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*

```
$BroadcastNotmsg(i, m) \triangleq electionMsgs' = [electionMsgs$ 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  $\{\langle chosen, Cardinality(Remaining)\rangle\} \cup InitializeIdTable(re)$
```

$IdTable \triangleq InitializeIdTable(Server)$

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

```
$IdCompare(id1, id2) \triangleq$ LET $item1 \triangleq$ CHOOSE $item \in IdTable : item[1] = id1$
                                  $item2 \triangleq$ CHOOSE $item \in IdTable : item[1] = id2$
                             IN   $item1[2] > item2[2]$
```

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

```
$ZxidCompare(zxid1, zxid2) \triangleq$ $\lor$ $zxid1[1] > zxid2[1]$
                                $\lor$ $\land$ $zxid1[1] = zxid2[1]$
                                   $\land$ $zxid1[2] > zxid2[2]$
```

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

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

```
$TotalOrderPredicate(vote1, vote2) \triangleq$ $\lor$ $vote1.proposedEpoch > vote2.proposedEpoch$
                                        $\lor$ $\land$ $vote1.proposedEpoch = vote2.proposedEpoch$
                                           $\land$ $\lor$ $ZxidCompare(vote1.proposedZxid, vote2.proposedZxid)$
                                              $\lor$ $\land$ $ZxidEqual(vote1.proposedZxid, vote2.proposedZxid)$
                                                 $\land$ $IdCompare(vote1.proposedLeader, vote2.proposedLeader)$
```

```
$VoteEqual(vote1, round1, vote2, round2) \triangleq$ $\land$ $vote1.proposedLeader = vote2.proposedLeader$
                                              $\land$ $ZxidEqual(vote1.proposedZxid, vote2.proposedZxid)$
                                              $\land$ $vote1.proposedEpoch = vote2.proposedEpoch$
                                              $\land$ $round1 = round2$
```

```
$InitLastProcessed(i) \triangleq$ IF $Len(history[i]) = 0$ THEN $[index \mapsto 0,$
                                                        $zxid \mapsto \langle 0, 0 \rangle]$
                           ELSE
                          LET $lastIndex \triangleq Len(history[i])$
                              $entry \triangleq history[i][lastIndex]$
                          IN  $[index \mapsto lastIndex,$
                               $zxid \mapsto entry.zxid]$
```

```
RECURSIVE $InitAcksidInTxns(\_, \_)$
$InitAcksidInTxns(txns, src) \triangleq$ IF $Len(txns) = 0$ THEN $\langle \rangle$
                                  ELSE LET $newTxn \triangleq [zxid \mapsto txns[1].zxid,$
                                                         $value \mapsto txns[1].value,$
                                                         $ackSid \mapsto \{src\},$
                                                         $epoch \mapsto txns[1].epoch]$
                                       IN $\langle newTxn \rangle \circ InitAcksidInTxns(Tail(txns), src)$
```

```
$InitHistory(i) \triangleq$ LET $newState \triangleq state'[i]$IN
                     IF $newState = LEADING$ THEN $InitAcksidInTxns(history[i], i)$
                      ELSE $history[i]$
```

Processing of *currentVote*

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

$$
\begin{array}{l}
\mathit{SelfVote}(i) \triangleq [\mathit{proposedLeader} \mapsto i,\\
\qquad\qquad\quad \mathit{proposedZxid} \mapsto \mathit{lastProcessed}[i].\mathit{zxid},\\
\qquad\qquad\quad \mathit{proposedEpoch} \mapsto \mathit{currentEpoch}[i]]
\end{array}
$$

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

Processing of *receiveVotes* and *outOfElection*

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

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

$$
\begin{array}{l}
\mathit{Put}(i, \mathit{id}, \mathit{rcvset}, \mathit{mvote}, \mathit{mround}, \mathit{mstate}) \triangleq \text{CASE } \mathit{rcvset}[\mathit{id}].\mathit{round} < \mathit{mround} \rightarrow [\mathit{rcvset} \text{ EXCEPT } ![\mathit{id}].\mathit{vote}\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![\mathit{id}].\mathit{round}\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![\mathit{id}].\mathit{state}\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![\mathit{id}].\mathit{versio}\\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \Box \quad \mathit{rcvset}[\mathit{id}].\mathit{round} = \mathit{mround} \rightarrow [\mathit{rcvset} \text{ EXCEPT } ![\mathit{id}].\mathit{vote}\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![\mathit{id}].\mathit{state}\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![\mathit{id}].\mathit{versio}\\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \Box \quad \mathit{rcvset}[\mathit{id}].\mathit{round} > \mathit{mround} \rightarrow \mathit{rcvset}
\end{array}
$$

