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*

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: $id1 > id2$

```
$IdCompare(id1, id2) \triangleq$ LET $item1 \triangleq$ CHOOSE $item \in idTable : item[1] = id1$
```

$$\begin{aligned} & \quad\; item2 \triangleq \text{CHOOSE } item \in idTable : item[1] = id2 \\ & \text{IN} \quad item1[2] > item2[2] \end{aligned}$$

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

$$\begin{aligned} ZxidCompare(zxid1, zxid2) \triangleq\; & \lor\; zxid1[1] > zxid2[1] \\ & \lor\; \land\; zxid1[1] = zxid2[1] \\ & \quad\;\; \land\; zxid1[2] > zxid2[2] \end{aligned}$$

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

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

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

$$\begin{aligned} VoteEqual(vote1, round1, vote2, round2) \triangleq\; & \land\; vote1.proposedLeader = vote2.proposedLeader \\ & \land\; ZxidEqual(vote1.proposedZxid, vote2.proposedZxid) \\ & \land\; vote1.proposedEpoch = vote2.proposedEpoch \\ & \land\; round1 = round2 \end{aligned}$$

Processing of $currentVote$

$$\begin{aligned} InitialVote \triangleq\; & [proposedLeader \mapsto NullPoint, \\ & \;\; proposedZxid \mapsto \langle 0, 0 \rangle, \\ & \;\; proposedEpoch \mapsto 0] \end{aligned}$$

$$\begin{aligned} SelfVote(i) \triangleq\; & [proposedLeader \mapsto i, \\ & \;\; proposedZxid \mapsto lastZxid[i], \\ & \;\; proposedEpoch \mapsto currentEpoch[i]] \end{aligned}$$

$$\begin{aligned} UpdateProposal(i, nid, nzxid, nepoch) \triangleq currentVote' = [currentVote \text{ EXCEPT } & ![i].proposedLeader = nid, \quad \text{no} \\ & ![i].proposedZxid = nzxid, \\ & ![i].proposedEpoch = nepoch] \end{aligned}$$

Processing of $receiveVotes$ and $outOfElection$

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

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

```
                                              $\Box\quad receiveVotes[i][id].round > mround \rightarrow receiveV$

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

$RvClearAndPut(i, id, vote, round) \triangleq receiveVotes' =$ 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 $outOfElection[i][id].round < mround \rightarrow [ou$

                                                  $\Box\quad outOfElection[i][id].round = mround \rightarrow [ou$
```

$\Box \quad \mathit{outOfElection}[i][id].round > mround \rightarrow out$

$$\begin{aligned}
\mathit{InitServerVars} \triangleq\ & \land\ state = [s \in Server \mapsto \mathit{LOOKING}] \\
& \land\ currentEpoch = [s \in Server \mapsto 0] \\
& \land\ lastZxid = [s \in Server \mapsto \langle 0,\ 0\rangle]
\end{aligned}$$

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

$$\mathit{InitLeaderVars} \triangleq\ \land\ leadingVoteSet = [s \in Server \mapsto \{\}]$$

$$\begin{aligned}
Init \triangleq\ & \land\ InitServerVars \\
& \land\ InitElectionVars \\
& \land\ InitLeaderVars \\
& \land\ electionMsgs = [s \in Server \mapsto [v \in Server \mapsto \langle\rangle]] \\
& \land\ idTable = InitializeIdTable(Server)
\end{aligned}$$

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

$$\begin{aligned}
& ZabTimeout(i) \triangleq \\
& \quad \land\ state[i] \in \{\mathit{LEADING},\ \mathit{FOLLOWING}\} \\
& \quad \land\ state' = [state\ \text{EXCEPT}\ ![i] = \mathit{LOOKING}] \\
& \quad \land\ logicalClock' = [logicalClock\ \text{EXCEPT}\ ![i] = logicalClock[i] + 1] \\
& \quad \land\ currentVote' = [currentVote\ \text{EXCEPT}\ ![i] = [proposedLeader \mapsto i, \\
& \qquad\qquad proposedZxid \mapsto lastZxid[i], \\
& \qquad\qquad proposedEpoch \mapsto currentEpoch[i]]] \\
& \quad \land\ receiveVotes' = [receiveVotes\ \text{EXCEPT}\ ![i] = [v \in Server \mapsto [vote \mapsto InitialVote, \\
& \qquad\qquad round \mapsto 0, \\
& \qquad\qquad state \mapsto \mathit{LOOKING}, \\
& \qquad\qquad version \mapsto 0]]] \\
& \quad \land\ outOfElection' = [outOfElection\ \text{EXCEPT}\ ![i] = [v \in Server \mapsto [vote \mapsto InitialVote, \\
& \qquad\qquad round \mapsto 0, \\
& \qquad\qquad state \mapsto \mathit{LOOKING}, \\
& \qquad\qquad version \mapsto 0]]]
\end{aligned}$$

```
$\land$ $recvQueue'$       $= [recvQueue$       EXCEPT $![i]$ $= \langle\rangle]$
$\land$ $waitNotmsg'$      $= [waitNotmsg$      EXCEPT $![i] =$ FALSE]
$\land$ $leadingVoteSet' = [leadingVoteSet$ EXCEPT $![i]$ $= \{\}]$
$\land$ $BroadcastNotmsg(i, [mtype \mapsto NOTIFICATION,$
                          $msource \mapsto i,$
                          $mstate \mapsto LOOKING,$
                          $mround \mapsto logicalClock'[i],$
                          $mvote \mapsto currentVote'[i]])$
$\land$ UNCHANGED $\langle currentEpoch, lastZxid, idTable\rangle$
```

