



The leader's epoch or the last new epoch proposal the follower acknowledged  
(namely epoch of the last *NEWEPOCH* accepted,  $f.p$  in paper).

VARIABLE *currentEpoch*

The last new leader proposal the follower acknowledged  
(namely epoch of the last *NEWLEADER* accepted,  $f.a$  in paper).

VARIABLE *leaderEpoch*

The identifier of the leader for followers.

VARIABLE *leaderOracle*

The history of servers as the sequence of transactions.

VARIABLE *history*

The messages representing requests and responses sent from one server to another.  
 $msgs[i][j]$  means the input buffer of server  $j$  from server  $i$ .

VARIABLE *msgs*

The set of servers which the leader think follow itself ( $Q$  in paper).

VARIABLE *cluster*

The set of followers who has successfully sent *CEPOCH* to pleader in pleader.

VARIABLE *epochRecv*

The set of followers who has successfully sent *ACK-E* to pleader in pleader.

VARIABLE *ackRecv*

The set of followers who has successfully sent *ACK-LD* to pleader in pleader.

VARIABLE *ackldRecv*

$ackIndex[i][j]$  means leader  $i$  has received how many *ACK* messages from follower  $j$ .  
So  $ackIndex[i][i]$  is not used.

VARIABLE *ackIndex*

$currentCounter[i]$  means the count of transactions client requests leader.

VARIABLE *currentCounter*

$sendCounter[i]$  means the count of transactions leader has broadcast.

VARIABLE *sendCounter*

$initialHistory[i]$  means the initial history of leader  $i$  in epoch  $currentEpoch[i]$ .

VARIABLE *initialHistory*

$commitIndex[i]$  means leader/follower  $i$  should commit how many proposals and sent *COMMIT* messages.

It should be more formal to add variable *applyIndex*/*deliverIndex* to represent the prefix entries of the history that has applied to state machine, but we can tolerate that  $applyIndex(deliverIndex \text{ here}) = commitIndex$ .

This does not violate correctness. (*commitIndex* increases monotonically before restarting)

VARIABLE *commitIndex*

$commitIndex[i]$  means leader  $i$  has committed how many proposals and sent *COMMIT* messages.  
 VARIABLE *committedIndex*

Helper matrix for follower to stop sending *CEPOCH* to pleader in followers.  
 Because *CEPOCH* is the sole message which follower actively sends to pleader.  
 VARIABLE *ceepochSent*

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

the maximum *leaderEpoch* and most up-to-date history in *ACKE* pleader received from followers.  
 VARIABLE *tempMaxLastEpoch*

Because pleader updates state and broadcasts *NEWLEADER* when it receives *ACKE* from a quorum of followers,  
 and *initialHistory* is determined. But *tempInitialHistory* may change when receiving other *ACKEs* after entering into *phase2*.  
 So it is necessary to split *initialHistory* with *tempInitialHistory*.  
 VARIABLE *tempInitialHistory*

the set of all broadcast messages whose type is proposal that any leader has sent, only used in verifying properties.  
 So the variable will only be changed in transition *LeaderBroadcast1*.  
 VARIABLE *proposalMsgsLog*

Helper set for server who restarts to collect which servers has responded to it.  
 VARIABLE *recoveryRespRecv*

the maximum epoch and corresponding *leaderOracle* in *RECOVERYRESPONSE* from followers.  
 VARIABLE *recoveryMaxEpoch*

VARIABLE *recoveryMEOracle*

Helper variable for server after restart to stop sending *RECOVERYREQUEST* continuously.  
 VARIABLE *recoverySent*

variables that uniquely used for constraining state space in model checking  
 VARIABLES *electionNum*, the round of leader election, not equal to  $Maximum\{currentEpoch[i] : i \in Server\}$ ,  
 because *currentEpoch* will increase only when follower receives *NEWEPOCH*,  
 and it is common that some round of election ends without leader broadcasting *NEWEPOCH*  
 or follower receiving *NEWEPOCH*.  
*totalRestartNum* the number of restart from all servers, also as a global variable.

Persistent state of a server: history, *currentEpoch*, *leaderEpoch*  
 $serverVars \triangleq \langle state, currentEpoch, leaderEpoch, leaderOracle, history, commitIndex \rangle$   
 $leaderVars \triangleq \langle cluster, cepochRecv, ackRecv, ackldRecv, ackIndex, currentCounter, sendCounter, initialHistory \rangle$   
 $tempVars \triangleq \langle tempMaxEpoch, tempMaxLastEpoch, tempInitialHistory \rangle$   
 $recoveryVars \triangleq \langle recoveryRespRecv, recoveryMaxEpoch, recoveryMEOracle, recoverySent \rangle$   
 $testVars \triangleq \langle electionNum, totalRestartNum \rangle$   
 $vars \triangleq \langle serverVars, msgs, leaderVars, tempVars, recoveryVars, cepochSent, proposalMsgsLog, testVars \rangle$

---

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

Add a message to  $msgs$  – add a message  $m$  to  $msgs[i][j]$

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

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

Remove a message from  $msgs$  – discard head of  $msgs[i][j]$

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

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

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

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

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

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

$Reply2(i, j, m1, m2) \triangleq msgs' = [msgs \text{ EXCEPT } ![j][i] = Tail(msgs[j][i]),$   
 $![i][j] = Append(Append(msgs[i][j], m1), m2)]$

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

---

Define initial values for all variables

$Init \triangleq \wedge state = [s \in Server \mapsto Follower]$   
 $\wedge currentEpoch = [s \in Server \mapsto 0]$   
 $\wedge leaderEpoch = [s \in Server \mapsto 0]$   
 $\wedge leaderOracle = [s \in Server \mapsto NullPoint]$   
 $\wedge history = [s \in Server \mapsto \langle \rangle]$   
 $\wedge msgs = [i \in Server \mapsto [j \in Server \mapsto \langle \rangle]]$   
 $\wedge cluster = [i \in Server \mapsto \{\}]$   
 $\wedge epochRecv = [s \in Server \mapsto \{\}]$   
 $\wedge ackRecv = [s \in Server \mapsto \{\}]$   
 $\wedge ackldRecv = [s \in Server \mapsto \{\}]$   
 $\wedge ackIndex = [i \in Server \mapsto [j \in Server \mapsto 0]]$   
 $\wedge currentCounter = [s \in Server \mapsto 0]$   
 $\wedge sendCounter = [s \in Server \mapsto 0]$   
 $\wedge commitIndex = [s \in Server \mapsto 0]$   
 $\wedge committedIndex = [s \in Server \mapsto 0]$

$$\begin{aligned}
\wedge \text{initialHistory} &= [s \in \text{Server} \mapsto \langle \rangle] \\
\wedge \text{epochSent} &= [s \in \text{Server} \mapsto \text{FALSE}] \\
\wedge \text{tempMaxEpoch} &= [s \in \text{Server} \mapsto 0] \\
\wedge \text{tempMaxLastEpoch} &= [s \in \text{Server} \mapsto 0] \\
\wedge \text{tempInitialHistory} &= [s \in \text{Server} \mapsto \langle \rangle] \\
\wedge \text{recoveryRespRecv} &= [s \in \text{Server} \mapsto \{\}] \\
\wedge \text{recoveryMaxEpoch} &= [s \in \text{Server} \mapsto 0] \\
\wedge \text{recoveryMEOracle} &= [s \in \text{Server} \mapsto \text{NullPoint}] \\
\wedge \text{recoverySent} &= [s \in \text{Server} \mapsto \text{FALSE}] \\
\wedge \text{proposalMsgsLog} &= \{\} \\
\wedge \text{electionNum} &= 0 \\
\wedge \text{totalRestartNum} &= 0
\end{aligned}$$

---

A server becomes pleader and a quorum servers knows that.

