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*

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*

NOTE: Additional message types used for recovery in *synchronization*(*TRUNC*/*DIFF*/*SNAP*) are not needed since we abstract this part.(*see action RECOVERYSYNC*)
NOTE: In production, there is no message type *ACKLD*. Server judges if counter of *ACK* is 0 to distinguish one *ACK* represents *ACKLD* or not. Here we divide *ACK* into *ACKLD* and *ACK*, to enhance readability of spec.
TODO: Consider in the future replacing the magic atomic synchronization *RECOVERYSYNC* with *DIFF* based message passing.

[*MaxTimeoutFailures*, *MaxTransactionNum*, *MaxEpoch*]
CONSTANT *Parameters*
TODO: Here we can add more constraints to decrease space.

$MAXEPOCH \triangleq 10$

Variables that all servers need to use.
VARIABLES *zabState*, The current phase of server, in {*ELECTION*, *DISCOVERY*, *SYNCHRONIZATION*, *BROADCAST*}.

acceptedEpoch, The epoch number of the last *LEADERINFO* packet accepted, namely $f.p$ in paper.

history, The history of servers as the sequence of transactions.

commitIndex Maximum index known to be committed. Starts from 0, and increases monotonically before restarting. Equals to 'lastCommitted' in code.

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.

Variables only used when $state = LEADING$.

VARIABLES $learners$, The set of servers which leader i think are connected wich i.

$cepochRecv$, The set of followers who has successfully sent $FOLLOWERINFO$ to leader. Equals to 'connectingFollowers' in code.

$ackeRecv$, The set of followers who has successfully sent ACK-E to leader. Equals to 'electingFollowers' in code.

$ackldRecv$, The set of followers who has successfully sent ACK-LD to leader in leader. Equals to 'newLeaderProposal' in code.

$forwarding$, The set of servers which leader i should broadcast $PROPOSAL$ and $COMMIT$ to. Equals to 'forwardingFollowers' in code.

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

$currentCounter$, $[i]$: The count of transactions that clients have requested leader i.

$sendCounter$, $\backslash * [i]$: The count of transactions that leader i has broadcast via $PROPOSAL$.

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

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

Variables only used when $state = LEADING$ & $zabState! = BROADCAST$.

VARIABLES $initialHistory$, $[i]$: The initial history of leader i in epoch $acceptedEpoch[i]$.

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

Variables only used when $state = FOLLOWING$.

VARIABLES $cepochSent$, Express whether follower has sent $FOLLOWERINFO$ to leader.

$leaderAddr$, Express whether follower i has connected or lost connection. $[i]$: The leader id of follower i.

$synced$ Express whether follower has completed sync with leader.

Variables about network channel.

VARIABLE $msgs$ Simulates network channel. $msgs[i][j]$ means the input buffer of server j from server i.

Variables only used in verifying properties.

VARIABLES $epochLeader$, The set of leaders in every epoch.

$proposalMsgsLog$, The set of all broadcast messages.

$inherentViolated$ Check whether there are conditions contrary to the facts.

Variable used for recording data to constrain state space.
VARIABLE $recorder$ Consists: members of *Parameters* and *pc*.

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

$electionVarsZ \triangleq electionVars$ 6 variables

$leaderVarsZ \triangleq \langle leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv, forwarding,$
$ackIndex, currentCounter, committedCounter \rangle$ 9 variables

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

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

$ServersIncNullPoint \triangleq Server \cup \{NullPoint\}$

$Zxid \triangleq$
$Seq(Nat)$
$\cup$ [epoch: Nat, counter: Nat]

$HistoryItem \triangleq$
$[epoch : Nat,$
$counter : Nat,$
$value : Value]$

$Proposal \triangleq$
$[msource : Server, mtype : \{\text{“RECOVERYSYNC”}\}, mepoch : Nat, mproposals : Seq(HistoryItem)] \cup$
$[msource : Server, mtype : \{PROPOSAL\}, mepoch : Nat, mproposal : HistoryItem]$

$Message \triangleq$
$[mtype : \{FOLLOWERINFO\}, mepoch : Nat] \cup$
$[mtype : \{NEWLEADER\}, mepoch : Nat, mlastZxid : Zxid] \cup$
$[mtype : \{ACKLD\}, mepoch : Nat] \cup$
$[mtype : \{LEADERINFO\}, mepoch : Nat] \cup$
$[mtype : \{ACKEPOCH\}, mepoch : Nat, mlastEpoch : Nat, mlastZxid : Zxid] \cup$
$[mtype : \{UPTODATE\}, mepoch : Nat, mcommit : Nat] \cup$
$[mtype : \{PROPOSAL\}, mepoch : Nat, mproposal : HistoryItem] \cup$
$[mtype : \{ACK\}, mepoch : Nat, mzxid : Zxid] \cup$
$[mtype : \{COMMIT\}, mepoch : Nat, mzxid : Zxid]$

$ElectionState \triangleq \{LOOKING, FOLLOWING, LEADING\}$

$Vote \triangleq$
$[proposedLeader : ServersIncNullPoint,$
$proposedZxid : Zxid,$
$proposedEpoch : Nat]$

$ElectionVote \triangleq$
$[vote : Vote,\ round : Nat,\ state : ElectionState,\ version : Nat]$

$ElectionMsg \triangleq$
$[mtype : \{NOTIFICATION\},\ msource : Server,\ mstate : ElectionState,\ mround : Nat,\ mvote : Vote] \cup$
$[mtype : \{NONE\}]$

$TypeOK \triangleq$
$\land \quad zabState \in [Server \to \{ELECTION,\ DISCOVERY,\ SYNCHRONIZATION,\ BROADCAST\}]$
$\land \quad acceptedEpoch \in [Server \to Nat]$
$\land \quad history \in [Server \to Seq(HistoryItem)]$
$\land \quad commitIndex \in [Server \to Nat]$
$\land \quad learners \in [Server \to \text{SUBSET } ServersIncNullPoint]$
$\land \quad cepochRecv \in [Server \to \text{SUBSET } ServersIncNullPoint]$
$\land \quad ackeRecv \in [Server \to \text{SUBSET } ServersIncNullPoint]$
$\land \quad ackldRecv \in [Server \to \text{SUBSET } ServersIncNullPoint]$
$\land \quad ackIndex \in [Server \to [Server \to Nat]]$
$\land \quad currentCounter \in [Server \to Nat]$
$\land\ sendCounter \in\ [Server \to Nat]$
$\land\ committedIndex \in\ [Server \to Nat]$
$\land\ committedCounter \in [Server \to [Server \to Nat]]$
$\land\ forwarding \in [Server \to \text{SUBSET } ServersIncNullPoint]$
$\land\ initialHistory \in [Server \to Seq(HistoryItem)]$
$\land\ tempMaxEpoch \in [Server \to Nat]$
$\land\ cepochSent \in [Server \to \text{BOOLEAN }]$
$\land\ leaderAddr \in [Server \to ServersIncNullPoint]$
$\land\ synced \in [Server \to \text{BOOLEAN }]$
$\land\ proposalMsgsLog \in \text{SUBSET } Proposal$
$\land\ epochLeader \in [1\ ..\ MAXEPOCH \to \text{SUBSET } ServersIncNullPoint]$
$\land\ inherentViolated \in \text{BOOLEAN}$
$\land\ forwarding \in [Server \to \text{SUBSET } Server]$
$\land\ msgs \in [Server \to [Server \to Seq(Message)]]$
Fast Leader Election
$\land\ electionMsgs \in [Server \to [Server \to Seq(ElectionMsg)]]$
$\land\ recvQueue \in [Server \to Seq(ElectionMsg)]$
$\land\ leadingVoteSet \in [Server \to \text{SUBSET } Server]$
$\land\ receiveVotes \in [Server \to [Server \to ElectionVote]]$
$\land\ currentVote \in [Server \to Vote]$
$\land\ outOfElection \in [Server \to [Server \to ElectionVote]]$
$\land\ lastZxid \in [Server \to Zxid]$
$\land\ state \in [Server \to ElectionState]$

$\land$ $waitNotmsg \in [Server \to \text{BOOLEAN}\ ]$
$\land$ $currentEpoch \in [Server \to Nat]$
$\land$ $logicalClock \in [Server \to Nat]$

Return the maximum value from the set S

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

Return the minimum value from the set S

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

Check server state

$IsLeader(s) \triangleq state[s] = LEADING$
$IsFollower(s) \triangleq state[s] = FOLLOWING$
$IsLooking(s) \triangleq state[s] = LOOKING$

$IsMyLearner(i, j) \triangleq j \in learners[i]$
$IsMyLeader(i, j) \triangleq leaderAddr[i] = j$
$HasNoLeader(i) \triangleq leaderAddr[i] = NullPoint$
$HasLeader(i) \triangleq leaderAddr[i] \neq NullPoint$

