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

$leaderVarsZ \triangleq \langle leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv, forwarding,$
$\quad ackIndex, currentCounter \rangle$ 8 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

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

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

$Zxid \triangleq$
$\quad Seq(Nat)$
$\quad$ ∪ [epoch: *Nat*, counter: *Nat*]

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

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

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

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

$Vote \triangleq$

$[proposedLeader : ServersIncNullPoint,$
$proposedZxid : Zxid,$
$proposedEpoch : Nat]$

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

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

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

```
    $\land\ currentEpoch \in [Server \to Nat]$
    $\land\ logicalClock \ \ \in [Server \to Nat]$
```

Return the maximum value from the set S

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

Return the minimum value from the set S

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

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

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

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

$RecorderIncTimeout \triangleq RecorderIncHelper($"nTimeout"$)$
$RecorderGetTimeout \triangleq RecorderGetHelper($"nTimeout"$)$
$RecorderSetTransactionNum(pc) \triangleq ($"nTransaction" $:>$
                                    IF $pc[1] =$ "ClientRequestAndLeaderBroadcastProposal" THEN
                                       LET $s \triangleq$ CHOOSE $i \in Server$ :
                                              $\forall j \in Server : Len(history'[i]) \geq Len(history'[j])$
                                          IN  $Len(history'[s])$
                                        ELSE  $recorder[\text{"nTransaction"}])$
$RecorderSetMaxEpoch(pc)$          $\triangleq$ $(\text{"maxEpoch"} :>$
                                        IF $pc[1] = \text{"LeaderBroadcastLEADERINFO"}$ THEN
                                           LET $s \triangleq$ CHOOSE $i \in Server$ :
                                              $\forall j \in Server : acceptedEpoch'[i] \geq acceptedEpoch'[j]$
                                           IN  $acceptedEpoch'[s]$
                                        ELSE  $recorder[\text{"maxEpoch"}])$
$RecorderSetPc(pc)$       $\triangleq$ $(\text{"pc"} :> pc)$
$RecorderSetFailure(pc)$  $\triangleq$ CASE $pc[1] = \text{"Timeout"}$          $\rightarrow RecorderIncTimeout$
                             $\Box$    $pc[1] = \text{"LeaderTimeout"}$    $\rightarrow RecorderIncTimeout$
                             $\Box$    $pc[1] = \text{"FollowerTimeout"}$  $\rightarrow RecorderIncTimeout$
                             $\Box$    OTHER                           $\rightarrow RecorderGetTimeout$

$UpdateRecorder(pc)$ $\triangleq$ $recorder' = RecorderSetFailure(pc)$     $@@\ RecorderSetTransactionNum(pc)$
                          $@@\ RecorderSetMaxEpoch(pc)$  $@@\ RecorderSetPc(pc)\ @@\ recorder$
$UnchangeRecorder$ $\triangleq$ UNCHANGED $recorder$

$CheckParameterHelper(n, p, Comp(\_, \_))$ $\triangleq$ IF $p \in$ DOMAIN $Parameters$
                                             THEN $Comp(n, Parameters[p])$
                                             ELSE TRUE
$CheckParameterLimit(n, p)$ $\triangleq$ $CheckParameterHelper(n, p,$ LAMBDA $i, j : i < j)$

$CheckTimeout$         $\triangleq$ $CheckParameterLimit(recorder.nTimeout,$      $\text{"MaxTimeoutFailures"})$
$CheckTransactionNum$  $\triangleq$ $CheckParameterLimit(recorder.nTransaction,$  $\text{"MaxTransactionNum"})$
$CheckEpoch$           $\triangleq$ $CheckParameterLimit(recorder.maxEpoch,$      $\text{"MaxEpoch"})$

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

```
Actions about network
$PendingFOLLOWERINFO(i, j)$ $\triangleq$ $\land\ msgs[j][i] \neq \langle\rangle$
                                 $\land\ msgs[j][i][1].mtype = FOLLOWERINFO$
$PendingLEADERINFO(i, j)$   $\triangleq$ $\land\ msgs[j][i] \neq \langle\rangle$
                                 $\land\ msgs[j][i][1].mtype = LEADERINFO$
$PendingACKEPOCH(i, j)$     $\triangleq$ $\land\ msgs[j][i] \neq \langle\rangle$
                                 $\land\ msgs[j][i][1].mtype = ACKEPOCH$
$PendingNEWLEADER(i, j)$    $\triangleq$ $\land\ msgs[j][i] \neq \langle\rangle$
                                 $\land\ msgs[j][i][1].mtype = NEWLEADER$
$PendingACKLD(i, j)$        $\triangleq$ $\land\ msgs[j][i] \neq \langle\rangle$
                                 $\land\ msgs[j][i][1].mtype = ACKLD$
$PendingUPTODATE(i, j)$     $\triangleq$ $\land\ msgs[j][i] \neq \langle\rangle$
                                 $\land\ msgs[j][i][1].mtype = UPTODATE$
$PendingPROPOSAL(i, j)$     $\triangleq$ $\land\ msgs[j][i] \neq \langle\rangle$
                                 $\land\ msgs[j][i][1].mtype = PROPOSAL$
$PendingACK(i, j)$          $\triangleq$ $\land\ msgs[j][i] \neq \langle\rangle$
```

```
                                $\land\ msgs[j][i][1].mtype = ACK$
$PendingCOMMIT(i, j) \triangleq$   $\land\ msgs[j][i] \neq \langle\rangle$
                                $\land\ msgs[j][i][1].mtype = COMMIT$
```

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*.