$\text{Election}(i, Q) \triangleq$

test restrictions

$\wedge \text{electionNum} < \text{MaxElectionNum}$

$\wedge i \in Q$

$\wedge \text{electionNum}' = \text{electionNum} + 1$

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

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

$\wedge \text{epochRecv}' = [\text{epochRecv} \text{ EXCEPT } ![i] = \{i\}]$

$\wedge \text{ackRecv}' = [\text{ackRecv} \text{ EXCEPT } ![i] = \{i\}]$

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

$\wedge \text{ackIndex}' = [ii \in \text{Server} \mapsto [ij \in \text{Server} \mapsto$   
IF  $ii = i$  THEN 0

ELSE  $\text{ackIndex}[ii][ij]$ ]

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

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

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

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

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

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

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

$\wedge \text{epochSent}' = [s \in \text{Server} \mapsto \text{IF } s \in Q \text{ THEN FALSE}$

ELSE  $\text{epochSent}[s]$ ]

$\wedge \text{msgs}' = [ii \in \text{Server} \mapsto [ij \in \text{Server} \mapsto$   
IF  $ii \in Q \vee ij \in Q$  THEN  $\langle \rangle$

ELSE  $\text{msgs}[ii][ij]$ ]

$\wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{history}, \text{commitIndex}, \text{currentCounter}, \text{sendCounter}, \text{proposalMsgsLog} \rangle$

The action should be triggered once at the beginning.

Because we abstract the part of leader election, we can use global variables in this action.

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

The leader finds timeout with another follower.

$$\begin{aligned} \text{LeaderTimeout}(i, j) &\triangleq \\ &\wedge \text{state}[i] \neq \text{Follower} \\ &\wedge j \neq i \\ &\wedge j \in \text{cluster}[i] \\ &\wedge \text{LET } \text{newCluster} \triangleq \text{cluster}[i] \setminus \{j\} \\ &\quad \text{IN } \wedge \vee \wedge \text{newCluster} \in \text{Quorums} \\ &\quad \wedge \text{cluster}' = [\text{cluster} \text{ EXCEPT } ![i] = \text{newCluster}] \\ &\quad \wedge \text{clean}(i, j) \\ &\quad \wedge \text{UNCHANGED} \langle \text{state}, \text{cepcvRecv}, \text{ackRecv}, \text{ackldRecv}, \text{ackIndex}, \text{committedIndex}, \text{initialHistory}, \\ &\quad \quad \text{tempMaxEpoch}, \text{tempMaxLastEpoch}, \text{tempInitialHistory}, \text{leaderOracle}, \text{le} \rangle \\ &\quad \vee \wedge \text{newCluster} \notin \text{Quorums} \\ &\quad \wedge \text{LET } Q \triangleq \text{CHOOSE } q \in \text{Quorums} : i \in q \\ &\quad \quad v \triangleq \text{CHOOSE } s \in Q : \text{TRUE} \\ &\quad \quad \text{IN } \text{Election}(v, Q) \\ &\quad \exists Q \in \text{Quorums} : \wedge i \in Q \\ &\quad \quad \wedge \exists v \in Q : \text{Election}(v, Q) \\ &\wedge \text{UNCHANGED} \langle \text{currentEpoch}, \text{history}, \text{commitIndex}, \text{currentCounter}, \text{sendCounter}, \text{recoveryVars}, \text{pro} \rangle \end{aligned}$$

A follower finds timeout with the leader.

$$\begin{aligned} \text{FollowerTimeout}(i) &\triangleq \\ &\wedge \text{state}[i] = \text{Follower} \\ &\wedge \text{leaderOracle}[i] \neq \text{NullPoint} \\ &\wedge \exists Q \in \text{Quorums} : \wedge i \in Q \\ &\quad \wedge \exists v \in Q : \text{Election}(v, Q) \\ &\wedge \text{UNCHANGED} \langle \text{currentEpoch}, \text{history}, \text{commitIndex}, \text{currentCounter}, \text{sendCounter}, \text{recoveryVars}, \text{pro} \rangle \end{aligned}$$


---

A server halts and restarts.

Like Recovery protocol in View-stamped Replication, we let a server join in cluster by broadcast recovery and wait until receiving responses from a quorum of servers.

$$\begin{aligned} \text{Restart}(i) &\triangleq \\ &\text{test restrictions} \\ &\wedge \text{totalRestartNum} < \text{MaxTotalRestartNum} \\ &\wedge \text{totalRestartNum}' = \text{totalRestartNum} + 1 \\ &\wedge \text{state}' = [\text{state} \text{ EXCEPT } ![i] = \text{Follower}] \\ &\wedge \text{leaderOracle}' = [\text{leaderOracle} \text{ EXCEPT } ![i] = \text{NullPoint}] \\ &\wedge \text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = 0] \\ &\wedge \text{cepcvSent}' = [\text{cepcvSent} \text{ EXCEPT } ![i] = \text{FALSE}] \end{aligned}$$

$$\begin{aligned}
\wedge \text{msgs}' &= [ii \in \text{Server} \mapsto [ij \in \text{Server} \mapsto \text{IF } ij = i \text{ THEN } \langle \rangle \\
&\quad \text{ELSE } \text{msgs}[ii][ij]]] \\
\wedge \text{recoverySent}' &= [\text{recoverySent} \text{ EXCEPT } ![i] = \text{FALSE}] \\
\wedge \text{UNCHANGED } &\langle \text{currentEpoch}, \text{leaderEpoch}, \text{history}, \text{leaderVars}, \text{tempVars}, \\
&\quad \text{recoveryRespRecv}, \text{recoveryMaxEpoch}, \text{recoveryMEOracle}, \text{proposalMsgsLog}, \text{electionRe} \rangle \\
\text{RecoveryAfterRestart}(i) &\triangleq \\
&\quad \text{test restrictions} \\
&\quad \wedge \text{totalRestartNum} < \text{MaxTotalRestartNum} \\
&\quad \wedge \text{state}[i] = \text{Follower} \\
&\quad \wedge \text{leaderOracle}[i] = \text{NullPoint} \\
&\quad \wedge \neg \text{recoverySent}[i] \\
&\quad \wedge \text{recoveryRespRecv}' = [\text{recoveryRespRecv} \text{ EXCEPT } ![i] = \{\}] \\
&\quad \wedge \text{recoveryMaxEpoch}' = [\text{recoveryMaxEpoch} \text{ EXCEPT } ![i] = \text{currentEpoch}[i]] \\
&\quad \wedge \text{recoveryMEOracle}' = [\text{recoveryMEOracle} \text{ EXCEPT } ![i] = \text{NullPoint}] \\
&\quad \wedge \text{recoverySent}' = [\text{recoverySent} \text{ EXCEPT } ![i] = \text{TRUE}] \\
&\quad \wedge \text{BroadcastToAll}(i, [\text{mtype} \mapsto \text{RECOVERYREQUEST}]) \\
&\quad \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{epochSent}, \text{proposalMsgsLog}, \text{testVars} \rangle \\
\text{HandleRecoveryRequest}(i, j) &\triangleq \\
&\quad \wedge \text{msgs}[j][i] \neq \langle \rangle \\
&\quad \wedge \text{msgs}[j][i][1].\text{mtype} = \text{RECOVERYREQUEST} \\
&\quad \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{RECOVERYRESPONSE}, \\
&\quad \quad \text{moracle} \mapsto \text{leaderOracle}[i], \\
&\quad \quad \text{mepoch} \mapsto \text{currentEpoch}[i]]) \\
&\quad \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{epochSent}, \text{recoveryVars}, \text{proposalMsgsLog}, \text{testVars} \rangle \\
\text{HandleRecoveryResponse}(i, j) &\triangleq \\
&\quad \wedge \text{msgs}[j][i] \neq \langle \rangle \\
&\quad \wedge \text{msgs}[j][i][1].\text{mtype} = \text{RECOVERYRESPONSE} \\
&\quad \wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1] \\
&\quad \quad \text{infoOk} \triangleq \wedge \text{msg.mepoch} \geq \text{recoveryMaxEpoch}[i] \\
&\quad \quad \wedge \text{msg.moracle} \neq \text{NullPoint} \\
&\quad \text{IN } \vee \wedge \text{infoOk} \\
&\quad \quad \wedge \text{recoveryMaxEpoch}' = [\text{recoveryMaxEpoch} \text{ EXCEPT } ![i] = \text{msg.mepoch}] \\
&\quad \quad \wedge \text{recoveryMEOracle}' = [\text{recoveryMEOracle} \text{ EXCEPT } ![i] = \text{msg.moracle}] \\
&\quad \vee \wedge \neg \text{infoOk} \\
&\quad \quad \wedge \text{UNCHANGED } \langle \text{recoveryMaxEpoch}, \text{recoveryMEOracle} \rangle \\
&\quad \wedge \text{Discard}(j, i) \\
&\quad \wedge \text{recoveryRespRecv}' = [\text{recoveryRespRecv} \text{ EXCEPT } ![i] = \text{IF } j \in \text{recoveryRespRecv}[i] \text{ THEN } \text{recoveryRe} \\
&\quad \quad \quad \text{ELSE } \text{recoveryRe} \\
&\quad \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{epochSent}, \text{recoverySent}, \text{proposalMsgsLog}, \text{testVars} \rangle \\
\text{FindCluster}(i) &\triangleq \\
&\quad \wedge \text{state}[i] = \text{Follower} \\
&\quad \wedge \text{leaderOracle}[i] = \text{NullPoint}
\end{aligned}$$