Check if s is a quorum

$IsQuorum(s) \triangleq s \in Quorums$

Check *zxid* state

$ToZxid(z) \triangleq [epoch \mapsto z[1],\ counter \mapsto z[2]]$

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

$TransactionEqual(t1, t2) \triangleq \land\, t1.epoch = t2.epoch$
$\land\, t1.counter = t2.counter$

$TransactionPrecede(t1, t2) \triangleq \lor\, t1.epoch < t2.epoch$
$\lor\, \land\, t1.epoch = t2.epoch$
$\land\, t1.counter < t2.counter$

Actions about recorder

$GetParameter(p) \triangleq \text{IF}\ p \in \text{DOMAIN}\ Parameters\ \text{THEN}\ Parameters[p]\ \text{ELSE}\ 0$

$RecorderGetHelper(m) \triangleq (m :> recorder[m])$
$RecorderIncHelper(m) \triangleq (m :> recorder[m] + 1)$

$RecorderIncTimeout \triangleq RecorderIncHelper(\text{“nTimeout”})$
$RecorderGetTimeout \triangleq RecorderGetHelper(\text{“nTimeout”})$
$RecorderSetTransactionNum \triangleq (\text{“nTransaction”} :>$
$\text{LET}\ s \triangleq \text{CHOOSE}\ i \in Server :$

$$\begin{array}{l}
\qquad\qquad\qquad\qquad\quad \forall\, j \in Server : Len(history'[i]) \geq Len(history'[j]) \\
\qquad\qquad\qquad\quad \text{IN} \quad Len(history'[s])) \\
RecorderSetMaxEpoch \triangleq (\text{“maxEpoch”} :> \\
\qquad\qquad\quad \text{LET } s \triangleq \text{CHOOSE } i \in Server : \\
\qquad\qquad\qquad\quad \forall\, j \in Server : acceptedEpoch'[i] \geq acceptedEpoch'[j] \\
\qquad\qquad\quad \text{IN} \quad acceptedEpoch'[s]) \\
RecorderSetPc(pc) \triangleq (\text{“pc”} :> pc) \\
RecorderSetFailure(pc) \triangleq \text{CASE } pc[1] = \text{“Timeout”} \quad\quad\quad\quad \rightarrow RecorderIncTimeout \\
\qquad\qquad\qquad\qquad\quad \square \quad pc[1] = \text{“LeaderTimeout”} \rightarrow RecorderIncTimeout \\
\qquad\qquad\qquad\qquad\quad \square \quad pc[1] = \text{“FollowerTimout”} \rightarrow RecorderIncTimeout \\
\qquad\qquad\qquad\qquad\quad \square \quad \text{OTHER} \quad\quad\quad\quad\quad\quad\quad\quad \rightarrow RecorderGetTimeout
\end{array}$$

$$\begin{array}{l}
UpdateRecorder(pc) \triangleq recorder' = RecorderSetFailure(pc) \ @@ \ RecorderSetTransactionNum \\
\qquad\qquad\qquad\qquad\qquad\qquad @@ \ RecorderSetMaxEpoch \ @@ \ RecorderSetPc(pc) \ @@ \ recorder \\
UnchangeRecorder \triangleq \text{UNCHANGED } recorder
\end{array}$$

$$\begin{array}{l}
CheckParameterHelper(n,\, p,\, Comp(_,\, _)) \triangleq \text{IF } p \in \text{DOMAIN } Parameters \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{THEN } Comp(n,\, Parameters[p]) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{ELSE TRUE} \\
CheckParameterLimit(n,\, p) \triangleq CheckParameterHelper(n,\, p,\, \text{LAMBDA } i,\, j : i < j)
\end{array}$$

$$\begin{array}{lll}
CheckTimeout & \triangleq & CheckParameterLimit(recorder.nTimeout, \quad \text{“MaxTimeoutFailures”}) \\
CheckTransactionNum & \triangleq & CheckParameterLimit(recorder.nTransaction, \ \text{“MaxTransactionNum”}) \\
CheckEpoch & \triangleq & CheckParameterLimit(recorder.maxEpoch, \quad \text{“MaxEpoch”})
\end{array}$$

$$CheckStateConstraints \triangleq CheckTimeout \land CheckTransactionNum \land CheckEpoch$$

Actions about network

$$\begin{array}{lll}
PendingFOLLOWERINFO(i,\, j) & \triangleq & \land\, msgs[j][i] \neq \langle\rangle \\
 & & \land\, msgs[j][i][1].mtype = FOLLOWERINFO \\
PendingLEADERINFO(i,\, j) & \triangleq & \land\, msgs[j][i] \neq \langle\rangle \\
 & & \land\, msgs[j][i][1].mtype = LEADERINFO \\
PendingACKEPOCH(i,\, j) & \triangleq & \land\, msgs[j][i] \neq \langle\rangle \\
 & & \land\, msgs[j][i][1].mtype = ACKEPOCH \\
PendingNEWLEADER(i,\, j) & \triangleq & \land\, msgs[j][i] \neq \langle\rangle \\
 & & \land\, msgs[j][i][1].mtype = NEWLEADER \\
PendingACKLD(i,\, j) & \triangleq & \land\, msgs[j][i] \neq \langle\rangle \\
 & & \land\, msgs[j][i][1].mtype = ACKLD \\
PendingUPTODATE(i,\, j) & \triangleq & \land\, msgs[j][i] \neq \langle\rangle \\
 & & \land\, msgs[j][i][1].mtype = UPTODATE \\
PendingPROPOSAL(i,\, j) & \triangleq & \land\, msgs[j][i] \neq \langle\rangle \\
 & & \land\, msgs[j][i][1].mtype = PROPOSAL \\
PendingACK(i,\, j) & \triangleq & \land\, msgs[j][i] \neq \langle\rangle \\
 & & \land\, msgs[j][i][1].mtype = ACK \\
PendingCOMMIT(i,\, j) & \triangleq & \land\, msgs[j][i] \neq \langle\rangle \\
 & & \land\, msgs[j][i][1].mtype = COMMIT
\end{array}$$

Add a message to $msgs-$ add a message m to $msgs$.

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

Remove a message from $msgs-$ discard head of $msgs$.

$Discard(i,\ j) \triangleq msgs' = \text{IF } msgs[i][j] \neq \langle\rangle \text{ THEN } [msgs \text{ EXCEPT } ![i][j] = Tail(msgs[i][j])] \text{ ELSE } msgs$

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

$Broadcast(i,\ m) \triangleq msgs' = [msgs \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } \land v \in forwarding[i] \land v \neq i \text{ THEN } Append(msgs[i][v],\ m) \text{ ELSE } msgs[i][v]]]$

$Broadcast(i,\ m) \triangleq msgs' = [msgs \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } \land v \in forwarding[i] \land v \neq i \land \lor \land m.mtype = PROPOSAL \land ackIndex[i][v] < Len(initialHistory[i]) + m.mproposal.counter \lor \land m.mtype = COMMIT \land committedCounter[i][v] < m.mzxid[2] \text{ THEN } Append(msgs[i][v],\ m) \text{ ELSE } msgs[i][v]]]$

$DiscardAndBroadcast(i,\ j,\ m) \triangleq msgs' = [msgs \text{ EXCEPT } ![j][i] = Tail(msgs[j][i]), ![i] = [v \in Server \mapsto \text{IF } \land v \in forwarding[i] \land v \neq i \text{ THEN } Append(msgs[i][v],\ m) \text{ ELSE } msgs[i][v]]]$

Leader broadcasts $LEADERINFO$ to all other servers in $connectingFollowers$.

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

Leader broadcasts $UPTODATE$ to all other servers in $newLeaderProposal$.

