―――――――――――――――― MODULE $P4RuntimeElection$ ――――――――――――――――

EXTENDS $Naturals$, $FiniteSets$, $Sequences$, TLC

The set of all $ONOS$ nodes
CONSTANTS $Nodes$

Stream states
CONSTANTS $Open$, $Closed$

Master arbitration message types
CONSTANTS $MasterArbitrationUpdate$

Write message types
CONSTANTS $WriteRequest$, $WriteResponse$

Response status constants
CONSTANTS Ok, $AlreadyExists$, $PermissionDenied$

Empty value
CONSTANT Nil

The current state of mastership elections
VARIABLES $term$, $master$, $backups$

The current mastership event queue for each node
VARIABLE $events$

The current mastership state for each node
VARIABLE $masterships$

Whether the node has received a $MasterArbitrationUpdate$ indicating it is the current master
VARIABLE $isMaster$

The state of all streams and their requests and responses
VARIABLE $streams$, $requests$, $responses$

The current set of elections for the switch, the greatest of which is the current master
VARIABLE $elections$

Counting variables used to enforce state constraints
VARIABLES $mastershipChanges$, $streamChanges$, $messageCount$

A sequence of successful writes to the switch used for model checking
VARIABLE $writes$

――

Mastership/consensus related variables
$mastershipVars \triangleq \langle term, master, backups, mastershipChanges \rangle$

1

Node related variables
$nodeVars \triangleq \langle events,\ masterships,\ isMaster \rangle$

Stream related variables
$streamVars \triangleq \langle streams,\ streamChanges \rangle$

Message related variables
$messageVars \triangleq \langle requests,\ responses,\ messageCount \rangle$

Device related variables
$deviceVars \triangleq \langle elections,\ writes \rangle$

A sequence of all variables
$vars \triangleq \langle mastershipVars,\ nodeVars,\ streamVars,\ messageVars,\ deviceVars \rangle$

Helpers

Returns a sequence with the head removed
$Pop(q) \triangleq SubSeq(q,\ 2,\ Len(q))$

Returns a sequences with the element at the given index removed
$Drop(q,\ i) \triangleq SubSeq(q,\ 1,\ i-1) \circ SubSeq(q,\ i+1,\ Len(q))$

Returns the set of values in f
$Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$

Returns the maximum value from a set or undefined if the set is empty
$Max(s) \triangleq \text{CHOOSE } x \in s : \forall\, y \in s : x \geq y$

Messaging between the *Nodes* and the device are modelled on *TCP*. For each node, a request and response sequence provides ordered messaging between the two points. Requests and responses are always received from the head of the queue and are never duplicated or reordered, and request and response queues only last the lifetime of the stream. When a stream is closed, all that stream's requests and responses are lost.

Sends request 'm' on the stream for node 'n'
$SendRequest(n,\ m) \triangleq$
$\qquad \wedge requests' = [requests \text{ EXCEPT } ![n] = Append(requests[n],\ m)]$
$\qquad \wedge messageCount' = messageCount + 1$

Indicates whether any requests are in the queue for node 'n'
$HasRequest(n,\ t) \triangleq Len(requests[n]) > 0 \wedge requests[n][1].type = t$

Returns the next request in the queue for node 'n'
$NextRequest(n) \triangleq requests[n][1]$

Discards the request at the head of the queue for node 'n'
$DiscardRequest(n) \triangleq requests' = [requests \text{ EXCEPT } ![n] = Pop(requests[n])]$

Sends response 'm' on the stream for node 'n'

$SendResponse(n, m) \triangleq$
$\quad \land responses' = [responses \text{ EXCEPT } ![n] = Append(responses[n], m)]$
$\quad \land messageCount' = messageCount + 1$

Indicates whether any responses are in the queue for node 'n'

$HasResponse(n, t) \triangleq Len(responses[n]) > 0 \land responses[n][1].type = t$

Returns the next response in the queue for node 'n'

$NextResponse(n) \triangleq responses[n][1]$

Discards the response at the head of the queue for node 'n'

$DiscardResponse(n) \triangleq responses' = [responses \text{ EXCEPT } ![n] = Pop(responses[n])]$

This section models mastership arbitration on the controller side. Mastership election occurs in two disctinct types of state changes. One state change occurs to change the mastership in the consensus layer, and the other occurs when a node actually learns of the mastership change. Nodes will always learn of mastership changes in the order in which they occur, and nodes will always learn of a mastership change. This, of course, is not representative of practice but is sufficient for modelling the mastership election algorithm.

