$\overline{}$ MODULE *Zab* $\overline{}$

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

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, currently not used
CONSTANT *Epoches*

$\vdash$

Return the maximum value from the set S
$Maximum(S) \triangleq$ IF $S = \{\}$ THEN -1
 ELSE CHOOSE $n \in S : \forall m \in S : n \geq m$

Return the minimum value from the set S
$Minimum(S) \triangleq$ IF $S = \{\}$ THEN -1
 ELSE CHOOSE $n \in S : \forall m \in S : n \leq m$

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

$None \triangleq$ CHOOSE $v : v \notin Value$

$NullPoint \triangleq$ CHOOSE $p : p \notin Server$

$\vdash$

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*

1

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

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

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

Add a message to $msgs$ − add a message m to $msgs[i][j]$
$Send(i, j, m) \triangleq msgs' = [msgs$ EXCEPT $![i][j] = Append(msgs[i][j], m)]$

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

Remove a message from $msgs$ − discard head of $msgs[i][j]$
$Discard(i, j) \triangleq msgs' =$ IF $msgs[i][j] \neq \langle \rangle$ THEN $[msgs$ EXCEPT $![i][j] = Tail(msgs[i][j])]$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ ELSE $msgs$

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

3

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

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

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

A server becomes pleader and a quorum servers knows that.

4

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

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

The leader finds timeout with another follower.

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

5

$$\lor \land newCluster \notin Quorums$$
$$\land \text{ LET } Q \;\triangleq\; \text{CHOOSE } q \in Quorums\colon i \in q$$
$$\phantom{\land \text{ LET }} v \;\triangleq\; \text{CHOOSE } s \in Q\colon \text{TRUE}$$
$$\phantom{\land \text{ LET }} \text{IN} \quad Election(v,\, Q)$$
$$\exists\, Q \in Quorums : \land i \in Q$$
$$ \land \exists\, v \in Q : Election(v,\, Q)$$
$$\land \text{UNCHANGED } \langle currentEpoch,\, history,\, commitIndex,\, currentCounter,\, sendCounter,\, recoveryVars,\, pr$$

A follower finds timeout with the leader.

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

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.

$Restart(i) \;\triangleq\;$
$$\land state' \qquad\quad = [state \qquad\quad\;\; \text{EXCEPT } ![i] \quad = Follower]$$
$$\land leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] \quad = NullPoint]$$
$$\land commitIndex' = [commitIndex \;\;\; \text{EXCEPT } ![i] = 0]$$
$$\land cepochSent' \;\;\; = [cepochSent \quad\; \text{EXCEPT } ![i] \;\; = \text{FALSE}]$$
$$\land msgs' \qquad\quad = [ii \in Server \mapsto [ij \in Server \mapsto \text{IF } ij = i \text{ THEN } \langle\rangle$$
$$ \text{ELSE } \quad msgs[ii][ij]]]$$
$$\land recoverySent' = [recoverySent \text{ EXCEPT } ![i] = \text{FALSE}]$$
$$\land \text{UNCHANGED } \langle currentEpoch,\, leaderEpoch,\, history,\, leaderVars,\, tempVars,$$
$$\phantom{\land \text{UNCHANGED } \langle} recoveryRespRecv,\, recoveryMaxEpoch,\, recoveryMEOracle,\, proposalMsgsLog\rangle$$

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

$HandleRecoveryRequest(i,\, j) \;\triangleq\;$
$$\land msgs[j][i] \neq \langle\rangle$$
$$\land msgs[j][i][1].mtype = RECOVERYREQUEST$$
$$\land Reply(i,\, j,\, [mtype \quad\mapsto RECOVERYRESPONSE,$$

$$
\begin{aligned}
&\qquad\qquad\qquad moracle \mapsto leaderOracle[i],\\
&\qquad\qquad\qquad mepoch \;\mapsto currentEpoch[i]])\\
&\qquad\land \text{UNCHANGED } \langle serverVars,\ leaderVars,\ tempVars,\ cepochSent,\ recoveryVars,\ proposalMsgsLog\rangle
\end{aligned}
$$

