initialHistory \text{ EXCEPT } ![i] = \langle\rangle]$
$\quad \wedge\ tempMaxEpoch' = [tempMaxEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$
$\quad \wedge\ tempMaxLastEpoch' = [tempMaxLastEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$
$\quad \wedge\ tempInitialHistory' = [tempInitialHistory \text{ EXCEPT } ![i] = history[i]]$
$\quad \wedge\ leaderOracle' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } i \text{ ELSE } leaderOracle[s]]$
$\quad \wedge\ leaderEpoch' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } currentEpoch[s] \text{ ELSE } leaderEpoch[s]]$
$\quad \wedge\ cepochSent' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN FALSE} \text{ ELSE } cepochSent[s]]$
$\quad \wedge\ msgs' = [ii \in Server \mapsto [ij \in Server \mapsto \text{IF } ii \in Q \vee ij \in Q \text{ THEN } \langle\rangle \text{ ELSE } msgs[ii][ij]]]$

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.

$InitialElection(i,\ Q)\ \triangleq$
$\quad \wedge \forall\, s \in Server : state[i] = Follower \wedge leaderOracle[i] = NullPoint$
$\quad \wedge\ Election(i,\ Q)$
$\quad \wedge \text{UNCHANGED } \langle currentEpoch,\ history,\ commitIndex,\ currentCounter,\ sendCounter,\ recoveryVars,\ pr$

The leader finds timeout with another follower.

$LeaderTimeout(i,\ j)\ \triangleq$
$\quad \wedge\ state[i] \neq Follower$
$\quad \wedge\ j \neq i$
$\quad \wedge\ j \in cluster[i]$
$\quad \wedge \text{LET } newCluster \triangleq cluster[i] \setminus \{j\}$
$\quad \text{IN} \quad \wedge \vee \wedge\ newCluster \in Quorums$
$\qquad\qquad\qquad \wedge\ cluster' = [cluster \text{ EXCEPT } ![i] = newCluster]$
$\qquad\qquad\qquad \wedge\ clean(i,\ j)$
$\qquad\qquad\qquad \wedge \text{UNCHANGED } \langle state,\ cepochRecv,\ ackeRecv,\ ackldRecv,\ ackIndex,\ committedIndex,\ initi$
$\qquad\qquad\qquad\qquad\qquad tempMaxEpoch,\ tempMaxLastEpoch,\ tempInitialHistory,\ leaderOracle,\ le$

$\lor \land newCluster \notin Quorums$
$\land$ LET $Q \triangleq$ CHOOSE $q \in Quorums$: $i \in q$
$v \triangleq$ CHOOSE $s \in Q$: 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.
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.

$Restart(i) \triangleq$
$\land state' = [state$ 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 $msgs[ii][ij]]]$
$\land recoverySent' = [recoverySent$ EXCEPT $![i] =$ FALSE$]$
$\land$ UNCHANGED $\langle currentEpoch, leaderEpoch, history, leaderVars, tempVars,$
$recoveryRespRecv, recoveryMaxEpoch, recoveryMEOracle, proposalMsgsLog\rangle$

$RecoveryAfterRestart(i) \triangleq$
$\land state[i] = Follower$
$\land leaderOracle[i] = NullPoint$
$\land \neg recoverySent[i]$
$\land recoveryRespRecv' = [recoveryRespRecv$ EXCEPT $![i] = \{\}]$
$\land recoveryMaxEpoch' = [recoveryMaxEpoch$ EXCEPT $![i] = currentEpoch[i]]$
$\land recoveryMEOracle' = [recoveryMEOracle$ EXCEPT $![i] = NullPoint]$
$\land recoverySent' = [recoverySent$ EXCEPT $![i] =$ TRUE$]$
$\land BroadcastToAll(i, [mtype \mapsto RECOVERYREQUEST])$
$\land$ 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 ↦ leaderOracle[i],
                    mepoch  ↦ currentEpoch[i]])
        ∧ UNCHANGED ⟨serverVars, leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog⟩

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

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

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

```
FollowerDiscovery1(i) ≜
        ∧ state[i] = Follower
        ∧ leaderOracle[i] ≠ NullPoint
        ∧ ¬cepochSent[i]
        ∧ LET leader ≜ leaderOracle[i]
```

IN $Send(i,\ leader,\ [mtype \mapsto CEPOCH,$
$mepoch \mapsto currentEpoch[i]])$
$\land\ cepochSent' = [cepochSent \text{ EXCEPT } ![i] = \text{TRUE}]$
$\land \text{ UNCHANGED } \langle serverVars,\ leaderVars,\ tempVars,\ recoveryVars,\ proposalMsgsLog\rangle$