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

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

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

```
$InitServerVarsZ \triangleq$ $\land\ InitServerVars$
                     $\land\ zabState \quad = [s \in Server \mapsto ELECTION]$
                     $\land\ acceptedEpoch = [s \in Server \mapsto 0]$
                     $\land\ history \quad = [s \in Server \mapsto \langle\rangle]$
                     $\land\ commitIndex \ = [s \in Server \mapsto 0]$

$InitLeaderVarsZ \triangleq$ $\land\ InitLeaderVars$
                     $\land\ learners \quad = [s \in Server \mapsto \{\}]$
```

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

$InitElectionVarsZ \triangleq InitElectionVars$

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

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

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

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

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

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

$ZabTurnToLeading(i) \triangleq$

$$\begin{aligned}
&\land\ zabState' &&= [zabState \ \text{EXCEPT}\ ![i] = DISCOVERY]\\
&\land\ learners' &&= [learners \ \text{EXCEPT}\ ![i] = \{i\}]\\
&\land\ cepochRecv' &&= [cepochRecv \ \text{EXCEPT}\ ![i] = \{i\}]\\
&\land\ ackeRecv' &&= [ackeRecv \ \text{EXCEPT}\ ![i] = \{i\}]\\
&\land\ ackldRecv' &&= [ackldRecv \ \text{EXCEPT}\ ![i] = \{i\}]\\
&\land\ forwarding' &&= [forwarding \ \text{EXCEPT}\ ![i] = \{\}]
\end{aligned}$$

```
          ∧ ackIndex'        = [ackIndex    EXCEPT ![i] = [v ∈ Server ↦ IF v = i THEN Len(history[i])
                                                                          ELSE 0]]
          ∧ currentCounter'  = [currentCounter  EXCEPT ![i] = 0]
          ∧ commitIndex'     = [commitIndex     EXCEPT ![i] = 0]
          ∧ sendCounter'     = [sendCounter     EXCEPT ![i] = 0]
          ∧ committedIndex'  = [committedIndex  EXCEPT ![i] = 0]
          ∧ initialHistory'  = [initialHistory EXCEPT ![i]    = history[i]]
          ∧ tempMaxEpoch'    = [tempMaxEpoch   EXCEPT ![i] = acceptedEpoch[i]]

ZabTurnToFollowing(i) ≜
          ∧ zabState'    = [zabState    EXCEPT ![i]  = DISCOVERY]
          ∧ cepochSent'  = [cepochSent EXCEPT ![i] = FALSE]
          ∧ synced'      = [synced      EXCEPT ![i]  = FALSE]
          ∧ commitIndex' = [commitIndex       EXCEPT ![i] = 0]
```

Fast Leader Election

```
FLEReceiveNotmsg(i, j) ≜
          ∧ ReceiveNotmsg(i, j)
          ∧ UNCHANGED ⟨zabState, acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackl
                       forwarding, ackIndex, currentCounter, tempVarsZ, followerVarsZ, verifyVarsZ, msgs
          ∧ UpdateRecorder(⟨"FLEReceiveNotmsg", i, j⟩)

FLENotmsgTimeout(i) ≜
          ∧ NotmsgTimeout(i)
          ∧ UNCHANGED ⟨zabState, acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackl
                       forwarding, ackIndex, currentCounter, tempVarsZ, followerVarsZ, verifyVarsZ, msgs
          ∧ UpdateRecorder(⟨"FLENotmsgTimeout", i⟩)

FLEHandleNotmsg(i) ≜
          ∧ HandleNotmsg(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
             ∨ ∧ newState = LOOKING
               ∧ UNCHANGED ⟨zabState, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex,
                            currentCounter, commitIndex, tempVarsZ, cepochSent, synced⟩
          ∧ UNCHANGED ⟨acceptedEpoch, history, leaderAddr, verifyVarsZ, msgs⟩
          ∧ UpdateRecorder(⟨"FLEHandleNotmsg", i⟩)
```

On the premise that $ReceiveVotes.HasQuorums = \text{TRUE}$, corresponding to logic in line $1050 - 1055$ in *LFE.java*.

```
FLEWaitNewNotmsg(i) ≜
```

```
        $\land$ $WaitNewNotmsg(i)$
        $\land$ UNCHANGED $\langle zabState, acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackld$
                        $forwarding, ackIndex, currentCounter, tempVarsZ, followerVarsZ, verifyVarsZ, msgs$
        $\land$ $UpdateRecorder(\langle$“FLEWaitNewNotmsg”, $i\rangle)$
```

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

```
$FLEWaitNewNotmsgEnd(i) \triangleq$
        $\land$ $WaitNewNotmsgEnd(i)$
        $\land$ 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 $\langle learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex, currentCounte$
          $\lor$ $\land$ $newState = LOOKING$
             $\land$ $PrintT($“Note: New state is LOOKING in FLEWaitNewNotmsgEnd, which should not happen.”$)$
             $\land$ UNCHANGED $\langle zabState, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackIndex,$
                              $currentCounter, commitIndex, tempVarsZ, cepochSent, synced\rangle$
        $\land$ UNCHANGED $\langle acceptedEpoch, history, leaderAddr, verifyVarsZ, msgs\rangle$
        $\land$ $UpdateRecorder(\langle$“FLEWaitNewNotmsgEnd”, $i\rangle)$
```