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

$FindCluster(i) \triangleq$
$\quad \land \quad state[i] = Follower$
$\quad \land \quad leaderOracle[i] = NullPoint$
$\quad \land \quad recoveryRespRecv[i] \in Quorums$
$\quad \land \quad \text{LET } infoOk \triangleq \land recoveryMEOracle[i] \neq i$
$\qquad\qquad\qquad\qquad\qquad\quad \land recoveryMEOracle[i] \neq NullPoint$
$\qquad\qquad\qquad\qquad\qquad\quad \land currentEpoch[i] \leq recoveryMaxEpoch[i]$
$\qquad\qquad \text{IN} \quad \lor \land \neg infoOk$
$\qquad\qquad\qquad\quad \land recoverySent' = [recoverySent \text{ EXCEPT } ![i] = \text{FALSE}]$
$\qquad\qquad\qquad\quad \land \text{UNCHANGED } \langle currentEpoch,\ leaderOracle,\ msgs\rangle$
$\qquad\qquad\qquad \lor \land infoOk$
$\qquad\qquad\qquad\quad \land currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = recoveryMaxEpoch[i]]$
$\qquad\qquad\qquad\quad \land leaderOracle' \; = [leaderOracle \text{ EXCEPT } ![i] \; = recoveryMEOracle[i]]$
$\qquad\qquad\qquad\quad \land Send(i, recoveryMEOracle[i], [mtype \;\; \mapsto CEPOCH,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad mepoch \mapsto recoveryMaxEpoch[i]])$
$\qquad\qquad\qquad\quad \land \text{UNCHANGED } recoverySent$
$\quad \land \quad \text{UNCHANGED } \langle state,\ leaderEpoch,\ history,\ commitIndex,\ leaderVars,\ tempVars,$
$\qquad\qquad\qquad\qquad\qquad recoveryRespRecv,\ recoveryMaxEpoch,\ recoveryMEOracle,\ cepochSent,\ proposalMsgs$

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

$FollowerDiscovery1(i) \triangleq$
$\quad \land state[i] = Follower$
$\quad \land leaderOracle[i] \neq NullPoint$
$\quad \land \neg cepochSent[i]$
$\quad \land \text{LET } leader \triangleq leaderOracle[i]$

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

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

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

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.
$FollowerDiscovery2(i, j) \triangleq$
   $\wedge\ state[i] = Follower$
   $\wedge\ msgs[j][i] \neq \langle\rangle$
   $\wedge\ msgs[j][i][1].mtype = NEWEPOCH$
   $\wedge\ \text{LET } msg \triangleq msgs[j][i][1]$
    $\text{IN}\quad \vee\ $ new $NEWEPOCH -$ accept and reply
      $\wedge\ currentEpoch[i] < msg.mepoch$
      $\wedge\ \vee\ \wedge\ leaderOracle[i] = j$
        $\wedge\ currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch]$
        $\wedge\ Reply(i, j, [mtype \quad\ \mapsto ACKE,$
             $mepoch \quad\ \mapsto msg.mepoch,$
             $mlastEpoch \mapsto leaderEpoch[i],$
             $mhf \quad\quad\ \mapsto history[i]])$
       $\vee\ \wedge\ leaderOracle[i] \neq j$
        $\wedge\ Discard(j, i)$
        $\wedge\ \text{UNCHANGED } currentEpoch$
     $\vee\ \wedge\ currentEpoch[i] = msg.mepoch$
      $\wedge\ \vee\ \wedge\ leaderOracle[i] = j$
        $\wedge\ Reply(i, j, [mtype \quad\ \mapsto ACKE,$
             $mepoch \quad\ \mapsto msg.mepoch,$
             $mlastEpoch \mapsto leaderEpoch[i],$
             $mhf \quad\quad\ \mapsto history[i]])$
        $\wedge\ \text{UNCHANGED } currentEpoch$
       $\vee\ $ It may happen when a leader do not update new epoch to all followers in Q, and a new election begi
        $\wedge\ leaderOracle[i] \neq j$
        $\wedge\ Discard(j, i)$
        $\wedge\ \text{UNCHANGED } currentEpoch$
    $\vee\ $ stale $NEWEPOCH - diacard$
     $\wedge\ currentEpoch[i] > msg.mepoch$
     $\wedge\ Discard(j, i)$
     $\wedge\ \text{UNCHANGED } currentEpoch$
   $\wedge\ \text{UNCHANGED } \langle state, leaderEpoch, leaderOracle, history, leaderVars,$
         $commitIndex, cepochSent, tempVars, recoveryVars, proposalMsgsLog\rangle$

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.
$LeaderHandleACKE(i, j) \triangleq$
   $\wedge\ state[i] = ProspectiveLeader$
   $\wedge\ msgs[j][i] \neq \langle\rangle$
   $\wedge\ msgs[j][i][1].mtype = ACKE$
   $\wedge\ \text{LET } msg \quad\ \triangleq msgs[j][i][1]$
     $infoOk \triangleq\ \vee\ msg.mlastEpoch > tempMaxLastEpoch[i]$
          $\vee\ \wedge\ msg.mlastEpoch = tempMaxLastEpoch[i]$

9

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

IN $\quad \lor \land leaderEpoch[i] = msg.mepoch$
$\quad\quad \land \lor \land infoOk$
$\quad\quad\quad\quad \land tempMaxLastEpoch' \quad = [tempMaxLastEpoch \quad$ EXCEPT $![i] = msg.mlastEpoch]$
$\quad\quad\quad\quad \land tempInitialHistory' \quad = [tempInitialHistory$ EXCEPT $![i] \quad = msg.mhf]$
$\quad\quad\quad \lor \land \neg infoOk$
$\quad\quad\quad\quad \land$ UNCHANGED $\langle tempMaxLastEpoch, tempInitialHistory\rangle$