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

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

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

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

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

Define initial values for all variables

$InitServerVarsZ \triangleq \land InitServerVars \land zabState = [s \in Server \mapsto ELECTION]$

$$
\begin{array}{l}
\quad \land\ acceptedEpoch = [s \in Server \mapsto 0] \\
\quad \land\ history \quad\quad = [s \in Server \mapsto \langle\rangle] \\
\quad \land\ commitIndex = [s \in Server \mapsto 0]
\end{array}
$$

$$
\begin{array}{l}
InitLeaderVarsZ \triangleq \land\ InitLeaderVars \\
\quad \land\ learners \quad\quad = [s \in Server \mapsto \{\}] \\
\quad \land\ cepochRecv \quad = [s \in Server \mapsto \{\}] \\
\quad \land\ ackeRecv \quad\quad = [s \in Server \mapsto \{\}] \\
\quad \land\ ackldRecv \quad\quad = [s \in Server \mapsto \{\}] \\
\quad \land\ ackIndex \quad\quad = [s \in Server \mapsto [v \in Server \mapsto 0]] \\
\quad \land\ currentCounter = [s \in Server \mapsto 0] \\
\quad \land\ sendCounter \quad = [s \in Server \mapsto 0] \\
\quad \land\ committedIndex = [s \in Server \mapsto 0] \\
\quad \land\ committedCounter = [s \in Server \mapsto [v \in Server \mapsto 0]] \\
\quad \land\ forwarding \quad\quad = [s \in Server \mapsto \{\}]
\end{array}
$$

$$
InitElectionVarsZ \triangleq InitElectionVars
$$

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

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

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

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

$$
\begin{array}{l}
InitRecorder \triangleq recorder = [nTimeout \quad\quad \mapsto 0, \\
\qquad\qquad\qquad\qquad\quad nTransaction \mapsto 0, \\
\qquad\qquad\qquad\qquad\quad maxEpoch \quad\quad \mapsto 0, \\
\qquad\qquad\qquad\qquad\quad pc \quad\quad\quad\quad \mapsto \langle \text{“InitZ”} \rangle]
\end{array}
$$

$$
\begin{array}{l}
InitZ \triangleq \land\ InitServerVarsZ \\
\quad \land\ InitLeaderVarsZ \\
\quad \land\ InitElectionVarsZ \\
\quad \land\ InitTempVarsZ \\
\quad \land\ InitFollowerVarsZ \\
\quad \land\ InitVerifyVarsZ \\
\quad \land\ InitMsgVarsZ \\
\quad \land\ idTable = InitializeIdTable(Server) \\
\quad \land\ InitRecorder
\end{array}
$$

$ZabTurnToLeading(i) \triangleq$
$\land\ zabState' = [zabState \text{ EXCEPT } ![i] = DISCOVERY]$
$\land\ learners' = [learners \text{ EXCEPT } ![i] = \{i\}]$
$\land\ cepochRecv' = [cepochRecv \text{ EXCEPT } ![i] = \{i\}]$
$\land\ ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \{i\}]$
$\land\ ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] = \{i\}]$
$\land\ forwarding' = [forwarding \text{ EXCEPT } ![i] = \{\}]$
$\land\ ackIndex' = [ackIndex \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } v = i \text{ THEN } Len(history[i]) \text{ ELSE } 0]]$
$\land\ currentCounter' = [currentCounter \text{ EXCEPT } ![i] = 0]$
$\land\ commitIndex' = [commitIndex \text{ EXCEPT } ![i] = 0]$
$\land\ sendCounter' = [sendCounter \text{ EXCEPT } ![i] = 0]$
$\land\ committedIndex' = [committedIndex \text{ EXCEPT } ![i] = 0]$
$\land\ committedCounter' = [committedCounter \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } v = i \text{ THEN } Len(history[i]) \text{ ELSE } 0]]$
$\land\ initialHistory' = [initialHistory \text{ EXCEPT } ![i] = history[i]]$
$\land\ tempMaxEpoch' = [tempMaxEpoch \text{ EXCEPT } ![i] = acceptedEpoch[i]]$

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

Fast Leader Election

$FLEReceiveNotmsg(i, j) \triangleq$
$\land\ ReceiveNotmsg(i, j)$
$\land\ \text{UNCHANGED } \langle zabState, acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex, currentCounter, committedCounter, tempVarsZ, followerVarsZ, verifyVarsZ, msgs\rangle$
$\land\ UpdateRecorder(\langle \text{“FLEReceiveNotmsg”}, i, j\rangle)$

$FLENotmsgTimeout(i) \triangleq$
$\land\ NotmsgTimeout(i)$
$\land\ \text{UNCHANGED } \langle zabState, acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackl$
$forwarding, ackIndex, currentCounter, committedCounter, tempVarsZ, followerVarsZ, verifyVarsZ, msgs\rangle$
$\land\ UpdateRecorder(\langle \text{“FLENotmsgTimeout”}, i\rangle)$

$FLEHandleNotmsg(i) \triangleq$
$\land\ HandleNotmsg(i)$
$\land\ \text{LET } newState \triangleq state'[i]$
IN
$\lor\ \land\ newState = LEADING$
$\land\ ZabTurnToLeading(i)$

```
                /\ UNCHANGED ⟨cepochSent, synced⟩
             \/ /\ newState = FOLLOWING
                /\ ZabTurnToFollowing(i)
                /\ UNCHANGED ⟨learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex, currentCounte
                             committedCounter, tempVarsZ⟩
             \/ /\ newState = LOOKING
                /\ UNCHANGED ⟨zabState, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex,
                              currentCounter, commitIndex,
                              committedCounter, tempVarsZ, cepochSent, synced⟩
          /\ UNCHANGED ⟨acceptedEpoch, history, leaderAddr, verifyVarsZ, msgs⟩
          /\ UpdateRecorder(⟨"FLEHandleNotmsg", i⟩)

 On the premise that ReceiveVotes.HasQuorums = TRUE, corresponding to logic in line 1050 − 1055 in LFE.java.
FLEWaitNewNotmsg(i) ≜
          /\ WaitNewNotmsg(i)
          /\ UNCHANGED ⟨zabState, acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv,
                        ackldRecv, forwarding, ackIndex, currentCounter,
                        committedCounter, tempVarsZ, followerVarsZ, verifyVarsZ, msgs⟩
          /\ UpdateRecorder(⟨"FLEWaitNewNotmsg", i⟩)

 On the premise that ReceiveVotes.HasQuorums = TRUE, corresponding to logic in line 1061 − 1066 in LFE.java.
FLEWaitNewNotmsgEnd(i) ≜
          /\ WaitNewNotmsgEnd(i)
          /\ LET newState ≜ state'[i]
             IN
             \/ /\ newState = LEADING
                /\ ZabTurnToLeading(i)
                /\ UNCHANGED ⟨cepochSent, synced⟩
             \/ /\ newState = FOLLOWING
                /\ ZabTurnToFollowing(i)
                /\ UNCHANGED ⟨learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex, currentCounte
                             committedCounter, tempVarsZ⟩
             \/ /\ newState = LOOKING
                /\ PrintT("New state is LOOKING in FLEWaitNewNotmsgEnd, which should not happen.")
                /\ UNCHANGED ⟨zabState, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex,
                              currentCounter, commitIndex,
                              committedCounter, tempVarsZ, cepochSent, synced⟩
          /\ UNCHANGED ⟨acceptedEpoch, history, leaderAddr, verifyVarsZ, msgs⟩
          /\ UpdateRecorder(⟨"FLEWaitNewNotmsgEnd", i⟩)
```

```
 Describe how a server transitions from LEADING/FOLLOWING to LOOKING.
FollowerShutdown(i) ≜
          /\ ZabTimeout(i)
          /\ zabState'    = [zabState    EXCEPT ![i] = ELECTION]
          /\ leaderAddr' = [leaderAddr EXCEPT ![i] = NullPoint]
```

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

$RemoverLearner(i, j) \triangleq$
$\land\ learners' = [learners \text{ EXCEPT } ![i] = learners[i] \setminus \{j\}]$
$\land\ forwarding' = [forwarding \text{ EXCEPT } ![i] = \text{IF } j \in forwarding[i] \text{ THEN } forwarding[i] \setminus \{j\} \text{ ELSE } forw$
$\land\ cepochRecv' = [cepochRecv \text{ EXCEPT } ![i] = \text{IF } j \in cepochRecv[i] \text{ THEN } cepochRecv[i] \setminus \{j\} \text{ ELSE } cepo$
$\land\ ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \text{IF } j \in ackeRecv[i] \text{ THEN } ackeRecv[i] \setminus \{j\} \text{ ELSE } acke$
$\land\ ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] = \text{IF } j \in ackldRecv[i] \text{ THEN } ackldRecv[i] \setminus \{j\} \text{ ELSE } ackl$
$\land\ ackIndex' = [ackIndex \text{ EXCEPT } ![i][j] = 0]$
$\land\ committedCounter' = [committedCounter \text{ EXCEPT } ![i][j] = 0]$

$FollowerTimout(i) \triangleq$
$\land\ IsFollower(i)$
$\land\ HasNoLeader(i)$
$\land\ FollowerShutdown(i)$
$\land\ msgs' = [s \in Server \mapsto [v \in Server \mapsto \text{IF } v = i \text{ THEN } \langle\rangle \text{ ELSE } msgs[s][v]]]$
$\land\ \text{UNCHANGED } \langle acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex, currentCounter, committedCounter, tempVarsZ, cepochSent, synced, verifyVarsZ\rangle$
$\land\ UpdateRecorder(\langle \text{“FollowerTimout”}, i\rangle)$