Adds a node to the mastership election

$JoinMastershipElection(n) \triangleq$
$\quad \land \lor \land master = Nil$
$\qquad \land term' = term + 1$
$\qquad \land master' = n$
$\qquad \land backups' = \langle\rangle$
$\qquad \land events' = [i \in Nodes \mapsto Append(events[i], [$
$\qquad\qquad\qquad\qquad\qquad\qquad term \mapsto term',$
$\qquad\qquad\qquad\qquad\qquad\qquad master \mapsto master',$
$\qquad\qquad\qquad\qquad\qquad\qquad backups \mapsto backups'])]$
$\quad\quad \lor \land master \neq Nil$
$\qquad \land master \neq n$
$\qquad \land n \notin Range(backups)$
$\qquad \land backups' = Append(backups, n)$
$\qquad \land events' = [i \in Nodes \mapsto Append(events[i], [$
$\qquad\qquad\qquad\qquad\qquad\qquad term \mapsto term,$
$\qquad\qquad\qquad\qquad\qquad\qquad master \mapsto master,$
$\qquad\qquad\qquad\qquad\qquad\qquad backups \mapsto backups'])]$
$\qquad \land \text{UNCHANGED } \langle term, master \rangle$
$\quad \land mastershipChanges' = mastershipChanges + 1$
$\quad \land \text{UNCHANGED } \langle masterships, isMaster, streamVars, messageVars, deviceVars \rangle$

Removes a node from the mastership election

$LeaveMastershipElection(n) \triangleq$
$\quad \land \lor \land master = n$
$\qquad \land \lor \land Len(backups) > 0$

3

$\quad\quad\quad\quad\quad\quad \wedge\ term' = term + 1$
$\quad\quad\quad\quad\quad\quad \wedge\ master' = backups[1]$
$\quad\quad\quad\quad\quad\quad \wedge\ backups' = Pop(backups)$
$\quad\quad\quad\quad\quad\quad \wedge\ events' = [i\ \ \in\ Nodes \mapsto Append(events[i], [$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad term \mapsto term',$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad master \mapsto master',$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad backups \mapsto backups'])]$
$\quad\quad\quad\quad \vee\ \wedge\ Len(backups) = 0$
$\quad\quad\quad\quad\quad\ \wedge\ master' = Nil$
$\quad\quad\quad\quad\quad\ \wedge\ \text{UNCHANGED}\ \langle term,\ backups,\ events\rangle$
$\quad\quad \vee\ \wedge\ n \in Range(backups)$
$\quad\quad\quad\ \wedge\ backups' = Drop(backups,\ \text{CHOOSE}\ j \in \text{DOMAIN}\ backups : backups[j] = n)$
$\quad\quad\quad\ \wedge\ \text{UNCHANGED}\ \langle term,\ master,\ events\rangle$
$\quad \wedge\ mastershipChanges' = mastershipChanges + 1$
$\quad \wedge\ \text{UNCHANGED}\ \langle masterships,\ isMaster,\ streamVars,\ messageVars,\ deviceVars\rangle$

$SetMastership(n) \ \triangleq$
$\quad \vee\ \wedge\ master = n$
$\quad\quad\ \wedge\ \text{UNCHANGED}\ \langle mastershipVars\rangle$
$\quad \vee\ \wedge\ master \neq n$
$\quad\quad\ \wedge\ term' = term + 1$
$\quad\quad\ \wedge\ master' = n$
$\quad\quad\ \wedge\ \vee\ \wedge\ n\ \in Range(backups)$
$\quad\quad\quad\quad\quad \wedge\ backups' = Drop(backups,\ \text{CHOOSE}\ j \in \text{DOMAIN}\ backups : backups[j] = n)$
$\quad\quad\quad\ \vee\ \wedge\ n \notin Range(backups)$
$\quad\quad\quad\quad\quad \wedge\ \text{UNCHANGED}\ \langle backups\rangle$
$\quad\quad\ \wedge\ mastershipChanges' = mastershipChanges + 1$

