MODULE *FastLeaderElection*

This is the formal specification for Fast Leader Election in *Zab* protocol.

Reference: *FastLeaderElection.java*, *Vote.java*, *QuorumPeer.java* in https://*github.com*/apache/zookeeper. Medeiros A. *ZooKeeper*'s atomic broadcast protocol: Theory and *practice*[*J*]. Aalto University School of Science, 2012.

EXTENDS *Integers*, *FiniteSets*, *Sequences*, *Naturals*, *TLC*

The set of server identifiers

CONSTANT *Server*

Server states

CONSTANTS *LOOKING*, *FOLLOWING*, *LEADING*

NOTE: In spec, we do not discuss servers whose *ServerState* is OBSERVING.

Message types

CONSTANTS *NOTIFICATION*

Timeout signal

CONSTANT *NONE*

$Quorums \triangleq \{Q \in \text{SUBSET } Server : Cardinality(Q) * 2 > Cardinality(Server)\}$

$NullPoint \triangleq \text{CHOOSE } p : p \notin Server$

Server's *state*(*LOOKING*, *FOLLOWING*, *LEADING*).

VARIABLE *state*

VARIABLE *history*

The epoch number of the last *NEWLEADER* packet accepted, used for comparing.

VARIABLE *currentEpoch*

The index and *zxid* of the last processed transaction in history.

VARIABLE *lastProcessed*

currentVote[*i*]: The server who *i* thinks is the current *leader*(*id*, *zxid*, *peerEpoch*, ...).

VARIABLE *currentVote*

Election instance.(*logicalClock in code*)

VARIABLE *logicalClock*

The votes from the current leader election are stored in *ReceiveVotes*.

VARIABLE *receiveVotes*

The votes from previous leader elections, as well as the votes from the current leader election are stored in outofelection. Note that notifications in a *LOOKING* state are not stored in outofelection. Only *FOLLOWING* or *LEADING* notifications are stored in outofelection.

VARIABLE *outOfElection*

recvQueue[*i*]: The queue of received notifications or timeout signals in server *i*.
VARIABLE *recvQueue*

A veriable to wait for new notifications, corresponding to line 1050 in *FastLeaderElection.java*.
VARIABLE *waitNotmsg*

leadingVoteSet[*i*]: The set of voters that follow *i*.
VARIABLE *leadingVoteSet*

The messages about election sent from one server to another. *electionMsgs*[*i*][*j*] means the input buffer of server *j* from server *i*.
VARIABLE *electionMsgs*

Set used for mapping *Server* to *Integers*, to compare ids from different servers.
VARIABLE *idTable*

$serverVarsL \triangleq \langle state,\ currentEpoch,\ lastProcessed,\ history \rangle$

$electionVarsL \triangleq \langle currentVote,\ logicalClock,\ receiveVotes,\ outOfElection,\ recvQueue,\ waitNotmsg \rangle$

$leaderVarsL \triangleq \langle leadingVoteSet \rangle$

$varsL \triangleq \langle serverVarsL,\ electionVarsL,\ leaderVarsL,\ electionMsgs \rangle$

Processing of *electionMsgs*

$BroadcastNotmsg(i,\ m) \triangleq electionMsgs' = [electionMsgs \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } v \neq i$
$\text{THEN } Append(electi$
$\text{ELSE } electionMsgs[i$

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

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

Processing of *recvQueue*

RECURSIVE $RemoveNone(\_)$

$RemoveNone(seq) \triangleq \text{CASE } seq = \langle\rangle \to \langle\rangle$
$\square \quad seq \neq \langle\rangle \to \text{IF } Head(seq).mtype = NONE \text{ THEN } RemoveNone(Tail(seq))$
$\text{ELSE } \langle Head(seq)\rangle \circ RemoveNone(Tail$

Processing of *idTable* and order comparing

RECURSIVE $InitializeIdTable(\_)$

$InitializeIdTable(Remaining) \triangleq \text{IF } Remaining = \{\} \text{ THEN } \{\}$
$\text{ELSE LET } chosen \triangleq \text{CHOOSE } i \in Remaining : \text{TRUE}$
$re \triangleq Remaining \setminus \{chosen\}$
$\text{IN } \{\langle chosen,\ Cardinality(Remaining)\rangle\} \cup InitializeIdTable(re)$