$LeaderTimeout(i) \triangleq$
$\land\ IsLeader(i)$
$\land\ \neg IsQuorum(learners[i])$
$\land\ LeaderShutdown(i)$
$\land\ \text{UNCHANGED } \langle acceptedEpoch, history, commitIndex, cepochRecv, ackeRecv, ackldRecv, ackIndex, committedCounter, tempVarsZ, cepochSent, synced, verifyVarsZ\rangle$
$\land\ UpdateRecorder(\langle \text{“LeaderTimeout”}, i\rangle)$

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

$EstablishConnection(i, j) \triangleq$
$\land\ IsLeader(i)$
$\land\ IsFollower(j)$
$\land\ \neg IsMyLearner(i, j)$
$\land\ HasNoLeader(i)$
$\land\ currentVote[j].proposedLeader = i$
$\land\ learners' = [learners \text{ EXCEPT } ![i] = learners[i] \cup \{j\}]$ Leader: '$addLearnerHandler(peer)$'
$\land\ leaderAddr' = [leaderAddr \text{ EXCEPT } ![j] = i]$ Follower: '$connectToLeader(addr, hostname)$'

$\land$ UNCHANGED $\langle serverVarsZ,\ electionVarsZ,\ leadingVoteSet,\ cepochRecv,\ ackeRecv,\ ackldRecv,\ forwa$
$ackIndex,\ \ currentCounter,\ committedCounter,\ tempVarsZ,$
$cepochSent,\ synced,\ verifyVarsZ,\ msgVarsZ,\ idTable\rangle$
$\land\ UpdateRecorder(\langle$"EstablishConnection"$,\ i,\ j\rangle)$

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

$Timeout(i,\ j)\ \triangleq$
$\land\ IsLeader(i)$
$\land\ IsFollower(j)$
$\land\ IsMyLearner(i,\ j)$
$\land\ IsMyLeader(j,\ i)$
The action of leader i.($corresponding\ to\ function'removeLearnerHandler(peer)'$.)
$\land\ RemoverLearner(i,\ j)$
The action of follower j.
$\land\ FollowerShutdown(j)$
Clean channel between i and j.
$\land\ Clean(i,\ j)$
$\land$ UNCHANGED $\langle acceptedEpoch,\ history,\ commitIndex,\ currentCounter,$
$tempVarsZ,\ cepochSent,\ synced,\ verifyVarsZ\rangle$
$\land\ UpdateRecorder(\langle$"Timeout"$,\ i,\ j\rangle)$

Follower sends $f.p$ to leader via $FOLLOWERINFO(CEPOCH)$.

$FollowerSendFOLLOWERINFO(i)\ \triangleq$
$\land\ IsFollower(i)$
$\land\ zabState[i] = DISCOVERY$
$\land\ HasLeader(i)$
$\land\ \neg cepochSent[i]$
$\land\ Send(i,\ leaderAddr[i],\ [mtype\ \mapsto FOLLOWERINFO,$
$mepoch \mapsto acceptedEpoch[i]])$
$\land\ cepochSent' = [cepochSent$ EXCEPT $![i] =$ TRUE$]$
$\land$ UNCHANGED $\langle serverVarsZ,\ leaderVarsZ,\ electionVarsZ,\ tempVarsZ,\ leaderAddr,\ synced,$
$verifyVarsZ,\ electionMsgs,\ idTable\rangle$
$\land\ UpdateRecorder(\langle$"FollowerSendFOLLOWERINFO"$,\ i\rangle)$

Leader waits for receiving $FOLLOWERINFO$ from a quorum, and then chooses a new epoch e' as its own epoch and broadcasts $LEADERINFO$.

$LeaderHandleFOLLOWERINFO(i,\ j)\ \triangleq$
$\land\ IsLeader(i)$
$\land\ PendingFOLLOWERINFO(i,\ j)$
$\land$ LET $msg\ \triangleq\ msgs[j][i][1]$
IN $\lor$ 1. has not broadcast $LEADERINFO$ – modify $tempMaxEpoch$
$\land\ NullPoint \notin cepochRecv[i]$
$\land$ LET $newEpoch\ \triangleq\ Maximum(\{tempMaxEpoch[i],\ msg.mepoch\})$
IN $tempMaxEpoch' = [tempMaxEpoch$ EXCEPT $![i] = newEpoch]$
$\land\ Discard(j,\ i)$

```
                \/  2. has broadcast LEADERINFO − no need to handle the msg, just send LEADERINFO to correspondin
                    /\ NullPoint \in cepochRecv[i]
                    /\ Reply(i, j, [mtype  |-> LEADERINFO,
                                    mepoch |-> acceptedEpoch[i]])
                    /\ UNCHANGED tempMaxEpoch
            /\ cepochRecv' = [cepochRecv EXCEPT ![i] = IF j \in cepochRecv[i] THEN cepochRecv[i]
                                                                              ELSE cepochRecv[i] \cup {j}]
            /\ UNCHANGED <<serverVarsZ, followerVarsZ, electionVarsZ, initialHistory, leadingVoteSet, learners,
                           ackeRecv, ackldRecv, forwarding, ackIndex, currentCounter,
                           committedCounter, verifyVarsZ, electionMsgs, idTable>>
            /\ UpdateRecorder(<<"LeaderHandleFOLLOWERINFO", i, j>>)

LeaderBroadcastLEADERINFO(i) ==
        /\ IsLeader(i)
        /\ zabState[i] = DISCOVERY
        /\ IsQuorum(cepochRecv[i])
        /\ acceptedEpoch' = [acceptedEpoch EXCEPT ![i] = tempMaxEpoch[i] + 1]
        /\ cepochRecv'    = [cepochRecv    EXCEPT ![i] = cepochRecv[i] \cup {NullPoint}]
        /\ BroadcastLEADERINFO(i, [mtype  |-> LEADERINFO,
                                   mepoch |-> acceptedEpoch'[i]])
        /\ UNCHANGED <<state, currentEpoch, lastZxid, zabState, history, commitIndex, electionVarsZ,
                       leadingVoteSet, learners, ackeRecv, ackldRecv, forwarding, ackIndex, currentCounter
                       committedCounter, tempVarsZ, followerVarsZ, verifyVarsZ,
                       electionMsgs, idTable>>
        /\ UpdateRecorder(<<"LeaderBroadcastLEADERINFO", i>>)

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) ==
        /\ IsFollower(i)
        /\ PendingLEADERINFO(i, j)
        /\ LET msg       == msgs[j][i][1]
               infoOk    == IsMyLeader(i, j)
               epochOk   == /\ infoOk
                            /\ msg.mepoch >= acceptedEpoch[i]
               correct   == /\ epochOk
                            /\ zabState[i] = DISCOVERY
           IN  /\ infoOk
               /\ \/ 1. Normal case
                     /\ epochOk
                     /\ \/ /\ correct
                           /\ acceptedEpoch' = [acceptedEpoch EXCEPT ![i] = msg.mepoch]
                           /\ Reply(i, j, [mtype      |-> ACKEPOCH,
                                           mepoch     |-> msg.mepoch,
                                           mlastEpoch |-> currentEpoch[i],
```

```
                                        mlastZxid   ↦ lastZxid[i]])
                        ∧ cepochSent' = [cepochSent EXCEPT ![i] = TRUE]
                        ∧ UNCHANGED inherentViolated
                    ∨ ∧ ¬correct
                        ∧ PrintT("Exception: Follower receives LEADERINFO while its ZabState is not DISCO
                        ∧ inherentViolated' = TRUE
                        ∧ Discard(j, i)
                        ∧ UNCHANGED ⟨acceptedEpoch, cepochSent⟩
                ∧ zabState' = [zabState EXCEPT ![i] = IF zabState[i] = DISCOVERY THEN SYNCHRON
                                                                                 ELSE zabState[i]]
                ∧ UNCHANGED ⟨varsL, leaderAddr⟩
            ∨ 2. Abnormal case - go back to election
                ∧ ¬epochOk
                ∧ FollowerShutdown(i)
                ∧ Clean(i, j)
                ∧ UNCHANGED ⟨acceptedEpoch, cepochSent, inherentViolated⟩
        ∧ UNCHANGED ⟨history, commitIndex, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackInd
                     currentCounter, committedCounter, tempVarsZ, synced, proposalMsgsLog, epochLead
        ∧ UpdateRecorder(⟨"FollowerHandleLEADERINFO", i, j⟩)

Abstraction of actions making follower synced with leader before leader sending NEWLEADER.
SubRECOVERYSYNC(i, j) ≜
    LET canSync ≜ ∧ IsLeader(i)   ∧ zabState[i] ≠ DISCOVERY          ∧ IsMyLearner(i, j)
                  ∧ IsFollower(j) ∧ zabState[j] = SYNCHRONIZATION ∧ IsMyLeader(j, i)
                  ∧ synced[j] = FALSE
    IN
    ∨ ∧ canSync
      ∧ history'     = [history     EXCEPT ![j] = history[i]]
      ∧ lastZxid'    = [lastZxid    EXCEPT ![j] = lastZxid[i]]
      ∧ UpdateProposal(j, leaderAddr[j], lastZxid'[j], currentEpoch[j])
      ∧ commitIndex' = [commitIndex EXCEPT ![j] = commitIndex[i]]
      ∧ synced'      = [synced      EXCEPT ![j]   = TRUE]
      ∧ forwarding'  = [forwarding  EXCEPT ![i]   = forwarding[i] ∪ {j}]    j will join traffic, and receive PI
      ∧ ackIndex'    = [ackIndex    EXCEPT ![i][j] = Len(history[i])]
      ∧ committedCounter' = [committedCounter EXCEPT ![i][j] = Maximum({commitIndex[i] - Len(ini
      ∧ LET ms ≜ [msource ↦ i, mtype ↦ "RECOVERYSYNC", mepoch ↦ acceptedEpoch[i], mproposals
        IN  proposalMsgsLog' = IF ms ∈ proposalMsgsLog THEN proposalMsgsLog
                                                       ELSE proposalMsgsLog ∪ {ms}
      ∧ UNCHANGED inherentViolated
    ∨ ∧ ¬canSync
      ∧ PrintT("Exception: Leader wants to sync with follower while the condition doesn't allow.")
      ∧ inherentViolated' = TRUE
      ∧ UNCHANGED ⟨history, lastZxid, currentVote, commitIndex, synced, forwarding, ackIndex, commi

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) ≜
    ∧ IsLeader(i)
    ∧ PendingACKEPOCH(i, j)
    ∧ LET msg ≜ msgs[j][i][1]
          infoOk ≜ ∧ IsMyLearner(i, j)
                   ∧ acceptedEpoch[i] = msg.mepoch
          logOk  ≜ logOk represents whether leader is more up-to-date than follower
                   ∧ infoOk
                   ∧ ∨ currentEpoch[i] > msg.mlastEpoch
                     ∨ ∧ currentEpoch[i] = msg.mlastEpoch
                       ∧ ∨ lastZxid[i][1] > msg.mlastZxid[1]
                         ∨ ∧ lastZxid[i][1] = msg.mlastZxid[1]
                           ∧ lastZxid[i][2] ≥ msg.mlastZxid[2]
          replyOk ≜ ∧ infoOk
                    ∧ NullPoint ∈ ackeRecv[i]
      IN  ∧ infoOk
          ∧ ∨ ∧ replyOk
              ∧ SubRECOVERYSYNC(i, j)
              ∧ ackeRecv' = [ackeRecv EXCEPT ![i] = IF j ∉ ackeRecv[i] THEN ackeRecv[i] ∪ {j}
                                                                       ELSE ackeRecv[i]]
              ∧ Reply(i, j, [mtype      ↦ NEWLEADER,
                             mepoch     ↦ acceptedEpoch[i],
                             mlastZxid  ↦ lastZxid[i]])
              ∧ UNCHANGED ⟨state, currentEpoch, logicalClock, receiveVotes, outOfElection, recvQueue
                           leadingVoteSet, electionMsgs, idTable, zabState, leaderAddr, learners⟩
            ∨ ∧ ¬replyOk
              ∧ ∨ normal case
                  ∧ logOk
                  ∧ ackeRecv' = [ackeRecv EXCEPT ![i] = IF j ∉ ackeRecv[i] THEN ackeRecv[i] ∪ {j}
                                                                           ELSE ackeRecv[i]]
                  ∧ Discard(j, i)
                  ∧ UNCHANGED ⟨varsL, zabState, leaderAddr, learners, forwarding⟩
                ∨ go back to election since there exists follower more up-to-date than leader
                  ∧ ¬logOk
                  ∧ LeaderShutdown(i)
                  ∧ UNCHANGED ackeRecv
              ∧ UNCHANGED ⟨history, commitIndex, synced, forwarding, ackIndex,
                           committedCounter, proposalMsgsLog, inherentViolated⟩
    ∧ UNCHANGED ⟨acceptedEpoch, cepochRecv, ackldRecv, currentCounter,
                 tempVarsZ, cepochSent, epochLeader⟩
    ∧ UpdateRecorder(⟨"LeaderHandleACKEPOCH", i, j⟩)

LeaderTransitionToSynchronization(i) ≜
    ∧ IsLeader(i)
    ∧ zabState[i] = DISCOVERY
```