$LearnMastership(n)\ \triangleq$
$\quad \wedge\ Len(events[n]) > 0$
$\quad \wedge\ \text{LET}\ e\ \ \triangleq\ events[n][1]$
$\quad\quad\quad\ \ m\ \ \triangleq\ masterships[n]$
$\quad\quad \text{IN}$
$\quad\quad\quad \vee\ \wedge\ e.term > m.term$
$\quad\quad\quad\quad\ \wedge\ masterships' = [masterships\ \text{EXCEPT}\ ![n] = [$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad term\quad\ \mapsto e.term,$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad master\ \mapsto e.master,$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad backups \mapsto e.backups,$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad sent\quad\ \mapsto \text{FALSE}]]$
$\quad\quad\quad \vee\ \wedge\ e.term = m.term$
$\quad\quad\quad\quad\ \wedge\ masterships' = [masterships\ \text{EXCEPT}\ ![n] = [$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad term\quad\ \mapsto e.term,$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad master\ \mapsto e.master,$

$$\qquad\qquad\qquad\qquad backups \mapsto e.backups,$$
$$\qquad\qquad\qquad\qquad sent \quad\ \mapsto m.sent]]$$
$\land\ events' = [events\ \text{EXCEPT}\ ![n] = Pop(events[n])]$
$\land\ \text{UNCHANGED}\ \langle mastershipVars,\ isMaster,\ streamVars,\ messageVars,\ deviceVars\rangle$

Notifies the device of node 'n' mastership info if it hasn't already been sent
$SendMasterArbitrationUpdateRequest(n)\ \triangleq$
$\quad \land\ streams[n] = Open$
$\quad \land\ \text{LET}\ m\ \triangleq\ masterships[n]$
$\qquad \text{IN}$
$\qquad\quad \land\ m.term > 0$
$\qquad\quad \land\ \neg m.sent$
$\qquad\quad \land\ \lor\ \land\ m.master = n$
$\qquad\qquad\qquad \land\ SendRequest(n, [$
$\qquad\qquad\qquad\qquad type \qquad\quad \mapsto MasterArbitrationUpdate,$
$\qquad\qquad\qquad\qquad election\_id \mapsto m.term + Cardinality(Nodes),$
$\qquad\qquad\qquad\qquad term \qquad\quad \mapsto m.term])$
$\qquad\qquad \lor\ \land\ m.master \neq n$
$\qquad\qquad\qquad \land\ n \in Range(m.backups)$
$\qquad\qquad\qquad \land\ SendRequest(n, [$
$\qquad\qquad\qquad\qquad type \qquad\quad \mapsto MasterArbitrationUpdate,$
$\qquad\qquad\qquad\qquad election_id \mapsto m.term + Cardinality(Nodes) - \text{CHOOSE}\ i \in \text{DOMAIN}\ m.backups : m.back\ $
$\qquad\qquad\qquad\qquad term \qquad\quad \mapsto m.term])$
$\quad \land\ masterships' = [masterships\ \text{EXCEPT}\ ![n].sent = \text{TRUE}]$
$\quad \land\ \text{UNCHANGED}\ \langle mastershipVars,\ events,\ isMaster,\ deviceVars,\ streamVars,\ responses\rangle$

Receives a master arbitration update response on node 'n'
$ReceiveMasterArbitrationUpdateResponse(n)\ \triangleq$
$\quad \land\ streams[n] = Open$
$\quad \land\ HasResponse(n,\ MasterArbitrationUpdate)$
$\quad \land\ \text{LET}\ m\ \triangleq\ NextResponse(n)$
$\qquad \text{IN}$
$\qquad\quad \lor\ \land\ m.status = Ok$
$\qquad\qquad\quad \land\ isMaster' = [isMaster\ \text{EXCEPT}\ ![n] = \text{TRUE}]$
$\qquad\qquad\quad \land\ SetMastership(n)$
$\qquad\quad \lor\ \land\ m.status = AlreadyExists$
$\qquad\qquad\quad \land\ isMaster' = [isMaster\ \text{EXCEPT}\ ![n] = \text{FALSE}]$
$\qquad\qquad\quad \land\ \text{UNCHANGED}\ \langle mastershipVars\rangle$
$\quad \land\ DiscardResponse(n)$
$\quad \land\ \text{UNCHANGED}\ \langle events,\ masterships,\ deviceVars,\ streamVars,\ requests,\ messageCount\rangle$