$\quad\quad \land ackeRecv' = [ackeRecv$ EXCEPT $![i] =$ IF $j \notin ackeRecv[i]$ THEN $ackeRecv[i] \cup \{j\}$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ ELSE $\quad ackeRecv[i]]$
$\quad\quad \lor \land leaderEpoch[i] \neq msg.mepoch$
$\quad\quad\quad \land$ UNCHANGED $\langle tempMaxLastEpoch, tempInitialHistory, ackeRecv\rangle$
$\land Discard(j, i)$
$\land$ UNCHANGED $\langle serverVars, cluster, cepochRecv, ackldRecv, ackIndex, currentCounter,$
$\quad\quad\quad\quad sendCounter, initialHistory, committedIndex, cepochSent, tempMaxEpoch, recoveryVa$

$LeaderDiscovery2Sync1(i) \triangleq$
$\quad \land state[i] = ProspectiveLeader$
$\quad \land ackeRecv[i] \in Quorums$
$\quad \land currentEpoch' \quad = [currentEpoch \quad$ EXCEPT $![i] \quad = leaderEpoch[i]]$
$\quad \land history' \quad\quad = [history \quad\quad\quad$ EXCEPT $![i] \quad\quad = tempInitialHistory[i]]$
$\quad \land initialHistory' \quad = [initialHistory$ EXCEPT $![i] \quad\quad = tempInitialHistory[i]]$
$\quad \land ackeRecv' \quad\quad = [ackeRecv \quad\quad$ EXCEPT $![i] \quad = \{NullPoint\}]$
$\quad \land ackIndex' \quad\quad = [ackIndex \quad\quad$ EXCEPT $![i][i] = Len(tempInitialHistory[i])]$

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

$FollowerSync1(i, j) \triangleq$
$\quad \land state[i] = Follower$

10

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

In phase $l22$, pleader receives ACK-LD from a quorum of followers, and sends $COMMIT$-LD to followers.
$LeaderHandleACKLD(i,\ j)\ \triangleq$
$\qquad \wedge\ state[i] = ProspectiveLeader$
$\qquad \wedge\ msgs[j][i] \neq \langle\rangle$
$\qquad \wedge\ msgs[j][i][1].mtype = ACKLD$
$\qquad \wedge\ \text{LET } msg\ \triangleq\ msgs[j][i][1]$
$\qquad\quad \text{IN}\quad \vee\ $ new ACK-LD - accept
$\qquad\qquad\qquad \wedge\ currentEpoch[i] = msg.mepoch$
$\qquad\qquad\qquad \wedge\ ackIndex'\ = [ackIndex\ \ \text{EXCEPT } ![i][j] = Len(initialHistory[i])]$
$\qquad\qquad\qquad \wedge\ ackldRecv' = [ackldRecv\ \text{EXCEPT } ![i] = \text{IF } j \notin ackldRecv[i]\ \text{THEN } ackldRecv[i] \cup \{j\}$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE}\ \ ackldRecv[i]]$
$\qquad\qquad\quad \vee\ $ stale ACK-LD - discard
$\qquad\qquad\qquad \wedge\ currentEpoch[i] \neq msg.mepoch$
$\qquad\qquad\qquad \wedge\ \text{UNCHANGED } \langle ackldRecv,\ ackIndex\rangle$
$\qquad \wedge\ Discard(j,\ i)$
$\qquad \wedge\ \text{UNCHANGED } \langle serverVars,\ cluster,\ cepochRecv,\ ackeRecv,\ currentCounter,$
$\qquad\qquad\qquad\qquad\qquad sendCounter,\ initialHistory,\ committedIndex,\ tempVars,\ cepochSent,\ recoveryVars,\ p$

$LeaderSync2(i)\ \triangleq$
$\qquad \wedge\quad state[i] = ProspectiveLeader$
$\qquad \wedge\quad ackldRecv[i] \in Quorums$
$\qquad \wedge\quad commitIndex'\qquad = [commitIndex\qquad \text{EXCEPT } ![i] = Len(history[i])]$
$\qquad \wedge\quad committedIndex' = [committedIndex\ \text{EXCEPT } ![i] = Len(history[i])]$
$\qquad \wedge\quad state'\qquad\qquad = [state\qquad\qquad \text{EXCEPT } ![i]\quad = Leader]$
$\qquad \wedge\quad currentCounter' = [currentCounter\ \text{EXCEPT } ![i] = 0]$
$\qquad \wedge\quad sendCounter'\qquad = [sendCounter\qquad \text{EXCEPT } ![i] = 0]$

$\wedge \quad ackldRecv' \qquad = [ackldRecv \qquad \text{EXCEPT } ![i] = \{NullPoint\}]$
$\wedge \quad Broadcast(i, [mtype \quad \mapsto COMMITLD,$
$\qquad\qquad\qquad mepoch \mapsto currentEpoch[i],$
$\qquad\qquad\qquad mlength \mapsto Len(history[i])])$
$\wedge \quad \text{UNCHANGED } \langle currentEpoch, leaderEpoch, leaderOracle, history, cluster, cepochRecv,$
$\qquad\qquad\qquad ackeRecv, ackIndex, initialHistory, tempVars, cepochSent, recoveryVars, proposalM$

