MODULE *ZabWithFLE*

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

Reference:
FLE: *FastLeaderElection.java*, *Vote.java*, *QuorumPeer.java* in https://*github.com*/*apache*/zookeeper.
ZAB: *QuorumPeer.java*, *Learner.java*, *Follower.java*, *LearnerHandler.java*, *Leader.java* in https://*github.com*/*apache*/zookeeper. https://*cwiki.apache.org*/confluence/display/*ZOOKEEPER*/*Zab*1.0.

EXTENDS *FastLeaderElection*

Defined in *FastLeaderElection.tla*:
* The set of server identifiers
CONSTANT *Server*

* *Server* states CONSTANTS *LOOKING*, *FOLLOWING*, *LEADING*

* Message types
CONSTANTS *NOTIFICATION*

* *Timeout* signal
CONSTANT *NONE*

The set of requests that can go into history
CONSTANT *Value*

Zab states
CONSTANTS *ELECTION*, *DISCOVERY*, *SYNCHRONIZATION*, *BROADCAST*

Message types
CONSTANTS *FOLLOWERINFO*, *LEADERINFO*, *ACKEPOCH*, *NEWLEADER*, *ACKLD*, *UPTODATE*, *PR*

Additional message types used for recovery in *synchronization*(*TRUNC*/*DIFF*/*SNAP*) are not needed since we abstract this part.(*see action RECOVERYSYNC*)

Defined in *FastLeaderElection.tla*: *Quorums*, *NullPoint*
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
$Minimum(S) \triangleq \text{IF } S = \{\} \text{ THEN } -1$
$\text{ELSE CHOOSE } n \in S : \forall\, m \in S : n \leq m$

$MAXEPOCH \triangleq 10$

Defined in *FastLeaderElection.tla*: *serverVars*: ⟨*state*, *currentEpoch*, *lastZxid*⟩, *electionVars*: ⟨*currentVote*, *logicalClock*, *receiveVotes*, *outOfElection*, *recvQueue*, *waitNotmsg*⟩, *leaderVars*: ⟨*leadingVoteSet*⟩, *electionMsgs*,
idTable

The current phase of $server$($ELECTION$, $DISCOVERY$, $SYNCHRONIZATION$, $BROADCAST$)
VARIABLE $zabState$

The epoch number of the last $NEWEPOCH$($LEADERINFO$) packet accepted
namely $f.p$ in paper, and $currentEpoch$ in $Zab.tla$.
VARIABLE $acceptedEpoch$

The history of servers as the sequence of transactions.
VARIABLE $history$

$commitIndex[i]$: The maximum index of transactions that have been saved in a quorum of servers
in the perspective of server i.(*increases monotonically before restarting*)
VARIABLE $commitIndex$

These transactions whose index \le $commitIndex[i]$ can be applied to state machine immediately.
So if we have a variable $applyIndex$, we can suppose that $applyIndex[i] = commitIndex[i]$
when verifying properties. But in phase $SYNC$, follower will apply all queued proposals to
state machine when receiving $NEWLEADER$. But follower only serves traffic after receiving
$UPTODATE$, so sequential consistency is not violated.

So when we verify properties, we still suppose $applyIndex[i] = commitIndex[i]$, because this is
an engineering detail.

$learners[i]$: The set of servers which leader i think are connected wich i.
VARIABLE $learners$

The messages representing 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 followers who has successfully sent $CEPOCH$($FOLLOWERINFO$) to leader.(*equals to connectingFollowers in code*)
VARIABLE $cepochRecv$

The set of followers who has successfully sent ACK-E to leader.(*equals to electingFollowers in code*)
VARIABLE $ackeRecv$

The set of followers who has successfully sent ACK-LD to leader in leader.(*equals to newLeaderProposal in code*)
VARIABLE $ackldRecv$

The set of servers which leader i broadcasts $PROPOSAL$ and $COMMIT$ to.(*equals to forwardingFollowers in code*)
VARIABLE $forwarding$

$ackIndex[i][j]$: The latest index that leader i has received from follower j via ACK.
VARIABLE $ackIndex$

$currentCounter[i]$: The count of transactions that clients request leader i.
VARIABLE $currentCounter$

$sendCounter[i]$: The count of transactions that leader i has broadcast in $PROPOSAL$.
VARIABLE $sendCounter$

$committedIndex[i]$: The maximum index of trasactions that leader i has broadcast in $COMMIT$.

VARIABLE *committedIndex*

committedCounter[*i*][*j*]: The latest counter of transaction that leader *i* has confirmed that follower *j* has committed.
VARIABLE *committedCounter*

initialHistory[*i*]: The initial history if leader *i* in epoch *acceptedEpoch*[*i*].
VARIABLE *initialHistory*

the maximum epoch in *CEPOCH* the prospective leader received from followers.
VARIABLE *tempMaxEpoch*

cepochSent[*i*] = TRUE means follower *i* has sent *CEPOCH*(*FOLLOWERINFO*) to leader.
VARIABLE *cepochSent*

leaderAddr[*i*]: The leader *id* of follower *i*. We use *leaderAddr* to express whether follower *i* has connected or lost connection.
VARIABLE *leaderAddr*

synced[*i*] = TRUE: follower *i* has completed sync with leader.
VARIABLE *synced*

The set of leaders in every epoch, only used in verifying properties.
VARIABLE *epochLeader*

The set of all broadcast messages, only used in verifying properties.
VARIABLE *proposalMsgsLog*

A variable used to check whether there are conditions contrary to the facts.
VARIABLE *inherentViolated*

$serverVarsZ \triangleq \langle state, currentEpoch, lastZxid, zabState, acceptedEpoch, history, commitIndex\rangle$ 7 varia

$electionVarsZ \triangleq electionVars$ 6 variables

$leaderVarsZ \triangleq \langle leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv, forwarding,$
$ackIndex, currentCounter, sendCounter, committedIndex, committedCounter\rangle$ 11 var

$tempVarsZ \triangleq \langle initialHistory, tempMaxEpoch\rangle$ 2 variables

$followerVarsZ \triangleq \langle cepochSent, leaderAddr, synced\rangle$ 3 variables

$verifyVarsZ \triangleq \langle proposalMsgsLog, epochLeader, inherentViolated\rangle$ 3 variables

$msgVarsZ \triangleq \langle msgs, electionMsgs\rangle$ 2 variables

$vars \triangleq \langle serverVarsZ, electionVarsZ, leaderVarsZ, tempVarsZ, followerVarsZ, verifyVarsZ, msgVarsZ, idTa$

Add a message to *msgs* – add a message *m* to *msgs*[*i*][*j*].
$Send(i, j, m) \triangleq msgs' = [msgs \text{ EXCEPT } ![i][j] = Append(msgs[i][j], m)]$

Remove a message from *msgs* – discard head of *msgs*[*i*][*j*].
$Discard(i, j) \triangleq msgs' = \text{IF } msgs[i][j] \neq \langle\rangle \text{ THEN } [msgs \text{ EXCEPT } ![i][j] = Tail(msgs[i][j])]$

ELSE $msgs$

Leader broadcasts a *message*(*PROPOSAL*/*COMMIT*) to all other servers in *forwardingFollowers*.

$$\begin{aligned}
Broadcast(i,\ m) \triangleq\ & msgs' = [msgs \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } \land v \in forwarding[i] \\
& \qquad \land v \neq i \\
& \qquad \land \lor \land m.mtype = PROPOSAL \\
& \qquad\quad \land ackIndex[i][v] < Len(initialHistor \\
& \qquad\ \lor \land m.mtype = COMMIT \\
& \qquad\quad \land committedCounter[i][v] < m.mzxi \\
& \qquad \text{THEN } Append(msgs[i][v],\ m) \\
& \qquad \text{ELSE } msgs[i][v]]]
\end{aligned}$$

$$\begin{aligned}
BroadcastLEADERINFO(i,\ m) \triangleq\ & msgs' = [msgs \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } \land v \in cepochRecv[i] \\
& \qquad \land v \in learners[i] \\
& \qquad \land v \neq i \text{ THEN } Append(ms \\
& \qquad\quad \text{ELSE } msgs[i][v]]]
\end{aligned}$$

$$\begin{aligned}
BroadcastUPTODATE(i,\ m) \triangleq\ & msgs' = [msgs \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } \land v \in ackldRecv[i] \\
& \qquad \land v \in learners[i] \\
& \qquad \land v \neq i \text{ THEN } Append(msgs \\
& \qquad\quad \text{ELSE } msgs[i][v]]]
\end{aligned}$$

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

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

shuffle the input buffer from server $j(i)$ in server $i(j)$.