$IdTable \triangleq InitializeIdTable(Server)$

FALSE: $id1 < id2$; TRUE: $id1 > id2$

$$\begin{array}{l} IdCompare(id1,\ id2) \triangleq \text{LET } item1 \triangleq \text{CHOOSE } item \in IdTable : item[1] = id1 \\ \qquad\qquad\qquad\qquad\quad item2 \triangleq \text{CHOOSE } item \in IdTable : item[1] = id2 \\ \qquad\qquad\qquad\quad \text{IN } \ item1[2] > item2[2] \end{array}$$

FALSE: $zxid1 \leq zxid2$; TRUE: $zxid1 > zxid2$

$$\begin{array}{l} ZxidCompare(zxid1,\ zxid2) \triangleq \ \lor\ zxid1[1] > zxid2[1] \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \lor\ \land\ zxid1[1] = zxid2[1] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \ \land\ zxid1[2] > zxid2[2] \end{array}$$

$$ZxidEqual(zxid1,\ zxid2) \triangleq zxid1[1] = zxid2[1] \land zxid1[2] = zxid2[2]$$

FALSE: $vote1 \leq vote2$; TRUE: $vote1 > vote2$

$$\begin{array}{l} TotalOrderPredicate(vote1,\ vote2) \triangleq \ \lor\ vote1.proposedEpoch > vote2.proposedEpoch \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \lor\ \land\ vote1.proposedEpoch = vote2.proposedEpoch \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ \lor\ ZxidCompare(vote1.proposedZxid,\ vote2.proposedZxid) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ \land\ ZxidEqual(vote1.proposedZxid,\ vote2.proposedZxid) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ IdCompare(vote1.proposedLeader,\ vote2.proposedLeader) \end{array}$$

$$\begin{array}{l} VoteEqual(vote1,\ round1,\ vote2,\ round2) \triangleq \ \land\ vote1.proposedLeader = vote2.proposedLeader \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ ZxidEqual(vote1.proposedZxid,\ vote2.proposedZxid) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ vote1.proposedEpoch\ = vote2.proposedEpoch \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ round1 = round2 \end{array}$$

$$\begin{array}{l} InitLastProcessed(i) \triangleq \text{IF } Len(history[i]) = 0 \text{ THEN } [index \mapsto 0, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad zxid \mapsto \langle 0,\ 0\rangle] \\ \qquad\qquad\qquad\qquad\quad \text{ELSE} \\ \qquad\qquad\qquad\qquad\quad \text{LET } lastIndex \triangleq Len(history[i]) \\ \qquad\qquad\qquad\qquad\qquad\ \ entry \quad\ \triangleq history[i][lastIndex] \\ \qquad\qquad\qquad\qquad\quad \text{IN } \ [index \mapsto lastIndex, \\ \qquad\qquad\qquad\qquad\qquad\ \ zxid \ \mapsto entry.zxid] \end{array}$$

RECURSIVE $InitAcksidInTxns(\_,\ \_)$

$$\begin{array}{l} InitAcksidInTxns(txns,\ src) \triangleq \text{IF } Len(txns) = 0 \text{ THEN } \langle\rangle \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{ELSE LET } newTxn \triangleq [zxid \quad \mapsto txns[1].zxid, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad value \ \mapsto txns[1].value, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ackSid \mapsto \{src\}, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad epoch \ \mapsto txns[1].epoch] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{IN } \ \langle newTxn\rangle \circ InitAcksidInTxns(Tail(txns),\ src) \end{array}$$

$$\begin{array}{l} InitHistory(i) \triangleq \text{LET } newState \triangleq state'[i] \text{IN} \\ \qquad\qquad\qquad\qquad\quad \text{IF } newState = LEADING \text{ THEN } InitAcksidInTxns(history[i],\ i) \\ \qquad\qquad\qquad\qquad\quad \text{ELSE } history[i] \end{array}$$

Processing of *currentVote*

$$InitialVote \triangleq [proposedLeader \mapsto NullPoint,\ proposedZxid \mapsto \langle 0, 0\rangle,\ proposedEpoch \mapsto 0]$$

$$SelfVote(i) \triangleq [proposedLeader \mapsto i,\ proposedZxid \mapsto lastProcessed[i].zxid,\ proposedEpoch \mapsto currentEpoch[i]]$$

$$UpdateProposal(i, nid, nzxid, nepoch) \triangleq currentVote' = [currentVote \text{ EXCEPT } ![i].proposedLeader = nid,\ ![i].proposedZxid = nzxid,\ ![i].proposedEpoch = nepoch]$$