Describe how a server transitions from $LEADING/FOLLOWING$ to $LOOKING$.

```
$FollowerShutdown(i) \triangleq$
        $\land$ $ZabTimeout(i)$
        $\land$ $zabState'$   $= [zabState$   EXCEPT $![i] = ELECTION]$
        $\land$ $leaderAddr' = [leaderAddr$ EXCEPT $![i] = NullPoint]$

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

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

```
$FollowerTimeout(i) \triangleq$
        $\land$ $CheckTimeout$   test restrictions of timeout 1
        $\land$ $IsFollower(i)$
        $\land$ $HasNoLeader(i)$
        $\land$ $FollowerShutdown(i)$
        $\land$ $msgs' = [s \in Server \mapsto [v \in Server \mapsto \text{IF } v = i \text{ THEN } \langle\rangle \text{ ELSE } msgs[s][v]]]$
        $\land$ $\text{UNCHANGED } \langle acceptedEpoch, history, commitIndex, learners, cepochRecv, ackeRecv, ackldRecv,$
                    $forwarding, ackIndex, currentCounter, tempVarsZ, cepochSent, synced, verifyVarsZ\rangle$
        $\land$ $UpdateRecorder(\langle$"FollowerTimeout"$, i\rangle)$

$LeaderTimeout(i) \triangleq$
        $\land$ $CheckTimeout$   test restrictions of timeout 2
        $\land$ $IsLeader(i)$
        $\land$ $\neg IsQuorum(learners[i])$
        $\land$ $LeaderShutdown(i)$
        $\land$ $\text{UNCHANGED } \langle acceptedEpoch, history, commitIndex, cepochRecv, ackeRecv, ackldRecv, ackIndex,$
                    $currentCounter, tempVarsZ, cepochSent, synced, verifyVarsZ\rangle$
        $\land$ $UpdateRecorder(\langle$"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: '$connectToLeader(addr, hostname)$'
        $\land$ $\text{UNCHANGED } \langle serverVarsZ, electionVarsZ, leadingVoteSet, cepochRecv, ackeRecv, ackldRecv, forwa$
                    $ackIndex, currentCounter, tempVarsZ, cepochSent, synced, verifyVarsZ, msgVarsZ,$
        $\land$ $UpdateRecorder(\langle$"EstablishConnection"$, i, j\rangle)$
```

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

```
$Timeout(i, j) \triangleq$
        $\land$ $CheckTimeout$   test restrictions of timeout 3
        $\land$ $IsLeader(i)$
        $\land$ $IsFollower(j)$
        $\land$ $IsMyLearner(i, j)$
        $\land$ $IsMyLeader(j, i)$
        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 $j$.
        ∧ Clean(i, j)
        ∧ UNCHANGED ⟨acceptedEpoch, history, commitIndex, currentCounter,
                     tempVarsZ, cepochSent, synced, verifyVarsZ⟩
        ∧ UpdateRecorder(⟨"Timeout", i, j⟩)
├──────────────────────────────────────────────────────────────────────┤
 Follower sends f.p to leader via FOLLOWERINFO(CEPOCH).
FollowerSendFOLLOWERINFO(i) ≜
      ∧ IsFollower(i)
      ∧ zabState[i] = DISCOVERY
      ∧ HasLeader(i)
      ∧ ¬cepochSent[i]
      ∧ Send(i, leaderAddr[i], [mtype   ↦ FOLLOWERINFO,
                                mepoch ↦ acceptedEpoch[i]])
      ∧ cepochSent′ = [cepochSent EXCEPT ![i] = TRUE]
      ∧ UNCHANGED ⟨serverVarsZ, leaderVarsZ, electionVarsZ, tempVarsZ, leaderAddr, synced,
                   verifyVarsZ, electionMsgs, idTable⟩
      ∧ UpdateRecorder(⟨"FollowerSendFOLLOWERINFO", i⟩)

 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) ≜
      ∧ IsLeader(i)
      ∧ PendingFOLLOWERINFO(i, j)
      ∧ LET msg ≜ msgs[j][i][1]
        IN  ∨ 1. has not broadcast LEADERINFO − modify tempMaxEpoch
              ∧ NullPoint ∉ cepochRecv[i]
              ∧ LET newEpoch ≜ Maximum({tempMaxEpoch[i], msg.mepoch})
                IN  tempMaxEpoch′ = [tempMaxEpoch EXCEPT ![i] = newEpoch]
              ∧ Discard(j, i)
            ∨  2. has broadcast LEADERINFO − no need to handle the msg, just send LEADERINFO to correspondin
              ∧ NullPoint ∈ cepochRecv[i]
              ∧ Reply(i, j, [mtype   ↦ LEADERINFO,
                             mepoch ↦ acceptedEpoch[i]])
              ∧ UNCHANGED tempMaxEpoch
      ∧ cepochRecv′ = [cepochRecv EXCEPT ![i] = IF j ∈ cepochRecv[i] THEN cepochRecv[i]
                                                                     ELSE cepochRecv[i] ∪ {j}]
      ∧ UNCHANGED ⟨serverVarsZ, followerVarsZ, electionVarsZ, initialHistory, leadingVoteSet, learners,
                   ackeRecv, ackldRecv, forwarding, ackIndex, currentCounter, verifyVarsZ, electionMs
      ∧ UpdateRecorder(⟨"LeaderHandleFOLLOWERINFO", i, j⟩)

LeaderBroadcastLEADERINFO(i) ≜
      ∧ CheckEpoch   test restrictions of max epoch
      ∧ IsLeader(i)
      ∧ zabState[i] = DISCOVERY
      ∧ IsQuorum(cepochRecv[i])
```