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

$$PZxidEqual(p,\ z) \triangleq\ p.epoch = z[1] \land p.counter = z[2]$$

$$\begin{aligned}
TransactionEqual(t1,\ t2) \triangleq\ & \land t1.epoch\ \ = t2.epoch \\
& \land t1.counter = t2.counter
\end{aligned}$$

$$\begin{aligned}
TransactionPrecede(t1,\ t2) \triangleq\ & \lor t1.epoch < t2.epoch \\
& \lor \land t1.epoch\ \ = t2.epoch \\
& \quad \land t1.counter < t2.counter
\end{aligned}$$

Define initial values for all variables

$$\begin{aligned}
InitServerVarsZ \triangleq\ & \land InitServerVars \\
& \land zabState\ \ = [s\ \in Server \mapsto ELECTION] \\
& \land acceptedEpoch = [s\ \in Server \mapsto 0] \\
& \land history\ \ = [s \in Server \mapsto \langle\rangle] \\
& \land commitIndex\ = [s\ \in Server \mapsto 0]
\end{aligned}$$

$$InitLeaderVarsZ \triangleq\ \land InitLeaderVars$$

$$
\begin{aligned}
&\land\ learners &&= [s \in Server \mapsto \{\}]\\
&\land\ cepochRecv &&= [s \in Server \mapsto \{\}]\\
&\land\ ackeRecv &&= [s \in Server \mapsto \{\}]\\
&\land\ ackldRecv &&= [s \in Server \mapsto \{\}]\\
&\land\ ackIndex &&= [s \in Server \mapsto [v \in Server \mapsto 0]]\\
&\land\ currentCounter &&= [s \in Server \mapsto 0]\\
&\land\ sendCounter &&= [s \in Server \mapsto 0]\\
&\land\ committedIndex &&= [s \in Server \mapsto 0]\\
&\land\ committedCounter &&= [s \in Server \mapsto [v \in Server \mapsto 0]]\\
&\land\ forwarding &&= [s \in Server \mapsto \{\}]
\end{aligned}
$$

$$InitElectionVarsZ \triangleq InitElectionVars$$

$$
\begin{aligned}
InitTempVarsZ \triangleq\ &\land\ initialHistory &&= [s \in Server \mapsto \langle\rangle]\\
&\land\ tempMaxEpoch &&= [s \in Server \mapsto 0]
\end{aligned}
$$

$$
\begin{aligned}
InitFollowerVarsZ \triangleq\ &\land\ cepochSent &&= [s \in Server \mapsto \text{FALSE}]\\
&\land\ leaderAddr &&= [s \in Server \mapsto NullPoint]\\
&\land\ synced &&= [s \in Server \mapsto \text{FALSE}]
\end{aligned}
$$

$$
\begin{aligned}
InitVerifyVarsZ \triangleq\ &\land\ proposalMsgsLog &&= \{\}\\
&\land\ epochLeader &&= [i \in 1 \,..\, MAXEPOCH \mapsto \{\}]\\
&\land\ inherentViolated &&= \text{FALSE}
\end{aligned}
$$

$$
\begin{aligned}
InitMsgVarsZ \triangleq\ &\land\ msgs &&= [s \in Server \mapsto [v \in Server \mapsto \langle\rangle]]\\
&\land\ electionMsgs &&= [s \in Server \mapsto [v \in Server \mapsto \langle\rangle]]
\end{aligned}
$$

$$
\begin{aligned}
InitZ \triangleq\ &\land\ InitServerVarsZ\\
&\land\ InitLeaderVarsZ\\
&\land\ InitElectionVarsZ\\
&\land\ InitTempVarsZ\\
&\land\ InitFollowerVarsZ\\
&\land\ InitVerifyVarsZ\\
&\land\ InitMsgVarsZ\\
&\land\ idTable = InitializeIdTable(Server)
\end{aligned}
$$

$$
\begin{aligned}
&ZabTurnToLeading(i) \triangleq\\
&\quad\land\ zabState' &&= [zabState \ \text{EXCEPT}\ ![i] = DISCOVERY]\\
&\quad\land\ learners' &&= [learners \ \text{EXCEPT}\ ![i] = \{i\}]\\
&\quad\land\ cepochRecv' &&= [cepochRecv \ \text{EXCEPT}\ ![i] = \{i\}]\\
&\quad\land\ ackeRecv' &&= [ackeRecv \ \text{EXCEPT}\ ![i] = \{i\}]\\
&\quad\land\ ackldRecv' &&= [ackldRecv \ \text{EXCEPT}\ ![i] = \{i\}]\\
&\quad\land\ forwarding' &&= [forwarding \ \text{EXCEPT}\ ![i] = \{\}]\\
&\quad\land\ ackIndex' &&= [ackIndex \ \text{EXCEPT}\ ![i] = [v \in Server \mapsto \text{IF}\ v = i\ \text{THEN}\ Len(history[i])\\
&&&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE}\ 0]]\\
&\quad\land\ currentCounter' &&= [currentCounter \ \text{EXCEPT}\ ![i] = 0]
\end{aligned}
$$

$\land\ sendCounter' = [sendCounter \text{ EXCEPT } ![i] = 0]$
$\land\ commitIndex' = [commitIndex \text{ EXCEPT } ![i] = 0]$
$\land\ committedIndex' = [committedIndex \text{ EXCEPT } ![i] = 0]$
$\land\ committedCounter' = [committedCounter \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } v = i \text{ THEN } Len(history[$
$\text{ELSE } 0]]$
$\land\ initialHistory' = [initialHistory \text{ EXCEPT } ![i] = history[i]]$
$\land\ tempMaxEpoch' = [tempMaxEpoch \text{ EXCEPT } ![i] = acceptedEpoch[i]]$

$ZabTurnToFollowing(i) \triangleq$
$\land\ zabState' = [zabState \text{ EXCEPT } ![i] = DISCOVERY]$
$\land\ cepochSent' = [cepochSent \text{ EXCEPT } ![i] = \text{FALSE}]$
$\land\ synced' = [synced \text{ EXCEPT } ![i] = \text{FALSE}]$
$\land\ commitIndex' = [commitIndex \text{ EXCEPT } ![i] = 0]$

Fast Leader Election
$FLEReceiveNotmsg(i, j) \triangleq$
$\land\ ReceiveNotmsg(i, j)$
$\land$ UNCHANGED $\langle zabState, acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackl$
$sendCounter, committedIndex, committedCounter, tempVarsZ, followerVarsZ, verify$

$FLENotmsgTimeout(i) \triangleq$
$\land\ NotmsgTimeout(i)$
$\land$ UNCHANGED $\langle zabState, acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackl$
$sendCounter, committedIndex, committedCounter, tempVarsZ, followerVarsZ, verify$

$FLEHandleNotmsg(i) \triangleq$
$\land\ HandleNotmsg(i)$
$\land$ LET $newState \triangleq state'[i]$
IN
$\lor\ \land\ newState = LEADING$
$\land\ ZabTurnToLeading(i)$
$\land$ UNCHANGED $\langle cepochSent, synced\rangle$
$\lor\ \land\ newState = FOLLOWING$
$\land\ ZabTurnToFollowing(i)$
$\land$ UNCHANGED $\langle learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex, currentCounte$
$\lor\ \land\ newState = LOOKING$
$\land$ UNCHANGED $\langle zabState, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex, curr$
$committedIndex, committedCounter, tempVarsZ, cepochSent, synced\rangle$
$\land$ UNCHANGED $\langle acceptedEpoch, history, leaderAddr, verifyVarsZ, msgs\rangle$

On the premise that $ReceiveVotes.HasQuorums = \text{TRUE}$, corresponding to logic in line $1050 - 1055$ in $LFE.java$.
$FLEWaitNewNotmsg(i) \triangleq$
$\land\ WaitNewNotmsg(i)$
$\land$ UNCHANGED $\langle zabState, acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackl$
$sendCounter, committedIndex, committedCounter, tempVarsZ, followerVarsZ, verify$

On the premise that $ReceiveVotes.HasQuorums = \text{TRUE}$, corresponding to logic in line $1061 - 1066$ in $LFE.java$.