Processing of *receiveVotes* and *outOfElection*

$$RvClear(i) \triangleq receiveVotes' = [receiveVotes \text{ EXCEPT } ![i] = [v \in Server \mapsto [vote \mapsto InitialVote,\ round \mapsto 0,\ state \mapsto LOOKING,\ version \mapsto 0]]]$$

$$RvPut(i, id, mvote, mround, mstate) \triangleq receiveVotes' = \text{CASE } receiveVotes[i][id].round < mround \to [receive$$

$$\square \quad receiveVotes[i][id].round = mround \to [receive$$

$$\square \quad receiveVotes[i][id].round > mround \to receive$$

$$Put(i, id, rcvset, mvote, mround, mstate) \triangleq \text{CASE } rcvset[id].round < mround \to [rcvset \text{ EXCEPT } ![id].vote ![id].round ![id].state ![id].versio$$

$$\square \quad rcvset[id].round = mround \to [rcvset \text{ EXCEPT } ![id].vote ![id].state ![id].versio$$

$$\square \quad rcvset[id].round > mround \to rcvset$$

$$RvClearAndPut(i, id, vote, round) \triangleq receiveVotes' = \text{LET } oneVote \triangleq [vote \mapsto vote,\ round \mapsto round,\ state \mapsto LOOKING,\ version \mapsto 1] \text{ IN } [receiveVotes \text{ EXCEPT } ![i] = [v \in Server \mapsto \text{IF } v =$$

$$VoteSet(i,\ msource,\ rcvset,\ thisvote,\ thisround) \triangleq \{msource\} \cup \{s \in (Server \setminus \{msource\}) : VoteEqual(rcvset$$
$$rcvset$$
$$thisvo$$
$$thisro$$

$$HasQuorums(i,\ msource,\ rcvset,\ thisvote,\ thisround) \triangleq \text{LET } Q \triangleq VoteSet(i,\ msource,\ rcvset,\ thisvote,\ thisr$$
$$\text{IN IF } Q \in Quorums \text{ THEN TRUE ELSE FALSE}$$

$$CheckLeader(i,\ votes,\ thisleader,\ thisround) \triangleq \text{IF } thisleader = i \text{ THEN (IF } thisround = logicalClock[i] \text{ THEN T}$$
$$\text{ELSE (IF } votes[thisleader].vote.proposedLeader = NullPoint$$
$$\text{ELSE (IF } votes[thisleader].state = LEADING \text{ THEN}$$
$$\text{ELSE}$$

$$OoeClear(i) \triangleq outOfElection' = [outOfElection \text{ EXCEPT } ![i] = [v \in Server \mapsto [vote \mapsto InitialVote,$$
$$round \mapsto 0,$$
$$state \mapsto LOOKING,$$
$$version \mapsto 0]]]$$

$$OoePut(i,\ id,\ mvote,\ mround,\ mstate) \triangleq outOfElection' = \text{CASE } outOfElection[i][id].round < mround \to [ou$$

$$\square \quad outOfElection[i][id].round = mround \to [ou$$

$$\square \quad outOfElection[i][id].round > mround \to out$$

$$InitServerVarsL \triangleq \land state = [s \in Server \mapsto LOOKING]$$
$$\land currentEpoch = [s \in Server \mapsto 0]$$
$$\land lastProcessed = [s \in Server \mapsto [index \mapsto 0,$$
$$zxid \mapsto \langle 0, 0\rangle]]$$
$$\land history = [s \in Server \mapsto \langle\rangle]$$

$$InitElectionVarsL \triangleq \land currentVote = [s \in Server \mapsto SelfVote(s)]$$
$$\land logicalClock = [s \in Server \mapsto 0]$$
$$\land receiveVotes = [s \in Server \mapsto [v \in Server \mapsto [vote \mapsto InitialVote,$$
$$round \mapsto 0,$$
$$state \mapsto LOOKING,$$
$$version \mapsto 0]]]$$
$$\land outOfElection = [s \in Server \mapsto [v \in Server \mapsto [vote \mapsto InitialVote,$$
$$round \mapsto 0,$$
$$state \mapsto LOOKING,$$
$$version \mapsto 0]]]$$
$$\land recvQueue = [s \in Server \mapsto \langle\rangle]$$
$$\land waitNotmsg = [s \in Server \mapsto \text{FALSE}]$$