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

$$\begin{array}{l}
\mathit{VoteSet}(i,\ \mathit{msource},\ \mathit{rcvset},\ \mathit{thisvote},\ \mathit{thisround}) \triangleq \{\mathit{msource}\} \cup \{s \in (\mathit{Server} \setminus \{\mathit{msource}\}) : \mathit{VoteEqual}(\mathit{rcvset} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{rcvset} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{thisvo} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{thisro}
\end{array}$$

$$\begin{array}{l}
\mathit{HasQuorums}(i,\ \mathit{msource},\ \mathit{rcvset},\ \mathit{thisvote},\ \mathit{thisround}) \triangleq \textsc{let}\ Q \triangleq \mathit{VoteSet}(i,\ \mathit{msource},\ \mathit{rcvset},\ \mathit{thisvote},\ \mathit{thisr} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textsc{in}\quad \textsc{if}\ Q \in \mathit{Quorums}\ \textsc{then true else false}
\end{array}$$

$$\begin{array}{l}
\mathit{CheckLeader}(i,\ \mathit{votes},\ \mathit{thisleader},\ \mathit{thisround}) \triangleq \textsc{if}\ \mathit{thisleader} = i\ \textsc{then}\ (\textsc{if}\ \mathit{thisround} = \mathit{logicalClock}[i]\ \textsc{then t} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textsc{else}\ \ (\textsc{if}\ \mathit{votes}[\mathit{thisleader}].\mathit{vote}.\mathit{proposedLeader} = \mathit{NullPoint} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textsc{else}\ \ (\textsc{if}\ \mathit{votes}[\mathit{thisleader}].\mathit{state} = \mathit{LEADING}\ \textsc{then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textsc{else}
\end{array}$$

$$\begin{array}{l}
\mathit{OoeClear}(i) \triangleq \mathit{outOfElection}' = [\mathit{outOfElection}\ \textsc{except}\ ![i] = [v \in \mathit{Server} \mapsto [\begin{array}[t]{ll}
\mathit{vote} & \mapsto \mathit{InitialVote}, \\
\mathit{round} & \mapsto 0, \\
\mathit{state} & \mapsto \mathit{LOOKING}, \\
\mathit{version} & \mapsto 0]]]
\end{array}
\end{array}$$

$$\begin{array}{l}
\mathit{OoePut}(i,\ \mathit{id},\ \mathit{mvote},\ \mathit{mround},\ \mathit{mstate}) \triangleq \mathit{outOfElection}' = \textsc{case}\ \ \mathit{outOfElection}[i][\mathit{id}].\mathit{round} < \mathit{mround} \rightarrow [\mathit{ou} \\
\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \Box\quad \mathit{outOfElection}[i][\mathit{id}].\mathit{round} = \mathit{mround} \rightarrow [\mathit{ou} \\
\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \Box\quad \mathit{outOfElection}[i][\mathit{id}].\mathit{round} > \mathit{mround} \rightarrow \mathit{out}
\end{array}$$

$$\begin{array}{ll}
\mathit{InitServerVarsL} \triangleq & \land\ \mathit{state} \qquad\qquad\ = [s \in \mathit{Server} \mapsto \mathit{LOOKING}] \\
& \land\ \mathit{currentEpoch} = [s \in \mathit{Server} \mapsto 0] \\
& \land\ \mathit{lastProcessed} = [s \in \mathit{Server} \mapsto [\begin{array}[t]{ll}\mathit{index} & \mapsto 0, \\ \mathit{zxid} & \mapsto \langle 0,\ 0\rangle]]\end{array} \\
& \land\ \mathit{history} \qquad\ \ = [s \in \mathit{Server} \mapsto \langle\rangle]
\end{array}$$

