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. *ZooKeepers* 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*

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

VARIABLE *currentEpoch*

The *zxid* of the last transaction in history.

VARIABLE *lastZxid*

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*

$serverVars \triangleq \langle state, currentEpoch, lastZxid \rangle$

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

$leaderVars \triangleq \langle leadingVoteSet \rangle$

$varsL \triangleq \langle serverVars, electionVars, leaderVars, electionMsgs, idTable \rangle$

Processing of *electionMsgs*

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

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

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

Processing of *recvQueue*

```
RECURSIVE $RemoveNone(\_)$
$RemoveNone(seq) \triangleq$ CASE $seq = \langle\rangle \rightarrow \langle\rangle$
                       $\Box$    $seq \neq \langle\rangle \rightarrow$ IF $Head(seq).mtype = NONE$ THEN $RemoveNone(Tail(seq))$
                                                                     ELSE $\langle Head(seq)\rangle \circ RemoveNone(Tail$
```

Processing of *idTable* and order comparing

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

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

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

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

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

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

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

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

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

Processing of *currentVote*

$$\begin{array}{l} InitialVote \triangleq \begin{array}[t]{l} [proposedLeader \mapsto NullPoint, \\ \ proposedZxid \quad \mapsto \langle 0,\ 0 \rangle, \\ \ proposedEpoch \ \mapsto 0] \end{array} \end{array}$$

$$\begin{array}{l} SelfVote(i) \triangleq \begin{array}[t]{l} [proposedLeader \mapsto i, \\ \ proposedZxid \quad \mapsto lastZxid[i], \\ \ proposedEpoch \ \mapsto currentEpoch[i]] \end{array} \end{array}$$

$$\begin{array}{l} UpdateProposal(i,\ nid,\ nzxid,\ nepoch) \triangleq currentVote' = [currentVote \text{ EXCEPT } \begin{array}[t]{l} ![i].proposedLeader = nid, \quad \text{nc} \\ ![i].proposedZxid \quad = nzxid, \\ ![i].proposedEpoch \ = nepoch] \end{array} \end{array}$$

Processing of *receiveVotes* and *outOfElection*

$$\begin{array}{l} RvClear(i) \triangleq receiveVotes' = [receiveVotes \text{ EXCEPT } ![i] = [v \in Server \mapsto \begin{array}[t]{l} [vote \quad \mapsto InitialVote, \\ \ round \quad \mapsto 0, \\ \ state \quad \mapsto LOOKING, \\ \ version \mapsto 0]]] \end{array} \end{array}$$

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

```
                                          $\Box$  $receiveVotes[i][id].round = mround \rightarrow [receive$

                                          $\Box$  $receiveVotes[i][id].round > mround \rightarrow receiveV$
$Put(i, id, rcvset, mvote, mround, mstate) \triangleq$ CASE $rcvset[id].round < mround \rightarrow [rcvset$ EXCEPT $![id].vote$
                                                                                   $![id].round$
                                                                                   $![id].state$
                                                                                   $![id].versio$
                                   $\Box$  $rcvset[id].round = mround \rightarrow [rcvset$ EXCEPT $![id].vote$
                                                                                   $![id].state$
                                                                                   $![id].versio$
                                   $\Box$  $rcvset[id].round > mround \rightarrow rcvset$

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

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

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

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

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

$OoePut(i, id, mvote, mround, mstate) \triangleq outOfElection' =$ CASE $outOfElection[i][id].round < mround \rightarrow [ou$
```

```
□    outOfElection[i][id].round = mround → [ou

□    outOfElection[i][id].round > mround → out
```

```
InitServerVars ≜ ∧ state        = [s ∈ Server ↦ LOOKING]
                 ∧ currentEpoch = [s ∈ Server ↦ 0]
                 ∧ lastZxid     = [s ∈ Server ↦ ⟨0, 0⟩]

InitElectionVars ≜ ∧ currentVote   = [s ∈ Server ↦ SelfVote(s)]
                   ∧ logicalClock  = [s ∈ Server ↦ 0]
                   ∧ receiveVotes  = [s ∈ Server ↦ [v ∈ Server ↦ [vote    ↦ InitialVote,
                                                                 round   ↦ 0,
                                                                 state   ↦ LOOKING,
                                                                 version ↦ 0]]]
                   ∧ outOfElection = [s ∈ Server ↦ [v ∈ Server ↦ [vote    ↦ InitialVote,
                                                                 round   ↦ 0,
                                                                 state   ↦ LOOKING,
                                                                 version ↦ 0]]]
                   ∧ recvQueue     = [s ∈ Server ↦ ⟨⟩]
                   ∧ waitNotmsg    = [s ∈ Server ↦ FALSE]

InitLeaderVars ≜ ∧ leadingVoteSet = [s ∈ Server ↦ {}]

Init ≜ ∧ InitServerVars
       ∧ InitElectionVars
       ∧ InitLeaderVars
       ∧ electionMsgs = [s ∈ Server ↦ [v ∈ Server ↦ ⟨⟩]]
       ∧ idTable = InitializeIdTable(Server)
```