$FLEWaitNewNotmsgEnd(i) \triangleq$
$\quad \land\ WaitNewNotmsgEnd(i)$
$\quad \land\ \text{LET } newState \triangleq state'[i]$
$\quad\quad \text{IN}$
$\quad\quad \lor\ \land\ newState = LEADING$
$\quad\quad\quad \land\ ZabTurnToLeading(i)$
$\quad\quad\quad \land\ \text{UNCHANGED } \langle cepochSent, synced\rangle$
$\quad\quad \lor\ \land\ newState = FOLLOWING$
$\quad\quad\quad \land\ ZabTurnToFollowing(i)$
$\quad\quad\quad \land\ \text{UNCHANGED } \langle learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex, currentCounte$
$\quad\quad \lor\ \land\ newState = LOOKING$
$\quad\quad\quad \land\ PrintT($"New state is LOOKING in FLEWaitNewNotmsgEnd, which should not happen."$)$
$\quad\quad\quad \land\ \text{UNCHANGED } \langle zabState, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex, curr$
$\quad\quad\quad\quad committedIndex, committedCounter, tempVarsZ, cepochSent, synced\rangle$
$\quad \land\ \text{UNCHANGED } \langle acceptedEpoch, history, leaderAddr, verifyVarsZ, msgs\rangle$

A sub-action describing how a server transitions from $LEADING/FOLLOWING$ to $LOOKING$. Initially I call it 'ZabTimeoutZ', but it will be called not only when timeout, but also when finding a low epoch from leader.

$FollowerShutdown(i) \triangleq$
$\quad \land\ ZabTimeout(i)$
$\quad \land\ zabState' = [zabState \text{ EXCEPT } ![i] = ELECTION]$
$\quad \land\ leaderAddr' = [leaderAddr \text{ EXCEPT } ![i] = NullPoint]$

$LeaderShutdown(i) \triangleq$
$\quad \land\ ZabTimeout(i)$
$\quad \land\ zabState' = [zabState \text{ EXCEPT } ![i] = ELECTION]$
$\quad \land\ leaderAddr' = [s \in Server \mapsto \text{IF } s \in learners[i] \text{ THEN } NullPoint \text{ ELSE } leaderAddr[s]]$
$\quad \land\ learners' = [learners \text{ EXCEPT } ![i] = \{\}]$
$\quad \land\ forwarding' = [forwarding \text{ EXCEPT } ![i] = \{\}]$
$\quad \land\ msgs' = [s \in Server \mapsto [v \in Server \mapsto \text{IF } v \in learners[i] \lor s \in learners[i] \text{ THEN } \langle\rangle \text{ ELSE } msgs$

$FollowerTimout(i) \triangleq$
$\quad \land\ state[i] = FOLLOWING$
$\quad \land\ leaderAddr[i] = NullPoint$
$\quad \land\ FollowerShutdown(i)$
$\quad \land\ msgs' = [s \in Server \mapsto [v \in Server \mapsto \text{IF } v = i \text{ THEN } \langle\rangle \text{ ELSE } msgs[s][v]]]$
$\quad \land\ \text{UNCHANGED } \langle acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackldRecv, for$
$\quad\quad currentCounter, sendCounter, committedIndex, committedCounter, tempVarsZ, cepoc$

$LeaderTimeout(i) \triangleq$
$\quad \land\ state[i] = LEADING$
$\quad \land\ learners[i] \notin Quorums$
$\quad \land\ LeaderShutdown(i)$

$\land$ UNCHANGED $\langle acceptedEpoch, history, commitIndex, cepochRecv, ackeRecv, ackldRecv, ackIndex, cu$
$tempVarsZ, cepochSent, synced, verifyVarsZ\rangle$

Establish connection between leader i and follower j. It means i creates a *learnerHandler* for communicating with j, and j finds i's address.

$EstablishConnection(i, j) \triangleq$
$\land state[i] = LEADING \quad \land state[j] = FOLLOWING$
$\land j \notin learners[i] \quad \land leaderAddr[j] = NullPoint$
$\land currentVote[j].proposedLeader = i$
$\land learners' = [learners \text{ EXCEPT } ![i] = learners[i] \cup \{j\}]$ Leader: '$addLearnerHandler(peer)$'
$\land leaderAddr' = [leaderAddr \text{ EXCEPT } ![j] = i]$ Follower: '$connectToLeader(addr, hostname)$'
$\land$ UNCHANGED $\langle serverVarsZ, electionVarsZ, leadingVoteSet, cepochRecv, ackeRecv, ackldRecv, forwa$
$currentCounter, sendCounter, committedIndex, committedCounter, tempVarsZ, cepo$

The leader i finds timeout and *TCP* connection between i and j closes.

$Timeout(i, j) \triangleq$
$\land state[i] = LEADING \land state[j] = FOLLOWING$
$\land j \in learners[i] \quad \land leaderAddr[j] = i$
The action of leader i.(*corresponding to function'removeLearnerHandler(peer)'*.)
$\land learners' = [learners \text{ EXCEPT } ![i] = learners[i] \setminus \{j\}]$
$\land forwarding' = [forwarding \text{ EXCEPT } ![i] = \text{IF } j \in forwarding[i] \text{ THEN } forwarding[i] \setminus \{j\} \text{ ELSE } fo$
$\land cepochRecv' = [cepochRecv \text{ EXCEPT } ![i] = \text{IF } j \in cepochRecv[i] \text{ THEN } cepochRecv[i] \setminus \{j\} \text{ ELSE } ce$
$\land ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \text{IF } j \in ackeRecv[i] \text{ THEN } ackeRecv[i] \setminus \{j\} \text{ ELSE } ackeR$
$\land ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] = \text{IF } j \in ackldRecv[i] \text{ THEN } ackldRecv[i] \setminus \{j\} \text{ ELSE } ackl$
$\land ackIndex' = [ackIndex \text{ EXCEPT } ![i][j] = 0]$
$\land committedCounter' = [committedCounter \text{ EXCEPT } ![i][j] = 0]$
The action of follower j.
$\land FollowerShutdown(j)$
Clean input buffer.
$\land Clean(i, j)$
$\land$ UNCHANGED $\langle acceptedEpoch, history, commitIndex, currentCounter, sendCounter, committedInde$
$tempVarsZ, cepochSent, synced, verifyVarsZ\rangle$

In phase $f11$, follower sends $f.p$ to leader via $FOLLOWERINFO(CEPOCH)$.

$FollowerSendFOLLOWERINFO(i) \triangleq$
$\land state[i] = FOLLOWING$
$\land zabState[i] = DISCOVERY$
$\land leaderAddr[i] \neq NullPoint$
$\land \neg cepochSent[i]$
$\land Send(i, leaderAddr[i], [mtype \mapsto FOLLOWERINFO,$
$mepoch \mapsto acceptedEpoch[i]])$
$\land cepochSent' = [cepochSent \text{ EXCEPT } ![i] = \text{TRUE}]$
$\land$ UNCHANGED $\langle serverVarsZ, leaderVarsZ, electionVarsZ, tempVarsZ, leaderAddr, synced, verifyVars$

In phase $l11$, leader waits for receiving *FOLLOWERINFO* from a quorum, and then chooses a new epoch e' as its own epoch and broadcasts *LEADERINFO*.