$InitLeaderVarsL \triangleq leadingVoteSet = [s \in Server \mapsto \{\}]$

$InitL \triangleq \land InitServerVarsL$
$\quad \land InitElectionVarsL$
$\quad \land InitLeaderVarsL$
$\quad \land electionMsgs = [s \in Server \mapsto [v \in Server \mapsto \langle\rangle]]$
$\quad \land idTable = InitializeIdTable(Server)$

The beginning part of *FLE*'s main function *lookForLeader*()

$ZabTimeout(i) \triangleq$
$\quad \land state[i] \in \{LEADING, FOLLOWING\}$
$\quad \land state' = [state \text{ EXCEPT } ![i] = LOOKING]$
$\quad \land lastProcessed' = [lastProcessed \text{ EXCEPT } ![i] = InitLastProcessed(i)]$
$\quad \land logicalClock' = [logicalClock \text{ EXCEPT } ![i] = logicalClock[i] + 1]$
$\quad \land currentVote' = [currentVote \text{ EXCEPT } ![i] = [proposedLeader \mapsto i,$
$\qquad proposedZxid \mapsto lastProcessed'[i].zxid,$
$\qquad proposedEpoch \mapsto currentEpoch[i]]]$
$\quad \land receiveVotes' = [receiveVotes \text{ EXCEPT } ![i] = [v \in Server \mapsto [vote \mapsto InitialVote,$
$\qquad round \mapsto 0,$
$\qquad state \mapsto LOOKING,$
$\qquad version \mapsto 0]]]$
$\quad \land outOfElection' = [outOfElection \text{ EXCEPT } ![i] = [v \in Server \mapsto [vote \mapsto InitialVote,$
$\qquad round \mapsto 0,$
$\qquad state \mapsto LOOKING,$
$\qquad version \mapsto 0]]]$
$\quad \land recvQueue' = [recvQueue \text{ EXCEPT } ![i] = \langle\rangle]$
$\quad \land waitNotmsg' = [waitNotmsg \text{ EXCEPT } ![i] = \text{FALSE}]$
$\quad \land leadingVoteSet' = [leadingVoteSet \text{ EXCEPT } ![i] = \{\}]$
$\quad \land BroadcastNotmsg(i, [mtype \mapsto NOTIFICATION,$
$\qquad msource \mapsto i,$
$\qquad mstate \mapsto LOOKING,$
$\qquad mround \mapsto logicalClock'[i],$
$\qquad mvote \mapsto currentVote'[i]])$
$\quad \land \text{UNCHANGED } \langle currentEpoch, history\rangle$

Abstraction of *WorkerReceiver.run*()