$\land$ $IsQuorum(ackeRecv[i])$
$\land$ $zabState'$ $= [zabState$ 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 $\langle state,\ lastZxid,\ acceptedEpoch,\ history,\ commitIndex,\ logicalClock,\ receiveVotes,\ outO$
$recvQueue,\ waitNotmsg,\ leadingVoteSet,\ learners,\ cepochRecv,\ ackldRecv,\ forwarding$
$committedCounter,\ tempMaxEpoch,\ followerVarsZ,\ proposalMsgsLog,$
$inherentViolated,\ msgVarsZ,\ idTable\rangle$
$\land$ $UpdateRecorder(\langle$"LeaderTransitionToSynchronization", $i\rangle)$

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.

$RECOVERYSYNC(i,\ j) \triangleq$
LET $canSync \triangleq \land IsLeader(i) \land zabState[i] \neq DISCOVERY \land IsMyLearner(i,\ j)$
$\land IsFollower(j) \land zabState[j] = SYNCHRONIZATION \land IsMyLeader(j,\ i)$
IN
$\land$ $canSync$
$\land$ $SubRECOVERYSYNC(i,\ j)$
$\land$ $Send(i,\ j,\ [mtype \mapsto NEWLEADER,$
$mepoch \mapsto acceptedEpoch[i],$
$mlastZxid \mapsto lastZxid[i]])$
$\land$ UNCHANGED $\langle state,\ zabState,\ acceptedEpoch,\ currentEpoch,\ logicalClock,\ receiveVotes,\ outOfElectio$
$leadingVoteSet,\ learners,\ cepochRecv,\ ackeRecv,\ ackldRecv,\ currentCounter,$
$tempVarsZ,\ cepochSent,\ leaderAddr,\ epochLeader,\ electionMsgs,\ idTable\rangle$
$\land$ $UpdateRecorder(\langle$"RECOVERYSYNC", $i,\ j\rangle)$

Follower receives *NEWLEADER*. The follower updates its epoch and history, and sends back *ACK*-LD to leader.

$FollowerHandleNEWLEADER(i,\ j) \triangleq$
$\land$ $IsFollower(i)$
$\land$ $PendingNEWLEADER(i,\ j)$
$\land$ LET $msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \land IsMyLeader(i,\ j)$
$\land acceptedEpoch[i] = msg.mepoch$
$correct \triangleq \land infoOk$
$\land zabState[i] = SYNCHRONIZATION$
$\land synced[i]$
$\land ZxidEqual(lastZxid[i],\ msg.mlastZxid)$
IN $\land$ $infoOk$

$\land\ currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch]$
$\land\ UpdateProposal(i,\ j,\ lastZxid[i],\ currentEpoch'[i])$
$\land\ \lor\ \land\ correct$
$\land\ Reply(i,\ j,\ [mtype \mapsto ACKLD,$
$mepoch \mapsto msg.mepoch])$
$\land$ UNCHANGED $inherentViolated$
$\lor\ \land\ \neg correct$
$\land\ PrintT(\text{“Exception: Follower receives NEWLEADER while it has not completed sync with l}$
$\land\ inherentViolated' = \text{TRUE}$
$\land\ Discard(j,\ i)$
$\land$ UNCHANGED $\langle state,\ lastZxid,\ zabState,\ acceptedEpoch,\ history,\ commitIndex,\ logicalClock,\ receiveV$
$outOfElection,\ recvQueue,\ waitNotmsg,\ leaderVarsZ,\ tempVarsZ,\ followerVarsZ,\ pro$
$epochLeader,\ electionMsgs,\ idTable\rangle$
$\land\ UpdateRecorder(\langle \text{“FollowerHandleNEWLEADER”},\ i,\ j\rangle)$