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


---

In phase  $f11$ , follower sends  $f.p$  to pleader via  $\text{CEPOCH}$ .

$$\begin{aligned}
& \text{FollowerDiscovery1}(i) \triangleq \\
& \quad \wedge \text{ state}[i] = \text{Follower} \\
& \quad \wedge \text{ leaderOracle}[i] \neq \text{NullPoint} \\
& \quad \wedge \neg \text{ cepochSent}[i] \\
& \quad \wedge \text{ LET } \text{leader} \triangleq \text{ leaderOracle}[i] \\
& \quad \text{ IN } \quad \text{ Send}(i, \text{leader}, [\text{ mtype} \mapsto \text{CEPOCH}, \\
& \quad \quad \quad \text{ mepoch} \mapsto \text{ currentEpoch}[i]]) \\
& \quad \wedge \text{ cepochSent}' = [\text{ cepochSent} \text{ EXCEPT } ![i] = \text{TRUE}] \\
& \quad \wedge \text{ UNCHANGED } \langle \text{ serverVars}, \text{ leaderVars}, \text{ tempVars}, \text{ recoveryVars}, \text{ proposalMsgsLog}, \text{ testVars} \rangle
\end{aligned}$$

In phase  $l11$ , pleader receives  $\text{CEPOCH}$  from a quorum, and choose a new epoch  $e'$  as its own  $l.p$  and sends  $\text{NEWPOCH}$  to followers.

$$\begin{aligned}
& \text{LeaderHandleCEPOCH}(i, j) \triangleq \\
& \quad \wedge \text{ state}[i] = \text{ProspectiveLeader} \\
& \quad \wedge \text{ msgs}[j][i] \neq \langle \rangle \\
& \quad \wedge \text{ msgs}[j][i][1].\text{mtype} = \text{CEPOCH} \\
& \quad \wedge \vee \text{ new message - modify } \text{ tempMaxEpoch} \text{ and } \text{ cepochRecv} \\
& \quad \quad \wedge \text{ NullPoint} \notin \text{ cepochRecv}[i] \\
& \quad \quad \wedge \text{ LET } \text{ newEpoch} \triangleq \text{ Maximum}(\{\text{ tempMaxEpoch}[i], \text{ msgs}[j][i][1].\text{mepoch}\}) \\
& \quad \quad \text{ IN } \quad \text{ tempMaxEpoch}' = [\text{ tempMaxEpoch} \text{ EXCEPT } ![i] = \text{ newEpoch}] \\
& \quad \quad \wedge \text{ cepochRecv}' = [\text{ cepochRecv} \text{ EXCEPT } ![i] = \text{ IF } j \in \text{ cepochRecv}[i] \text{ THEN } \text{ cepochRecv}[i] \\
& \quad \quad \quad \text{ ELSE } \text{ cepochRecv}[i] \cup \{j\}] \\
& \quad \wedge \text{ Discard}(j, i) \\
& \quad \vee \text{ new follower who joins in cluster / follower whose history and } \text{ commitIndex} \text{ do not match} \\
& \quad \quad \wedge \text{ NullPoint} \in \text{ cepochRecv}[i] \\
& \quad \quad \wedge \vee \wedge \text{ NullPoint} \notin \text{ ackeRecv}[i] \\
& \quad \quad \quad \wedge \text{ Reply}(i, j, [\text{ mtype} \mapsto \text{NEWPOCH}],
\end{aligned}$$



$$\begin{aligned}
& mepoch \mapsto leaderEpoch[i]) \\
\vee \wedge \text{NullPoint} \in \text{ackRecv}[i] \\
& \wedge \text{Reply2}(i, j, [mtype \mapsto \text{NEWEPOCH}, \\
& \quad mepoch \mapsto leaderEpoch[i], \\
& \quad [mtype \mapsto \text{NEWLEADER}, \\
& \quad mepoch \mapsto currentEpoch[i], \\
& \quad minitialHistory \mapsto initialHistory[i]]) \\
& \wedge \text{UNCHANGED} \langle \text{ceepochRecv}, \text{tempMaxEpoch} \rangle \\
& \wedge cluster' = [cluster \text{ EXCEPT } ![i] = \text{IF } j \in cluster[i] \text{ THEN } cluster[i] \text{ ELSE } cluster[i] \cup \{j\}] \\
& \wedge \text{UNCHANGED} \langle \text{serverVars}, \text{ackRecv}, \text{ackldRecv}, \text{ackIndex}, \text{currentCounter}, \text{sendCounter}, \text{initialHistory}, \\
& \quad \text{committedIndex}, \text{ceepochSent}, \text{tempMaxLastEpoch}, \text{tempInitialHistory}, \text{recoveryVars}, p \rangle
\end{aligned}$$

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

$$\begin{aligned}
\text{LeaderDiscovery1}(i) & \triangleq \\
& \wedge state[i] = \text{ProspectiveLeader} \\
& \wedge \text{ceepochRecv}[i] \in \text{Quorums} \\
& \wedge leaderEpoch' = [leaderEpoch \text{ EXCEPT } ![i] = \text{tempMaxEpoch}[i] + 1] \\
& \wedge \text{ceepochRecv}' = [\text{ceepochRecv} \text{ EXCEPT } ![i] = \{\text{NullPoint}\}] \\
& \wedge \text{Broadcast}(i, [mtype \mapsto \text{NEWEPOCH}, \\
& \quad mepoch \mapsto leaderEpoch'[i]]) \\
& \wedge \text{UNCHANGED} \langle state, currentEpoch, leaderOracle, history, cluster, \text{ackRecv}, \text{ackldRecv}, \text{ackIndex}, \text{currentCounter}, \\
& \quad \text{initialHistory}, \text{commitIndex}, \text{committedIndex}, \text{ceepochSent}, \text{tempVars}, \text{recoveryVars}, p \rangle
\end{aligned}$$