Abstraction of $WorkerReceiver.run()$

```
$ReceiveNotmsg(i, j) \triangleq$
    $\land$ $electionMsgs[j][i] \neq \langle\rangle$
    $\land$ LET $notmsg \triangleq electionMsgs[j][i][1]$
            $toSend \triangleq [mtype \mapsto NOTIFICATION,$
                               $msource \mapsto i,$
                               $mstate \mapsto state[i],$
                               $mround \mapsto logicalClock[i],$
                               $mvote \mapsto currentVote[i]]$
      IN  $\lor$ $\land$ $state[i] = LOOKING$
               $\land$ $recvQueue' = [recvQueue$ EXCEPT $![i] = Append(RemoveNone(recvQueue[i]), notmsg)]$
               $\land$ LET $replyOk \triangleq$ $\land$ $notmsg.mstate = LOOKING$
                                    $\land$ $notmsg.mround < logicalClock[i]$
                  IN
                   $\lor$ $\land$ $replyOk$
                      $\land$ $ReplyNotmsg(i, j, toSend)$
                   $\lor$ $\land$ $\neg replyOk$
                      $\land$ $DiscardNotmsg(j, i)$
          $\lor$ $\land$ $state[i] \in \{LEADING, FOLLOWING\}$
               $\land$ $\lor$  Only reply when sender's state is $LOOKING$
                      $\land$ $notmsg.mstate = LOOKING$
                      $\land$ $ReplyNotmsg(i, j, toSend)$
                  $\lor$  sender's state and mine are both not $LOOKING$, just discard
                      $\land$ $notmsg.mstate \neq LOOKING$
                      $\land$ $DiscardNotmsg(j, i)$
               $\land$ UNCHANGED $recvQueue$
    $\land$ UNCHANGED $\langle serverVars, currentVote, logicalClock, receiveVotes, outOfElection, waitNotmsg, leade$

$NotmsgTimeout(i) \triangleq$
    $\land$ $state[i] = LOOKING$
    $\land$ $\forall j \in Server : electionMsgs[j][i] = \langle\rangle$
    $\land$ $recvQueue[i] = \langle\rangle$
    $\land$ $recvQueue' = [recvQueue$ EXCEPT $![i] = Append(recvQueue[i], [mtype \mapsto NONE])]$
    $\land$ UNCHANGED $\langle serverVars, currentVote, logicalClock, receiveVotes, outOfElection, waitNotmsg, leade$
```

```
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.proposec
                  ∨ ∧ ¬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
```

```
                                        $\land$ UNCHANGED $\langle currentVote, electionMsgs \rangle$
                                   $\land$ UNCHANGED $logicalClock$
                              $\land$ LET $rcvsetModifiedTwice \triangleq n.mround > logicalClock[i]$
                                 IN   $\lor \land rcvsetModifiedTwice$      Since a variable cannot be changed more than once in o
                                         $\land RvClearAndPut(i, n.msource, n.mvote, n.mround)$   clear + put
                                      $\lor \land \neg rcvsetModifiedTwice$
                                         $\land RvPut(i, n.msource, n.mvote, n.mround, n.mstate)$        put
                              $\land$ LET $hasQuorums \triangleq HasQuorums(i, i, receiveVotes'[i], currentVote'[i], n.mrou$
                                 IN   $\lor \land hasQuorums$  If $hasQuorums$, see action $WaitNewNotmsg$ and $WaitNewNotmsg$
                                         $\land waitNotmsg' = [waitNotmsg$ EXCEPT $![i] =$ TRUE$]$
                                      $\lor \land \neg hasQuorums$
                                         $\land$ UNCHANGED $waitNotmsg$
                           $\lor$  $n.round <$ my round, just discard it.
                              $\land n.mround < logicalClock[i]$
                              $\land$ UNCHANGED $\langle logicalClock, currentVote, electionMsgs, receiveVotes, waitNotmsg$
                        $\land$ UNCHANGED $\langle state, outOfElection, leadingVoteSet \rangle$
                     $\lor$  mainly contains $receivedFollowingNotification(line\ 1146)$, $receivedLeadingNotification(line\ 1185)$.
                        $\land n.mstate \in \{LEADING, FOLLOWING\}$
                        $\land ReceivedFollowingAndLeadingNotification(i, n)$
                        $\land$ UNCHANGED $\langle electionMsgs, waitNotmsg \rangle$
        $\land recvQueue' = [recvQueue$ EXCEPT $![i] = Tail(recvQueue[i])]$
        $\land$ UNCHANGED $\langle currentEpoch, lastZxid, idTable \rangle$

On the premise that $ReceiveVotes.HasQuorums =$ TRUE, corresponding to logic in line $1050 - 1055$ in $LFE.java$.
$WaitNewNotmsg(i) \triangleq$
        $\land state[i] = LOOKING$
        $\land waitNotmsg[i] =$ 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 $![i] =$ FALSE$]$
                  $\land recvQueue' = [recvQueue$ EXCEPT $![i] = Append(delQ, n)]$
               $\lor \land \neg peerOk$
                  $\land recvQueue' = [recvQueue$ EXCEPT $![i] = delQ]$
                  $\land$ UNCHANGED $waitNotmsg$
        $\land$ UNCHANGED $\langle serverVars, currentVote, logicalClock, receiveVotes, outOfElection, leaderVars, electio$

On the premise that $ReceiveVotes.HasQuorums =$ TRUE, corresponding to logic in line $1061 - 1066$ in $LFE.java$.
$WaitNewNotmsgEnd(i) \triangleq$
        $\land state[i] = LOOKING$
        $\land waitNotmsg[i] =$ TRUE
        $\land \lor recvQueue[i] = \langle\rangle$
        ∨ ∧ recvQueue[i] ≠ ⟨⟩
          ∧ recvQueue[i][1].mtype = NONE
      ∧ state'          = [state          EXCEPT ![i] = IF currentVote[i].proposedLeader = i THEN LEAD
                                                                                           ELSE FOLL
      ∧ leadingVoteSet' = [leadingVoteSet EXCEPT ![i] = IF currentVote[i].proposedLeader = i THEN VoteS
                                                                                           ELSE @]
      ∧ UNCHANGED ⟨currentEpoch, lastZxid, electionVars, electionMsgs, idTable⟩
```