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

$LeaderHandleACKLD(i,\ j) \triangleq$
$\land\ IsLeader(i)$
$\land\ PendingACKLD(i,\ j)$
$\land$ LET $msg \triangleq msgs[j][i][1]$
$infoOk \triangleq\ \land\ acceptedEpoch[i] = msg.mepoch$
$\land\ IsMyLearner(i,\ j)$
$replyOk \triangleq\ \land\ infoOk$
$\land\ NullPoint \in ackldRecv[i]$
IN $\land\ infoOk$
$\land\ \lor$ leader has broadcast $UPTODATE$ – just reply
$\land\ replyOk$
$\land\ Reply(i,\ j,\ [mtype \mapsto UPTODATE,$
$mepoch \mapsto acceptedEpoch[i],$
$mcommit \mapsto commitIndex[i]])$
$\land\ committedCounter' = [committedCounter \text{ EXCEPT } ![i][j] = Maximum(\{commitIndex[i]$
$\lor$ leader still waits for a quorum's $ACKLD$ to broadcast $UPTODATE$
$\land\ \neg replyOk$
$\land\ Discard(j,\ i)$
$\land$ UNCHANGED $committedCounter$
$\land\ ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] = \text{IF } j \notin ackldRecv[i] \text{ THEN } ackldRecv[i] \cup \{j\}$
ELSE $ackldRecv[i]]$
$\land$ UNCHANGED $\langle serverVarsZ,\ electionVarsZ,\ leadingVoteSet,\ learners,\ cepochRecv,\ ackeRecv,\ forward$
$ackIndex,\ currentCounter,\ tempVarsZ,\ followerVarsZ,\ verifyVarsZ,\ electionMsgs,\ id'$
$\land\ UpdateRecorder(\langle \text{“LeaderHandleACKLD”},\ i,\ j\rangle)$

$LeaderTransitionToBroadcast(i) \triangleq$
$\land\ IsLeader(i)$
$\land\ zabState[i] = SYNCHRONIZATION$
$\land\ IsQuorum(ackldRecv[i])$

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

$FollowerHandleUPTODATE(i, j) \triangleq$
$\land\ IsFollower(i)$
$\land\ PendingUPTODATE(i, j)$
$\land\ \text{LET } msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \land\ IsMyLeader(i, j)$
$\land\ acceptedEpoch[i] = msg.mepoch$
$correct \triangleq \land\ infoOk$
$\land\ zabState[i] = SYNCHRONIZATION$
$\land\ currentEpoch[i] = msg.mepoch$
$\text{IN} \quad \land\ infoOk$
$\land\ \lor\ \land\ correct$
$\land\ commitIndex' = [commitIndex \text{ EXCEPT } ![i] = Maximum(\{commitIndex[i], msg.mcomm$
$\land\ zabState' = [zabState \text{ EXCEPT } ![i] = BROADCAST]$
$\land\ \text{UNCHANGED } inherentViolated$
$\lor\ \land\ \neg correct$
$\land\ PrintT(\text{"Exception: Follower receives UPTODATE while its ZabState is not SYNCHRONIZ}$
$\land\ inherentViolated' = \text{TRUE}$
$\land\ \text{UNCHANGED } \langle commitIndex, zabState\rangle$
$\land\ Discard(j, i)$
$\land\ \text{UNCHANGED } \langle state, currentEpoch, lastZxid, acceptedEpoch, history, electionVarsZ, leaderVarsZ, te$
$proposalMsgsLog, epochLeader, electionMsgs, idTable\rangle$
$\land\ UpdateRecorder(\langle \text{"FollowerHandleUPTODATE"}, i, j\rangle)$

Leader receives client request and broadcasts *PROPOSAL*.

*Note*1: 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.

*Note*2: To compress state space, we now choose to merge action client request and action leader broadcasts proposal into one action.

$ClientRequestAndLeaderBroadcastProposal(i, v) \triangleq$
$\land\ IsLeader(i)$
$\land\ zabState[i] = BROADCAST$

$\land\ currentCounter' = [currentCounter \text{ EXCEPT } ![i] = currentCounter[i] + 1]$
$\land$ LET $newTransaction \triangleq [epoch \mapsto acceptedEpoch[i],$
$counter \mapsto currentCounter'[i],$
$value \mapsto v]$
IN $\land\ history' = [history \text{ EXCEPT } ![i] = Append(history[i], newTransaction)]$
$\land\ lastZxid' = [lastZxid \text{ EXCEPT } ![i] = \langle acceptedEpoch[i], currentCounter'[i]\rangle]$
$\land\ ackIndex' = [ackIndex \text{ EXCEPT } ![i][i] = Len(history'[i])]$
$\land\ UpdateProposal(i, i, lastZxid'[i], currentEpoch[i])$
$\land\ Broadcast(i, [mtype \mapsto PROPOSAL,$
$mepoch \mapsto acceptedEpoch[i],$
$mproposal \mapsto newTransaction])$
$\land$ LET $m \triangleq [msource \mapsto i, mepoch \mapsto acceptedEpoch[i], mtype \mapsto PROPOSAL, mproposal \mapsto$
IN $proposalMsgsLog' = proposalMsgsLog \cup \{m\}$
$\land$ UNCHANGED $\langle state, currentEpoch, zabState, acceptedEpoch, commitIndex, logicalClock, receiveVotes$
$recvQueue, waitNotmsg, leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv,$
$committedCounter, tempVarsZ, followerVarsZ, epochLeader, inherentViolated,$
$electionMsgs, idTable\rangle$
$\land\ UpdateRecorder(\langle$"ClientRequestAndLeaderBroadcastProposal", $i, v\rangle)$

$LeaderBroadcastProposal(i) \triangleq$
$\land\ IsLeader(i)$
$\land\ zabState[i] = BROADCAST$
$\land\ sendCounter[i] < currentCounter[i]$
$\land$ LET $toBeSentCounter \triangleq sendCounter[i] + 1$
$toBeSentIndex \triangleq Len(initialHistory[i]) + toBeSentCounter$
$toBeSentEntry \triangleq history[i][toBeSentIndex]$
IN $\land\ Broadcast(i, [mtype \mapsto PROPOSAL,$
$mepoch \mapsto acceptedEpoch[i],$
$mproposal \mapsto toBeSentEntry])$
$\land\ sendCounter' = [sendCounter \text{ EXCEPT } ![i] = toBeSentCounter]$
$\land$ LET $m \triangleq [msource \mapsto i, mepoch \mapsto acceptedEpoch[i], mtype \mapsto PROPOSAL,$
$mproposal \mapsto toBeSentEntry]$
IN $proposalMsgsLog' = proposalMsgsLog \cup \{m\}$
$\land\ UpdateRecorder(\langle$"LeaderBroadcastProposal", $i\rangle)$
$\land$UNCHANGED $\langle serverVarsZ, electionVarsZ, leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv, forwarding,$
$ackIndex, currentCounter, committedCounter, tempVarsZ, followerVarsZ, epochLeader,$
$inherentViolated, electionMsgs, idTable\rangle$

Follower accepts proposal and append it to history.