$$\begin{array}{ll}
\mathit{InitElectionVarsL} \triangleq & \land\ \mathit{currentVote} \quad\ \ = [s\ \in \mathit{Server} \mapsto \mathit{SelfVote}(s)] \\
& \land\ \mathit{logicalClock} \quad\ = [s\ \in \mathit{Server} \mapsto 0] \\
& \land\ \mathit{receiveVotes} \quad = [s\ \in \mathit{Server} \mapsto [v \in \mathit{Server} \mapsto [\begin{array}[t]{ll}
\mathit{vote} & \mapsto \mathit{InitialVote}, \\
\mathit{round} & \mapsto 0, \\
\mathit{state} & \mapsto \mathit{LOOKING}, \\
\mathit{version} & \mapsto 0]]]
\end{array} \\
& \land\ \mathit{outOfElection} = [s\ \in \mathit{Server} \mapsto [v \in \mathit{Server} \mapsto [\begin{array}[t]{ll}
\mathit{vote} & \mapsto \mathit{InitialVote}, \\
\mathit{round} & \mapsto 0, \\
\mathit{state} & \mapsto \mathit{LOOKING}, \\
\mathit{version} & \mapsto 0]]]
\end{array} \\
& \land\ \mathit{recvQueue} \qquad = [s \in \mathit{Server} \mapsto \langle\rangle] \\
& \land\ \mathit{waitNotmsg} \quad\ \ = [s\ \in \mathit{Server} \mapsto \textsc{false}]
\end{array}$$

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

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

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

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

Abstraction of $WorkerReceiver.run()$

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

```
            $\land\ recvQueue' = [recvQueue$ 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 $\langle serverVarsL,\ currentVote,\ logicalClock,\ receiveVotes,\ outOfElection,\ waitNotmsg,\ lead$

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

----------------------------------------------------------------------------

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

```
        $\land$ UNCHANGED $\langle logicalClock, outOfElection \rangle$
    $\lor$ $\land \neg leaveOk1$
        $\land outOfElection' = [outOfElection$ EXCEPT $![i] = Put(i, n.msource, outOfElection[i], n.mvote, n$
        $\land$ LET $voteSet2 \triangleq VoteSet(i, n.msource, outOfElection'[i], n.mvote, n.mround)$
                $hasQuorums2 \triangleq voteSet2 \in Quorums$
                $check2 \triangleq CheckLeader(i, outOfElection'[i], n.mvote.proposedLeader, n.mround)$
                $leaveOk2 \triangleq \land hasQuorums2$
                                $\land check2$
          IN
          $\lor$ $\land leaveOk2$
             $\land PrintT($"leave with condition 2"$)$
             $\land logicalClock' = [logicalClock$ EXCEPT $![i] = n.mround]$
             $\land state' = [state$ EXCEPT $![i] =$ IF $n.mvote.proposedLeader = i$ THEN $LEADING$ ELSE $FO$
             $\land leadingVoteSet' = [leadingVoteSet$ EXCEPT $![i] =$ IF $n.mvote.proposedLeader = i$ THEN $vo$
             $\land UpdateProposal(i, n.mvote.proposedLeader, n.mvote.proposedZxid, n.mvote.proposedEpoc$
          $\lor$ $\land \neg leaveOk2$
             $\land$ LET $leaveOk3 \triangleq \land n.mstate = LEADING$
                                 $\land n.mround = logicalClock[i]$
               IN
               $\lor$ $\land leaveOk3$
                  $\land PrintT($"leave with condition 3"$)$
                  $\land state' = [state$ EXCEPT $![i] =$ IF $n.mvote.proposedLeader = i$ THEN $LEADING$ ELSE
                  $\land UpdateProposal(i, n.mvote.proposedLeader, n.mvote.proposedZxid, n.mvote.propose$
               $\lor$ $\land \neg leaveOk3$
                  $\land$ UNCHANGED $\langle state, currentVote \rangle$
             $\land$ UNCHANGED $\langle logicalClock, leadingVoteSet \rangle$

Main part of lookForLeader()
$HandleNotmsg(i) \triangleq$
        $\land state[i] = LOOKING$
        $\land \neg waitNotmsg[i]$
        $\land recvQueue[i] \neq \langle\rangle$
        $\land$ LET $n \triangleq recvQueue[i][1]$
                $rawToSend \triangleq [mtype \mapsto NOTIFICATION,$
                                 $msource \mapsto i,$
                                 $mstate \mapsto LOOKING,$
                                 $mround \mapsto logicalClock[i],$
                                 $mvote \mapsto currentVote[i]]$
          IN  $\lor$ $\land n.mtype = NONE$
                 $\land BroadcastNotmsg(i, rawToSend)$
                 $\land$ UNCHANGED $\langle history, logicalClock, currentVote, receiveVotes, waitNotmsg, outOfElection$
              $\lor$ $\land n.mtype = NOTIFICATION$
                 $\land$ $\lor$ $\land n.mstate = LOOKING$
                        $\land$ $\lor$  $n.round \geq$ my round, then update data and $receiveVotes$.
                            $\land n.mround \geq logicalClock[i]$
```