$FollowerSync2(i, j) \triangleq$
$\qquad \wedge state[i] = Follower$
$\qquad \wedge msgs[j][i] \neq \langle \rangle$
$\qquad \wedge msgs[j][i][1].mtype = COMMITLD$
$\qquad \wedge \text{LET } msg \triangleq msgs[j][i][1]$
$\qquad\qquad replyOk \triangleq \wedge currentEpoch[i] = msg.mepoch$
$\qquad\qquad\qquad\qquad \wedge leaderOracle[i] = j$
$\qquad\quad \text{IN} \quad \vee \quad$ new $COMMIT$-LD - commit all transactions in initial history
$\qquad\qquad\qquad\qquad$ Regradless of $Restart$, it must be true because one will receive $NEWLEADER$ before receiving $COMMIT$-L
$\qquad\qquad\qquad \wedge replyOk$
$\qquad\qquad\qquad \wedge \vee \wedge Len(history[i]) = msg.mlength$
$\qquad\qquad\qquad\qquad \wedge commitIndex' = [commitIndex \text{ EXCEPT } ![i] = Len(history[i])]$
$\qquad\qquad\qquad\qquad \wedge Discard(j, i)$
$\qquad\qquad\qquad\quad \vee \wedge Len(history[i]) \neq msg.mlength$
$\qquad\qquad\qquad\qquad \wedge Reply(i, j, [mtype \quad \mapsto CEPOCH,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad mepoch \mapsto currentEpoch[i]])$
$\qquad\qquad\qquad\qquad \wedge \text{UNCHANGED } commitIndex$
$\qquad\qquad \vee \quad > \;$: stale $COMMIT$-LD - discard
$\qquad\qquad\qquad < \;$: In our implementation, $' < '$ does not exist due to the guarantee of $Restart$
$\qquad\qquad\qquad \wedge \neg replyOk$
$\qquad\qquad\qquad \wedge Discard(j, i)$
$\qquad\qquad\qquad \wedge \text{UNCHANGED } commitIndex$
$\qquad \wedge \text{UNCHANGED } \langle state, currentEpoch, leaderEpoch, leaderOracle, history,$
$\qquad\qquad\qquad leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog \rangle$

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

12

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

In phase $f31$, follower accepts proposal and append it to history.
$FollowerBroadcast1(i, j) \triangleq$
        $\wedge\ state[i] = Follower$
        $\wedge\ msgs[j][i] \neq \langle\rangle$
        $\wedge\ msgs[j][i][1].mtype = PROPOSE$
        $\wedge\ \text{LET } msg \triangleq msgs[j][i][1]$
                $replyOk \triangleq \wedge\ currentEpoch[i] = msg.mepoch$
                              $\wedge\ leaderOracle[i] = j$
      $\text{IN} \quad \vee\ $ It should be that $\vee\ msg.mproposal.counter = 1$
                              $\vee\ msg.mrpoposal.counter = history[Len(history)].counter + 1$
             $\wedge\ replyOk$
             $\wedge\ history' = [history \text{ EXCEPT } ![i] = Append(history[i], msg.mproposal)]$
             $\wedge\ Reply(i, j, [mtype \quad \mapsto ACK,$
                            $mepoch \mapsto currentEpoch[i],$
                            $mindex \mapsto Len(history'[i])])$
          $\vee\ $ If happens, $\neq$ must be $>$, namely a stale leader sends it.
             $\wedge\ \neg replyOk$
             $\wedge\ Discard(j, i)$
             $\wedge\ \text{UNCHANGED } history$
        $\wedge\ \text{UNCHANGED } \langle state, currentEpoch, leaderEpoch, leaderOracle, commitIndex,$
                        $leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog\rangle$

In phase $l32$, leader receives ack from a quorum of followers to a certain proposal,
and commits the proposal.
$LeaderHandleACK(i, j) \triangleq$
        $\wedge\ state[i] = Leader$
        $\wedge\ msgs[j][i] \neq \langle\rangle$
        $\wedge\ msgs[j][i][1].mtype = ACK$
        $\wedge\ \text{LET } msg \triangleq msgs[j][i][1]$
      $\text{IN} \quad \vee\ $ It should be that $ackIndex[i][j] + 1 \triangleq msg.mindex$

13

$\qquad\qquad\qquad \wedge\ currentEpoch[i] = msg.mepoch$
$\qquad\qquad\qquad \wedge\ ackIndex' = [ackIndex\ \text{EXCEPT}\ ![i][j] = Maximum(\{ackIndex[i][j],\ msg.mindex\})]$
$\qquad\qquad \vee\ $ If happens, $\neq$ must be $>$, namely a stale follower sends it.
$\qquad\qquad\qquad \wedge\ currentEpoch[i] \neq msg.mepoch$
$\qquad\qquad\qquad \wedge\ \text{UNCHANGED}\ ackIndex$
$\qquad\quad \wedge\ Discard(j,\ i)$
$\qquad\quad \wedge\ \text{UNCHANGED}\ \langle serverVars,\ cluster,\ cepochRecv,\ ackeRecv,\ ackldRecv,\ currentCounter,$
$\qquad\qquad\qquad\qquad\qquad sendCounter,\ initialHistory,\ committedIndex,\ tempVars,\ cepochSent,\ recoveryVars,\ p$