```
$\land$ $acceptedEpoch' = [acceptedEpoch$ EXCEPT $![i] = tempMaxEpoch[i] + 1]$
$\land$ $cepochRecv' \quad = [cepochRecv \quad$ EXCEPT $![i] = cepochRecv[i] \cup \{NullPoint\}]$
$\land$ $BroadcastLEADERINFO(i, [mtype \mapsto LEADERINFO,$
                                    $mepoch \mapsto acceptedEpoch'[i]])$
$\land$ UNCHANGED $\langle state, currentEpoch, lastZxid, zabState, history, commitIndex, electionVarsZ,$
               $leadingVoteSet, learners, ackeRecv, ackldRecv, forwarding, ackIndex, currentCounter$
               $tempVarsZ, followerVarsZ, verifyVarsZ, electionMsgs, idTable\rangle$
$\land$ $UpdateRecorder(\langle$"LeaderBroadcastLEADERINFO", $i\rangle)$
```

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 $SYNCHRONIZATION$.

```
$FollowerHandleLEADERINFO(i, j) \triangleq$
    $\land$ $IsFollower(i)$
    $\land$ $PendingLEADERINFO(i, j)$
    $\land$ LET $msg \quad \triangleq msgs[j][i][1]$
            $infoOk \triangleq IsMyLeader(i, j)$
            $epochOk \triangleq \land infoOk$
                        $\land msg.mepoch \geq acceptedEpoch[i]$
            $correct \triangleq \land epochOk$
                        $\land zabState[i] = DISCOVERY$
       IN $\land infoOk$
          $\land \lor$ 1. Normal case
                $\land epochOk$
                $\land \lor \land correct$
                      $\land acceptedEpoch' = [acceptedEpoch$ EXCEPT $![i] = msg.mepoch]$
                      $\land Reply(i, j, [mtype \mapsto ACKEPOCH,$
                                    $mepoch \mapsto msg.mepoch,$
                                    $mlastEpoch \mapsto currentEpoch[i],$
                                    $mlastZxid \mapsto lastZxid[i]])$
                      $\land cepochSent' = [cepochSent$ EXCEPT $![i] =$ TRUE$]$
                      $\land$ UNCHANGED $inherentViolated$
                   $\lor \land \neg correct$
                      $\land PrintT($"Exception: Follower receives LEADERINFO while its ZabState is not DISCO
                      $\land inherentViolated' =$ TRUE
                      $\land Discard(j, i)$
                      $\land$ UNCHANGED $\langle acceptedEpoch, cepochSent\rangle$
                $\land zabState' = [zabState$ EXCEPT $![i] =$ IF $zabState[i] = DISCOVERY$ THEN $SYNCHRON$
                                                                         ELSE $zabState[i]]$
                $\land$ UNCHANGED $\langle varsL, leaderAddr\rangle$
             $\lor$ 2. Abnormal case - go back to election
                $\land \neg epochOk$
                $\land FollowerShutdown(i)$
                $\land Clean(i, j)$
                $\land$ UNCHANGED $\langle acceptedEpoch, cepochSent, inherentViolated\rangle$
```

```
        ∧ UNCHANGED ⟨history, commitIndex, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, ackInd
                     currentCounter, tempVarsZ, synced, proposalMsgsLog, epochLeader⟩
        ∧ 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])]
      ∧ 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, propos
```

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
```

```
$\land \lor \land replyOk$
        $\land SubRECOVERYSYNC(i, j)$
        $\land ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \text{IF } j \notin ackeRecv[i] \text{ THEN } ackeRecv[i] \cup \{j\}$
                                                                                   $\text{ELSE } ackeRecv[i]]$
        $\land Reply(i, j, [mtype \mapsto NEWLEADER,$
                            $mepoch \mapsto acceptedEpoch[i],$
                            $mlastZxid \mapsto lastZxid[i]])$
        $\land \text{UNCHANGED } \langle state, currentEpoch, logicalClock, receiveVotes, outOfElection, recvQueue$
                                $leadingVoteSet, electionMsgs, idTable, zabState, leaderAddr, learners\rangle$
    $\lor \land \neg replyOk$
        $\land \lor$  normal case
              $\land logOk$
              $\land ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \text{IF } j \notin ackeRecv[i] \text{ THEN } ackeRecv[i] \cup \{j\}$
                                                                                         $\text{ELSE } ackeRecv[i]]$
              $\land Discard(j, i)$
              $\land \text{UNCHANGED } \langle varsL, zabState, leaderAddr, learners, forwarding\rangle$
           $\lor$  go back to election since there exists follower more up-to-date than leader
              $\land \neg logOk$
              $\land LeaderShutdown(i)$
              $\land \text{UNCHANGED } ackeRecv$
        $\land \text{UNCHANGED } \langle history, commitIndex, synced, forwarding, ackIndex, proposalMsgsLog, ir$
$\land \text{UNCHANGED } \langle acceptedEpoch, cepochRecv, ackldRecv, currentCounter,$
                        $tempVarsZ, cepochSent, epochLeader\rangle$
$\land UpdateRecorder(\langle \text{"LeaderHandleACKEPOCH"}, i, j\rangle)$
```