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

```
ZabTimeout(i) ≜
      ∧ state[i] ∈ {LEADING, FOLLOWING}
      ∧ state'         = [state         EXCEPT ![i] = LOOKING]
      ∧ logicalClock'  = [logicalClock  EXCEPT ![i] = logicalClock[i] + 1]
      ∧ currentVote'   = [currentVote   EXCEPT ![i] = [proposedLeader ↦ i,
                                                       proposedZxid   ↦ lastZxid[i],
                                                       proposedEpoch  ↦ currentEpoch[i]]]
      ∧ receiveVotes'  = [receiveVotes  EXCEPT ![i] = [v ∈ Server ↦ [vote    ↦ InitialVote,
                                                                    round   ↦ 0,
                                                                    state   ↦ LOOKING,
                                                                    version ↦ 0]]]
      ∧ outOfElection' = [outOfElection EXCEPT ![i] = [v ∈ Server ↦ [vote    ↦ InitialVote,
                                                                    round   ↦ 0,
                                                                    state   ↦ LOOKING,
                                                                    version ↦ 0]]]
      ∧ recvQueue'     = [recvQueue     EXCEPT ![i]  = ⟨⟩]
      ∧ waitNotmsg'    = [waitNotmsg    EXCEPT ![i] = FALSE]
      ∧ leadingVoteSet' = [leadingVoteSet EXCEPT ![i]  = {}]
      ∧ BroadcastNotmsg(i, [mtype   ↦ NOTIFICATION,
                            msource ↦ i,
                            mstate  ↦ LOOKING,
                            mround  ↦ logicalClock'[i],
                            mvote   ↦ currentVote'[i]])
      ∧ UNCHANGED ⟨currentEpoch, lastZxid, idTable⟩

Abstraction of WorkerReceiver.run()
ReceiveNotmsg(i, j) ≜
      ∧ electionMsgs[j][i] ≠ ⟨⟩
      ∧ LET notmsg ≜ electionMsgs[j][i][1]
            toSend ≜ [mtype   ↦ NOTIFICATION,
                      msource ↦ i,
                      mstate  ↦ state[i],
                      mround  ↦ logicalClock[i],
                      mvote   ↦ currentVote[i]]
        IN   ∨ ∧ 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 ⟨serverVars, currentVote, logicalClock, receiveVotes, outOfElection, waitNotmsg, leade

NotmsgTimeout(i) ≜
      ∧ state[i] = LOOKING
      ∧ ∀ j ∈ Server : electionMsgs[j][i] = ⟨⟩
      ∧ recvQueue[i] = ⟨⟩
      ∧ recvQueue' = [recvQueue EXCEPT ![i] = Append(recvQueue[i], [mtype ↦ NONE])]
```

```
        ∧ UNCHANGED ⟨serverVars, currentVote, logicalClock, receiveVotes, outOfElection, waitNotmsg, leade
```

```
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 FOLLOV
             ∧ 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 ⟨logicalClock, currentVote, receiveVotes, waitNotmsg, outOfElection, state, l
           ∨ ∧ n.mtype = NOTIFICATION
             ∧ ∨ ∧ n.mstate = LOOKING
                 ∧ ∨  n.round ≥  my round, then update data and receiveVotes.
                    ∧ n.mround ≥ logicalClock[i]
                    ∧ ∨  n.round >  my round, update round and decide new proposed leader.
                       ∧ n.mround > logicalClock[i]
                       ∧ logicalClock' = [logicalClock EXCEPT ![i] = n.mround]  There should be RvCle
                       ∧ LET selfinfo ≜ [proposedLeader ↦ i,
                                          proposedZxid    ↦ lastZxid[i],
                                          proposedEpoch  ↦ currentEpoch[i]]
                             peerOk ≜ TotalOrderPredicate(n.mvote, selfinfo)
                         IN   ∨ ∧ peerOk
                                ∧ UpdateProposal(i, n.mvote.proposedLeader, n.mvote.proposedZxi
                              ∨ ∧ ¬peerOk
                                ∧ UpdateProposal(i, i, lastZxid[i], currentEpoch[i])
                       ∧ BroadcastNotmsg(i, [mtype    ↦ NOTIFICATION,
                                              msource ↦ i,
                                              mstate   ↦ LOOKING,