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

$LeaderBroadcast2(i)\ \triangleq$
$\qquad\quad \wedge\ state[i] = Leader$
$\qquad\quad \wedge\ committedIndex[i] < commitIndex[i]$
$\qquad\quad \wedge\ \text{LET}\ newCommittedIndex\ \triangleq\ committedIndex[i] + 1$
$\qquad\qquad \text{IN}\quad \wedge\ Broadcast(i,\ [mtype\qquad \mapsto COMMIT,$
$\qquad\qquad\qquad\qquad\qquad\qquad\ mepoch\quad \mapsto currentEpoch[i],$
$\qquad\qquad\qquad\qquad\qquad\qquad\ mindex\quad \mapsto newCommittedIndex,$
$\qquad\qquad\qquad\qquad\qquad\qquad\ mcounter \mapsto history[i][newCommittedIndex].counter])$
$\qquad\qquad\qquad\ \wedge\ committedIndex' = [committedIndex\ \text{EXCEPT}\ ![i] = committedIndex[i] + 1]$
$\qquad\quad \wedge\ \text{UNCHANGED}\ \langle serverVars,\ cluster,\ cepochRecv,\ ackeRecv,\ ackldRecv,\ ackIndex,\ currentCounter,$
$\qquad\qquad\qquad\qquad\qquad sendCounter,\ initialHistory,\ tempVars,\ cepochSent,\ recoveryVars,\ proposalMsgsLog\rangle$

In phase $f32$, follower receives $COMMIT$ and commits transaction.
$FollowerBroadcast2(i,\ j)\ \triangleq$
$\qquad\quad \wedge\ state[i] = Follower$
$\qquad\quad \wedge\ msgs[j][i] \neq \langle\rangle$
$\qquad\quad \wedge\ msgs[j][i][1].mtype = COMMIT$
$\qquad\quad \wedge\ \text{LET}\ msg\ \triangleq\ msgs[j][i][1]$
$\qquad\qquad\qquad\ replyOk\ \triangleq\ \wedge\ currentEpoch[i] = msg.mepoch$
$\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ leaderOracle[i]\ = j$
$\qquad\qquad \text{IN}\quad \vee\ \wedge\ replyOk$
$\qquad\qquad\qquad\quad \wedge\ \text{LET}\ infoOk\ \triangleq\ \wedge\ Len(history[i]) \geq msg.mindex$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \wedge\ \vee\ \wedge\ msg.mindex > 0$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ history[i][msg.mindex].epoch = msg.mepoch$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ history[i][msg.mindex].counter = msg.mcounter$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \vee\ msg.mindex = 0$

14

IN $\lor$ new $COMMIT -$ commit transaction in history
  $\land infoOk$
  $\land commitIndex' = [commitIndex$ EXCEPT $![i] = Maximum(\{commitIndex[i], msg.$
  $\land Discard(j, i)$
  $\lor$ It may happen when the server is a new follower who joined in the cluster,
    and it misses the corresponding $PROPOSE$.
  $\land \neg infoOk$
  $\land Reply(i, j, [mtype \mapsto CEPOCH,$
    $mepoch \mapsto currentEpoch[i]])$
  $\land$ UNCHANGED $commitIndex$
$\lor$ stale $COMMIT -$ discard
  $\land \neg replyOk$
  $\land Discard(j, i)$
  $\land$ UNCHANGED $commitIndex$
$\land$ UNCHANGED $\langle state, currentEpoch, leaderEpoch, history, leaderOracle,$
  $leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog\rangle$

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 $l33$, 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 $\lor \land currentEpoch[i] \geq msg.mepoch$
      $\land Reply2(i, j, [mtype \mapsto NEWEPOCH,$
        $mepoch \mapsto currentEpoch[i]],$
        $[mtype \mapsto NEWLEADER,$
        $mepoch \mapsto currentEpoch[i],$
        $minitialHistory \mapsto history[i]])$
      $\land$ LET $m \triangleq [msource \mapsto i, mtype \mapsto NEWLEADER, mepoch \mapsto currentEpoch[i], mproposal$
        IN $proposalMsgsLog' =$ IF $m \in proposalMsgsLog$ THEN $proposalMsgsLog$
          ELSE $proposalMsgsLog \cup \{m\}$
    $\lor \land currentEpoch[i] < msg.mepoch$
      $\land$ UNCHANGED $\langle msgs, proposalMsgsLog\rangle$
  $\land$ UNCHANGED $\langle serverVars, leaderVars, tempVars, cepochSent, recoveryVars\rangle$