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

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

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

```
In phase f12, follower receives NEWEPOCH. If e' > f.p then sends back ACKE,
and ACKE contains f.a and hf to help pleader choose a newer history.
FollowerDiscovery2(i, j) ≜
    ∧ state[i] = Follower
    ∧ msgs[j][i] ≠ ⟨⟩
    ∧ msgs[j][i][1].mtype = NEWEPOCH
    ∧ LET msg ≜ msgs[j][i][1]
      IN  ∨ new NEWEPOCH − accept and reply
          ∧ currentEpoch[i] < msg.mepoch
          ∧ currentEpoch' = [currentEpoch EXCEPT ![i] = msg.mepoch]
          ∧ leaderOracle' = [leaderOracle EXCEPT ![i] = j]
          ∧ Reply(i, j, [mtype      ↦ ACKE,
                         mepoch     ↦ msg.mepoch,
                         mlastEpoch ↦ leaderEpoch[i],
                         mhf        ↦ history[i]])
        ∨ ∧ currentEpoch[i] = msg.mepoch
          ∧ ∨ ∧ leaderOracle[i] = j
              ∧ Reply(i, j, [mtype      ↦ ACKE,
                             mepoch     ↦ msg.mepoch,
                             mlastEpoch ↦ leaderEpoch[i],
                             mhf        ↦ history[i]])
              ∧ UNCHANGED ⟨currentEpoch, leaderOracle⟩
            ∨ It may happen when a leader do not update new epoch to all followers in Q, and a new election begi
              ∧ leaderOracle[i] ≠ j
              ∧ leaderOracle' = [leaderOracle EXCEPT ![i] = j]
              ∧ Reply(i, j, [mtype      ↦ ACKE,
                             mepoch     ↦ msg.mepoch,
                             mlastEpoch ↦ leaderEpoch[i],
                             mhf        ↦ history[i]])
              ∧ UNCHANGED currentEpoch
        ∨ stale NEWEPOCH − diacard
          ∧ currentEpoch[i] > msg.mepoch
          ∧ Discard(j, i)
          ∧ UNCHANGED ⟨currentEpoch, leaderOracle⟩
    ∧ UNCHANGED ⟨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) ≜
    ∧ state[i] = ProspectiveLeader
    ∧ msgs[j][i] ≠ ⟨⟩
    ∧ msgs[j][i][1].mtype = ACKE
    ∧ LET msg ≜ msgs[j][i][1]
          infoOk ≜ ∨ msg.mlastEpoch > tempMaxLastEpoch[i]
                   ∨ ∧ msg.mlastEpoch = tempMaxLastEpoch[i]
```

$\land \lor LastZxid(msg.mhf)[1] > LastZxid(tempInitialHistory[i])[1]$
$\lor \land LastZxid(msg.mhf)[1] = LastZxid(tempInitialHistory[i])[1]$
$\land LastZxid(msg.mhf)[2] \geq LastZxid(tempInitialHistory[i])[2]$