Sends a write request to the device from node 'n'
$SendWriteRequest(n)\ \triangleq$
$\quad \land\ streams[n] = Open$
$\quad \land\ \text{LET}\ m\ \triangleq\ masterships[n]$
$\qquad \text{IN}$

5

$\quad\quad\quad \wedge\ m.term > 0$
$\quad\quad\quad \wedge\ m.master = n$
$\quad\quad\quad \wedge\ isMaster[n]$
$\quad\quad\quad \wedge\ SendRequest(n,\ [$
$\quad\quad\quad\quad\quad type \quad\quad\quad \mapsto WriteRequest,$
$\quad\quad\quad\quad\quad election\_id \mapsto m.term + Cardinality(Nodes),$
$\quad\quad\quad\quad\quad term \quad\quad\quad \mapsto m.term])$
$\quad\quad \wedge\ \text{UNCHANGED}\ \langle mastershipVars,\ nodeVars,\ deviceVars,\ streamVars,\ responses \rangle$

Receives a write response on node 'n'
$ReceiveWriteResponse(n)\ \triangleq$
$\quad\quad \wedge\ streams[n] = Open$
$\quad\quad \wedge\ HasResponse(n,\ WriteResponse)$
$\quad\quad \wedge\ \text{LET}\ m\ \triangleq\ NextResponse(n)$
$\quad\quad\quad \text{IN}$

$\quad\quad\quad\quad TODO$: This should be used to determine whether writes from old masters are allowed
$\quad\quad\quad\quad \vee\ m.status = Ok$
$\quad\quad\quad\quad \vee\ m.status = PermissionDenied$
$\quad\quad \wedge\ DiscardResponse(n)$
$\quad\quad \wedge\ \text{UNCHANGED}\ \langle mastershipVars,\ nodeVars,\ deviceVars,\ streamVars,\ requests,\ messageCount \rangle$

This section models the $P4$ switch. The switch side manages stream states between the device and the controller. Streams are opened and closed in a single state transition for the purposes of this model. Switches can handle two types of messages from the controller nodes: $MasterArbitrationUpdate$ and Write.

Returns the highest election ID for the given elections
$ElectionId(e)\ \triangleq\ Max(Range(e))$

Returns the master for the given elections
$Master(e)\ \triangleq$
$\quad\quad \text{IF}\ Cardinality(\{i \in Range(e) : i > 0\}) > 0\ \text{THEN}$
$\quad\quad\quad \text{CHOOSE}\ n \in \text{DOMAIN}\ e : e[n] = ElectionId(e)$
$\quad\quad \text{ELSE}$
$\quad\quad\quad Nil$

Opens a new stream between node 'n' and the device
When a new stream is opened, the 'requests' and 'responses' queues for the node are cleared and the 'streams' state is set to 'Open'.
$ConnectStream(n)\ \triangleq$
$\quad\quad \wedge\ streams[n] = Closed$
$\quad\quad \wedge\ streams' = [streams\ \text{EXCEPT}\ ![n] = Open]$
$\quad\quad \wedge\ streamChanges' = streamChanges + 1$
$\quad\quad \wedge\ \text{UNCHANGED}\ \langle mastershipVars,\ nodeVars,\ deviceVars,\ messageVars \rangle$