$ReceiveNotmsg(i, j) \triangleq$
$\quad \land electionMsgs[j][i] \neq \langle\rangle$
$\quad \land \text{LET } notmsg \triangleq electionMsgs[j][i][1]$
$\qquad toSend \triangleq [mtype \mapsto NOTIFICATION,$
$\qquad\quad msource \mapsto i,$
$\qquad\quad mstate \mapsto state[i],$
$\qquad\quad mround \mapsto logicalClock[i],$
$\qquad\quad mvote \mapsto currentVote[i]]$
$\quad \text{IN} \quad \lor \land state[i] = LOOKING$

```
                    ∧ recvQueue' = [recvQueue EXCEPT ![i] = Append(RemoveNone(recvQueue[i]), notmsg)]
                    ∧ LET replyOk ≜ ∧ notmsg.mstate  = LOOKING
                                    ∧ notmsg.mround < logicalClock[i]
                      IN
                       ∨ ∧ replyOk
                         ∧ ReplyNotmsg(i, j, toSend)
                       ∨ ∧ ¬replyOk
                         ∧ DiscardNotmsg(j, i)
                  ∨ ∧ state[i] ∈ {LEADING, FOLLOWING}
                    ∧ ∨  Only reply when sender's state is LOOKING
                         ∧ notmsg.mstate = LOOKING
                         ∧ ReplyNotmsg(i, j, toSend)
                       ∨  sender's state and mine are both not LOOKING, just discard
                         ∧ notmsg.mstate ≠ LOOKING
                         ∧ DiscardNotmsg(j, i)
                    ∧ UNCHANGED recvQueue
            ∧ UNCHANGED ⟨serverVarsL, currentVote, logicalClock, receiveVotes, outOfElection, waitNotmsg, lead

NotmsgTimeout(i) ≜
        ∧ state[i] = LOOKING
        ∧ ∀ j ∈ Server : electionMsgs[j][i] = ⟨⟩
        ∧ recvQueue[i] = ⟨⟩
        ∧ recvQueue' = [recvQueue EXCEPT ![i] = Append(recvQueue[i], [mtype ↦ NONE])]
        ∧ UNCHANGED ⟨serverVarsL, currentVote, logicalClock, receiveVotes, outOfElection, waitNotmsg, lead
```

```
 Sub-action in HandleNotmsg
ReceivedFollowingAndLeadingNotification(i, n) ≜
        LET newVotes      ≜ Put(i, n.msource, receiveVotes[i], n.mvote, n.mround, n.mstate)
            voteSet1      ≜ VoteSet(i, n.msource, newVotes, n.mvote, n.mround)
            hasQuorums1   ≜ voteSet1 ∈ Quorums
            check1        ≜ CheckLeader(i, newVotes, n.mvote.proposedLeader, n.mround)
            leaveOk1      ≜ ∧ n.mround = logicalClock[i]
                            ∧ hasQuorums1
                            ∧ check1      state and leadingVoteSet cannot be changed twice in the first '∧' and second
        IN
        ∧ ∨ ∧ n.mround = logicalClock[i]
            ∧ receiveVotes' = [receiveVotes EXCEPT ![i] = newVotes]
          ∨ ∧ n.mround ≠ logicalClock[i]
            ∧ UNCHANGED receiveVotes
        ∧ ∨ ∧ leaveOk1
            ∧ PrintT("leave with condition 1")
            ∧ state' = [state EXCEPT ![i] = IF n.mvote.proposedLeader = i THEN LEADING ELSE FOLLO
            ∧ leadingVoteSet' = [leadingVoteSet EXCEPT ![i] = IF n.mvote.proposedLeader = i THEN voteSet
            ∧ UpdateProposal(i, n.mvote.proposedLeader, n.mvote.proposedZxid, n.mvote.proposedEpoch)
```

```
                ∧ UNCHANGED ⟨logicalClock, outOfElection⟩
            ∨ ∧ ¬leaveOk1
              ∧ outOfElection' = [outOfElection EXCEPT ![i] = Put(i, n.msource, outOfElection[i], n.mvote, n
              ∧ LET voteSet2      ≜ VoteSet(i, n.msource, outOfElection'[i], n.mvote, n.mround)
                    hasQuorums2 ≜ voteSet2 ∈ Quorums
                    check2      ≜ CheckLeader(i, outOfElection'[i], n.mvote.proposedLeader, n.mround)
                    leaveOk2    ≜ ∧ hasQuorums2
                                  ∧ check2
                IN
                ∨ ∧ leaveOk2
                  ∧ PrintT("leave with condition 2")
                  ∧ logicalClock' = [logicalClock EXCEPT ![i] = n.mround]
                  ∧ state' = [state EXCEPT ![i] = IF n.mvote.proposedLeader = i THEN LEADING ELSE FO
                  ∧ leadingVoteSet' = [leadingVoteSet EXCEPT ![i] = IF n.mvote.proposedLeader = i THEN vo
                  ∧ UpdateProposal(i, n.mvote.proposedLeader, n.mvote.proposedZxid, n.mvote.proposedEpoc
                ∨ ∧ ¬leaveOk2
                  ∧ LET leaveOk3 ≜ ∧ n.mstate  = LEADING
                                   ∧ n.mround = logicalClock[i]
                    IN
                    ∨ ∧ leaveOk3
                      ∧ PrintT("leave with condition 3")
                      ∧ state' = [state EXCEPT ![i] = IF n.mvote.proposedLeader = i THEN LEADING ELSE
                      ∧ UpdateProposal(i, n.mvote.proposedLeader, n.mvote.proposedZxid, n.mvote.proposed
                    ∨ ∧ ¬leaveOk3
                      ∧ UNCHANGED ⟨state, currentVote⟩
                  ∧ UNCHANGED ⟨logicalClock, leadingVoteSet⟩

Main part of lookForLeader()
HandleNotmsg(i) ≜
        ∧ state[i] = LOOKING
        ∧ ¬waitNotmsg[i]
        ∧ recvQueue[i] ≠ ⟨⟩
        ∧ LET n           ≜ recvQueue[i][1]
              rawToSend ≜ [mtype   ↦ NOTIFICATION,
                           msource ↦ i,
                           mstate  ↦ LOOKING,
                           mround  ↦ logicalClock[i],
                           mvote   ↦ currentVote[i]]
          IN  ∨ ∧ n.mtype = NONE
                ∧ BroadcastNotmsg(i, rawToSend)
                ∧ UNCHANGED ⟨history, logicalClock, currentVote, receiveVotes, waitNotmsg, outOfElection
              ∨ ∧ n.mtype = NOTIFICATION
                ∧ ∨ ∧ n.mstate = LOOKING
                    ∧ ∨ n.round ≥ my round, then update data and receiveVotes.
                      ∧ n.mround ≥ logicalClock[i]
```