In phase $l34$, upon receiving ack from f of the $NEWLEADER$, it sends a commit message to f.
Leader l also makes $Q := Q \cup \{f\}$.
$LeaderHandleACKLDinPhase3(i, j) \triangleq$
  $\land state[i] = Leader$
  $\land msgs[j][i] \neq \langle\rangle$
  $\land msgs[j][i][1].mtype = ACKLD$

15

$\wedge$ LET $msg \;\triangleq\; msgs[j][i][1]$
$\qquad aimCommitIndex \;\triangleq\; Minimum(\{commitIndex[i],\ Len(msg.mhistory)\})$
$\qquad aimCommitCounter \;\triangleq\;$ IF $aimCommitIndex = 0$ THEN 0 ELSE $history[i][aimCommitIndex].co$
$\quad$ IN $\quad \vee \; \wedge currentEpoch[i] = msg.mepoch$
$\qquad\qquad \wedge ackIndex' = [ackIndex$ EXCEPT $![i][j] = Len(msg.mhistory)]$
$\qquad\qquad \wedge Reply(i,\ j,\ [mtype \quad \mapsto COMMIT,$
$\qquad\qquad\qquad\qquad\qquad\quad mepoch \quad \mapsto currentEpoch[i],$
$\qquad\qquad\qquad\qquad\qquad\quad mindex \quad \mapsto aimCommitIndex,$
$\qquad\qquad\qquad\qquad\qquad\quad mcounter \mapsto aimCommitCounter])$
$\qquad\quad \vee \; \wedge currentEpoch[i] \neq msg.mepoch$
$\qquad\qquad \wedge Discard(j,\ i)$
$\qquad\qquad \wedge$ UNCHANGED $ackIndex$
$\wedge cluster' = [cluster$ EXCEPT $![i] =$ IF $j \in cluster[i]$ THEN $cluster[i]$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ ELSE $cluster[i] \cup \{j\}]$
$\wedge$ UNCHANGED $\langle serverVars,\ cepochRecv,\ ackeRecv,\ ackldRecv,\ currentCounter,\ sendCounter,$
$\qquad\qquad\qquad initialHistory,\ committedIndex,\ tempVars,\ cepochSent,\ recoveryVars,\ proposalMsgsLo$

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 $mindex$,

$\quad$ which means there are some transactions before $mindex$ miss. Here we choose to send $CEPOCH$

$\quad$ again, to receive the newest history from leader.

$BecomeFollower(i) \;\triangleq$
$\qquad \wedge state[i] \neq Follower$
$\qquad \wedge \exists j \in Server \setminus \{i\} : \wedge msgs[j][i] \neq \langle \rangle$
$\qquad\qquad\qquad\qquad\qquad\qquad \wedge msgs[j][i][1].mtype \neq RECOVERYREQUEST$
$\qquad\qquad\qquad\qquad\qquad\qquad \wedge msgs[j][i][1].mtype \neq RECOVERYRESPONSE$
$\qquad\qquad\qquad\qquad\qquad\qquad \wedge$ LET $msg \;\triangleq\; msgs[j][i][1]$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad$ IN $\quad \wedge NullPoint \in cepochRecv[i]$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge Maximum(\{currentEpoch[i],\ leaderEpoch[i]\}) < msg.mepoch$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge \vee msg.mtype = NEWEPOCH$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee msg.mtype = NEWLEADER$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee msg.mtype = COMMITLD$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee msg.mtype = PROPOSE$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee msg.mtype = COMMIT$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge state' \qquad\quad = [state \qquad\quad$ EXCEPT $![i] \quad = Follower]$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge currentEpoch' = [currentEpoch$ EXCEPT $![i] = msg.mepoch]$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge Reply(i,\ j,\ [mtype \; \mapsto CEPOCH,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad mepoch \mapsto currentEpoch[i]])$

Here we should not use $Discard$.

$\qquad \wedge$ UNCHANGED $\langle leaderEpoch,\ history,\ commitIndex,\ leaderVars,\ tempVars,\ cepochSent,\ recoveryVars,$

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

Defines how the variables may transition.

$Next \triangleq$
$\quad\quad \vee \exists\, i \in Server,\, Q \in Quorums : InitialElection(i,\, Q)$
$\quad\quad \vee \exists\, i \in Server : \quad\quad Restart(i)$
$\quad\quad \vee \exists\, i \in Server : \quad\quad RecoveryAfterRestart(i)$
$\quad\quad \vee \exists\, i,\, j \in Server : \quad HandleRecoveryRequest(i,\, j)$
$\quad\quad \vee \exists\, i,\, j \in Server : \quad HandleRecoveryResponse(i,\, j)$
$\quad\quad \vee \exists\, i,\, j \in Server : \quad FindCluster(i)$
$\quad\quad \vee \exists\, i,\, j \in Server : \quad LeaderTimeout(i,\, j)$
$\quad\quad \vee \exists\, i \in Server : \quad\quad FollowerTimeout(i)$
$\quad\quad \vee \exists\, i \in Server : \quad\quad FollowerDiscovery1(i)$