Note: Since we abstract the process 'syncFollower' for leader and 'syncWithLeader' for follower, we cannot discribe how transactions being committed. One way we choose is let *commitIndex* be changed when leader receives quorum of *ACKEPOCH* and truns to *SYNCHRONIZATION*. It is actually different from code which uses *DIFF*/*TRUNC*/SNAP *syncMode* to complete sync.

```
$LeaderTransitionToSynchronization(i) \triangleq$
    $\land IsLeader(i)$
    $\land zabState[i] = DISCOVERY$
    $\land IsQuorum(ackeRecv[i])$
    $\land zabState' = [zabState \text{ EXCEPT } ![i] = SYNCHRONIZATION]$
    $\land currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = acceptedEpoch[i]]$
    $\land commitIndex' = [commitIndex \text{ EXCEPT } ![i] = Len(history[i])]$
    $\land initialHistory' = [initialHistory \text{ EXCEPT } ![i] = history[i]]$
    $\land ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = ackeRecv[i] \cup \{NullPoint\}]$
    $\land ackIndex' = [ackIndex \text{ EXCEPT } ![i][i] = Len(history[i])]$
    $\land UpdateProposal(i, i, lastZxid[i], currentEpoch'[i])$
    $\land \text{LET } epoch \triangleq acceptedEpoch[i]$
      $\text{IN } epochLeader' = [epochLeader \text{ EXCEPT } ![epoch] = epochLeader[epoch] \cup \{i\}]$
    $\land \text{UNCHANGED } \langle state, lastZxid, acceptedEpoch, history, logicalClock, receiveVotes, outOfElection,$
                        $recvQueue, waitNotmsg, leadingVoteSet, learners, cepochRecv, ackldRecv, forwarding$
                        $tempMaxEpoch, followerVarsZ, proposalMsgsLog, inherentViolated, msgVarsZ, idTab$
```

```
        ∧ UpdateRecorder(⟨"LeaderTransitionToSynchronization", i⟩)
```

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) ≜
        LET     canSync ≜ ∧ IsLeader(i)   ∧ zabState[i] ≠ DISCOVERY            ∧ IsMyLearner(i, j)
                          ∧ IsFollower(j) ∧ zabState[j] = SYNCHRONIZATION ∧ IsMyLeader(j, i)
        IN
        ∧ canSync
        ∧ SubRECOVERYSYNC(i, j)
        ∧ Send(i, j, [mtype      ↦ NEWLEADER,
                      mepoch     ↦ acceptedEpoch[i],
                      mlastZxid  ↦ lastZxid[i]])
        ∧ UNCHANGED ⟨state, zabState, acceptedEpoch, currentEpoch, logicalClock, receiveVotes, outOfElectio
                     leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv, currentCounter,
                     tempVarsZ, cepochSent, leaderAddr, epochLeader, electionMsgs, idTable⟩
        ∧ UpdateRecorder(⟨"RECOVERYSYNC", i, j⟩)
```

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

```
FollowerHandleNEWLEADER(i, j) ≜
        ∧ IsFollower(i)
        ∧ PendingNEWLEADER(i, j)
        ∧ LET msg     ≜ msgs[j][i][1]
              infoOk  ≜ ∧ IsMyLeader(i, j)
                        ∧ acceptedEpoch[i] = msg.mepoch
              correct ≜ ∧ infoOk
                        ∧ zabState[i] = SYNCHRONIZATION
                        ∧ synced[i]
                        ∧ ZxidEqual(lastZxid[i], msg.mlastZxid)
          IN  ∧ infoOk
              ∧ currentEpoch' = [currentEpoch EXCEPT ![i] = msg.mepoch]
              ∧ UpdateProposal(i, j, lastZxid[i], currentEpoch'[i])
              ∧ ∨ ∧ correct
                  ∧ Reply(i, j, [mtype  ↦ ACKLD,
                                 mepoch ↦ msg.mepoch])
                  ∧ UNCHANGED inherentViolated
                ∨ ∧ ¬correct
                  ∧ PrintT("Exception: Follower receives NEWLEADER while it has not completed sync with l
                  ∧ inherentViolated' = TRUE
                  ∧ Discard(j, i)
        ∧ UNCHANGED ⟨state, lastZxid, zabState, acceptedEpoch, history, commitIndex, logicalClock, receiveV
                     outOfElection, recvQueue, waitNotmsg, leaderVarsZ, tempVarsZ, followerVarsZ, pro
                     epochLeader, electionMsgs, idTable⟩
        $\land$ $UpdateRecorder(\langle$"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
                      $\land replyOk$
                      $\land Reply(i, j, [mtype \mapsto UPTODATE,$
                                        $mepoch \mapsto acceptedEpoch[i],$
                                        $mcommit \mapsto commitIndex[i]])$
                   $\lor$ leader still waits for a quorum's $ACKLD$ to broadcast $UPTODATE$
                      $\land \neg replyOk$
                      $\land Discard(j, i)$
                $\land ackldRecv' = [ackldRecv$ EXCEPT $![i] =$ IF $j \notin ackldRecv[i]$ THEN $ackldRecv[i] \cup \{j\}$
                                                                          ELSE $ackldRecv[i]]$
        $\land$ UNCHANGED $\langle serverVarsZ, electionVarsZ, leadingVoteSet, learners, cepochRecv, ackeRecv, forward$
                        $ackIndex, currentCounter, tempVarsZ, followerVarsZ, verifyVarsZ, electionMsgs, id$
        $\land$ $UpdateRecorder(\langle$"LeaderHandleACKLD", $i, j\rangle)$