                                              mround  ↦ n.mround,
                                              mvote    ↦ currentVote'[i]])
                     ∨  n.round =  my round & n.vote >  my vote
                       ∧ n.mround = logicalClock[i]
                       ∧ LET peerOk ≜ TotalOrderPredicate(n.mvote, currentVote[i])
                         IN   ∨ ∧ peerOk
                                ∧ UpdateProposal(i, n.mvote.proposedLeader, n.mvote.proposedZxi
                                ∧ BroadcastNotmsg(i, [mtype    ↦ NOTIFICATION,
                                                       msource ↦ i,
                                                          mstate    ↦ LOOKING,
                                                          mround   ↦ logicalClock[i],
                                                          mvote     ↦ n.mvote])
                                          ∨ ∧ ¬peerOk
                                            ∧ UNCHANGED ⟨currentVote, electionMsgs⟩
                                      ∧ UNCHANGED logicalClock
                                   ∧ LET rcvsetModifiedTwice ≜ n.mround > logicalClock[i]
                                     IN   ∨ ∧ rcvsetModifiedTwice    Since a variable cannot be changed more than once in o
                                            ∧ RvClearAndPut(i, n.msource, n.mvote, n.mround)   clear + put
                                          ∨ ∧ ¬rcvsetModifiedTwice
                                            ∧ RvPut(i, n.msource, n.mvote, n.mround, n.mstate)          put
                                   ∧ LET hasQuorums ≜ HasQuorums(i, i, receiveVotes′[i], currentVote′[i], n.mrou
                                     IN   ∨ ∧ hasQuorums  If hasQuorums, see action WaitNewNotmsg and WaitNewNotmsg
                                            ∧ waitNotmsg′ = [waitNotmsg EXCEPT ![i] = TRUE]
                                          ∨ ∧ ¬hasQuorums
                                            ∧ UNCHANGED waitNotmsg
                                ∨   n.round <  my round, just discard it.
                                   ∧ n.mround < logicalClock[i]
                                   ∧ UNCHANGED ⟨logicalClock, currentVote, electionMsgs, receiveVotes, waitNotmsg
                             ∧ UNCHANGED ⟨state, outOfElection, leadingVoteSet⟩
                          ∨  mainly contains receivedFollowingNotification(line 1146), receivedLeadingNotification(line 1185).
                             ∧ n.mstate ∈ {LEADING, FOLLOWING}
                             ∧ ReceivedFollowingAndLeadingNotification(i, n)
                             ∧ UNCHANGED ⟨electionMsgs, waitNotmsg⟩
          ∧ recvQueue′ = [recvQueue EXCEPT ![i] = Tail(recvQueue[i])]
          ∧ UNCHANGED ⟨currentEpoch, lastZxid, idTable⟩

On the premise that ReceiveVotes.HasQuorums = TRUE, corresponding to logic in line 1050 − 1055 in LFE.java.
WaitNewNotmsg(i) ≜
          ∧ state[i] = LOOKING
          ∧ waitNotmsg[i] = TRUE
          ∧ recvQueue[i] ≠ ⟨⟩
          ∧ recvQueue[i][1].mtype = NOTIFICATION
          ∧ LET n        ≜ recvQueue[i][1]
                peerOk ≜ TotalOrderPredicate(n.mvote, currentVote[i])
                delQ    ≜ Tail(recvQueue[i])
            IN   ∨ ∧ peerOk
                   ∧ waitNotmsg′ = [waitNotmsg EXCEPT ![i] = FALSE]
                   ∧ recvQueue′   = [recvQueue  EXCEPT ![i]  = Append(delQ, n)]
                 ∨ ∧ ¬peerOk
                   ∧ recvQueue′ = [recvQueue EXCEPT ![i] = delQ]
                   ∧ UNCHANGED waitNotmsg
          ∧ UNCHANGED ⟨serverVars, currentVote, logicalClock, receiveVotes, outOfElection, leaderVars, electio

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

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

Test - simulate modifying *currentEpoch* and *lastZxid*. 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.