$$\lor \exists\, i,\, j \in Server: \quad LeaderHandleCEPOCH(i,\, j)$$
$$\lor \exists\, i \in Server: \qquad LeaderDiscovery1(i)$$
$$\lor \exists\, i,\, j \in Server: \quad FollowerDiscovery2(i,\, j)$$
$$\lor \exists\, i,\, j \in Server: \quad LeaderHandleACKE(i,\, j)$$
$$\lor \exists\, i \in Server: \qquad LeaderDiscovery2Sync1(i)$$
$$\lor \exists\, i,\, j \in Server: \quad FollowerSync1(i,\, j)$$
$$\lor \exists\, i,\, j \in Server: \quad LeaderHandleACKLD(i,\, j)$$
$$\lor \exists\, i \in Server: \qquad LeaderSync2(i)$$
$$\lor \exists\, i,\, j \in Server: \quad FollowerSync2(i,\, j)$$
$$\lor \exists\, i \in Server,\, v \in Value: ClientRequest(i,\, v)$$
$$\lor \exists\, i \in Server: \qquad LeaderBroadcast1(i)$$
$$\lor \exists\, i,\, j \in Server: \quad FollowerBroadcast1(i,\, j)$$
$$\lor \exists\, i,\, j \in Server: \quad LeaderHandleACK(i,\, j)$$
$$\lor \exists\, i \in Server: \qquad LeaderAdvanceCommit(i)$$
$$\lor \exists\, i \in Server: \qquad LeaderBroadcast2(i)$$
$$\lor \exists\, i,\, j \in Server: \quad FollowerBroadcast2(i,\, j)$$
$$\lor \exists\, i,\, j \in Server: \quad LeaderHandleCEPOCHinPhase3(i,\, j)$$
$$\lor \exists\, i,\, j \in Server: \quad LeaderHandleACKLDinPhase3(i,\, j)$$
$$\lor \exists\, i \in Server: \qquad DiscardStaleMessage(i)$$
$$\lor \exists\, i \in Server: \qquad BecomeFollower(i)$$

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

Safety properties of *Zab* consensus algorithm

There is most one leader/prospective leader in a certain epoch.
$$Leadership \;\triangleq\; \forall\, i,\, j \in Server:$$
$$\land\; \lor\; state[i] = Leader$$
$$\qquad \lor\; \land\; state[i] = ProspectiveLeader$$
$$\qquad\qquad \land\; NullPoint \in ackeRecv[i] \quad \text{prospective leader determines its epoch after broadcasting } NEWLE$$
$$\land\; \lor\; state[j] = Leader$$
$$\qquad \lor\; \land\; state[j] = ProspectiveLeader$$
$$\qquad\qquad \land\; NullPoint \in ackeRecv[j]$$
$$\land\; currentEpoch[i] = currentEpoch[j]$$
$$\Rightarrow i = j$$

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

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

18

$PrefixConsistency$: The prefix that have been delivered in history in any process is the same.
$PrefixConsistency \triangleq \forall i, j \in Server :$
$\qquad\qquad\qquad\qquad$ LET $smaller \triangleq Minimum(\{commitIndex[i], commitIndex[j]\})$
$\qquad\qquad\qquad\qquad$ IN $\quad \lor smaller = 0$
$\qquad\qquad\qquad\qquad\qquad \lor \land smaller > 0$
$\qquad\qquad\qquad\qquad\qquad\qquad \land \forall index \in 1 .. smaller : equal(history[i][index], history[j][index])$

Integrity: If some follower delivers one transaction, then some primary has broadcast it.
$Integrity \triangleq \forall i \in Server :$
$\qquad\qquad\qquad state[i] = Follower \land commitIndex[i] > 0$
$\qquad\qquad\qquad \Rightarrow \forall index \in 1 .. commitIndex[i] : \exists msg \in proposalMsgsLog :$
$\qquad\qquad\qquad\qquad \lor \land msg.mtype = PROPOSE$
$\qquad\qquad\qquad\qquad\quad \land equal(msg.mproposal, history[i][index])$
$\qquad\qquad\qquad\qquad \lor \land msg.mtype = NEWLEADER$
$\qquad\qquad\qquad\qquad\quad \land \exists pindex \in 1 .. Len(msg.mproposals) : equal(msg.mproposals[pindex], history[i][ind$

Agreement: If some follower f delivers transaction a and some follower f' delivers transaction b,
$\qquad$ then f' delivers a or f delivers b.
$Agreement \triangleq \forall i, j \in Server :$
$\qquad\qquad\qquad \land state[i] = Follower \land commitIndex[i] > 0$
$\qquad\qquad\qquad \land state[j] = Follower \land commitIndex[j] > 0$
$\qquad\qquad\qquad \Rightarrow$
$\qquad\qquad\qquad \forall index1 \in 1 .. commitIndex[i], index2 \in 1 .. commitIndex[j] :$
$\qquad\qquad\qquad\qquad \lor \exists indexj \in 1 .. commitIndex[j] :$
$\qquad\qquad\qquad\qquad\quad equal(history[j][indexj], history[i][index1])$
$\qquad\qquad\qquad\qquad \lor \exists indexi \in 1 .. commitIndex[i] :$
$\qquad\qquad\qquad\qquad\quad equal(history[i][indexi], history[j][index2])$