In phase  $f12$ , follower receives *NEWEPOCH*. If  $e' > f.p$  then sends back *ACKE*, and *ACKE* contains  $f.a$  and  $hf$  to help pleader choose a newer history.

$$\begin{aligned}
\text{FollowerDiscovery2}(i, j) & \triangleq \\
& \wedge state[i] = \text{Follower} \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].mtype = \text{NEWEPOCH} \\
& \wedge \text{LET } msg \triangleq \text{msgs}[j][i][1] \\
& \text{IN } \vee \text{new NEWEPOCH} - \text{accept and reply} \\
& \wedge currentEpoch[i] < msg.mepoch \\
& \wedge \vee \wedge leaderOracle[i] = j \\
& \quad \wedge currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch] \\
& \quad \wedge \text{Reply}(i, j, [mtype \mapsto \text{ACKE}, \\
& \quad \quad mepoch \mapsto msg.mepoch, \\
& \quad \quad mlastEpoch \mapsto leaderEpoch[i], \\
& \quad \quad mhf \mapsto history[i]]) \\
& \quad \wedge \text{ceepochSent}' = [\text{ceepochSent} \text{ EXCEPT } ![i] = \text{TRUE}] \\
& \vee \wedge leaderOracle[i] \neq j \\
& \quad \wedge \text{Discard}(j, i) \\
& \quad \wedge \text{UNCHANGED} \langle currentEpoch, \text{ceepochSent} \rangle \\
& \vee \wedge currentEpoch[i] = msg.mepoch \\
& \quad \wedge \vee \wedge leaderOracle[i] = j
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{Reply}(i, j, [mtype \mapsto \text{ACKE}, \\
& \quad mepoch \mapsto \text{msg.mepoch}, \\
& \quad mlastEpoch} \mapsto \text{leaderEpoch}[i], \\
& \quad mhf \mapsto \text{history}[i]]) \\
& \wedge \text{cepochSent}' = [\text{cepochSent} \text{ EXCEPT } ![i] = \text{TRUE}] \\
& \wedge \text{UNCHANGED } \text{currentEpoch} \\
\vee & \text{ It may happen when a leader do not update new epoch to all followers in } Q, \text{ and a new election begi} \\
& \wedge \text{leaderOracle}[i] \neq j \\
& \wedge \text{Discard}(j, i) \\
& \wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{cepochSent} \rangle \\
\vee & \text{ stale } \text{NEWPOCH} - \text{diacard} \\
& \wedge \text{currentEpoch}[i] > \text{msg.mepoch} \\
& \wedge \text{Discard}(j, i) \\
& \wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{cepochSent} \rangle \\
& \wedge \text{UNCHANGED } \langle \text{state}, \text{leaderEpoch}, \text{leaderOracle}, \text{history}, \text{leaderVars}, \\
& \quad \text{commitIndex}, \text{tempVars}, \text{recoveryVars}, \text{proposalMsgsLog}, \text{testVars} \rangle
\end{aligned}$$

In phase *l12*, pleader receives *ACKE* from a quorum,  
and select the history of one most up-to-date follower to be the initial history.

$$\begin{aligned}
& \text{LeaderHandleACKE}(i, j) \triangleq \\
& \wedge \text{state}[i] = \text{ProspectiveLeader} \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{ACKE} \\
& \wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1] \\
& \quad \text{infoOk} \triangleq \vee \text{msg.mlastEpoch} > \text{tempMaxLastEpoch}[i] \\
& \quad \vee \wedge \text{msg.mlastEpoch} = \text{tempMaxLastEpoch}[i] \\
& \quad \wedge \vee \text{LastZxid}(\text{msg.mhf})[1] > \text{LastZxid}(\text{tempInitialHistory}[i])[1] \\
& \quad \vee \wedge \text{LastZxid}(\text{msg.mhf})[1] = \text{LastZxid}(\text{tempInitialHistory}[i])[1] \\
& \quad \wedge \text{LastZxid}(\text{msg.mhf})[2] \geq \text{LastZxid}(\text{tempInitialHistory}[i])[2] \\
\text{IN } & \vee \wedge \text{leaderEpoch}[i] = \text{msg.mepoch} \\
& \wedge \vee \wedge \text{infoOk} \\
& \quad \wedge \text{tempMaxLastEpoch}' = [\text{tempMaxLastEpoch} \text{ EXCEPT } ![i] = \text{msg.mlastEpoch}] \\
& \quad \wedge \text{tempInitialHistory}' = [\text{tempInitialHistory} \text{ EXCEPT } ![i] = \text{msg.mhf}] \\
& \vee \wedge \neg \text{infoOk} \\
& \quad \wedge \text{UNCHANGED } \langle \text{tempMaxLastEpoch}, \text{tempInitialHistory} \rangle \\
& \text{Followers not in } Q \text{ will not receive } \text{NEWPOCH}, \text{ so leader will receive } \text{ACKE} \text{ only when the source is in } Q \\
& \wedge \text{ackRecv}' = [\text{ackRecv} \text{ EXCEPT } ![i] = \text{IF } j \notin \text{ackRecv}[i] \text{ THEN } \text{ackRecv}[i] \cup \{j\} \\
& \quad \text{ELSE } \text{ackRecv}[i]] \\
& \vee \wedge \text{leaderEpoch}[i] \neq \text{msg.mepoch} \\
& \quad \wedge \text{UNCHANGED } \langle \text{tempMaxLastEpoch}, \text{tempInitialHistory}, \text{ackRecv} \rangle \\
& \wedge \text{Discard}(j, i) \\
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{cluster}, \text{cepochRecv}, \text{ackldRecv}, \text{ackIndex}, \text{currentCounter}, \\
& \quad \text{sendCounter}, \text{initialHistory}, \text{committedIndex}, \text{cepochSent}, \text{tempMaxEpoch}, \text{recoveryVars} \rangle
\end{aligned}$$

$$\text{LeaderDiscovery2Sync1}(i) \triangleq$$

$$\begin{aligned}
& \wedge state[i] = ProspectiveLeader \\
& \wedge ackeRecv[i] \in Quorums \\
& \wedge currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = leaderEpoch[i]] \\
& \wedge history' = [history \text{ EXCEPT } ![i] = tempInitialHistory[i]] \\
& \wedge initialHistory' = [initialHistory \text{ EXCEPT } ![i] = tempInitialHistory[i]] \\
& \wedge ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \{NullPoint\}] \\
& \wedge ackIndex' = [ackIndex \text{ EXCEPT } ![i][i] = Len(tempInitialHistory[i])] \\
& \text{until now, phase1(Discovery) ends} \\
& \wedge Broadcast(i, [mtype \mapsto NEWLEADER, \\
& \quad mepoch \mapsto currentEpoch'[i], \\
& \quad minitialHistory \mapsto history'[i]]) \\
& \wedge LET m \triangleq [msource \mapsto i, mtype \mapsto NEWLEADER, mepoch \mapsto currentEpoch'[i], mproposals \mapsto history'[i]] \\
& \quad IN proposalMsgsLog' = IF m \in proposalMsgsLog THEN proposalMsgsLog \\
& \quad \quad ELSE proposalMsgsLog \cup \{m\} \\
& \wedge UNCHANGED \langle state, leaderEpoch, leaderOracle, commitIndex, cluster, cepochRecv, ackldRecv, \\
& \quad currentCounter, sendCounter, committedIndex, cepochSent, tempVars, recoveryVars, p \rangle
\end{aligned}$$

Note1: Delete the change of *commitIndex* in *LeaderDiscovery2Sync1* and *FollowerSync1*, then we can promise that *commitIndex* of every server increases monotonically, except that some server halts and restarts.