$FollowerHandlePROPOSAL(i, j) \triangleq$
$\land\ IsFollower(i)$
$\land\ PendingPROPOSAL(i, j)$
$\land$ LET $msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \land\ IsMyLeader(i, j)$
$\land\ acceptedEpoch[i] = msg.mepoch$
$correct \triangleq \land\ infoOk$

```
                    ∧ zabState[i] ≠ DISCOVERY
                    ∧ synced[i]
        logOk  ≜    the first PROPOSAL in this epoch
                    ∨ ∧ msg.mproposal.counter = 1
                      ∧ ∨ Len(history[i]) = 0
                        ∨ ∧ Len(history[i]) > 0
                          ∧ history[i][Len(history[i])].epoch < msg.mepoch
                    not the first PROPOSAL in this epoch
                    ∨ ∧ msg.mproposal.counter > 1
                      ∧ Len(history[i]) > 0
                      ∧ history[i][Len(history[i])].epoch = msg.mepoch
                      ∧ history[i][Len(history[i])].counter = msg.mproposal.counter − 1
    IN  ∧ infoOk
        ∧ ∨ ∧ correct
          ∧ ∨ ∧ logOk
            ∧ history′  = [history  EXCEPT ![i] = Append(history[i], msg.mproposal)]
            ∧ lastZxid′ = [lastZxid EXCEPT ![i] = ⟨msg.mepoch, msg.mproposal.counter⟩]
            ∧ UpdateProposal(i, j, lastZxid′[i], currentEpoch[i])
            ∧ Reply(i, j, [mtype  ↦ ACK,
                           mepoch ↦ acceptedEpoch[i],
                           mzxid  ↦ ⟨msg.mepoch, msg.mproposal.counter⟩])
            ∧ UNCHANGED inherentViolated
          ∨ ∧ ¬logOk
            ∧ PrintT("Exception: Follower receives PROPOSAL while the transaction is not the nex
            ∧ inherentViolated′ = TRUE
            ∧ Discard(j, i)
            ∧ UNCHANGED ⟨history, lastZxid, currentVote⟩
        ∨ ∧ ¬correct
          ∧ PrintT("Exception: Follower receives PROPOSAL while it has not completed sync with lea
          ∧ inherentViolated′ = TRUE
          ∧ Discard(j, i)
          ∧ UNCHANGED ⟨history, lastZxid, currentVote⟩
    ∧ UNCHANGED ⟨state, currentEpoch, zabState, acceptedEpoch, commitIndex, logicalClock, receiveVotes
                 outOfElection, recvQueue, waitNotmsg, leaderVarsZ, tempVarsZ, followerVarsZ, pro
                 epochLeader, electionMsgs, idTable⟩
    ∧ UpdateRecorder(⟨"FollowerHandlePROPOSAL", i, j⟩)

Create a commit packet and send it to all the members of the quorum.
LeaderCommit(s, source, index, zxid) ≜
    ∧ commitIndex′ = [commitIndex EXCEPT ![s] = index]
    ∧ DiscardAndBroadcast(s, source, [mtype  ↦ COMMIT,
                                      mepoch ↦ acceptedEpoch[s],
                                      mzxid  ↦ ⟨zxid.epoch, zxid.counter⟩])

return true if committed, otherwise false.
```

```
LeaderTryToCommit(s, zxid, follower) ≜
    LET pindex ≜ Len(initialHistory[s]) + zxid.counter
        Only when all proposals before zxid all committed, this proposal can be permitted to be committed.
        allProposalsBeforeCommitted ≜ ∨ zxid.counter = 1
                                      ∨ ∧ zxid.counter > 1
                                        ∧ commitIndex[s] ≥ pindex − 1
        In order to be committed, a proposal must be accepted by a quorum.
        agreeSet ≜ {s} ∪ {k ∈ (Server \ {s}) : ackIndex′[s][k] ≥ pindex}
        hasAllQuorums ≜ IsQuorum(agreeSet)
        Commit proposals in order.
        ordered ≜ commitIndex[s] + 1 = pindex
    IN  ∨ ∧ ∨ ¬allProposalsBeforeCommitted
          ∨ ¬hasAllQuorums
        ∧ Discard(follower, s)
        ∧ UNCHANGED ⟨inherentViolated, commitIndex⟩
      ∨ ∧ allProposalsBeforeCommitted
        ∧ hasAllQuorums
        ∧ ∨ ∧ ¬ordered
            ∧ PrintT("Exception: Commiting zxid " ∘ zxid ∘ "not first.")
            ∧ inherentViolated′ = TRUE
          ∨ ∧ ordered
            ∧ UNCHANGED inherentViolated
        ∧ LeaderCommit(s, follower, pindex, zxid)

Keep a count of acks that are received by the leader for a particular
proposal, and commit the proposal.
LeaderProcessACK(i, j) ≜
    ∧ IsLeader(i)
    ∧ PendingACK(i, j)
    ∧ LET msg ≜ msgs[j][i][1]
          infoOk ≜ ∧ j ∈ forwarding[i]
                   ∧ acceptedEpoch[i] = msg.mepoch
          correct ≜ ∧ infoOk
                    ∧ zabState[i] = BROADCAST
                    ∧ currentCounter[i]     ≥ msg.mzxid[2]
          noOutstanding ≜ commitIndex[i] = Len(history[i])
                          outstandingProposals: proposals in history[commitIndex + 1 : Len(history)]
          hasCommitted ≜ commitIndex[i] ≥ Len(initialHistory[i]) + msg.mzxid[2]
                         namely, lastCommitted ≥ zxid
          logOk ≜ ∧ infoOk
                  ∧ ackIndex[i][j] + 1 = Len(initialHistory[i]) + msg.mzxid[2]
                    everytime, ackIndex should just increase by 1
      IN  ∧ infoOk
          ∧ ∨ ∧ correct
              ∧ logOk
```

$\land \lor \land \lor noOutstanding$
$\lor hasCommitted$
$\land PrintT(\text{“Note: outstanding is 0 / proposal has already been committed.”})$
$\land Discard(j,\ i)$
$\land \text{UNCHANGED } \langle ackIndex,\ commitIndex,\ inherentViolated\rangle$
$\lor \land \neg noOutstanding$
$\land \neg hasCommitted$
$\land ackIndex' = [ackIndex \text{ EXCEPT } ![i][j] = Len(initialHistory[i]) + msg.mzxid[2]]$
$\land LeaderTryToCommit(i,\ ToZxid(msg.mzxid),\ j)$
$\lor \land \lor \neg correct$
$\lor \neg logOk$
$\land PrintT(\text{“Exception: ackIndex doesn't increase monotonically.”})$
$\land inherentViolated' = \text{TRUE}$
$\land Discard(j,\ i)$
$\land \text{UNCHANGED } \langle ackIndex,\ commitIndex\rangle$
$\land \text{UNCHANGED } \langle state,\ currentEpoch,\ lastZxid,\ zabState,\ acceptedEpoch,\ history,\ electionVarsZ,$
$leadingVoteSet,\ learners,\ cepochRecv,\ ackeRecv,\ ackldRecv,$
$forwarding,\ currentCounter,\ committedCounter,\ tempVarsZ,$
$followerVarsZ,\ proposalMsgsLog,\ epochLeader,\ electionMsgs,\ idTable\rangle$
$\land UpdateRecorder(\langle \text{“LeaderProcessACK”},\ i,\ j\rangle)$

$LeaderAdvanceCommit(i) \triangleq$
$\land IsLeader(i)$
$\land zabState[i] = BROADCAST$
$\land commitIndex[i] < Len(history[i])$
$\land \text{LET } Agree(index) \triangleq \{i\} \cup \{k \in (Server \setminus \{i\}) : ackIndex[i][k] \geq index\}$
$agreeIndexes \triangleq \{index \in (commitIndex[i] + 1) \ldots Len(history[i]) :$
$Agree(index) \in Quorums\}$
$newCommitIndex \triangleq \text{IF } agreeIndexes \neq \{\} \text{ THEN } Maximum(agreeIndexes)$
$\text{ELSE } commitIndex[i]$
$\text{IN } commitIndex' = [commitIndex \text{ EXCEPT } ![i] = newCommitIndex]$
$\land UpdateRecorder(\langle \text{“}LeaderAdvanceCommit\text{”},\ i\rangle)$
$\land \text{UNCHANGED } \langle state,\ currentEpoch,\ lastZxid,\ zabState,\ acceptedEpoch,\ history,\ electionVarsZ,\ leaderVarsZ,$
$tempVarsZ,\ followerVarsZ,\ verifyVarsZ,\ msgVarsZ,\ idTable\rangle$

$LeaderBroadcastCommit(i) \triangleq$
$\land IsLeader(i)$
$\land zabState[i] = BROADCAST$
$\land committedIndex[i] < commitIndex[i]$
$\land Len(initialHistory[i]) + sendCounter[i] > committedIndex[i]$
$\land \text{LET } newCommittedIndex \triangleq committedIndex[i] + 1$
$\text{IN } \land Broadcast(i,\ [mtype \mapsto COMMIT,$
$mepoch \mapsto acceptedEpoch[i],$
$mzxid \mapsto \langle history[i][newCommittedIndex].epoch,\ history[i][newCommittedIndex].counter\rangle])$
$\land committedIndex' = [committedIndex \text{ EXCEPT } ![i] = committedIndex[i] + 1]$
$\land committedCounter' = [committedCounter \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } \land$
$v \in forwarding[i]$

$\wedge \quad committedCounter[i][v] \quad <$
$history[i][newCommittedIndex].counter$

THEN $history[i][newCommittedIndex].counter$

ELSE $committedCounter[i][v]]$
$\wedge\, UpdateRecorder(\langle$"$LeaderBroadcastCommit$", $i\rangle)$
$\wedge$UNCHANGED $\langle serverVarsZ,\ electionVarsZ,\ leadingVoteSet,\ learners,\ cepochRecv,\ ackeRecv,\ ackldRecv,$

$forwarding,\ ackIndex,\ currentCounter,\ sendCounter,\ tempVarsZ,\ followerVarsZ,\ verifyVarsZ,$

$electionMsgs,\ idTable\rangle$

Follower receives $COMMIT$ and commits transaction.