$LeaderHandleFOLLOWERINFO(i, j) \triangleq$
$\wedge\ state[i] = LEADING$
$\wedge\ msgs[j][i] \neq \langle\rangle$
$\wedge\ msgs[j][i][1].mtype = FOLLOWERINFO$
$\wedge$ LET $msg \triangleq msgs[j][i][1]$
IN $\vee\ \wedge\ NullPoint \notin cepochRecv[i]$ 1. has not broadcast *LEADERINFO* – modify *tempMaxEpoch*
$\wedge$ LET $newEpoch \triangleq Maximum(\{tempMaxEpoch[i], msg.mepoch\})$
IN $tempMaxEpoch' = [tempMaxEpoch$ EXCEPT $![i] = newEpoch]$
$\wedge\ Discard(j, i)$
$\vee\ \wedge\ NullPoint \in cepochRecv[i]$ 2. has broadcast *LEADERINFO* – no need to handle the *msg*, just
$\wedge\ Reply(i, j, [mtype \mapsto LEADERINFO,$
$mepoch \mapsto acceptedEpoch[i]])$
$\wedge$ UNCHANGED $tempMaxEpoch$
$\wedge\ cepochRecv' = [cepochRecv$ EXCEPT $![i] =$ IF $j \in cepochRecv[i]$ THEN $cepochRecv[i]$
ELSE $cepochRecv[i] \cup \{j\}]$
$\wedge$ UNCHANGED $\langle serverVarsZ, followerVarsZ, electionVarsZ, initialHistory, leadingVoteSet, learners,$
$forwarding, ackIndex, currentCounter, sendCounter, committedIndex, committedCou$

$LeaderDiscovery1(i) \triangleq$
$\wedge\ state[i] = LEADING$
$\wedge\ zabState[i] = DISCOVERY$
$\wedge\ cepochRecv[i] \in Quorums$
$\wedge\ acceptedEpoch' = [acceptedEpoch$ EXCEPT $![i] = tempMaxEpoch[i] + 1]$
$\wedge\ cepochRecv' = [cepochRecv$ EXCEPT $![i] = cepochRecv[i] \cup \{NullPoint\}]$
$\wedge\ BroadcastLEADERINFO(i, [mtype \mapsto LEADERINFO,$
$mepoch \mapsto acceptedEpoch'[i]])$
$\wedge$ UNCHANGED $\langle state, currentEpoch, lastZxid, zabState, history, commitIndex, electionVarsZ, leading$
$forwarding, ackIndex, currentCounter, sendCounter, committedIndex, committedCou$
$tempVarsZ, followerVarsZ, verifyVarsZ, electionMsgs, idTable\rangle$

In phase $f12$, follower receives *NEWEPOCH*. If $e' > f.p$, then follower sends *ACK*-E back, and *ACK*-E contains $f.a$ and *lastZxid* to let leader judge whether it is the latest. After handling *NEWEPOCH*, follower's *zabState* turns to *SYNCHRONIZATION*.

$FollowerHandleLEADERINFO(i, j) \triangleq$
$\wedge\ state[i] = FOLLOWING$
$\wedge\ msgs[j][i] \neq \langle\rangle$
$\wedge\ msgs[j][i][1].mtype = LEADERINFO$
$\wedge$ LET $msg \triangleq msgs[j][i][1]$
$infoOk \triangleq j = leaderAddr[i]$
$epochOk \triangleq \wedge\ infoOk$
$\wedge\ msg.mepoch \geq acceptedEpoch[i]$
$correct \triangleq \wedge\ epochOk$
$\wedge\ zabState[i] = DISCOVERY$
IN $\wedge\ infoOk$

$\land \lor \land epochOk$ 1. Normal case
$\land \lor \land correct$
$\land acceptedEpoch' = [acceptedEpoch \text{ EXCEPT } ![i] = msg.mepoch]$
$\land Reply(i, j, [mtype \mapsto ACKEPOCH,$
$mepoch \mapsto msg.mepoch,$
$mlastEpoch \mapsto currentEpoch[i],$
$mlastZxid \mapsto lastZxid[i]])$
$\land cepochSent' = [cepochSent \text{ EXCEPT } ![i] = \text{TRUE}]$
$\land \text{UNCHANGED } inherentViolated$
$\lor \land \neg correct$
$\land PrintT(\text{“Exception: Condition correct is false in FollowerHandleLEADERINFO(”} \circ To$
$\land inherentViolated' = \text{TRUE}$
$\land Discard(j, i)$
$\land \text{UNCHANGED } \langle acceptedEpoch, cepochSent \rangle$
$\land zabState' = [zabState \text{ EXCEPT } ![i] = \text{IF } zabState[i] = DISCOVERY \text{ THEN } SYNCHRON$
$\text{ELSE } zabState[i]]$
$\land \text{UNCHANGED } \langle varsL, leaderAddr \rangle$
$\lor \land \neg epochOk$ 2. Abnormal case - go back to election
$\land FollowerShutdown(i)$
$\land Clean(i, j)$
$\land \text{UNCHANGED } \langle acceptedEpoch, cepochSent, inherentViolated \rangle$
$\land \text{UNCHANGED } \langle history, commitIndex, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackInd$
$committedCounter, tempVarsZ, synced, proposalMsgsLog, epochLeader \rangle$

Abstraction of actions making follower *synced* with leader before leader sending *NEWLEADER*.
$subRECOVERYSYNC(i, j) \triangleq$
$\text{LET } canSync \triangleq \land state[i] = LEADING \quad \land zabState[i] \neq DISCOVERY \quad \land j \in learners[i]$
$\land state[j] = FOLLOWING \land zabState[j] = SYNCHRONIZATION \land leaderAddr[j] =$
IN
$\lor \land canSync$
$\land history' = [history \text{ EXCEPT } ![j] = history[i]]$
$\land lastZxid' = [lastZxid \text{ EXCEPT } ![j] = lastZxid[i]]$
$\land UpdateProposal(j, leaderAddr[j], lastZxid'[j], currentEpoch[j])$
$\land commitIndex' = [commitIndex \text{ EXCEPT } ![j] = commitIndex[i]]$
$\land synced' = [synced \text{ EXCEPT } ![j] = \text{TRUE}]$
$\land forwarding' = [forwarding \text{ EXCEPT } ![i] = forwarding[i] \cup \{j\}]$ j will receive *PROPOSAL* and
$\land ackIndex' = [ackIndex \text{ EXCEPT } ![i][j] = Len(history[i])]$
$\land committedCounter' = [committedCounter \text{ EXCEPT } ![i][j] = Maximum(\{commitIndex[i] - Len(ini$
$\land \text{LET } ms \triangleq [msource \mapsto i, mtype \mapsto \text{“RECOVERYSYNC”}, mepoch \mapsto acceptedEpoch[i], mproposals$
$\text{IN } proposalMsgsLog' = \text{IF } ms \in proposalMsgsLog \text{ THEN } proposalMsgsLog$
$\text{ELSE } proposalMsgsLog \cup \{ms\}$
$\land Reply(i, j, [mtype \mapsto NEWLEADER,$
$mepoch \mapsto acceptedEpoch[i],$
$mlastZxid \mapsto lastZxid[i]])$
$\lor \land \neg canSync$

$\land$ $Discard(j,\ i)$
$\land$ UNCHANGED $\langle history,\ lastZxid,\ currentVote,\ commitIndex,\ synced,\ forwarding,\ ackIndex,\ commi$

In phase $l12$, leader waits for receiving $ACKEPOPCH$ from a quorum, and check whether it has the latest history and epoch from them. If so, leader's $zabState$ turns to $SYNCHRONIZATION$.