Note2: Set *cepochRecv*, *ackeRecv*, *ackldRecv* to  $\{NullPoint\}$  in corresponding three actions to make sure that the prospective leader will not broadcast *NEWLEADER/NEWLEADER/COMMITLD* twice.

---

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

$$\begin{aligned}
FollowerSync1(i, j) & \triangleq \\
& \wedge state[i] = Follower \\
& \wedge msgs[j][i] \neq \langle \rangle \\
& \wedge msgs[j][i][1].mtype = NEWLEADER \\
& \wedge LET msg \triangleq msgs[j][i][1] \\
& \quad replyOk \triangleq \wedge currentEpoch[i] \leq msg.mepoch \\
& \quad \quad \wedge leaderOracle[i] = j \\
& \quad IN \quad \vee \text{new NEWLEADER} - \text{accept and reply} \\
& \quad \quad \wedge replyOk \\
& \quad \quad \wedge currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch] \\
& \quad \quad \wedge leaderEpoch' = [leaderEpoch \text{ EXCEPT } ![i] = msg.mepoch] \\
& \quad \quad \wedge history' = [history \text{ EXCEPT } ![i] = msg.minitialHistory] \\
& \quad \quad \wedge Reply(i, j, [mtype \mapsto ACKLD, \\
& \quad \quad \quad mepoch \mapsto msg.mepoch, \\
& \quad \quad \quad mhistory \mapsto msg.minitialHistory]) \\
& \quad \vee \text{stale NEWLEADER} - \text{discard} \\
& \quad \quad \wedge \neg replyOk \\
& \quad \quad \wedge Discard(j, i) \\
& \quad \quad \wedge UNCHANGED \langle currentEpoch, leaderEpoch, history \rangle \\
& \wedge UNCHANGED \langle state, commitIndex, leaderOracle, leaderVars, tempVars, cepochSent, recoveryVars, p \rangle
\end{aligned}$$

In phase *l22*, pleader receives *ACK-LD* from a quorum of followers, and sends *COMMIT-LD* to followers.

*LeaderHandleACKLD*(*i*, *j*)  $\triangleq$

- $\wedge \text{state}[i] = \text{ProspectiveLeader}$
- $\wedge \text{msgs}[j][i] \neq \langle \rangle$
- $\wedge \text{msgs}[j][i][1].\text{mtype} = \text{ACKLD}$
- $\wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1]$
- IN  $\vee$  new *ACK-LD* - accept
  - $\wedge \text{currentEpoch}[i] = \text{msg.mepoch}$
  - $\wedge \text{ackIndex}' = [\text{ackIndex} \text{ EXCEPT } ![i][j] = \text{Len}(\text{initialHistory}[i])]$
  - $\wedge \text{ackldRecv}' = [\text{ackldRecv} \text{ EXCEPT } ![i] = \text{IF } j \notin \text{ackldRecv}[i] \text{ THEN } \text{ackldRecv}[i] \cup \{j\} \text{ ELSE } \text{ackldRecv}[i]]$
- $\vee$  stale *ACK-LD* - discard
  - $\wedge \text{currentEpoch}[i] \neq \text{msg.mepoch}$
  - $\wedge \text{UNCHANGED } \langle \text{ackldRecv}, \text{ackIndex} \rangle$
- $\wedge \text{Discard}(j, i)$
- $\wedge \text{UNCHANGED } \langle \text{serverVars}, \text{cluster}, \text{cepocheRecv}, \text{ackeRecv}, \text{currentCounter}, \text{sendCounter}, \text{initialHistory}, \text{committedIndex}, \text{tempVars}, \text{cepocheSent}, \text{recoveryVars}, \text{proposalM} \rangle$

*LeaderSync2*(*i*)  $\triangleq$

- $\wedge \text{state}[i] = \text{ProspectiveLeader}$
- $\wedge \text{ackldRecv}[i] \in \text{Quorums}$
- $\wedge \text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = \text{Len}(\text{history}[i])]$
- $\wedge \text{committedIndex}' = [\text{committedIndex} \text{ EXCEPT } ![i] = \text{Len}(\text{history}[i])]$
- $\wedge \text{state}' = [\text{state} \text{ EXCEPT } ![i] = \text{Leader}]$
- $\wedge \text{currentCounter}' = [\text{currentCounter} \text{ EXCEPT } ![i] = 0]$
- $\wedge \text{sendCounter}' = [\text{sendCounter} \text{ EXCEPT } ![i] = 0]$
- $\wedge \text{ackldRecv}' = [\text{ackldRecv} \text{ EXCEPT } ![i] = \{\text{NullPoint}\}]$
- $\wedge \text{Broadcast}(i, [\text{mtype} \mapsto \text{COMMITLD}, \text{mepoch} \mapsto \text{currentEpoch}[i], \text{mlength} \mapsto \text{Len}(\text{history}[i])])$
- $\wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{leaderEpoch}, \text{leaderOracle}, \text{history}, \text{cluster}, \text{cepocheRecv}, \text{ackeRecv}, \text{ackIndex}, \text{initialHistory}, \text{tempVars}, \text{cepocheSent}, \text{recoveryVars}, \text{proposalM} \rangle$

In phase *f22*, follower receives *COMMIT-LD* and delivers all unprocessed transaction.

*FollowerSync2*(*i*, *j*)  $\triangleq$

- $\wedge \text{state}[i] = \text{Follower}$
- $\wedge \text{msgs}[j][i] \neq \langle \rangle$
- $\wedge \text{msgs}[j][i][1].\text{mtype} = \text{COMMITLD}$
- $\wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1]$
- $\text{replyOk} \triangleq \wedge \text{currentEpoch}[i] = \text{msg.mepoch}$
- $\wedge \text{leaderOracle}[i] = j$
- IN  $\vee$  new *COMMIT-LD* - commit all transactions in initial history
  - Regardless of *Restart*, it must be true because one will receive *NEWLEADER* before receiving *COMMIT-LD*
  - $\wedge \text{replyOk}$
  - $\wedge \vee \wedge \text{Len}(\text{history}[i]) = \text{msg.mlength}$

$$\begin{aligned}
& \wedge \text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = \text{Len}(\text{history}[i])] \\
& \wedge \text{Discard}(j, i) \\
& \vee \wedge \text{Len}(\text{history}[i]) \neq \text{msg.mlength} \\
& \quad \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{CEPOCH}, \\
& \quad \quad \text{mepoch} \mapsto \text{currentEpoch}[i]]) \\
& \quad \wedge \text{UNCHANGED } \text{commitIndex} \\
\vee & \text{ } > : \text{stale COMMIT-LD - discard} \\
& < : \text{In our implementation, ' < ' does not exist due to the guarantee of Restart} \\
& \wedge \neg \text{replyOk} \\
& \wedge \text{Discard}(j, i) \\
& \quad \wedge \text{UNCHANGED } \text{commitIndex} \\
& \wedge \text{UNCHANGED } \langle \text{state}, \text{currentEpoch}, \text{leaderEpoch}, \text{leaderOracle}, \text{history}, \\
& \quad \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog}, \text{testVars} \rangle
\end{aligned}$$