$LeaderTransitionToBroadcast(i) \triangleq$
        $\land IsLeader(i)$
        $\land zabState[i] = SYNCHRONIZATION$
        $\land IsQuorum(ackldRecv[i])$
        $\land zabState' = [zabState$ EXCEPT $![i] = BROADCAST]$
        $\land currentCounter' = [currentCounter$ EXCEPT $![i] = 0]$
        $\land sendCounter' = [sendCounter$ EXCEPT $![i] = 0]$
        $\land committedIndex' = [committedIndex$ EXCEPT $![i] = Len(history[i])]$
        $\land ackldRecv' = [ackldRecv$ 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 $\langle state, currentEpoch, lastZxid, acceptedEpoch, history, commitIndex, electionVarsZ, le$
                        $learners, cepochRecv, ackeRecv, forwarding, ackIndex, tempVarsZ, followerVarsZ,$
                        $verifyVarsZ, electionMsgs, idTable\rangle$
        $\land$ $UpdateRecorder(\langle$"LeaderTransitionToBroadcast", $i\rangle)$

$FollowerHandleUPTODATE(i, j) \triangleq$
        $\land IsFollower(i)$
        $\land PendingUPTODATE(i, j)$
```

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

```
$ClientRequestAndLeaderBroadcastProposal(i, v) \triangleq$
    $\land CheckTransactionNum$   test restrictions of transaction num
    $\land IsLeader(i)$
    $\land zabState[i] = BROADCAST$
    $\land currentCounter' = [currentCounter$ EXCEPT $![i] = currentCounter[i] + 1]$
    $\land$ LET $newTransaction \triangleq [epoch \mapsto acceptedEpoch[i],$
                                 $counter \mapsto currentCounter'[i],$
                                 $value \mapsto v]$
      IN  $\land history' \quad = [history$ EXCEPT $![i] = Append(history[i], newTransaction)]$
          $\land lastZxid' \quad = [lastZxid$ EXCEPT $![i] = \langle acceptedEpoch[i], currentCounter'[i] \rangle]$
          $\land ackIndex' = [ackIndex$ 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 $\langle state, currentEpoch, zabState, acceptedEpoch, commitIndex, logicalClock, receiveVote$
                     $recvQueue, waitNotmsg, leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv,$
```

```
                    $tempVarsZ, followerVarsZ, epochLeader, inherentViolated,$
                    $electionMsgs, idTable\rangle$
           $\land UpdateRecorder(\langle$“ClientRequestAndLeaderBroadcastProposal”$, i, v\rangle)$

Follower accepts proposal and append it to history.
$FollowerHandlePROPOSAL(i, j) \triangleq$
     $\land IsFollower(i)$
     $\land PendingPROPOSAL(i, j)$
     $\land$ LET $msg \triangleq msgs[j][i][1]$
             $infoOk \triangleq \land IsMyLeader(i, j)$
                             $\land acceptedEpoch[i] = msg.mepoch$
             $correct \triangleq \land infoOk$
                              $\land zabState[i] \neq DISCOVERY$
                              $\land synced[i]$
             $logOk \triangleq$ the first $PROPOSAL$ in this epoch
                           $\lor \land msg.mproposal.counter = 1$
                              $\land \lor Len(history[i]) = 0$
                                 $\lor \land Len(history[i]) > 0$
                                    $\land history[i][Len(history[i])].epoch < msg.mepoch$
                           not the first $PROPOSAL$ in this epoch
                           $\lor \land msg.mproposal.counter > 1$
                              $\land Len(history[i]) > 0$
                              $\land history[i][Len(history[i])].epoch = msg.mepoch$
                              $\land history[i][Len(history[i])].counter = msg.mproposal.counter - 1$
       IN  $\land infoOk$
           $\land \lor \land correct$
                 $\land \lor \land logOk$
                       $\land history' = [history$ EXCEPT $![i] = Append(history[i], msg.mproposal)]$
                       $\land lastZxid' = [lastZxid$ EXCEPT $![i] = \langle msg.mepoch, msg.mproposal.counter\rangle]$
                       $\land UpdateProposal(i, j, lastZxid'[i], currentEpoch[i])$
                       $\land Reply(i, j, [mtype \mapsto ACK,$
                                        $mepoch \mapsto acceptedEpoch[i],$
                                        $mzxid \mapsto \langle msg.mepoch, msg.mproposal.counter\rangle])$
                       $\land$ UNCHANGED $inherentViolated$
                    $\lor \land \neg logOk$
                       $\land PrintT($“Exception: Follower receives PROPOSAL while the transaction is not the ne
                       $\land inherentViolated' =$ TRUE
                       $\land Discard(j, i)$
                       $\land$ UNCHANGED $\langle history, lastZxid, currentVote\rangle$
              $\lor \land \neg correct$
                 $\land PrintT($“Exception: Follower receives PROPOSAL while it has not completed sync with lea
                 $\land inherentViolated' =$ TRUE
                 $\land Discard(j, i)$
                 $\land$ UNCHANGED $\langle history, lastZxid, currentVote\rangle$
     $\land$ UNCHANGED $\langle state, currentEpoch, zabState, acceptedEpoch, commitIndex, logicalClock, receiveVotes$
```