$LeaderHandleACKEPOCH(i,\ j) \triangleq$
$\land\ state[i] = LEADING$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = ACKEPOCH$
$\land$ LET $msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \land\ j \in learners[i]$
$\land\ acceptedEpoch[i] = msg.mepoch$
$logOk \triangleq \land\ infoOk$ $logOk$ = TRUE means leader is more up-to-date than follow
$\land\ \lor\ currentEpoch[i] > msg.mlastEpoch$
$\lor\ \land\ currentEpoch[i] = msg.mlastEpoch$
$\land\ \lor\ lastZxid[i][1] > msg.mlastZxid[1]$
$\lor\ \land\ lastZxid[i][1] = msg.mlastZxid[1]$
$\land\ lastZxid[i][2] \geq msg.mlastZxid[2]$
$replyOk \triangleq \land\ infoOk$
$\land\ NullPoint \in ackeRecv[i]$
IN $\land\ infoOk$
$\land\ \lor\ \land\ replyOk$
$\land\ subRECOVERYSYNC(i,\ j)$
$\land\ ackeRecv' = [ackeRecv$ EXCEPT $![i] =$ IF $j \notin ackeRecv[i]$ THEN $ackeRecv[i] \cup \{j\}$
ELSE $ackeRecv[i]]$
$\land$ UNCHANGED $\langle state,\ currentEpoch,\ logicalClock,\ receiveVotes,\ outOfElection,\ recvQueue$
$zabState,\ leaderAddr,\ learners\rangle$
$\lor\ \land\ \neg replyOk$
$\land\ \lor\ \land\ logOk$
$\land\ ackeRecv' = [ackeRecv$ EXCEPT $![i] =$ IF $j \notin ackeRecv[i]$ THEN $ackeRecv[i] \cup \{j\}$
ELSE $ackeRecv[i]]$
$\land\ Discard(j,\ i)$
$\land$ UNCHANGED $\langle varsL,\ zabState,\ leaderAddr,\ learners,\ forwarding\rangle$
$\lor\ \land\ \neg logOk$ go back to election
$\land\ LeaderShutdown(i)$
$\land$ UNCHANGED $ackeRecv$
$\land$ UNCHANGED $\langle history,\ commitIndex,\ synced,\ forwarding,\ ackIndex,\ committedCounter,$
$\land$ UNCHANGED $\langle acceptedEpoch,\ cepochRecv,\ ackldRecv,\ currentCounter,\ sendCounter,\ committedInde$

$LeaderDiscovery2(i) \triangleq$
$\land\ state[i] = LEADING$
$\land\ zabState[i] = DISCOVERY$
$\land\ ackeRecv[i] \in Quorums$
$\land\ zabState' = [zabState$ EXCEPT $![i] = SYNCHRONIZATION]$
$\land\ currentEpoch' = [currentEpoch$ EXCEPT $![i] = acceptedEpoch[i]]$

$\land$ $initialHistory' = [initialHistory$ EXCEPT $![i] = history[i]]$
$\land$ $ackeRecv' = [ackeRecv$ EXCEPT $![i] = ackeRecv[i] \cup \{NullPoint\}]$
$\land$ $ackIndex' = [ackIndex$ EXCEPT $![i][i] = Len(history[i]))]$
$\land$ $UpdateProposal(i, i, lastZxid[i], currentEpoch'[i])$
$\land$ LET $epoch \triangleq acceptedEpoch[i]$
IN $epochLeader' = [epochLeader$ EXCEPT $![epoch] = epochLeader[epoch] \cup \{i\}]$
$\land$ UNCHANGED $\langle state, lastZxid, acceptedEpoch, history, commitIndex, logicalClock, receiveVotes, outC$
$leadingVoteSet, learners, cepochRecv, ackldRecv, forwarding, currentCounter, sendCo$
$tempMaxEpoch, followerVarsZ, proposalMsgsLog, inherentViolated, msgVarsZ, idTab$

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

$RECOVERYSYNC(i, j) \triangleq$
$\land$ $state[i] = LEADING \land zabState[i] \neq DISCOVERY \land j \in learners[i] \land j \in ackeRecv$
$\land$ $state[j] = FOLLOWING \land zabState[j] = SYNCHRONIZATION \land leaderAddr[j] = i \land synced[j] =$ F
$\land$ $acceptedEpoch[i] = acceptedEpoch[j] \setminus *$ This condition is unnecessary.
$\land$ $history' = [history$ EXCEPT $![j] = history[i]]$
$\land$ $lastZxid' = [lastZxid$ EXCEPT $![j] = lastZxid[i]]$
$\land$ $UpdateProposal(j, leaderAddr[j], lastZxid'[j], currentEpoch[j])$
$\land$ $commitIndex' = [commitIndex$ EXCEPT $![j] = commitIndex[i]]$
$\land$ $synced' = [synced$ EXCEPT $![j] =$ TRUE$]$
$\land$ $forwarding' = [forwarding$ EXCEPT $![i] = forwarding[i] \cup \{j\}]$
$\land$ $ackIndex' = [ackIndex$ EXCEPT $![i][j] = Len(history[i]))]$
$\land$ $committedCounter' = [committedCounter$ EXCEPT $![i][j] = Maximum(\{commitIndex[i] - Len(initia$
$\land$ LET $ms \triangleq [msource \mapsto i, mtype \mapsto$ "RECOVERYSYNC", $mepoch \mapsto acceptedEpoch[i], mproposals \mapsto$
IN $proposalMsgsLog' =$ IF $ms \in proposalMsgsLog$ THEN $proposalMsgsLog$
ELSE $proposalMsgsLog \cup \{ms\}$
$\land$ $Send(i, j, [mtype \mapsto NEWLEADER,$
$mepoch \mapsto acceptedEpoch[i],$
$mlastZxid \mapsto lastZxid[i]])$
$\land$ UNCHANGED $\langle state, zabState, acceptedEpoch, currentEpoch, logicalClock, receiveVotes, outOfElectio$
$leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv, currentCounter, sendCou$
$tempVarsZ, cepochSent, leaderAddr, epochLeader, inherentViolated, electionMsgs, id'$

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

$FollowerHandleNEWLEADER(i, j) \triangleq$
$\land$ $state[i] = FOLLOWING$
$\land$ $msgs[j][i] \neq \langle\rangle$
$\land$ $msgs[j][i][1].mtype = NEWLEADER$
$\land$ LET $msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \land leaderAddr[i] = j$
$\land acceptedEpoch[i] = msg.mepoch$
$correct \triangleq \land infoOk$

$\wedge\ zabState[i] = SYNCHRONIZATION$
$\wedge\ synced[i]$
$\wedge\ ZxidEqual(lastZxid[i],\ msg.mlastZxid)$

IN $\wedge\ infoOk$
$\wedge\ currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch]$
$\wedge\ UpdateProposal(i,\ j,\ lastZxid[i],\ currentEpoch'[i])$
$\wedge\ \vee\ \wedge\ correct$
$\wedge\ Reply(i,\ j,\ [mtype \mapsto ACKLD,$
$mepoch \mapsto msg.mepoch])$
$\wedge$ UNCHANGED $inherentViolated$
$\vee\ \wedge\ \neg correct$
$\wedge\ PrintT($"Exception: Condition correct is false in FollowerHandleNEWLEADER(" $\circ\ ToString$
$\wedge\ inherentViolated' =$ TRUE
$\wedge\ Discard(j,\ i)$
$\wedge$ UNCHANGED $\langle state,\ lastZxid,\ zabState,\ acceptedEpoch,\ history,\ commitIndex,\ logicalClock,\ receiveV$
$leaderVarsZ,\ tempVarsZ,\ followerVarsZ,\ proposalMsgsLog,\ epochLeader,\ electionMsg$

In phase $l22$, leader receives ACK-LD from a quorum of followers, and sends $COMMIT\text{-}LD(UPTODATE)$ to followers.