Closes the open stream between node 'n' and the device

$CloseStream(n) \triangleq$
$\quad \wedge streams[n] = Open$
$\quad \wedge elections' = [elections \text{ EXCEPT } ![n] = 0]$
$\quad \wedge streams' = [streams \text{ EXCEPT } ![n] = Closed]$
$\quad \wedge requests' = [requests \text{ EXCEPT } ![n] = \langle\rangle]$
$\quad \wedge \text{ LET } oldMaster \triangleq Master(elections)$
$\qquad\qquad newMaster \triangleq Master(elections')$
$\quad\quad \text{ IN }$
$\qquad \vee \wedge oldMaster \neq newMaster$
$\qquad\quad \wedge responses' = [i \in \text{DOMAIN } streams' \mapsto$
$\qquad\qquad\qquad\qquad\qquad \text{IF } streams'[i] = Open \text{ THEN}$
$\qquad\qquad\qquad\qquad\qquad\quad \text{IF } i = newMaster \text{ THEN}$
$\qquad\qquad\qquad\qquad\qquad\qquad Append(responses[i], [$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad type \qquad\quad \mapsto MasterArbitrationUpdate,$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad status \qquad \mapsto Ok,$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad election\_id \mapsto ElectionId(elections')])$
$\qquad\qquad\qquad\qquad\qquad\quad \text{ELSE}$
$\qquad\qquad\qquad\qquad\qquad\qquad Append(responses[i], [$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad type \qquad\quad \mapsto MasterArbitrationUpdate,$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad status \qquad \mapsto AlreadyExists,$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad election\_id \mapsto ElectionId(elections')])$
$\qquad\qquad\qquad\qquad\qquad \text{ELSE}$
$\qquad\qquad\qquad\qquad\qquad\quad \langle\rangle]$
$\qquad\quad \wedge messageCount' = messageCount + 1$
$\qquad \vee \wedge oldMaster = newMaster$
$\qquad\quad \wedge responses' = [responses \text{ EXCEPT } ![n] = \langle\rangle]$
$\qquad\quad \wedge \text{UNCHANGED } \langle messageCount \rangle$
$\quad \wedge streamChanges' = streamChanges + 1$
$\quad \wedge \text{UNCHANGED } \langle mastershipVars, nodeVars, writes \rangle$

$HandleMasterArbitrationUpdate(n) \triangleq$
$\quad \wedge streams[n] = Open$
$\quad \wedge HasRequest(n, MasterArbitrationUpdate)$
$\quad \wedge \text{ LET } m \triangleq NextRequest(n)$
$\quad\quad \text{ IN }$
$\qquad \vee \wedge m.election\_id \in Range(elections)$
$\qquad\quad \wedge elections[n] \neq m.election\_id$
$\qquad\quad \wedge streams' = [streams \text{ EXCEPT } ![n] = Closed]$
$\qquad\quad \wedge requests' = [requests \text{ EXCEPT } ![n] = \langle\rangle]$

7

$$\wedge\ responses' = [responses\ \text{EXCEPT}\ ![n] = \langle\rangle]$$
$$\wedge\ \text{UNCHANGED}\ \langle deviceVars,\ streamChanges,\ messageCount\rangle$$
$$\vee\ \wedge\ m.election\_id \notin Range(elections)$$
$$\wedge\ elections' = [elections\ \text{EXCEPT}\ ![n] = m.election\_id]$$
$$\wedge\ \text{LET}\ oldMaster\ \triangleq\ Master(elections)$$
$$newMaster\ \triangleq\ Master(elections')$$
$$\text{IN}$$
$$\vee\ \wedge\ oldMaster \neq newMaster$$
$$\wedge\ responses' = [i \in \text{DOMAIN}\ streams \mapsto$$
$$\text{IF}\ streams[i] = Open\ \ \text{THEN}$$
$$\text{IF}\ i = newMaster\ \text{THEN}$$
$$Append(responses[i],\ [$$
$$type\ \ \ \ \ \ \ \ \ \ \mapsto MasterArbitrationUpdate,$$
$$status\ \ \ \ \ \ \ \mapsto Ok,$$
$$election\_id \mapsto ElectionId(elections')])$$
$$\text{ELSE}$$
$$Append(responses[i],\ [$$
$$type\ \ \ \ \ \ \ \ \ \ \mapsto MasterArbitrationUpdate,$$
$$status\ \ \ \ \ \ \ \mapsto AlreadyExists,$$
$$election\_id \mapsto ElectionId(elections')])$$
$$\text{ELSE}$$
$$responses[i]]$$
$$\wedge\ messageCount' = messageCount + 1$$
$$\vee\ \wedge\ oldMaster = newMaster$$
$$\wedge\ SendResponse(n,\ [$$
$$type\ \ \ \ \ \ \ \ \ \ \mapsto MasterArbitrationUpdate,$$
$$status\ \ \ \ \ \ \mapsto Ok,$$
$$election\_id \mapsto ElectionId(elections')])$$
$$\wedge\ \text{UNCHANGED}\ \langle streamVars\rangle$$
$$\wedge\ DiscardRequest(n)$$
$$\wedge\ \text{UNCHANGED}\ \langle mastershipVars,\ nodeVars,\ writes\rangle$$