```
                    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
                    $\land\ currentCounter[i] \geq msg.mzxid[2]$
        $noOutstanding \triangleq commitIndex[i] = Len(history[i])$
                    $outstandingProposals$: proposals in $history[commitIndex + 1 : Len(history))$
        $hasCommitted \triangleq commitIndex[i] \geq Len(initialHistory[i]) + msg.mzxid[2]$
                    namely, $lastCommitted \geq zxid$
        $logOk \triangleq \land\ infoOk$
                  $\land\ ackIndex[i][j] + 1 = Len(initialHistory[i]) + msg.mzxid[2]$
                      everytime, $ackIndex$ should just increase by 1
    IN  $\land\ infoOk$
        $\land\ \lor\ \land\ correct$
                $\land\ logOk$
                $\land\ ackIndex' = [ackIndex$ EXCEPT $![i][j] = ackIndex[i][j] + 1]$
                $\land\ \lor\ \land$  Note: outstanding is 0 / proposal has already been committed.
                        $\lor\ noOutstanding$
                        $\lor\ hasCommitted$
                      $\land\ Discard(j, i)$
                      $\land$ UNCHANGED $\langle commitIndex, inherentViolated\rangle$
                   $\lor\ \land\ \neg noOutstanding$
                      $\land\ \neg hasCommitted$
                      $\land\ LeaderTryToCommit(i, ToZxid(msg.mzxid), j)$
           $\lor\ \land\ \lor\ \neg correct$
                   $\lor\ \neg logOk$
                $\land\ PrintT($"Exception: ackIndex doesn't increase monotonically."$)$
                $\land\ inherentViolated' = $ TRUE
                $\land\ Discard(j, i)$
                $\land$ UNCHANGED $\langle ackIndex, commitIndex\rangle$
    $\land$ UNCHANGED $\langle state, currentEpoch, lastZxid, zabState, acceptedEpoch, history, electionVarsZ,$
                $leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv, forwarding, currentCount$
                $tempVarsZ, followerVarsZ, proposalMsgsLog, epochLeader, electionMsgs, idTable\rangle$
    $\land\ UpdateRecorder(\langle$"LeaderProcessACK"$, i, j\rangle)$

Follower receives $COMMIT$ and commits transaction.
$FollowerHandleCOMMIT(i, j) \triangleq$
    $\land\ IsFollower(i)$
    $\land\ PendingCOMMIT(i, j)$
    $\land$ LET $msg \triangleq msgs[j][i][1]$
        $infoOk \triangleq \land\ IsMyLeader(i, j)$
                   $\land\ acceptedEpoch[i] = msg.mepoch$
        $correct \triangleq \land\ infoOk$
                    $\land\ zabState[i] \neq DISCOVERY$
                    $\land\ synced[i]$
        $mindex \triangleq$ IF $Len(history[i]) = 0$ THEN $-1$
                    ELSE IF $\exists\, idx \in 1\, ..\, Len(history[i]) : PZxidEqual(history[i][idx], msg.mzxid)$
                            THEN CHOOSE $idx \in 1\, ..\, Len(history[i]) : PZxidEqual(history[i][idx], msg.$
                            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: Follower receives COMMIT while the index is not the next comm
                                ∧ inherentViolated′ = TRUE
                                ∧ UNCHANGED commitIndex
                        ∨ ∧ ¬logOk
                          ∧ PrintT(“Exception: Follower receives COMMIT while the transaction has not been sa
                          ∧ inherentViolated′ = TRUE
                          ∧ UNCHANGED commitIndex
                  ∨ ∧ ¬correct
                    ∧ PrintT(“Exception: Follower receives COMMIT while it has not completed sync with leade
                    ∧ inherentViolated′ = TRUE
                    ∧ UNCHANGED commitIndex
        ∧ Discard(j, i)
        ∧ UNCHANGED ⟨state, currentEpoch, lastZxid, zabState, acceptedEpoch, history, electionVarsZ, leader
                     tempVarsZ, followerVarsZ, proposalMsgsLog, epochLeader, electionMsgs, idTable⟩
        ∧ UpdateRecorder(⟨“FollowerHandleCOMMIT”, i, j⟩)
```

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
                                    ∨ ∧ IsLeader(i)
                                      ∧ 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]
                                          ∧ msg.mtype = FOLLOWERINFO
                                        ∨ ∧ ¬infoOk
                                          ∧ ∨ msg.mtype = ACKEPOCH
                                            ∨ msg.mtype = ACKLD
                                            ∨ msg.mtype = ACK
                                    ∨ ∧ IsFollower(i)
                                      ∧ LET infoOk ≜  ∧ j = leaderAddr[i]
                                                     ∧ acceptedEpoch[i] = msg.mepoch
                                       IN
                                        ∨ msg.mtype = FOLLOWERINFO
                                        ∨ msg.mtype = ACKEPOCH
                                        ∨ msg.mtype = ACKLD
                                        ∨ msg.mtype = ACK
                                        ∨ ∧ j ≠ leaderAddr[i]
                                          ∧ msg.mtype = LEADERINFO
                                        ∨ ∧ ¬infoOk
                                          ∧ ∨ msg.mtype = NEWLEADER
                                            ∨ msg.mtype = UPTODATE
                                            ∨ msg.mtype = PROPOSAL
                                            ∨ msg.mtype = COMMIT
                                    ∨ IsLooking(i)
                                 ∧ Discard(j, i)
        ∧ inherentViolated′ = TRUE
        ∧ UnchangeRecorder
        ∧ UNCHANGED ⟨serverVarsZ, electionVarsZ, leaderVarsZ, tempVarsZ, followerVarsZ, proposalMsgsL
                      epochLeader, electionMsgs, idTable⟩

Defines how the variables may transition.
NextZ ≜
    FLE modlue
      ∨ ∃ i, j ∈ Server : FLEReceiveNotmsg(i, j)
      ∨ ∃ i ∈ Server :    FLENotmsgTimeout(i)
      ∨ ∃ i ∈ Server :    FLEHandleNotmsg(i)
      ∨ ∃ i ∈ Server :    FLEWaitNewNotmsg(i)
      ∨ ∃ i ∈ Server :    FLEWaitNewNotmsgEnd(i)
    Some conditions like failure, network delay
      ∨ ∃ i ∈ Server :    FollowerTimeout(i)
      ∨ ∃ i ∈ Server :    LeaderTimeout(i)
      ∨ ∃ i, j ∈ Server : Timeout(i, j)
    Zab module - Discovery and Synchronization part
      ∨ ∃ i, j ∈ Server : EstablishConnection(i, j)
      ∨ ∃ i ∈ Server :    FollowerSendFOLLOWERINFO(i)
      ∨ ∃ i, j ∈ Server : LeaderHandleFOLLOWERINFO(i, j)
      ∨ ∃ i ∈ Server :    LeaderBroadcastLEADERINFO(i)
      ∨ ∃ i, j ∈ Server : FollowerHandleLEADERINFO(i, j)
      ∨ ∃ i, j ∈ Server : LeaderHandleACKEPOCH(i, j)
      ∨ ∃ i ∈ Server :    LeaderTransitionToSynchronization(i)
      ∨ ∃ i, j ∈ Server : RECOVERYSYNC(i, j)
```

```
        $\lor \exists i, j \in Server : FollowerHandleNEWLEADER(i, j)$
        $\lor \exists i, j \in Server : LeaderHandleACKLD(i, j)$
        $\lor \exists i \in Server : \quad LeaderTransitionToBroadcast(i)$
        $\lor \exists i, j \in Server : FollowerHandleUPTODATE(i, j)$
    Zab module $-$ *Broadcast* part
        $\lor \exists i \in Server, v \in Value : ClientRequestAndLeaderBroadcastProposal(i, v)$
         $\lor \exists i \in Server, v \in Value: ClientRequest(i, v)$
         $\lor \exists i \in Server: LeaderBroadcastProposal(i)$
        $\lor \exists i, j \in Server : FollowerHandlePROPOSAL(i, j)$
        $\lor \exists i, j \in Server : LeaderProcessACK(i, j)$
         $\lor \exists i \in Server: LeaderAdvanceCommit(i)$
         $\lor \exists i \in Server: LeaderBroadcastCommit(i)$
        $\lor \exists i, j \in Server : FollowerHandleCOMMIT(i, j)$
    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.

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

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

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

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

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