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

$LeaderDiscovery2Sync1(i) \triangleq$
$\land state[i] = ProspectiveLeader$
$\land ackeRecv[i] \in Quorums$
$\land currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = leaderEpoch[i]]$
$\land history' = [history \text{ EXCEPT } ![i] = tempInitialHistory[i]]$
$\land initialHistory' = [initialHistory \text{ EXCEPT } ![i] = tempInitialHistory[i]]$
$\land ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \{NullPoint\}]$
$\land ackIndex' = [ackIndex \text{ EXCEPT } ![i][i] = Len(tempInitialHistory[i])]$
until now, $phase1(Discovery)$ ends
$\land Broadcast(i, [mtype \mapsto NEWLEADER,$
$mepoch \mapsto currentEpoch'[i],$
$minitialHistory \mapsto history'[i]])$
$\land$ 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  ∨ new NEWLEADER − accept and reply
              ∧ currentEpoch[i] ≤ msg.mepoch
              ∧ currentEpoch' = [currentEpoch EXCEPT ![i] = msg.mepoch]
              ∧ leaderEpoch'  = [leaderEpoch  EXCEPT ![i] = msg.mepoch]
              ∧ leaderOracle' = [leaderOracle EXCEPT ![i] = j]
              ∧ history'      = [history      EXCEPT ![i] = msg.minitialHistory]
              ∧ Reply(i, j, [mtype     ↦ ACKLD,
                             mepoch    ↦ msg.mepoch,
                             mhistory  ↦ msg.minitialHistory])
            ∨ stale NEWLEADER − discard
              ∧ currentEpoch[i] > msg.mepoch
              ∧ Discard(j, i)
              ∧ UNCHANGED ⟨currentEpoch, leaderEpoch, leaderOracle, history⟩
      ∧ UNCHANGED ⟨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) ≜
      ∧ state[i] = ProspectiveLeader
      ∧ msgs[j][i] ≠ ⟨⟩
      ∧ msgs[j][i][1].mtype = ACKLD
      ∧ LET msg ≜ msgs[j][i][1]
        IN  ∨ new ACK-LD - accept
              ∧ currentEpoch[i] = msg.mepoch
              ∧ ackIndex'  = [ackIndex  EXCEPT ![i][j] = Len(initialHistory[i])]
              ∧ ackldRecv' = [ackldRecv EXCEPT ![i] = IF j ∉ ackldRecv[i] THEN ackldRecv[i] ∪ {j}
                                                                         ELSE ackldRecv[i]]
            ∨ stale ACK-LD - discard
              ∧ currentEpoch[i] ≠ msg.mepoch
              ∧ UNCHANGED ⟨ackldRecv, ackIndex⟩
      ∧ Discard(j, i)
      ∧ UNCHANGED ⟨serverVars, cluster, cepochRecv, ackeRecv, currentCounter,
                   sendCounter, initialHistory, committedIndex, tempVars, cepochSent, recoveryVars, p

LeaderSync2(i) ≜
      ∧  state[i] = ProspectiveLeader
      ∧  ackldRecv[i] ∈ Quorums
      ∧  commitIndex'     = [commitIndex     EXCEPT ![i] = Len(history[i])]
      ∧  committedIndex'  = [committedIndex  EXCEPT ![i] = Len(history[i])]
      ∧  state'           = [state           EXCEPT ![i] = Leader]
      ∧  currentCounter'  = [currentCounter  EXCEPT ![i] = 0]
      ∧  sendCounter'     = [sendCounter     EXCEPT ![i] = 0]
      ∧  ackldRecv'       = [ackldRecv       EXCEPT ![i] = {NullPoint}]
      ∧  Broadcast(i, [mtype   ↦ COMMITLD,
                       mepoch  ↦ currentEpoch[i],
                       mlength ↦ Len(history[i])])
```

$\land$ 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.

$FollowerSync2(i, j) \triangleq$
$\land\ state[i] = Follower$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = COMMITLD$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ new *COMMIT*-LD - commit all transactions in initial history
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$ unnecessary
$\land\ \lor\ \land\ Len(history[i]) = msg.mlength$
$\land\ commitIndex' = [commitIndex$ 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
$<$: 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 $\langle state, currentEpoch, leaderEpoch, history, leaderVars, tempVars, cepochSent, recovery$

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

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

$LeaderBroadcast1(i) \triangleq$
$\land\ state[i] = Leader$
$\land\ sendCounter[i] < currentCounter[i]$
$\land$ 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 $m \triangleq [msource \mapsto i,\ mtype \mapsto PROPOSE,\ mepoch \mapsto currentEpoch[i],\ mproposal \mapsto toB$
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 $\lor\ msg.mproposal.counter = 1$
$\lor\ msg.mrpoposal.counter = history[Len(history)].counter + 1$
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ history' = [history \text{ EXCEPT } ![i] = Append(history[i],\ msg.mproposal)]$
$\land\ leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] = j]$
$\land\ Reply(i,\ j,\ [mtype \mapsto ACK,$
$mepoch \mapsto currentEpoch[i],$
$mindex \mapsto Len(history'[i])])$
$\lor$ 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 $ackIndex[i][j] + 1 \triangleq msg.mindex$
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ ackIndex' = [ackIndex \text{ EXCEPT } ![i][j] = Maximum(\{ackIndex[i][j],\ msg.mindex\})]$
$\lor$ 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 $\langle serverVars,\ cluster,\ cepochRecv,\ ackeRecv,\ ackldRecv,\ currentCounter,$
$sendCounter,\ initialHistory,\ committedIndex,\ tempVars,\ cepochSent,\ recoveryVars,\ p$

$LeaderAdvanceCommit(i) \triangleq$