$FollowerHandleCOMMIT(i, j) \triangleq$
$\wedge\, IsFollower(i)$
$\wedge\, PendingCOMMIT(i, j)$
$\wedge$ LET $msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \wedge\, IsMyLeader(i, j)$
$\wedge\, acceptedEpoch[i] = msg.mepoch$
$correct \triangleq \wedge\, infoOk$
$\wedge\, zabState[i] \neq DISCOVERY$
$\wedge\, synced[i]$
$mindex \triangleq$ IF $Len(history[i]) = 0$ THEN -1
ELSE IF $\exists\, idx \in 1\,..\,Len(history[i]) : PZxidEqual(history[i][idx],\ msg.mzxid)$
THEN CHOOSE $idx \in 1\,..\,Len(history[i]) : PZxidEqual(history[i][idx],\ msg.m$
ELSE -1
$logOk \triangleq mindex > 0$
$latest \triangleq commitIndex[i] + 1 = mindex$
IN $\wedge\, infoOk$
$\wedge\ \vee\ \wedge\, correct$
$\wedge\ \vee\ \wedge\, logOk$
$\wedge\ \vee\ \wedge\, latest$
$\wedge\, commitIndex' = [commitIndex$ EXCEPT $![i] = commitIndex[i] + 1]$
$\wedge$ UNCHANGED $inherentViolated$
$\vee\ \wedge \neg latest$
$\wedge\, PrintT($"Note: Follower receives COMMIT while the index is not the next comm
$\wedge\, inherentViolated' =$ TRUE
$\wedge$ UNCHANGED $commitIndex$
$\vee\ \wedge \neg logOk$
$\wedge\, PrintT($"Exception: Follower receives COMMIT while the transaction has not been sa
$\wedge\, inherentViolated' =$ TRUE
$\wedge$ UNCHANGED $commitIndex$
$\vee\ \wedge \neg correct$
$\wedge\, PrintT($"Exception: Follower receives COMMIT while it has not completed sync with leade
$\wedge\, inherentViolated' =$ TRUE
$\wedge$ UNCHANGED $commitIndex$

$\land$ $Discard(j,\ i)$
$\land$ UNCHANGED $\langle state,\ currentEpoch,\ lastZxid,\ zabState,\ acceptedEpoch,\ history,\ electionVarsZ,\ leaderTempVarsZ,\ followerVarsZ,\ proposalMsgsLog,\ epochLeader,\ electionMsgs,\ idTable\rangle$
$\land$ $UpdateRecorder(\langle$"FollowerHandleCOMMIT", $i,\ j\rangle)$

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

$FilterNonexistentMessage(i) \triangleq$
$\land \exists j \in Server \setminus \{i\} :$ $\land msgs[j][i] \neq \langle\rangle$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN
$\lor \land IsLeader(i)$
$\land$ LET $infoOk \triangleq \land j \in learners[i]$
$\land acceptedEpoch[i] = msg.mepoch$
IN
$\lor msg.mtype = LEADERINFO$
$\lor msg.mtype = NEWLEADER$
$\lor msg.mtype = UPTODATE$
$\lor msg.mtype = PROPOSAL$
$\lor msg.mtype = COMMIT$
$\lor \land j \notin learners[i]$
$\land msg.mtype = FOLLOWERINFO$
$\lor \land \neg infoOk$
$\land \lor msg.mtype = ACKEPOCH$
$\lor msg.mtype = ACKLD$
$\lor msg.mtype = ACK$
$\lor \land IsFollower(i)$
$\land$ LET $infoOk \triangleq \land j = leaderAddr[i]$
$\land acceptedEpoch[i] = msg.mepoch$
IN
$\lor msg.mtype = FOLLOWERINFO$
$\lor msg.mtype = ACKEPOCH$
$\lor msg.mtype = ACKLD$
$\lor msg.mtype = ACK$
$\lor \land j \neq leaderAddr[i]$
$\land msg.mtype = LEADERINFO$
$\lor \land \neg infoOk$
$\land \lor msg.mtype = NEWLEADER$
$\lor msg.mtype = UPTODATE$
$\lor msg.mtype = PROPOSAL$
$\lor msg.mtype = COMMIT$
$\lor IsLooking(i)$
$\land Discard(j,\ i)$
$\land inherentViolated' =$ TRUE
$\land UnchangeRecorder$

$\wedge$ UNCHANGED $\langle serverVarsZ,\ electionVarsZ,\ leaderVarsZ,\ tempVarsZ,\ followerVarsZ,\ proposalMsgsL$
$epochLeader,\ electionMsgs,\ idTable\rangle$

Defines how the variables may transition.

$NextZ \triangleq$

FLE modlue

$\vee \exists\, i, j \in Server : FLEReceiveNotmsg(i, j)$
$\vee \exists\, i \in Server : \quad FLENotmsgTimeout(i)$
$\vee \exists\, i \in Server : \quad FLEHandleNotmsg(i)$
$\vee \exists\, i \in Server : \quad FLEWaitNewNotmsg(i)$
$\vee \exists\, i \in Server : \quad FLEWaitNewNotmsgEnd(i)$

Some conditions like failure, network delay

$\vee \exists\, i \in Server : \quad FollowerTimout(i)$
$\vee \exists\, i \in Server : \quad LeaderTimeout(i)$
$\vee \exists\, i, j \in Server : Timeout(i, j)$

Zab module - Discovery and Synchronization part

$\vee \exists\, i, j \in Server : EstablishConnection(i, j)$
$\vee \exists\, i \in Server : \quad FollowerSendFOLLOWERINFO(i)$
$\vee \exists\, i, j \in Server : LeaderHandleFOLLOWERINFO(i, j)$
$\vee \exists\, i \in Server : \quad LeaderBroadcastLEADERINFO(i)$
$\vee \exists\, i, j \in Server : FollowerHandleLEADERINFO(i, j)$
$\vee \exists\, i, j \in Server : LeaderHandleACKEPOCH(i, j)$
$\vee \exists\, i \in Server : \quad LeaderTransitionToSynchronization(i)$
$\vee \exists\, i, j \in Server : RECOVERYSYNC(i, j)$
$\vee \exists\, i, j \in Server : FollowerHandleNEWLEADER(i, j)$
$\vee \exists\, i, j \in Server : LeaderHandleACKLD(i, j)$
$\vee \exists\, i \in Server : \quad LeaderTransitionToBroadcast(i)$
$\vee \exists\, i, j \in Server : FollowerHandleUPTODATE(i, j)$

Zab module $-$ *Broadcast* part

$\vee \exists\, i \in Server,\ v \in Value : ClientRequestAndLeaderBroadcastProposal(i,\ v)$
$\vee \exists\, i \in Server,\ v \in Value : ClientRequest(i, v)$
$\vee \exists\, i \in Server : LeaderBroadcastProposal(i)$
$\vee \exists\, i, j \in Server : FollowerHandlePROPOSAL(i, j)$
$\vee \exists\, i, j \in Server : LeaderProcessACK(i, j)$
$\vee \exists\, i \in Server : LeaderAdvanceCommit(i)$
$\vee \exists\, i \in Server : LeaderBroadcastCommit(i)$
$\vee \exists\, i, j \in Server : FollowerHandleCOMMIT(i, j)$

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

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

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

Define safety properties of *Zab* 1.0 protocol.

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

There is most one established leader for a certain epoch.

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

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

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

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

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

$$
\begin{aligned}
Integrity \triangleq\ & \forall\, i \in Server : \\
& \land IsFollower(i) \\
& \land commitIndex[i] > 0 \\
& \Rightarrow \forall\, index \in 1 \,..\, commitIndex[i] : \exists\, msg \in proposalMsgsLog : \\
& \qquad \lor\ \land\ msg.mtype = PROPOSAL \\
& \qquad\quad \land\ TransactionEqual(msg.mproposal,\ history[i][index]) \\
& \qquad \lor\ \land\ msg.mtype = \text{“RECOVERYSYNC”} \\
& \qquad\quad \land \exists\, tindex \quad \in 1 \,..\, Len(msg.mproposals) : TransactionEqual(msg.mproposals[tindex],\ h
\end{aligned}
$$

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

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

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

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

$\land \exists\, indexj2 \in 1 \,..\, commitIndex[j] :$
$\land\ TransactionEqual(history[i][indexi2],\ history[j][indexj2])$
$\land \exists\, indexj1 \in 1 \,..\, (indexj2 - 1) : TransactionEqual(history[i][in$

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

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

Global primary order: A follower f delivers both a with epoch e and b with epoch e', and $e < e'$, then f must deliver a before b.

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

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.

$PrimaryIntegrity \triangleq \forall\, i,\ j \in Server :\ \land IsLeader(i)$
$\land IsFollower(j)$
$\land commitIndex[j] \geq 1$
$\Rightarrow \forall\, index \in 1 \,..\, commitIndex[j] :\ \lor history[j][index].epoch \geq currentEpoch[i]$
$\lor\ \land history[j][index].epoch < currentEpoch[i]$
$\land \exists\, idx \in 1 \,..\, commitIndex[i] : TransactionEqua$

\ * Modification History
\ * Last modified Sat *Oct* 09 11:05:52 *CST* 2021 by Dell
\ * Created *Tue Jun* 29 22:13:02 *CST* 2021 by Dell