---

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

$$\begin{aligned}
\text{ClientRequest}(i, v) & \triangleq \\
& \text{test restrictions} \\
& \wedge \text{Len}(\text{history}[i]) < \text{MaxTransactionNum} \\
& \wedge \text{state}[i] = \text{Leader} \\
& \wedge \text{currentCounter}' = [\text{currentCounter} \text{ EXCEPT } ![i] = \text{currentCounter}[i] + 1] \\
& \wedge \text{LET } \text{newTransaction} \triangleq [\text{epoch} \mapsto \text{currentEpoch}[i], \\
& \quad \text{counter} \mapsto \text{currentCounter}'[i], \\
& \quad \text{value} \mapsto v] \\
& \text{IN } \wedge \text{history}' = [\text{history} \text{ EXCEPT } ![i] = \text{Append}(\text{history}[i], \text{newTransaction})] \\
& \quad \wedge \text{ackIndex}' = [\text{ackIndex} \text{ EXCEPT } ![i][i] = \text{Len}(\text{history}'[i])] \text{ necessary, to push commitIndex} \\
& \wedge \text{UNCHANGED } \langle \text{msgs}, \text{state}, \text{currentEpoch}, \text{leaderEpoch}, \text{leaderOracle}, \text{commitIndex}, \text{cluster}, \text{ceepochR}, \\
& \quad \text{ackRecv}, \text{ackldRecv}, \text{sendCounter}, \text{initialHistory}, \text{committedIndex}, \text{tempVars}, \text{ceepoch} \rangle
\end{aligned}$$

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

In phase *f31*, follower accepts proposal and append it to history.

$$\text{FollowerBroadcast1}(i, j) \triangleq$$

$$\begin{aligned}
& \wedge state[i] = \text{Follower} \\
& \wedge msgs[j][i] \neq \langle \rangle \\
& \wedge msgs[j][i][1].mtype = \text{PROPOSE} \\
& \wedge \text{LET } msg \triangleq msgs[j][i][1] \\
& \quad \text{replyOk} \triangleq \wedge currentEpoch[i] = msg.mepoch \\
& \quad \quad \wedge leaderOracle[i] = j \\
& \quad \text{infoOk} \triangleq \vee \wedge msg.mproposal.counter = 1 \quad \text{the first PROPOSE in this epoch} \\
& \quad \quad \wedge \vee Len(history[i]) = 0 \\
& \quad \quad \quad \vee \wedge Len(history[i]) > 0 \\
& \quad \quad \quad \quad \wedge history[i][Len(history[i])].epoch < msg.mepoch \\
& \quad \vee \wedge msg.mproposal.counter > 1 \quad \text{not the first PROPOSE in this epoch} \\
& \quad \quad \wedge Len(history[i]) > 0 \\
& \quad \quad \wedge history[i][Len(history[i])].epoch = msg.mepoch \\
& \quad \quad \wedge history[i][Len(history[i])].counter = msg.mproposal.counter - 1 \\
\text{IN } & \vee \wedge \text{replyOk} \\
& \quad \wedge \vee \wedge \text{infoOk} \\
& \quad \quad \wedge history' = [history \text{ EXCEPT } ![i] = \text{Append}(history[i], msg.mproposal)] \\
& \quad \quad \wedge \text{Reply}(i, j, [mtype \mapsto \text{ACK}, \\
& \quad \quad \quad mepoch \mapsto currentEpoch[i], \\
& \quad \quad \quad mindex \mapsto Len(history'[i])]) \\
& \quad \vee \wedge \neg \text{infoOk} \\
& \quad \quad \wedge \text{Reply}(i, j, [mtype \mapsto \text{CEPOCH}, \\
& \quad \quad \quad mepoch \mapsto currentEpoch[i]]) \\
& \quad \quad \wedge \text{UNCHANGED } history \\
& \vee \wedge \neg \text{replyOk} \\
& \quad \wedge \text{Discard}(j, i) \\
& \quad \wedge \text{UNCHANGED } history \\
& \wedge \text{UNCHANGED } \langle state, currentEpoch, leaderEpoch, leaderOracle, commitIndex, \\
& \quad leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog, testVars \rangle
\end{aligned}$$

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

$$\begin{aligned}
& \text{LeaderHandleACK}(i, j) \triangleq \\
& \quad \wedge state[i] = \text{Leader} \\
& \quad \wedge msgs[j][i] \neq \langle \rangle \\
& \quad \wedge msgs[j][i][1].mtype = \text{ACK} \\
& \quad \wedge \text{LET } msg \triangleq msgs[j][i][1] \\
& \quad \text{IN } \vee \text{It should be that } ackIndex[i][j] + 1 \triangleq msg.mindex \\
& \quad \quad \wedge currentEpoch[i] = msg.mepoch \\
& \quad \quad \wedge ackIndex' = [ackIndex \text{ EXCEPT } ![i][j] = \text{Maximum}(\{ackIndex[i][j], msg.mindex\})] \\
& \quad \vee \text{If happens, } \neq \text{ must be } > , \text{ namely a stale follower sends it.} \\
& \quad \quad \wedge currentEpoch[i] \neq msg.mepoch \\
& \quad \quad \wedge \text{UNCHANGED } ackIndex \\
& \quad \wedge \text{Discard}(j, i) \\
& \quad \wedge \text{UNCHANGED } \langle serverVars, cluster, cepochRecv, ackeRecv, ackldRecv, currentCounter,
\end{aligned}$$

$sendCounter, initialHistory, committedIndex, tempVars, cepochSent, recoveryVars, p$

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

$LeaderBroadcast2(i) \triangleq$   
 $\wedge state[i] = Leader$   
 $\wedge committedIndex[i] < commitIndex[i]$   
 $\wedge LET newCommittedIndex \triangleq committedIndex[i] + 1$   
 $IN \wedge Broadcast(i, [mtype \mapsto COMMIT,$   
 $mepoch \mapsto currentEpoch[i],$   
 $mindex \mapsto newCommittedIndex,$   
 $mcounter \mapsto history[i][newCommittedIndex].counter])$   
 $\wedge committedIndex' = [committedIndex \text{ EXCEPT } ![i] = committedIndex[i] + 1]$   
 $\wedge UNCHANGED \langle serverVars, cluster, cepochRecv, ackRecv, ackldRecv, ackIndex, currentCounter,$   
 $sendCounter, initialHistory, tempVars, cepochSent, recoveryVars, proposalMsgsLog, t$

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

$FollowerBroadcast2(i, j) \triangleq$   
 $\wedge state[i] = Follower$   
 $\wedge msgs[j][i] \neq \langle \rangle$   
 $\wedge msgs[j][i][1].mtype = COMMIT$   
 $\wedge msgs[j][i][1].mtype = COMMIT$   
 $\wedge LET msg \triangleq msgs[j][i][1]$   
 $replyOk \triangleq \wedge currentEpoch[i] = msg.mepoch$   
 $\wedge leaderOracle[i] = j$   
 $IN \vee \wedge replyOk$   
 $\wedge LET infoOk \triangleq \wedge Len(history[i]) \geq msg.mindex$   
 $\wedge \vee \wedge msg.mindex > 0$   
 $\wedge history[i][msg.mindex].epoch = msg.mepoch$   
 $\wedge history[i][msg.mindex].counter = msg.mcounter$   
 $\vee msg.mindex = 0$   
 $IN \vee \wedge new COMMIT - \text{commit transaction in history}$   
 $\wedge infoOk$   
 $\wedge commitIndex' = [commitIndex \text{ EXCEPT } ![i] = Maximum(\{commitIndex[i], msg.mindex\})]$   
 $\wedge Discard(j, i)$   
 $\vee \wedge \text{It may happen when the server is a new follower who joined in the cluster,}$   
 $\text{and it misses the corresponding } PROPOSE.$

$$\begin{aligned}
& \wedge \neg infoOk \\
& \wedge Reply(i, j, [mtype \mapsto CEPOCH, \\
& \quad mepoch \mapsto currentEpoch[i]]) \\
& \wedge UNCHANGED commitIndex \\
\vee & \text{stale } COMMIT - \text{discard} \\
& \wedge \neg replyOk \\
& \wedge Discard(j, i) \\
& \wedge UNCHANGED commitIndex \\
& \wedge UNCHANGED \langle state, currentEpoch, leaderEpoch, history, leaderOracle, \\
& \quad leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog, testVars \rangle
\end{aligned}$$

---

There may be two ways to make sure all followers as up-to-date as the leader.  
*way1*: choose *Send* not *Broadcast* when leader is going to send *PROPOSE* and *COMMIT*.  
*way2*: When one follower receives *PROPOSE* or *COMMIT* which misses some entries between  
its history and the newest entry, the follower send *CEPOCH* to catch pace.  
Here I choose *way2*, which I need not to rewrite *PROPOSE* and *COMMIT*, but need to  
modify the code when follower receives *COMMIT-LD* and *COMMIT*.