```
$LeaderAdvanceEpoch(i) \triangleq$
        $\land state[i] = LEADING$
        $\land currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = @ + 1]$
        $\land \text{UNCHANGED } \langle state, lastZxid, electionVars, leaderVars, electionMsgs, idTable\rangle$

$FollowerUpdateEpoch(i, j) \triangleq$
        $\land state[i] = FOLLOWING$
        $\land currentVote[i].proposedLeader = j$
        $\land state[j] = LEADING$
        $\land currentEpoch[i] < currentEpoch[j]$
        $\land currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = currentEpoch[j]]$
        $\land \text{UNCHANGED } \langle state, lastZxid, electionVars, leaderVars, electionMsgs, idTable\rangle$

$LeaderAdvanceZxid(i) \triangleq$
        $\land state[i] = LEADING$
        $\land lastZxid' = [lastZxid \text{ EXCEPT } ![i] = \text{IF } lastZxid[i][1] = currentEpoch[i]$
                                                   $\text{THEN } \langle currentEpoch[i], lastZxid[i][2] + 1\rangle$
                                                   $\text{ELSE } \langle currentEpoch[i], 1\rangle]$
        $\land \text{UNCHANGED } \langle state, currentEpoch, electionVars, leaderVars, electionMsgs, idTable\rangle$

$FollowerUpdateZxid(i, j) \triangleq$
        $\land state[i] = FOLLOWING$
        $\land currentVote[i].proposedLeader = j$
        $\land state[j] = LEADING$
        $\land \text{LET } precede \triangleq \lor lastZxid[i][1] < lastZxid[j][1]$
                              $\lor \land lastZxid[i][1] = lastZxid[j][1]$
                                $\land lastZxid[i][2] < lastZxid[j][2]$
          $\text{IN } \land precede$
               $\land lastZxid' = [lastZxid \text{ EXCEPT } ![i] = lastZxid[j]]$
        $\land \text{UNCHANGED } \langle state, currentEpoch, electionVars, leaderVars, electionMsgs, idTable\rangle$
```

$$
\begin{array}{l}
\mathit{Next} \triangleq \\
\qquad \lor \exists\, i \in \mathit{Server} : \quad \mathit{ZabTimeout}(i) \\
\qquad \lor \exists\, i, j \in \mathit{Server} : \; \mathit{ReceiveNotmsg}(i, j) \\
\qquad \lor \exists\, i \in \mathit{Server} : \quad \mathit{NotmsgTimeout}(i) \\
\qquad \lor \exists\, i \in \mathit{Server} : \quad \mathit{HandleNotmsg}(i) \\
\qquad \lor \exists\, i \in \mathit{Server} : \quad \mathit{WaitNewNotmsg}(i) \\
\qquad \lor \exists\, i \in \mathit{Server} : \quad \mathit{WaitNewNotmsgEnd}(i) \\
\\
\qquad \lor \exists\, i \in \mathit{Server} : \quad \mathit{LeaderAdvanceEpoch}(i) \\
\qquad \lor \exists\, i, j \in \mathit{Server} : \; \mathit{FollowerUpdateEpoch}(i, j) \\
\qquad \lor \exists\, i \in \mathit{Server} : \quad \mathit{LeaderAdvanceZxid}(i) \\
\qquad \lor \exists\, i, j \in \mathit{Server} : \; \mathit{FollowerUpdateZxid}(i, j)
\end{array}
$$

$$\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{varsL}}$$

These invariants should be violated after running for minutes.

$$
\begin{array}{l}
\mathit{ShouldBeTriggered1} \triangleq \neg\exists\, Q \in \mathit{Quorums} : \land \forall\, i \in Q : \land\, \mathit{state}[i] \in \{\mathit{FOLLOWING}, \mathit{LEADING}\} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, \mathit{currentEpoch}[i] > 3 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, \mathit{logicalClock}[i] \;\; > 2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, \mathit{currentVote}[i].\mathit{proposedLeader} \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land \forall\, i, j \in Q : \mathit{currentVote}[i].\mathit{proposedLeader} = \mathit{currentVote}[j].\mathit{prop}
\end{array}
$$

$$
\begin{array}{l}
\mathit{ShouldBeTriggered2} \triangleq \neg\exists\, Q \in \mathit{Quorums}\colon \land \forall\, i \in Q\colon \land\, \mathit{state}[i] \in \{\mathit{FOLLOWING}, \mathit{LEADING}\} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land\, \mathit{currentEpoch}[i] > 3 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land\, \mathit{currentVote}[i].\mathit{proposedLeader} \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad \land \quad \forall\, i, \quad j \in Q\colon \quad \mathit{currentVote}[i].\mathit{proposedLeader} \quad = \\
\qquad\qquad\qquad\qquad\qquad \mathit{currentVote}[j].\mathit{proposedLeader}
\end{array}
$$

\ ∗ Modification History
\ ∗ Last modified Sun *Sep* 26 16:20:03 *CST* 2021 by Dell
\ ∗ Created *Fri* *Jun* 18 20:23:47 *CST* 2021 by Dell