$LeaderHandleACKLD(i,\ j) \triangleq$
$\wedge\ state[i] = LEADING$
$\wedge\ msgs[j][i] \neq \langle\rangle$
$\wedge\ msgs[j][i][1].mtype = ACKLD$
$\wedge$ LET $msg \triangleq msgs[j][i][1]$
$infoOk \triangleq\ \wedge\ acceptedEpoch[i] = msg.mepoch$
$\wedge\ j \in learners[i]$
$replyOk \triangleq\ \wedge\ infoOk$
$\wedge\ NullPoint \in ackldRecv[i]$
IN $\wedge\ infoOk$
$\wedge\ \vee\ \wedge\ replyOk$
$\wedge\ Reply(i,\ j,\ [mtype \mapsto UPTODATE,$
$mepoch \mapsto acceptedEpoch[i],$
$mcommit \mapsto commitIndex[i]])$
$\wedge\ committedCounter' = [committedCounter \text{ EXCEPT } ![i][j] = Maximum(\{commitIndex[i]$
$\vee\ \wedge\ \neg replyOk$
$\wedge\ Discard(j,\ i)$
$\wedge$ UNCHANGED $committedCounter$
$\wedge\ ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] =$ IF $j \notin ackldRecv[i]$ THEN $ackldRecv[i] \cup \{j\}$
ELSE $ackldRecv[i]]$
$\wedge$ UNCHANGED $\langle serverVarsZ,\ electionVarsZ,\ leadingVoteSet,\ learners,\ cepochRecv,\ ackeRecv,\ forward$
$ackIndex,\ currentCounter,\ sendCounter,\ committedIndex,\ tempVarsZ,\ followerVarsZ$

$LeaderSync2(i) \triangleq$
$\wedge\quad state[i] = LEADING$
$\wedge\quad zabState[i] = SYNCHRONIZATION$
$\wedge\quad ackldRecv[i] \in Quorums$

$\land$ $commitIndex' = [commitIndex \text{ EXCEPT } ![i] = Len(history[i])]$
$\land$ $committedIndex' = [committedIndex \text{ EXCEPT } ![i] = Len(history[i])]$
$\land$ $zabState' = [zabState \text{ EXCEPT } ![i] = BROADCAST]$
$\land$ $currentCounter' = [currentCounter \text{ EXCEPT } ![i] = 0]$
$\land$ $sendCounter' = [sendCounter \text{ EXCEPT } ![i] = 0]$
$\land$ $ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] = ackldRecv[i] \cup \{NullPoint\}]$
$\land$ $BroadcastUPTODATE(i, [mtype \mapsto UPTODATE, mepoch \mapsto acceptedEpoch[i], mcommit \mapsto Len(history[i])])$ In actual *UPTODATE* doesn't carry this info
$\land$ UNCHANGED $\langle state, currentEpoch, lastZxid, acceptedEpoch, history, electionVarsZ, leadingVoteSe$
$committedCounter, tempVarsZ, followerVarsZ, verifyVarsZ, electionMsgs, idTable$

$FollowerHandleUPTODATE(i, j) \triangleq$
$\land\ state[i] = FOLLOWING$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = UPTODATE$
$\land$ LET $msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \land\ leaderAddr[i] = j$
$\land\ acceptedEpoch[i] = msg.mepoch$
$correct \triangleq \land\ infoOk$
$\land\ zabState[i] = SYNCHRONIZATION$
$\land\ currentEpoch[i] = msg.mepoch$
IN $\land\ infoOk$
$\land\ \lor\ \land\ correct$
$\land\ commitIndex' = [commitIndex \text{ EXCEPT } ![i] = Maximum(\{commitIndex[i], msg.mcomm$
$\land\ zabState' = [zabState \text{ EXCEPT } ![i] = BROADCAST]$
$\land$ UNCHANGED $inherentViolated$
$\lor\ \land\ \neg correct$
$\land\ PrintT(\text{“Exception: Condition correct is false in FollowerHandleUPTODATE(”} \circ ToString($
$\land\ inherentViolated' = \text{TRUE}$
$\land$ UNCHANGED $\langle commitIndex, zabState\rangle$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $\langle state, currentEpoch, lastZxid, acceptedEpoch, history, electionVarsZ, leaderVarsZ, te$
$proposalMsgsLog, epochLeader, electionMsgs, idTable\rangle$

In phase $l31$, leader receives client request and broadcasts *PROPOSAL*. Note: In production, any server in traffic can receive requests and forward it to leader if necessary. We choose to let leader be the sole one who can receive requests, to simplify spec and keep correctness at the same time.

$ClientRequest(i, v) \triangleq$
$\land\ state[i] = LEADING$
$\land\ zabState[i] = BROADCAST$
$\land\ currentCounter' = [currentCounter \text{ EXCEPT } ![i] = currentCounter[i] + 1]$
$\land$ LET $newTransaction \triangleq [epoch \mapsto acceptedEpoch[i], counter \mapsto currentCounter'[i],$

```
                                    value    ↦ v]
    IN    ∧ history'   = [history   EXCEPT ![i] = Append(history[i], newTransaction)]
          ∧ lastZxid'  = [lastZxid  EXCEPT ![i] = ⟨acceptedEpoch[i], currentCounter'[i]⟩]
          ∧ ackIndex'  = [ackIndex  EXCEPT ![i][i] = Len(history'[i])]
          ∧ UpdateProposal(i, i, lastZxid'[i], currentEpoch[i])
  ∧ UNCHANGED ⟨state, currentEpoch, zabState, acceptedEpoch, commitIndex, logicalClock, receiveVotes
               leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, sendCounter,
               tempVarsZ, followerVarsZ, verifyVarsZ, msgVarsZ, idTable⟩

LeaderBroadcast1(i) ≜
  ∧ state[i] = LEADING
  ∧ zabState[i] = BROADCAST
  ∧ sendCounter[i] < currentCounter[i]
  ∧ LET toBeSentCounter ≜ sendCounter[i] + 1
        toBeSentIndex   ≜ Len(initialHistory[i]) + toBeSentCounter
        toBeSentEntry   ≜ history[i][toBeSentIndex]
    IN  ∧ Broadcast(i, [mtype     ↦ PROPOSAL,
                        mepoch    ↦ acceptedEpoch[i],
                        mproposal ↦ toBeSentEntry])
        ∧ sendCounter' = [sendCounter EXCEPT ![i] = toBeSentCounter]
        ∧ LET m ≜ [msource ↦ i, mepoch ↦ acceptedEpoch[i], mtype ↦ PROPOSAL, mproposal ↦ t
          IN  proposalMsgsLog' = proposalMsgsLog ∪ {m}
  ∧ UNCHANGED ⟨serverVarsZ, electionVarsZ, leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRe
               committedIndex, committedCounter, tempVarsZ, followerVarsZ, epochLeader, inhere

In phase f31, follower accepts proposal and append it to history.
FollowerHandlePROPOSAL(i, j) ≜
  ∧ state[i] = FOLLOWING
  ∧ msgs[j][i] ≠ ⟨⟩
  ∧ msgs[j][i][1].mtype = PROPOSAL
  ∧ LET msg     ≜ msgs[j][i][1]
        infoOk  ≜ ∧ leaderAddr[i] = j
                  ∧ acceptedEpoch[i] = msg.mepoch
        correct ≜ ∧ infoOk
                  ∧ zabState[i] ≠ DISCOVERY
                  ∧ synced[i]
        logOk   ≜ ∨ ∧ msg.mproposal.counter = 1        the first PROPOSAL in this epoch
                    ∧ ∨ Len(history[i]) = 0
                      ∨ ∧ Len(history[i]) > 0
                        ∧ history[i][Len(history[i])].epoch < msg.mepoch
                  ∨ ∧ msg.mproposal.counter > 1        not the first PROPOSAL in this epoch
                    ∧ Len(history[i]) > 0
                    ∧ history[i][Len(history[i])].epoch = msg.mepoch
                    ∧ history[i][Len(history[i])].counter = msg.mproposal.counter − 1
    IN  ∧ infoOk
```