```
        ∧ state[i] = Leader
        ∧ commitIndex[i] < Len(history[i])
        ∧ LET Agree(index)     ≜ {i} ∪ {k ∈ (Server \ {i}) : ackIndex[i][k] ≥ index}
              agreeIndexes     ≜ {index ∈ (commitIndex[i] + 1) .. Len(history[i]) : Agree(index) ∈ Quoru
              newCommitIndex   ≜ IF agreeIndexes ≠ {} THEN Maximum(agreeIndexes)
                                                      ELSE commitIndex[i]
          IN  commitIndex' = [commitIndex EXCEPT ![i] = newCommitIndex]
        ∧ UNCHANGED ⟨state, currentEpoch, leaderEpoch, leaderOracle, history,
                     msgs, leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog⟩

LeaderBroadcast2(i) ≜
        ∧ 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) ≜
        ∧ state[i] = Follower
        ∧ msgs[j][i] ≠ ⟨⟩
        ∧ msgs[j][i][1].mtype = COMMIT
        ∧ LET msg ≜ msgs[j][i][1]
          IN  ∨ ∧ currentEpoch[i] = msg.mepoch
                ∧ leaderOracle' = [leaderOracle EXCEPT ![i] = j]
                ∧ LET infoOk ≜ ∧ Len(history[i]) ≥ msg.mindex
                               ∧ ∨ ∧ msg.mindex > 0
                                   ∧ history[i][msg.mindex].epoch = msg.mepoch
                                   ∧ history[i][msg.mindex].counter = msg.mcounter
                                 ∨ msg.mindex = 0
                  IN  ∨  new COMMIT − commit transaction in history
                         ∧ infoOk
                         ∧ commitIndex' = [commitIndex EXCEPT ![i] = Maximum({commitIndex[i], msg.
                         ∧ Discard(j, i)
                      ∨  It may happen when the server is a new follower who joined in the cluster,
                         and it misses the corresponding PROPOSE.
                         ∧ ¬infoOk
                         ∧ Reply(i, j, [mtype  ↦ CEPOCH,
                                        mepoch ↦ currentEpoch[i]])
                         ∧ UNCHANGED commitIndex
```

$\lor$ stale *COMMIT* − discard
$\land$ $currentEpoch[i] \neq msg.mepoch$
$\land$ $Discard(j,\ i)$
$\land$ UNCHANGED $\langle commitIndex,\ leaderOracle\rangle$
$\land$ UNCHANGED $\langle state,\ currentEpoch,\ leaderEpoch,\ history,$
$leaderVars,\ tempVars,\ cepochSent,\ recoveryVars,\ proposalMsgsLog\rangle$

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.

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

$LeaderHandleCEPOCHinPhase3(i,\ j) \triangleq$
$\land$ $state[i] = Leader$
$\land$ $msgs[j][i] \neq \langle\rangle$
$\land$ $msgs[j][i][1].mtype = CEPOCH$
$\land$ 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 $Q := Q \cup \{f\}$.

$LeaderHandleACKLDinPhase3(i,\ j) \triangleq$
$\land$ $state[i] = Leader$
$\land$ $msgs[j][i] \neq \langle\rangle$
$\land$ $msgs[j][i][1].mtype = ACKLD$
$\land$ 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] ∪ {j}]
    ∧ UNCHANGED ⟨serverVars, cepochRecv, ackeRecv, ackldRecv, currentCounter, sendCounter,
                 initialHistory, committedIndex, tempVars, cepochSent, recoveryVars, proposalMsgsLo

  Let me suppose three 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) ≜
    ∧ state[i] ≠ Follower
    ∧ ∃ j ∈ Server \ {i} : ∧ msgs[j][i] ≠ ⟨⟩
                           ∧ msgs[j][i][1].mtype ≠ RECOVERYREQUEST
                           ∧ msgs[j][i][1].mtype ≠ RECOVERYRESPONSE
                           ∧ LET msg ≜ msgs[j][i][1]
                             IN  ∧ Maximum({currentEpoch[i], leaderEpoch[i]}) < msg.mepoch
                                 ∧ ∨ msg.mtype = NEWEPOCH
                                   ∨ msg.mtype = NEWLEADER
                                   ∨ msg.mtype = COMMITLD
                                   ∨ msg.mtype = PROPOSE
                                   ∨ msg.mtype = COMMIT
                                 ∧ state'       = [state       EXCEPT ![i] = Follower]
                                 ∧ currentEpoch' = [currentEpoch EXCEPT ![i] = msg.mepoch]
                                 ∧ leaderOracle' = [leaderOracle EXCEPT ![i] = j]
                                  Here we should not use Discard.
    ∧ UNCHANGED ⟨leaderEpoch, history, commitIndex, msgs, leaderVars, tempVars, cepochSent, recovery

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

DiscardStaleMessage(i) ≜
    ∧ ∃ j ∈ Server \ {i} : ∧ msgs[j][i] ≠ ⟨⟩
                           ∧ msgs[j][i][1].mtype ≠ RECOVERYREQUEST
                           ∧ msgs[j][i][1].mtype ≠ RECOVERYRESPONSE
                           ∧ LET msg ≜ msgs[j][i][1]
                             IN  ∨ ∧ state[i] = Follower
                                   ∧  ∨ msg.mepoch < currentEpoch[i]  \* Discussed before.
                                      ∨ msg.mtype = CEPOCH
                                      ∨ msg.mtype = ACKE
                                      ∨ msg.mtype = ACKLD
                                      ∨ msg.mtype = ACK
                                 ∨ ∧ state[i] ≠ Follower
                                   ∧ msg.mtype ≠ CEPOCH
                                   ∧ ∧ ∨ ∧ state[i] = ProspectiveLeader
                                         ∧ ∨ msg.mtype = ACK
                                           ∨ ∧ msg.mepoch ≤ Maximum({currentEpoch[i], leaderEpoch[i]
```