$\land \lor$ $n.round >$ my round, update round and decide new proposed leader.
$\land n.mround > logicalClock[i]$
$\land logicalClock' = [logicalClock \text{ EXCEPT } ![i] = n.mround]$ There should be *RvCle*
$\land \text{LET } selfinfo \triangleq [proposedLeader \mapsto i,$
$proposedZxid \mapsto lastProcessed[i].zxid,$
$proposedEpoch \mapsto currentEpoch[i]]$
$peerOk \triangleq TotalOrderPredicate(n.mvote, selfinfo)$
IN $\lor \land peerOk$
$\land UpdateProposal(i, n.mvote.proposedLeader, n.mvote.proposedZxi$
$\lor \land \neg peerOk$
$\land UpdateProposal(i, i, lastProcessed[i].zxid, currentEpoch[i])$
$\land BroadcastNotmsg(i, [mtype \mapsto NOTIFICATION,$
$msource \mapsto i,$
$mstate \mapsto LOOKING,$
$mround \mapsto n.mround,$
$mvote \mapsto currentVote'[i]])$
$\lor$ $n.round =$ my round & $n.vote >$ my vote
$\land n.mround = logicalClock[i]$
$\land \text{LET } peerOk \triangleq TotalOrderPredicate(n.mvote, currentVote[i])$
IN $\lor \land peerOk$
$\land UpdateProposal(i, n.mvote.proposedLeader, n.mvote.proposedZxi$
$\land BroadcastNotmsg(i, [mtype \mapsto NOTIFICATION,$
$msource \mapsto i,$
$mstate \mapsto LOOKING,$
$mround \mapsto logicalClock[i],$
$mvote \mapsto n.mvote])$
$\lor \land \neg peerOk$
$\land \text{UNCHANGED } \langle currentVote, electionMsgs \rangle$
$\land \text{UNCHANGED } logicalClock$
$\land \text{LET } rcvsetModifiedTwice \triangleq n.mround > logicalClock[i]$
IN $\lor \land rcvsetModifiedTwice$ Since a variable cannot be changed more than once in
$\land RvClearAndPut(i, n.msource, n.mvote, n.mround)$ clear + put
$\lor \land \neg rcvsetModifiedTwice$
$\land RvPut(i, n.msource, n.mvote, n.mround, n.mstate)$ put
$\land \text{LET } hasQuorums \triangleq HasQuorums(i, i, receiveVotes'[i], currentVote'[i], n.mrou$
IN $\lor \land hasQuorums$ If *hasQuorums*, see action *WaitNewNotmsg* and *WaitNewNotmsg*
$\land waitNotmsg' = [waitNotmsg \text{ EXCEPT } ![i] = \text{TRUE}]$
$\lor \land \neg hasQuorums$
$\land \text{UNCHANGED } waitNotmsg$
$\lor$ $n.round <$ my round, just discard it.
$\land n.mround < logicalClock[i]$
$\land \text{UNCHANGED } \langle logicalClock, currentVote, electionMsgs, receiveVotes, waitNotmsg$
$\land \text{UNCHANGED } \langle state, history, outOfElection, leadingVoteSet \rangle$
$\lor$ mainly contains *receivedFollowingNotification*(*line* 1146), *receivedLeadingNotification*(*line* 1185).
$\land n.mstate \in \{LEADING, FOLLOWING\}$