```
$Integrity \triangleq \forall i \in Server :$
               $\land IsFollower(i)$
               $\land commitIndex[i] > 0$
               $\Rightarrow \forall index \in 1 \,..\, commitIndex[i] : \exists msg \in proposalMsgsLog :$
                   $\lor \land msg.mtype = PROPOSAL$
                     $\land TransactionEqual(msg.mproposal, history[i][index])$
                   $\lor \land msg.mtype = \text{“RECOVERYSYNC”}$
```

```
                    $\land \exists\, tindex \in 1 \,..\, Len(msg.mproposals) : TransactionEqual(msg.mproposals[tindex], h$

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

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

 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 ≜ ∀ i ∈ Server : commitIndex[i] ≥ 2
                        ⇒ ∀ idx1, idx2 ∈ 1 .. commitIndex[i] : ∨ history[i][idx1].epoch ≥ history[i][idx2].ep
                                                               ∨ ∧ history[i][idx1].epoch < history[i][idx2]
                                                                 ∧ 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 ≜ ∀ i, j ∈ Server : ∧ IsLeader(i)
                                     ∧ IsFollower(j)
                                     ∧ commitIndex[j] ≥ 1
                     ⇒ ∀ index ∈ 1 .. commitIndex[j] : ∨ history[j][index].epoch ≥ currentEpoch[i]
                                                       ∨ ∧ history[j][index].epoch < currentEpoch[i]
                                                         ∧ ∃ idx ∈ 1 .. commitIndex[i] : TransactionEqua
```

```
LeaderBroadcastProposal(i) ≜
    ∧ IsLeader(i)
    ∧ 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 ↦ toBeSentEntry]
             IN proposalMsgsLog' = proposalMsgsLog ∪ {m}
    ∧ UpdateRecorder(⟨"LeaderBroadcastProposal", i⟩)
    ∧UNCHANGED ⟨serverVarsZ, electionVarsZ, leadingVoteSet, learners, cepochRecv, ackeRecv, ackldRecv, forwarding,

                ackIndex, currentCounter, committedCounter, tempVarsZ, followerVarsZ, epochLeader,

                inherentViolated, electionMsgs, idTable⟩

LeaderAdvanceCommit(i) ≜
    ∧ IsLeader(i)
    ∧ zabState[i] = BROADCAST
    ∧ commitIndex[i] < Len(history[i])
    ∧ LET Agree(index) ≜ {i} ∪ {k ∈ (Server \ {i}) : ackIndex[i][k] ≥ index}
          agreeIndexes ≜ {index ∈ (commitIndex[i] + 1) .. Len(history[i]) :
          Agree(index) ∈ Quorums}
          newCommitIndex ≜ IF agreeIndexes ≠ {} THEN Maximum(agreeIndexes)
                                                ELSE commitIndex[i]
      IN  commitIndex' = [commitIndex EXCEPT ![i] = newCommitIndex]
    ∧ UpdateRecorder(⟨"LeaderAdvanceCommit", i⟩)
    ∧UNCHANGED ⟨state, currentEpoch, lastZxid, zabState, acceptedEpoch, history, electionVarsZ, leaderVarsZ,
```

$tempVarsZ, followerVarsZ, verifyVarsZ, msgVarsZ, idTable\rangle$

\* Modification History
\* Last modified *Wed Oct* 20 20:56:37 *CST* 2021 by Dell
\* Created *Tue Jun* 29 22:13:02 *CST* 2021 by Dell