Handles a write request on the device

$HandleWrite(n)\ \triangleq$
$$\wedge\ streams[n] = Open$$
$$\wedge\ HasRequest(n,\ WriteRequest)$$
$$\wedge\ \text{LET}\ m\ \triangleq\ NextRequest(n)$$
$$\text{IN}$$
$$\vee\ \wedge\ elections[n] = m.election\_id$$
$$\wedge\ Master(elections) = n$$
$$\wedge\ writes' = Append(writes,\ [node \mapsto n,\ term \mapsto m.term])$$
$$\wedge\ SendResponse(n,\ [$$
$$type\ \ \ \mapsto WriteResponse,$$
$$status \mapsto Ok])$$
$$\vee\ \wedge\ \vee\ elections[n] \neq m.election\_id$$

8

$$\lor\ Master(elections) \neq n$$
$$\land\ SendResponse(n,\ [$$
$$\qquad type\quad \mapsto WriteResponse,$$
$$\qquad status \mapsto PermissionDenied])$$
$$\quad\land\ \text{UNCHANGED}\ \langle writes \rangle$$
$$\land\ DiscardRequest(n)$$
$$\land\ \text{UNCHANGED}\ \langle mastershipVars,\ nodeVars,\ elections,\ streamVars \rangle$$

The invariant asserts that no master can write to the switch after the switch
has been notified of a newer master

$TypeInvariant\ \triangleq\ \forall\,i \in \text{DOMAIN}\ writes : i = 1 \lor writes[i-1].term \leq writes[i].term$

$Init\ \triangleq$
$\quad \land\ term = 0$
$\quad \land\ master = Nil$
$\quad \land\ backups = \langle\rangle$
$\quad \land\ events = [n\ \in\ Nodes \mapsto \langle\rangle]$
$\quad \land\ masterships = [n \in Nodes \mapsto [term \mapsto 0,\ master \mapsto 0,\ backups \mapsto \langle\rangle,\ sent \mapsto \text{FALSE}]]$
$\quad \land\ isMaster = [n \in Nodes \mapsto \text{FALSE}]$
$\quad \land\ streams = [n \in Nodes \mapsto Closed]$
$\quad \land\ requests = [n \in Nodes \mapsto \langle\rangle]$
$\quad \land\ responses = [n \in Nodes \mapsto \langle\rangle]$
$\quad \land\ elections\ = [n \in Nodes \mapsto 0]$
$\quad \land\ mastershipChanges = 0$
$\quad \land\ streamChanges = 0$
$\quad \land\ messageCount = 0$
$\quad \land\ writes = \langle\rangle$

$Next\ \triangleq$
$\quad \lor\ \exists\,n \in Nodes : ConnectStream(n)$
$\quad \lor\ \exists\,n \in Nodes : CloseStream(n)$
$\quad \lor\ \exists\,n \in Nodes : JoinMastershipElection(n)$
$\quad \lor\ \exists\,n \in Nodes : LeaveMastershipElection(n)$
$\quad \lor\ \exists\,n \in Nodes : LearnMastership(n)$
$\quad \lor\ \exists\,n \in Nodes : SendMasterArbitrationUpdateRequest(n)$
$\quad \lor\ \exists\,n \in Nodes : HandleMasterArbitrationUpdate(n)$
$\quad \lor\ \exists\,n \in Nodes : ReceiveMasterArbitrationUpdateResponse(n)$
$\quad \lor\ \exists\,n \in Nodes : SendWriteRequest(n)$
$\quad \lor\ \exists\,n \in Nodes : HandleWrite(n)$
$\quad \lor\ \exists\,n \in Nodes : ReceiveWriteResponse(n)$

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

\ * Modification History
\ * Last modified *Mon Feb* 18 22:56:25 *PST* 2019 by *jordanhalterman*
\ * Created *Thu Feb* 14 11:33:03 *PST* 2019 by *jordanhalterman*