$\land$ *ReceivedFollowingAndLeadingNotification*$(i,\ n)$
$\land$ $history' = [history$ EXCEPT $![i] = InitHistory(i)]$
$\land$ UNCHANGED $\langle electionMsgs,\ waitNotmsg\rangle$
$\land$ $recvQueue' = [recvQueue$ EXCEPT $![i] = Tail(recvQueue[i])]$
$\land$ UNCHANGED $\langle currentEpoch,\ lastProcessed\rangle$

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

$WaitNewNotmsg(i) \triangleq$
$\land$ $state[i] = LOOKING$
$\land$ $waitNotmsg[i] = \text{TRUE}$
$\land$ $recvQueue[i] \neq \langle\rangle$
$\land$ $recvQueue[i][1].mtype = NOTIFICATION$
$\land$ LET $n \triangleq recvQueue[i][1]$
$peerOk \triangleq TotalOrderPredicate(n.mvote,\ currentVote[i])$
$delQ \triangleq Tail(recvQueue[i])$
IN $\lor \land peerOk$
$\land$ $waitNotmsg' = [waitNotmsg$ EXCEPT $![i] = \text{FALSE}]$
$\land$ $recvQueue' = [recvQueue$ EXCEPT $![i] = Append(delQ,\ n)]$
$\lor \land \neg peerOk$
$\land$ $recvQueue' = [recvQueue$ EXCEPT $![i] = delQ]$
$\land$ UNCHANGED $waitNotmsg$
$\land$ UNCHANGED $\langle serverVarsL,\ currentVote,\ logicalClock,\ receiveVotes,\ outOfElection,\ leaderVarsL,\ elec$

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

$WaitNewNotmsgEnd(i) \triangleq$
$\land$ $state[i] = LOOKING$
$\land$ $waitNotmsg[i] = \text{TRUE}$
$\land$ $\lor recvQueue[i] = \langle\rangle$
$\lor \land recvQueue[i] \neq \langle\rangle$
$\land$ $recvQueue[i][1].mtype = NONE$
$\land$ $state' = [state$ EXCEPT $![i] =$ IF $currentVote[i].proposedLeader = i$ THEN $LEAD$
ELSE $FOLL$
$\land$ $leadingVoteSet' = [leadingVoteSet$ EXCEPT $![i] =$ IF $currentVote[i].proposedLeader = i$ THEN $VoteS$
ELSE $@]$
$\land$ $history' = [history$ EXCEPT $![i] = InitHistory(i)]$
$\land$ UNCHANGED $\langle currentEpoch,\ lastProcessed,\ electionVarsL,\ electionMsgs\rangle$

Test - simulate modifying *currentEpoch* and *lastProcessed*. We want to reach violations to achieve some traces and see whether the whole state of system is advancing. The actions below are completely not equal to implementation in real, just simulate a process of leader updates state and followers get it.

$LeaderAdvanceEpoch(i) \triangleq$
$\land$ $state[i] = LEADING$
$\land$ $currentEpoch' = [currentEpoch$ EXCEPT $![i] = @ + 1]$
$\land$ UNCHANGED $\langle state,\ lastProcessed,\ history,\ electionVarsL,\ leaderVarsL,\ electionMsgs\rangle$

$FollowerUpdateEpoch(i,\ j)\ \triangleq$
$\quad \land\ state[i] = FOLLOWING$
$\quad \land\ currentVote[i].proposedLeader = j$
$\quad \land\ state[j] = LEADING$
$\quad \land\ currentEpoch[i] < currentEpoch[j]$
$\quad \land\ currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = currentEpoch[j]]$
$\quad \land \text{ UNCHANGED } \langle state,\ lastProcessed,\ history,\ electionVarsL,\ leaderVarsL,\ electionMsgs\rangle$