$\land \lor \land correct$
$\land \lor \land logOk$
$\land history' = [history \text{ EXCEPT } ![i] = Append(history[i], msg.mproposal)]$
$\land lastZxid' = [lastZxid \text{ EXCEPT } ![i] = \langle msg.mepoch, msg.mproposal.counter\rangle]$
$\land UpdateProposal(i, j, lastZxid'[i], currentEpoch[i])$
$\land Reply(i, j, [mtype \mapsto ACK,$
$mepoch \mapsto acceptedEpoch[i],$
$mzxid \mapsto \langle msg.mepoch, msg.mproposal.counter\rangle])$
$\land \text{UNCHANGED } inherentViolated$
$\lor \land \neg logOk$
$\land PrintT(\text{“Exception: Condition logOk is false in FollowerHandlePROPOSAL(”} \circ ToStr$
$\land inherentViolated' = \text{TRUE}$
$\land Discard(j, i)$
$\land \text{UNCHANGED } \langle history, lastZxid, currentVote\rangle$
$\lor \land \neg correct$
$\land PrintT(\text{“Exception: Condition correct is false in FollowerHandlePROPOSAL(”} \circ ToString($
$\land inherentViolated' = \text{TRUE}$
$\land Discard(j, i)$
$\land \text{UNCHANGED } \langle history, lastZxid, currentVote\rangle$
$\land \text{UNCHANGED } \langle state, currentEpoch, zabState, acceptedEpoch, commitIndex, logicalClock, receiveVotes$
$leaderVarsZ, tempVarsZ, followerVarsZ, proposalMsgsLog, epochLeader, electionMsg$

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

$LeaderHandleACK(i, j) \triangleq$
$\land state[i] = LEADING$
$\land msgs[j][i] \neq \langle\rangle$
$\land msgs[j][i][1].mtype = ACK$
$\land \text{LET } msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \land j \in forwarding[i]$
$\land acceptedEpoch[i] = msg.mepoch$
$correct \triangleq \land infoOk$
$\land zabState[i] = BROADCAST$
$\land sendCounter[i] \geq msg.mzxid[2]$
$logOk \triangleq \land infoOk$
$\land ackIndex[i][j] + 1 = Len(initialHistory[i]) + msg.mzxid[2]$
IN $\land infoOk$
$\land \lor \land correct$
$\land \lor \land logOk$
$\land ackIndex' = [ackIndex \text{ EXCEPT } ![i][j] = ackIndex[i][j] + 1]$
$\lor \land \neg logOk$
$\land PrintT(\text{“Note: redundant ACK.”})$
$\land \text{UNCHANGED } ackIndex$
$\land \text{UNCHANGED } inherentViolated$
$\lor \land \neg correct$
$\land PrintT(\text{“Exception: Condition correct is false in FollowerHandleACK(”} \circ ToString(i) \circ \text{“, ”}$

$\wedge\ inherentViolated' = \text{TRUE}$
$\wedge\ \text{UNCHANGED}\ ackIndex$
$\wedge\ Discard(j,\ i)$
$\wedge\ \text{UNCHANGED}\ \langle serverVarsZ,\ electionVarsZ,\ leadingVoteSet,\ learners,\ cepochRecv,\ ackeRecv,\ ackldRe$
$committedIndex,\ committedCounter,\ tempVarsZ,\ followerVarsZ,\ proposalMsgsLog,\ ep$

$LeaderAdvanceCommit(i)\ \triangleq$
$\wedge\ state[i] = LEADING$
$\wedge\ zabState[i] = BROADCAST$
$\wedge\ commitIndex[i] < Len(history[i])$
$\wedge\ \text{LET}\ Agree(index) \triangleq \{i\} \cup \{k \in (Server \setminus \{i\}) : ackIndex[i][k] \geq index\}$
$agreeIndexes \triangleq \{index \in (commitIndex[i]+1)\ ..\ Len(history[i]) : Agree(index) \in Quoru$
$newCommitIndex \triangleq \text{IF}\ agreeIndexes \neq \{\}\ \text{THEN}\ Maximum(agreeIndexes)$
$\text{ELSE}\ commitIndex[i]$
$\text{IN}\ \ commitIndex' = [commitIndex\ \text{EXCEPT}\ ![i] = newCommitIndex]$
$\wedge\ \text{UNCHANGED}\ \langle state,\ currentEpoch,\ lastZxid,\ zabState,\ acceptedEpoch,\ history,\ electionVarsZ,\ leader$
$verifyVarsZ,\ msgVarsZ,\ idTable\rangle$

$LeaderBroadcast2(i)\ \triangleq$
$\wedge\ state[i] = LEADING$
$\wedge\ zabState[i] = BROADCAST$
$\wedge\ committedIndex[i] < commitIndex[i]$
$\wedge\ Len(initialHistory[i]) + sendCounter[i] > committedIndex[i]$
$\wedge\ \text{LET}\ newCommittedIndex \triangleq committedIndex[i] + 1$
$\text{IN}\ \ \wedge\ Broadcast(i, [mtype \mapsto COMMIT,$
$mepoch \mapsto acceptedEpoch[i],$
$mzxid \mapsto \langle history[i][newCommittedIndex].epoch,\ history[i][newCommittedInd$
$\wedge\ committedIndex' = [committedIndex\ \text{EXCEPT}\ ![i] = committedIndex[i] + 1]$
$\wedge\ committedCounter' = [committedCounter\ \text{EXCEPT}\ ![i] = [v \in Server \mapsto \text{IF}\ \wedge v \in forwarding$
$\wedge committedCou$
$\text{THEN}\ history[i][ne$
$\text{ELSE}\ committedCo$
$\wedge\ \text{UNCHANGED}\ \langle serverVarsZ,\ electionVarsZ,\ leadingVoteSet,\ learners,\ cepochRecv,\ ackeRecv,\ ackldRe$
$sendCounter,\ tempVarsZ,\ followerVarsZ,\ verifyVarsZ,\ electionMsgs,\ idTable\rangle$

In phase $f32$, follower receives $COMMIT$ and commits transaction.

$FollowerHandleCOMMIT(i,\ j)\ \triangleq$
$\wedge\ state[i] = FOLLOWING$
$\wedge\ msgs[j][i] \neq \langle\rangle$
$\wedge\ msgs[j][i][1].mtype = COMMIT$
$\wedge\ \text{LET}\ msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \wedge\ leaderAddr[i] = j$
$\wedge\ acceptedEpoch[i] = msg.mepoch$
$correct \triangleq \wedge\ infoOk$
$\wedge\ zabState[i] \neq DISCOVERY$
$\wedge\ synced[i]$

```
        mindex ≜ IF Len(history[i]) = 0 THEN −1
                 ELSE IF ∃ idx ∈ 1 .. Len(history[i]) : PZxidEqual(history[i][idx], msg.mzxid)
                      THEN CHOOSE idx ∈ 1 .. Len(history[i]) : PZxidEqual(history[i][idx], msg.m
                      ELSE −1
        logOk ≜ mindex > 0
        latest ≜ commitIndex[i] + 1 = mindex
    IN  ∧ infoOk
        ∧ ∨ ∧ correct
            ∧ ∨ ∧ logOk
                ∧ ∨ ∧ latest
                    ∧ commitIndex' = [commitIndex EXCEPT ![i] = commitIndex[i] + 1]
                    ∧ UNCHANGED inherentViolated
                  ∨ ∧ ¬latest
                    ∧ PrintT("Note: Condition latest is false in FollowerHandleCOMMIT(" ∘ ToString
                    ∧ inherentViolated' = TRUE
                    ∧ UNCHANGED commitIndex
              ∨ ∧ ¬logOk
                ∧ PrintT("Exception: Condition logOk is false in FollowerHandleCOMMIT(" ∘ ToString
                ∧ inherentViolated' = TRUE
                ∧ UNCHANGED commitIndex
          ∨ ∧ ¬correct
            ∧ PrintT("Exception: Condition correct is false in FollowerHandleCOMMIT(" ∘ ToString(i) ∘
            ∧ inherentViolated' = TRUE
            ∧ UNCHANGED commitIndex
    ∧ Discard(j, i)
    ∧ UNCHANGED ⟨state, currentEpoch, lastZxid, zabState, acceptedEpoch, history, electionVarsZ, leader
                 proposalMsgsLog, epochLeader, electionMsgs, idTable⟩
```