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) ≜
      ∧ state[i] = LEADING
      ∧ currentEpoch' = [currentEpoch EXCEPT ![i] = @ + 1]
      ∧ UNCHANGED ⟨state, lastZxid, electionVars, leaderVars, electionMsgs, idTable⟩

FollowerUpdateEpoch(i, j) ≜
      ∧ state[i] = FOLLOWING
      ∧ currentVote[i].proposedLeader = j
      ∧ state[j] = LEADING
      ∧ currentEpoch[i] < currentEpoch[j]
      ∧ currentEpoch' = [currentEpoch EXCEPT ![i] = currentEpoch[j]]
      ∧ UNCHANGED ⟨state, lastZxid, electionVars, leaderVars, electionMsgs, idTable⟩

LeaderAdvanceZxid(i) ≜
      ∧ state[i] = LEADING
      ∧ lastZxid' = [lastZxid EXCEPT ![i] = IF lastZxid[i][1] = currentEpoch[i]
                                            THEN ⟨currentEpoch[i], lastZxid[i][2] + 1⟩
                                            ELSE ⟨currentEpoch[i], 1⟩]
      ∧ UNCHANGED ⟨state, currentEpoch, electionVars, leaderVars, electionMsgs, idTable⟩

FollowerUpdateZxid(i, j) ≜
      ∧ state[i] = FOLLOWING
      ∧ currentVote[i].proposedLeader = j
      ∧ state[j] = LEADING
      ∧ LET precede ≜ ∨ lastZxid[i][1] < lastZxid[j][1]
                      ∨ ∧ lastZxid[i][1] = lastZxid[j][1]
                        ∧ lastZxid[i][2] < lastZxid[j][2]
        IN  ∧ precede
            ∧ lastZxid' = [lastZxid EXCEPT ![i] = lastZxid[j]]
      ∧ UNCHANGED ⟨state, currentEpoch, electionVars, leaderVars, electionMsgs, idTable⟩

Next ≜
      ∨ ∃ i ∈ Server :     ZabTimeout(i)
      ∨ ∃ i, j ∈ Server :  ReceiveNotmsg(i, j)
```

$\vee \exists i \in \mathit{Server} : \quad \mathit{NotmsgTimeout}(i)$
$\vee \exists i \in \mathit{Server} : \quad \mathit{HandleNotmsg}(i)$
$\vee \exists i \in \mathit{Server} : \quad \mathit{WaitNewNotmsg}(i)$
$\vee \exists i \in \mathit{Server} : \quad \mathit{WaitNewNotmsgEnd}(i)$

$\vee \exists i \in \mathit{Server} : \quad \mathit{LeaderAdvanceEpoch}(i)$
$\vee \exists i, j \in \mathit{Server} : \quad \mathit{FollowerUpdateEpoch}(i, j)$
$\vee \exists i \in \mathit{Server} : \quad \mathit{LeaderAdvanceZxid}(i)$
$\vee \exists i, j \in \mathit{Server} : \quad \mathit{FollowerUpdateZxid}(i, j)$

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

These invariants should be violated after running for minutes.

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

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

\ ∗ Modification History
\ ∗ Last modified *Wed Jul* 14 21:02:21 *CST* 2021 by Dell
\ ∗ Created *Fri Jun* 18 20:23:47 *CST* 2021 by Dell