$LeaderAdvanceZxid(i)\ \triangleq$
$\quad \land\ state[i] = LEADING$
$\quad \land\ lastProcessed' = [lastProcessed \text{ EXCEPT } ![i] = \text{IF } lastProcessed[i].zxid[1] = currentEpoch[i]$
$\qquad\qquad\qquad\qquad\qquad\qquad \text{THEN } [\ index \mapsto lastProcessed[i].index + 1,$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \ zxid \mapsto \langle currentEpoch[i],\ lastProcessed[i].zxid[2] + 1\rangle]$
$\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE } [\ index \mapsto lastProcessed[i].index + 1,$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \ zxid \mapsto \langle currentEpoch[i],\ 1\rangle]]$
$\quad \land\ history' = [history \text{ EXCEPT } ![i] = Append(@,\ [zxid \mapsto lastProcessed'[i].zxid,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad value \mapsto NONE,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ackSid \mapsto \{\},$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad epoch \mapsto 0])]$
$\quad \land \text{ UNCHANGED } \langle state,\ currentEpoch,\ electionVarsL,\ leaderVarsL,\ electionMsgs\rangle$

$FollowerUpdateZxid(i,\ j)\ \triangleq$
$\quad \land\ state[i] = FOLLOWING$
$\quad \land\ currentVote[i].proposedLeader = j$
$\quad \land\ state[j] = LEADING$
$\quad \land \text{ LET } precede\ \triangleq\ \lor\ lastProcessed[i].zxid[1] < lastProcessed[j].zxid[1]$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad \lor\ \land\ lastProcessed[i].zxid[1] = lastProcessed[j].zxid[1]$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \land\ lastProcessed[i].zxid[2] < lastProcessed[j].zxid[2]$
$\qquad \text{IN } \quad \land\ precede$
$\qquad\qquad \land\ lastProcessed' = [lastProcessed \text{ EXCEPT } ![i] = lastProcessed[j]]$
$\qquad\qquad \land\ history' = [history \text{ EXCEPT } ![i] = history[j]]$
$\quad \land \text{ UNCHANGED } \langle state,\ currentEpoch,\ electionVarsL,\ leaderVarsL,\ electionMsgs\rangle$

$NextL\ \triangleq$
$\quad \lor \exists\, i \in Server :\ \ ZabTimeout(i)$
$\quad \lor \exists\, i, j \in Server :\ ReceiveNotmsg(i, j)$
$\quad \lor \exists\, i \in Server :\ \ NotmsgTimeout(i)$
$\quad \lor \exists\, i \in Server :\ \ HandleNotmsg(i)$
$\quad \lor \exists\, i \in Server :\ \ WaitNewNotmsg(i)$
$\quad \lor \exists\, i \in Server :\ \ WaitNewNotmsgEnd(i)$

$\quad \lor \exists\, i \in Server :\ \ LeaderAdvanceEpoch(i)$
$\quad \lor \exists\, i, j \in Server :\ FollowerUpdateEpoch(i, j)$
$\quad \lor \exists\, i \in Server :\ \ LeaderAdvanceZxid(i)$
$\quad \lor \exists\, i, j \in Server :\ FollowerUpdateZxid(i, j)$

$SpecL \triangleq InitL \land \Box[NextL]_{varsL}$

These invariants should be violated after running for minutes.

$$ShouldBeTriggered1 \triangleq \neg\exists\, Q \in Quorums : \land \forall\, i \in Q : \land state[i] \in \{FOLLOWING,\ LEADING\}$$
$$\land currentEpoch[i] > 3$$
$$\land logicalClock[i] > 2$$
$$\land currentVote[i].proposedLeader \in Q$$
$$\land \forall\, i,\ j \in Q : currentVote[i].proposedLeader = currentVote[j].pro$$

$$ShouldBeTriggered2 \triangleq \neg\exists\, Q \in Quorums : \land \forall\, i \in Q : \land state[i] \in \{FOLLOWING,\ LEADING\}$$
$$\land currentEpoch[i] > 3$$
$$\land currentVote[i].proposedLeader \in Q$$
$$\land \forall\, i,\ j \in Q : currentVote[i].proposedLeader = currentVote[j].proposedLeader$$

\ * Modification History
\ * Last modified Sun *Nov* 14 15:18:32 *CST* 2021 by Dell
\ * Created *Fri Jun* 18 20:23:47 *CST* 2021 by Dell