Used to discard some messages which should not exist in actual. This action should not be triggered.

```
FilterNonexistentMessage(i) ≜
    ∧ ∃ j ∈ Server \ {i} : ∧ msgs[j][i] ≠ ⟨⟩
                           ∧ LET msg ≜ msgs[j][i][1]
                             IN
                                 ∨ ∧ state[i] = LEADING
                                   ∧ LET infoOk ≜ ∧ j ∈ learners[i]
                                                 ∧ acceptedEpoch[i] = msg.mepoch
                                     IN
                                      ∨ msg.mtype = LEADERINFO
                                      ∨ msg.mtype = NEWLEADER
                                      ∨ msg.mtype = UPTODATE
                                      ∨ msg.mtype = PROPOSAL
                                      ∨ msg.mtype = COMMIT
                                      ∨ ∧ j ∉ learners[i]
```

$$
\begin{array}{l}
\quad \land\ msg.mtype = FOLLOWERINFO \\
\quad \lor\ \land\ \neg infoOk \\
\quad\quad \land\ \lor\ msg.mtype = ACKEPOCH \\
\quad\quad\quad \lor\ msg.mtype = ACKLD \\
\quad\quad\quad \lor\ msg.mtype = ACK \\
\lor\ \land\ state[i] = FOLLOWING \\
\quad \land \text{LET } infoOk \triangleq\ \land\ j = leaderAddr[i] \\
\quad\quad\quad\quad\quad\quad\ \land\ acceptedEpoch[i] = msg.mepoch \\
\quad \text{IN} \\
\quad\quad \lor\ msg.mtype = FOLLOWERINFO \\
\quad\quad \lor\ msg.mtype = ACKEPOCH \\
\quad\quad \lor\ msg.mtype = ACKLD \\
\quad\quad \lor\ msg.mtype = ACK \\
\quad\quad \lor\ \land\ j \neq leaderAddr[i] \\
\quad\quad\quad \land\ msg.mtype = LEADERINFO \\
\quad\quad \lor\ \land\ \neg infoOk \\
\quad\quad\quad \land\ \lor\ msg.mtype = NEWLEADER \\
\quad\quad\quad\quad \lor\ msg.mtype = UPTODATE \\
\quad\quad\quad\quad \lor\ msg.mtype = PROPOSAL \\
\quad\quad\quad\quad \lor\ msg.mtype = COMMIT \\
\quad\quad \lor\ state[i] = LOOKING \\
\land\ Discard(j,\ i) \\
\land\ inherentViolated' = \text{TRUE} \\
\land \text{UNCHANGED } \langle serverVarsZ,\ electionVarsZ,\ leaderVarsZ,\ tempVarsZ,\ followerVarsZ,\ proposalMsgsL
\end{array}
$$

Defines how the variables may transition.

$$
\begin{array}{l}
NextZ \triangleq \\
\quad \textit{FLE} \text{ modlue} \\
\quad \lor \exists\, i,\, j \in Server : FLEReceiveNotmsg(i,\, j) \\
\quad \lor \exists\, i \in Server : \quad FLENotmsgTimeout(i) \\
\quad \lor \exists\, i \in Server : \quad FLEHandleNotmsg(i) \\
\quad \lor \exists\, i \in Server : \quad FLEWaitNewNotmsg(i) \\
\quad \lor \exists\, i \in Server : \quad FLEWaitNewNotmsgEnd(i) \\
\quad \text{Some conditions like failure, network delay} \\
\quad \lor \exists\, i \in Server : \quad FollowerTimout(i) \\
\quad \lor \exists\, i \in Server : \quad LeaderTimeout(i) \\
\quad \lor \exists\, i,\, j \in Server : Timeout(i,\, j) \\
\quad \text{Zab module - Discovery and Synchronization part} \\
\quad \lor \exists\, i,\, j \in Server : EstablishConnection(i,\, j) \\
\quad \lor \exists\, i \in Server : \quad FollowerSendFOLLOWERINFO(i) \\
\quad \lor \exists\, i,\, j \in Server : LeaderHandleFOLLOWERINFO(i,\, j) \\
\quad \lor \exists\, i \in Server : \quad LeaderDiscovery1(i) \\
\quad \lor \exists\, i,\, j \in Server : FollowerHandleLEADERINFO(i,\, j) \\
\quad \lor \exists\, i,\, j \in Server : LeaderHandleACKEPOCH(i,\, j)
\end{array}
$$

$\lor \exists\, i \in Server : \quad LeaderDiscovery2(i)$
$\lor \exists\, i, j \in Server : RECOVERYSYNC(i, j)$
$\lor \exists\, i, j \in Server : FollowerHandleNEWLEADER(i, j)$
$\lor \exists\, i, j \in Server : LeaderHandleACKLD(i, j)$
$\lor \exists\, i \in Server : \quad LeaderSync2(i)$
$\lor \exists\, i, j \in Server : FollowerHandleUPTODATE(i, j)$

Zab module – *Broadcast* part

$\lor \exists\, i \in Server,\, v \in Value : ClientRequest(i,\, v)$
$\lor \exists\, i \in Server : \quad LeaderBroadcast1(i)$
$\lor \exists\, i, j \in Server : FollowerHandlePROPOSAL(i, j)$
$\lor \exists\, i, j \in Server : LeaderHandleACK(i, j)$
$\lor \exists\, i \in Server : \quad LeaderAdvanceCommit(i)$
$\lor \exists\, i \in Server : \quad LeaderBroadcast2(i)$
$\lor \exists\, i, j \in Server : FollowerHandleCOMMIT(i, j)$

An action used to judge whether there are redundant messages in network

$\lor \exists\, i \in Server : \quad FilterNonexistentMessage(i)$

$SpecZ \triangleq InitZ \land \Box[NextZ]_{vars}$

Define safety properties of *Zab* 1.0 protocol.

$ShouldNotBeTriggered \triangleq inherentViolated = \text{FALSE}$

There is most one established leader for a certain epoch.

$$\begin{aligned} Leadership1 \triangleq\ & \forall\, i, j \in Server : \\ & \land state[i] = LEADING \land zabState[i] \in \{SYNCHRONIZATION,\, BROADCAST\} \\ & \land state[j] = LEADING \land zabState[j] \in \{SYNCHRONIZATION,\, BROADCAST\} \\ & \land acceptedEpoch[i] = acceptedEpoch[j] \\ & \Rightarrow i = j \end{aligned}$$

$Leadership2 \triangleq \forall\, epoch \in 1 \,..\, MAXEPOCH : Cardinality(epochLeader[epoch]) < 2$

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

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

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

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

$\lor \land msg.mtype = \text{“RECOVERYSYNC”}$
$\land \exists\, tindex \in 1 \,..\, Len(msg.mproposals) : TransactionEqual(msg.mproposals[tindex], h$

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

$$
\begin{aligned}
Agreement \triangleq\ & \forall\, i, j \in Server : \\
& \land state[i] = FOLLOWING \land commitIndex[i] > 0 \\
& \land state[j] = FOLLOWING \land commitIndex[j] > 0 \\
& \Rightarrow \\
& \forall\, index1 \in 1 \,..\, commitIndex[i],\ index2 \in 1 \,..\, commitIndex[j] : \\
& \quad \lor \exists\, indexj \in 1 \,..\, commitIndex[j] : \\
& \qquad TransactionEqual(history[j][indexj],\ history[i][index1]) \\
& \quad \lor \exists\, indexi \in 1 \,..\, commitIndex[i] : \\
& \qquad TransactionEqual(history[i][indexi],\ history[j][index2])
\end{aligned}
$$

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

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

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

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

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.

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

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.

$$\begin{aligned} PrimaryIntegrity \triangleq\ & \forall\, i,\ j \in Server :\ \land\ state[i] = LEADING \\ & \qquad\qquad\ \land\ state[j] = FOLLOWING \\ & \qquad\qquad\ \land\ commitIndex[j] \geq 1 \\ & \qquad \Rightarrow \forall\, index \in 1\,..\,commitIndex[j] :\ \lor\ history[j][index].epoch \geq currentEpoch[i] \\ & \qquad\qquad\qquad\ \lor\ \land\ history[j][index].epoch < currentEpoch[i] \\ & \qquad\qquad\qquad\quad\ \land\ \exists\, idx \in 1\,..\,commitIndex[i] : TransactionEqua \end{aligned}$$

\ * Modification History
\ * Last modified *Thu Jul* 15 16:10:01 *CST* 2021 by Dell
\ * Created *Tue Jun* 29 22:13:02 *CST* 2021 by Dell