```
$\land \lor$  $n.round >$ my round, update round and decide new proposed leader.
        $\land n.mround > logicalClock[i]$
        $\land logicalClock' = [logicalClock$ EXCEPT $![i] = n.mround]$  There should be $RvCle$
        $\land$ 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
        $\land n.mround = logicalClock[i]$
        $\land$ LET $peerOk \triangleq TotalOrderPredicate(n.mvote, currentVote[i])$
          IN  $\lor \land peerOk$
                 $\land UpdateProposal(i, n.mvote.proposedLeader, n.mvote.proposedZxi$
                 $\land BroadcastNotmsg(i, [mtype \mapsto NOTIFICATION,$
                                           $msource \mapsto i,$
                                           $mstate \mapsto LOOKING,$
                                           $mround \mapsto logicalClock[i],$
                                           $mvote \mapsto n.mvote])$
              $\lor \land \neg peerOk$
                 $\land$ 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 $\langle state, history, outOfElection, leadingVoteSet\rangle$
$\lor$  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 $LFE.java$.

```
$WaitNewNotmsg(i) \triangleq$
        $\land$ $state[i] = LOOKING$
        $\land$ $waitNotmsg[i] =$ 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 $\langle serverVarsL, currentVote, logicalClock, receiveVotes, outOfElection, leaderVarsL, elec$
```

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

```
$WaitNewNotmsgEnd(i) \triangleq$
        $\land$ $state[i] = LOOKING$
        $\land$ $waitNotmsg[i] =$ 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 $\langle state, lastProcessed, history, electionVarsL, leaderVarsL, electionMsgs \rangle$
```

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

$LeaderAdvanceZxid(i) \triangleq$
        $\land\ state[i] = LEADING$
        $\land\ lastProcessed' = [lastProcessed\ \textsc{except}\ ![i] = \textsc{if}\ lastProcessed[i].zxid[1] = currentEpoch[i]$
                                                      $\textsc{then}\ [\ index \mapsto lastProcessed[i].index + 1,$
                                                      $\quad\ \ zxid \mapsto \langle currentEpoch[i], lastProcessed[i].zxid[2] + 1 \rangle]$
                                                      $\textsc{else}\ \ [\ index \mapsto lastProcessed[i].index + 1,$
                                                      $\quad\ \ zxid \mapsto \langle currentEpoch[i], 1 \rangle]]$
        $\land\ history' = [history\ \textsc{except}\ ![i] = Append(@, [zxid \mapsto lastProcessed'[i].zxid,$
                                                            $value \mapsto NONE,$
                                                            $ackSid \mapsto \{\},$
                                                            $epoch \mapsto 0])]$
        $\land\ \textsc{unchanged}\ \langle state, currentEpoch, electionVarsL, leaderVarsL, electionMsgs \rangle$

$FollowerUpdateZxid(i, j) \triangleq$
        $\land\ state[i] = FOLLOWING$
        $\land\ currentVote[i].proposedLeader = j$
        $\land\ state[j] = LEADING$
        $\land\ \textsc{let}\ precede \triangleq \lor\ lastProcessed[i].zxid[1] < lastProcessed[j].zxid[1]$
                                 $\lor\ \land\ lastProcessed[i].zxid[1] = lastProcessed[j].zxid[1]$
                                    $\land\ lastProcessed[i].zxid[2] < lastProcessed[j].zxid[2]$
          $\textsc{in}\quad \land\ precede$
                $\land\ lastProcessed' = [lastProcessed\ \textsc{except}\ ![i] = lastProcessed[j]]$
                $\land\ history' = [history\ \textsc{except}\ ![i] = history[j]]$
        $\land\ \textsc{unchanged}\ \langle state, currentEpoch, electionVarsL, leaderVarsL, electionMsgs \rangle$

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

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

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

These invariants should be violated after running for minutes.

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

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

\ ∗ 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