$$
\begin{array}{l}
\land \lor msg.mtype = NEWEPOCH \\
\quad \lor msg.mtype = NEWLEADER \\
\quad \lor msg.mtype = COMMITLD \\
\quad \lor msg.mtype = PROPOSE \\
\quad \lor msg.mtype = COMMIT \\
\lor \land state[i] = Leader \\
\quad \land \lor msg.mtype = ACKE \\
\quad \quad \lor \land msg.mepoch \leq currentEpoch[i] \\
\quad \quad \quad \land \lor msg.mtype = NEWEPOCH \\
\quad \quad \quad \quad \lor msg.mtype = NEWLEADER \\
\quad \quad \quad \quad \lor msg.mtype = COMMITLD \\
\quad \quad \quad \quad \lor msg.mtype = PROPOSE \\
\quad \quad \quad \quad \lor msg.mtype = COMMIT \\
\land Discard(j,\ i) \\
\land \text{UNCHANGED } \langle serverVars,\ leaderVars,\ tempVars,\ cepochSent,\ recoveryVars,\ proposalMsgsLog \rangle
\end{array}
$$

Defines how the variables may transition.

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

$\lor \exists\, i, j \in Server : \quad LeaderHandleCEPOCHinPhase3(i, j)$
$\lor \exists\, i, j \in Server : \quad LeaderHandleACKLDinPhase3(i, j)$
$\lor \exists\, i \in Server : \quad DiscardStaleMessage(i)$
$\lor \exists\, i \in Server : \quad BecomeFollower(i)$

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

Safety properties of *Zab* consensus algorithm

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

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

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

$$
\begin{aligned}
equal(entry1,\ entry2) \triangleq\ & \land\ entry1.epoch\ \ = entry2.epoch \\
& \land\ entry1.counter = entry2.counter
\end{aligned}
$$

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

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

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

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

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

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

$Agreement \triangleq \forall\, i, j \in Server :$

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

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

$$
\begin{array}{l}
TotalOrder \triangleq\ \forall\, i,\, j \in Server : commitIndex[i] \geq 2 \land commitIndex[j] \geq 2 \\
\qquad \Rightarrow \forall\, indexi1 \in 1\,..\,(commitIndex[i] - 1) : \forall\, indexi2 \in (indexi1 + 1)\,..\,commitIndex[i] : \\
\qquad\quad \text{LET}\ logOk \triangleq \exists\, index \in 1\,..\,commitIndex[j] : equal(history[i][indexi2],\, history[j][index] \\
\qquad\quad \text{IN}\quad \lor \neg logOk \\
\qquad\qquad \lor\ \land logOk \\
\qquad\qquad\quad \land \exists\, indexj2 \in 1\,..\,commitIndex[j] : \\
\qquad\qquad\qquad\qquad \land\, equal(history[i][indexi2],\, history[j][indexj2]) \\
\qquad\qquad\qquad\qquad \land \exists\, indexj1 \in 1\,..\,(indexj2 - 1) : equal(history[i][indexi1],\, histo
\end{array}
$$

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

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

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.

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

$\wedge\ idx1 < idx2$

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.

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

\ ∗ Modification History
\ ∗ Last modified *Thu Apr* 29 17:16:17 *CST* 2021 by Dell
\ ∗ Created Sat *Dec* 05 13:32:08 *CST* 2020 by Dell