In phase *l33*, upon receiving *CEPOCH*, leader *l* proposes back *NEWEPOCH* and *NEWLEADER*.  
 $LeaderHandleCEPOCHinPhase3(i, j) \triangleq$

$$\begin{aligned}
& \wedge state[i] = Leader \\
& \wedge msgs[j][i] \neq \langle \rangle \\
& \wedge msgs[j][i][1].mtype = CEPOCH \\
& \wedge LET msg \triangleq msgs[j][i][1] \\
& \quad IN \quad \vee \wedge currentEpoch[i] \geq msg.mepoch \\
& \quad \quad \wedge Reply2(i, j, [mtype \mapsto NEWEPOCH, \\
& \quad \quad \quad mepoch \mapsto currentEpoch[i], \\
& \quad \quad \quad [mtype \mapsto NEWLEADER, \\
& \quad \quad \quad mepoch \mapsto currentEpoch[i], \\
& \quad \quad \quad minitialHistory \mapsto history[i]]) \\
& \quad \wedge LET m \triangleq [msource \mapsto i, mtype \mapsto NEWLEADER, mepoch \mapsto currentEpoch[i], mproposal \\
& \quad \quad IN \quad proposalMsgsLog' = IF m \in proposalMsgsLog THEN proposalMsgsLog \\
& \quad \quad \quad ELSE proposalMsgsLog \cup \{m\} \\
& \vee \wedge currentEpoch[i] < msg.mepoch \\
& \quad \wedge UNCHANGED \langle msgs, proposalMsgsLog \rangle \\
& \wedge UNCHANGED \langle serverVars, leaderVars, tempVars, cepochSent, recoveryVars, testVars \rangle
\end{aligned}$$

In phase *l34*, upon receiving ack from *f* of the *NEWLEADER*, it sends a commit message to *f*.  
Leader *l* also makes  $Q := Q \cup \{f\}$ .  
 $LeaderHandleACKLDinPhase3(i, j) \triangleq$

$$\begin{aligned}
& \wedge state[i] = Leader \\
& \wedge msgs[j][i] \neq \langle \rangle \\
& \wedge msgs[j][i][1].mtype = ACKLD \\
& \wedge LET msg \triangleq msgs[j][i][1] \\
& \quad aimCommitIndex \triangleq Minimum(\{commitIndex[i], Len(msg.mhistory)\})
\end{aligned}$$



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

To ensure any follower can find the correct leader, the follower should modify *leaderOracle* anytime when it receive messages from leader, because a server may restart and join the cluster *Q* halfway and receive the first message which is not *NEWEPOCH*. But we can delete this restriction when we ensure *Broadcast* function acts on the followers in the cluster not any servers in the whole system, then one server must has correct *leaderOracle* before it receives messages.

Let me suppose two conditions when one follower sends *CEPOCH* to leader:

0. Usually, the server becomes follower in election and sends *CEPOCH* before receiving *NEWEPOCH*.
1. The follower wants to join the cluster halfway and get the newest history.
2. The follower has received *COMMIT*, but there exists the gap between its own history and *mindex*, which means there are some transactions before *mindex* miss. Here we choose to send *CEPOCH* again, to receive the newest history from leader.

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

Here we should not use *Discard*.

$\wedge \text{UNCHANGED } \langle \text{leaderEpoch}, \text{history}, \text{commitIndex}, \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars},$

---

$\text{DiscardStaleMessage}(i) \triangleq$

$\wedge \exists j \in \text{Server} \setminus \{i\} : \wedge \text{msgs}[j][i] \neq \langle \rangle$

$\wedge \text{msgs}[j][i][1].\text{mtype} \neq \text{RECOVERYREQUEST}$

$\wedge \text{msgs}[j][i][1].\text{mtype} \neq \text{RECOVERYRESPONSE}$

$\wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1]$

IN  $\vee \wedge \text{state}[i] = \text{Follower}$

$\wedge \vee \text{msg.mepoch} < \text{currentEpoch}[i] \setminus * \text{ Discussed before.}$

$\vee \text{msg.mtype} = \text{CEPOCH}$

$\vee \text{msg.mtype} = \text{ACKE}$

$\vee \text{msg.mtype} = \text{ACKLD}$

$\vee \text{msg.mtype} = \text{ACK}$

$\vee \wedge \text{state}[i] \neq \text{Follower}$

$\wedge \text{msg.mtype} \neq \text{CEPOCH}$

$\wedge \vee \wedge \text{state}[i] = \text{ProspectiveLeader}$

$\wedge \vee \text{msg.mtype} = \text{ACK}$

$\vee \wedge \text{msg.mepoch} \leq \text{Maximum}(\{\text{currentEpoch}[i], \text{leaderEpoch}[i]\})$

$\wedge \vee \text{msg.mtype} = \text{NEWPOCH}$

$\vee \text{msg.mtype} = \text{NEWLEADER}$

$\vee \text{msg.mtype} = \text{COMMITLD}$

$\vee \text{msg.mtype} = \text{PROPOSE}$

$\vee \text{msg.mtype} = \text{COMMIT}$

$\vee \wedge \text{state}[i] = \text{Leader}$

$\wedge \vee \text{msg.mtype} = \text{ACKE}$

$\vee \wedge \text{msg.mepoch} \leq \text{currentEpoch}[i]$

$\wedge \vee \text{msg.mtype} = \text{NEWPOCH}$

$\vee \text{msg.mtype} = \text{NEWLEADER}$

$\vee \text{msg.mtype} = \text{COMMITLD}$

$\vee \text{msg.mtype} = \text{PROPOSE}$

$\vee \text{msg.mtype} = \text{COMMIT}$

$\wedge \text{Discard}(j, i)$

$\wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog}, \text{testV}$

---

Defines how the variables may transition.

$\text{Next} \triangleq$

$\vee \exists i \in \text{Server}, Q \in \text{Quorums} : \text{InitialElection}(i, Q)$