Total order: If some follower delivers a before b, then any process that delivers b
$\qquad\qquad$ must also deliver a and deliver a before b.
$TotalOrder \triangleq \forall i, j \in Server : commitIndex[i] \geq 2 \land commitIndex[j] \geq 2$
$\qquad\qquad\qquad\qquad \Rightarrow \forall indexi1 \in 1 .. (commitIndex[i] - 1) : \forall indexi2 \in (indexi1 + 1) .. commitIndex[i] :$
$\qquad\qquad\qquad\qquad\quad$ LET $logOk \triangleq \exists index \in 1 .. commitIndex[j] : equal(history[i][indexi2], history[j][index]$
$\qquad\qquad\qquad\qquad\quad$ IN $\quad \lor \neg logOk$
$\qquad\qquad\qquad\qquad\qquad \lor \land logOk$
$\qquad\qquad\qquad\qquad\qquad\quad \land \exists indexj2 \in 1 .. commitIndex[j] :$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land equal(history[i][indexi2], history[j][indexj2])$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land \exists indexj1 \in 1 .. (indexj2 - 1) : equal(history[i][indexi1], histo$

Local primary order: If a primary broadcasts a before it broadcasts b, then a follower that
$\qquad\qquad$ delivers b must also deliver a before b.
$LocalPrimaryOrder \triangleq$ LET $mset(i, e) \triangleq \{msg \in proposalMsgsLog : \land msg.mtype = PROPOSE$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land msg.msource = i$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land msg.mepoch = e\}$
$\qquad\qquad\qquad\qquad\qquad mentries(i, e) \triangleq \{msg.mproposal : msg \in mset(i, e)\}$

19

IN $\forall i \in Server : \forall e \in 1 \mathinner{.\,.} currentEpoch[i] :$
     $\vee\ Cardinality(mentries(i,\, e)) < 2$
     $\vee\ \wedge\ Cardinality(mentries(i,\, e)) \geq 2$
       $\wedge\ \exists\, tsc1,\, tsc2 \in mentries(i,\, e) :$
       $\vee\ equal(tsc1,\, tsc2)$
       $\vee\ \wedge\ \neg equal(tsc1,\, tsc2)$
         $\wedge$ LET $tscPre\ \triangleq$ IF $precede(tsc1,\, tsc2)$ THEN $tsc1$ ELSE $tsc2$
                 $tscNext\ \triangleq$ IF $precede(tsc1,\, tsc2)$ THEN $tsc2$ ELSE $tsc1$
          IN $\forall j \in Server : \wedge commitIndex[j] \geq 2$
                        $\wedge\ \exists\, index \in 1 \mathinner{.\,.} commitIndex[j] : equal(history[j][ind$
        $\Rightarrow \exists\, index2 \in 1 \mathinner{.\,.} commitIndex[j] :$
           $\wedge\ equal(history[j][index2],\, tscNext)$
           $\wedge\ index2 > 1$
           $\wedge\ \exists\, index1 \in 1 \mathinner{.\,.} (index2 - 1) : equal(history[j][index1],\, tscPre$

Global primary order: A follower f delivers both a with epoch e and b with epoch e', and $e < e'$,
            then f must deliver a before b.
$GlobalPrimaryOrder\ \triangleq\ \forall i \in Server : commitIndex[i] \geq 2$
             $\Rightarrow \forall\, idx1,\, idx2 \in 1 \mathinner{.\,.} commitIndex[i] : \vee history[i][idx1].epoch \geq history[i][idx2].e$
                               $\vee\ \wedge history[i][idx1].epoch < history[i][idx2]$
                                  $\wedge idx1 < idx2$

Primary integrity: If primary p broadcasts a and some follower f delivers b such that b has epoch
            smaller than epoch of p, then p must deliver b before it broadcasts a.
$PrimaryIntegrity\ \triangleq\ \forall i,\, j \in Server : \wedge state[i] = Leader$
                         $\wedge state[j] = Follower \wedge commitIndex[j] \geq 1$
        $\Rightarrow \forall\, index \in 1 \mathinner{.\,.} commitIndex[j] : \vee history[j][index].epoch \geq currentEpoch[i]$
                                  $\vee\ \wedge history[j][index].epoch < currentEpoch[i]$
                                  $\wedge\ \exists\, idx \in 1 \mathinner{.\,.} commitIndex[i] : equal(history[i][i$

\ * Modification History
\ * Last modified *Wed* May 05 22:11:47 *CST* 2021 by Dell
\ * Created Sat *Dec* 05 13:32:08 *CST* 2020 by Dell