$\vee \exists i \in \text{Server} : \text{Restart}(i)$

$\vee \exists i \in \text{Server} : \text{RecoveryAfterRestart}(i)$

$\vee \exists i, j \in \text{Server} : \text{HandleRecoveryRequest}(i, j)$

$\vee \exists i, j \in \text{Server} : \text{HandleRecoveryResponse}(i, j)$

$\vee \exists i, j \in \text{Server} : \text{FindCluster}(i)$

$$\begin{aligned}
& \vee \exists i, j \in \text{Server} : \text{LeaderTimeout}(i, j) \\
& \vee \exists i \in \text{Server} : \text{FollowerTimeout}(i) \\
& \vee \exists i \in \text{Server} : \text{FollowerDiscovery1}(i) \\
& \vee \exists i, j \in \text{Server} : \text{LeaderHandleCEPOCH}(i, j) \\
& \vee \exists i \in \text{Server} : \text{LeaderDiscovery1}(i) \\
& \vee \exists i, j \in \text{Server} : \text{FollowerDiscovery2}(i, j) \\
& \vee \exists i, j \in \text{Server} : \text{LeaderHandleACKE}(i, j) \\
& \vee \exists i \in \text{Server} : \text{LeaderDiscovery2Sync1}(i) \\
& \vee \exists i, j \in \text{Server} : \text{FollowerSync1}(i, j) \\
& \vee \exists i, j \in \text{Server} : \text{LeaderHandleACKLD}(i, j) \\
& \vee \exists i \in \text{Server} : \text{LeaderSync2}(i) \\
& \vee \exists i, j \in \text{Server} : \text{FollowerSync2}(i, j) \\
& \vee \exists i \in \text{Server}, v \in \text{Value} : \text{ClientRequest}(i, v) \\
& \vee \exists i \in \text{Server} : \text{LeaderBroadcast1}(i) \\
& \vee \exists i, j \in \text{Server} : \text{FollowerBroadcast1}(i, j) \\
& \vee \exists i, j \in \text{Server} : \text{LeaderHandleACK}(i, j) \\
& \vee \exists i \in \text{Server} : \text{LeaderAdvanceCommit}(i) \\
& \vee \exists i \in \text{Server} : \text{LeaderBroadcast2}(i) \\
& \vee \exists i, j \in \text{Server} : \text{FollowerBroadcast2}(i, j) \\
& \vee \exists i, j \in \text{Server} : \text{LeaderHandleCEPOCHinPhase3}(i, j) \\
& \vee \exists i, j \in \text{Server} : \text{LeaderHandleACKLDinPhase3}(i, j) \\
& \vee \exists i \in \text{Server} : \text{DiscardStaleMessage}(i) \\
& \vee \exists i \in \text{Server} : \text{BecomeFollower}(i)
\end{aligned}$$

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

---

Define some variants, safety propoties, and liveness propoties of *Zab* consensus algorithm.

Safety properties

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

$\text{Leadership} \triangleq \forall i, j \in \text{Server} :$

$$\begin{aligned}
& \wedge \vee \text{state}[i] = \text{Leader} \\
& \quad \vee \wedge \text{state}[i] = \text{ProspectiveLeader} \\
& \quad \quad \wedge \text{NullPoint} \in \text{ackRecv}[i] \quad \text{prospective leader determines its epoch after broadcasting NEWLE} \\
& \wedge \vee \text{state}[j] = \text{Leader} \\
& \quad \vee \wedge \text{state}[j] = \text{ProspectiveLeader} \\
& \quad \quad \wedge \text{NullPoint} \in \text{ackRecv}[j] \\
& \wedge \text{currentEpoch}[i] = \text{currentEpoch}[j] \\
& \Rightarrow i = j
\end{aligned}$$

Here, delivering means deliver some transaction from history to replica. We can assume  $\text{deliverIndex} = \text{commitIndex}$ .

So we can assume the set of delivered transactions is the prefix of history with index from 1 to  $\text{commitIndex}$ .

We can express a transaction by two-tuple  $\langle \text{epoch}, \text{counter} \rangle$  according to its uniqueness.

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

$$\begin{aligned}
\text{precede}(\text{entry1}, \text{entry2}) &\triangleq \vee \text{entry1.epoch} < \text{entry2.epoch} \\
&\vee \wedge \text{entry1.epoch} = \text{entry2.epoch} \\
&\wedge \text{entry1.counter} < \text{entry2.counter}
\end{aligned}$$

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

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

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

$$\begin{aligned}
\text{Integrity} &\triangleq \forall i \in \text{Server} : \\
&\text{state}[i] = \text{Follower} \wedge \text{commitIndex}[i] > 0 \\
&\Rightarrow \forall \text{index} \in 1 \dots \text{commitIndex}[i] : \exists \text{msg} \in \text{proposalMsgsLog} : \\
&\vee \wedge \text{msg.mtype} = \text{PROPOSE} \\
&\wedge \text{equal}(\text{msg.mproposal}, \text{history}[i][\text{index}]) \\
&\vee \wedge \text{msg.mtype} = \text{NEWLEADER} \\
&\wedge \exists \text{pindex} \in 1 \dots \text{Len}(\text{msg.mproposals}) : \text{equal}(\text{msg.mproposals}[\text{pindex}], \text{history}[i][\text{index}])
\end{aligned}$$

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

$$\begin{aligned}
\text{Agreement} &\triangleq \forall i, j \in \text{Server} : \\
&\wedge \text{state}[i] = \text{Follower} \wedge \text{commitIndex}[i] > 0 \\
&\wedge \text{state}[j] = \text{Follower} \wedge \text{commitIndex}[j] > 0 \\
&\Rightarrow \\
&\forall \text{index1} \in 1 \dots \text{commitIndex}[i], \text{index2} \in 1 \dots \text{commitIndex}[j] : \\
&\vee \exists \text{indexj} \in 1 \dots \text{commitIndex}[j] : \\
&\quad \text{equal}(\text{history}[j][\text{indexj}], \text{history}[i][\text{index1}]) \\
&\vee \exists \text{indexi} \in 1 \dots \text{commitIndex}[i] : \\
&\quad \text{equal}(\text{history}[i][\text{indexi}], \text{history}[j][\text{index2}])
\end{aligned}$$

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

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

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

$$\text{LocalPrimaryOrder} \triangleq \text{LET } \text{mset}(i, e) \triangleq \{\text{msg} \in \text{proposalMsgsLog} : \wedge \text{msg.mtype} = \text{PROPOSE}$$

$$\begin{aligned}
& \wedge msg.msource = i \\
& \wedge msg.mepoch = e\} \\
mentries(i, e) & \triangleq \{msg.mproposal : msg \in mset(i, e)\} \\
IN \quad \forall i \in Server : \forall e \in 1 \dots currentEpoch[i] : \\
& \vee Cardinality(mentries(i, e)) < 2 \\
& \vee \wedge Cardinality(mentries(i, e)) \geq 2 \\
& \wedge \exists tsc1, tsc2 \in mentries(i, e) : \\
& \vee equal(tsc1, tsc2) \\
& \vee \wedge \neg equal(tsc1, tsc2) \\
& \wedge LET \quad tscPre \triangleq IF precede(tsc1, tsc2) THEN tsc1 ELSE \quad tsc2 \\
& \quad \quad tscNext \triangleq IF precede(tsc1, tsc2) THEN tsc2 ELSE \quad tsc1 \\
& IN \quad \forall j \in Server : \wedge commitIndex[j] \geq 2 \\
& \quad \wedge \exists index \in 1 \dots commitIndex[j] : equal(history[j][index], tscPre) \\
& \Rightarrow \exists index2 \in 1 \dots commitIndex[j] : \\
& \quad \wedge equal(history[j][index2], tscNext) \\
& \quad \wedge index2 > 1 \\
& \quad \wedge \exists index1 \in 1 \dots (index2 - 1) : equal(history[j][index1], tscPre)
\end{aligned}$$

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

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

Primary integrity: If primary  $p$  broadcasts  $a$  and some follower  $f$  delivers  $b$  such that  $b$  has epoch smaller than epoch of  $p$ , then  $p$  must deliver  $b$  before it broadcasts  $a$ .

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

Liveness property

Suppose that :

- A quorum  $Q$  of followers are up.
- The followers in  $Q$  elect the same process  $l$  and  $l$  is up.
- Messages between a follower in  $Q$  and  $l$  are received in a timely fashion.

If  $l$  proposes a transaction  $a$ , then  $a$  is eventually committed.

---

\ \* Modification History  
\ \* Last modified Sat May 08 20:33:14 CST 2021 by Dell  
\ \* Created Sat Dec 05 13:32:08 CST 2020 by Dell