MODULE *GenesisCeremony*

EXTENDS *FiniteSets*, *Integers*, *Sequences*

CONSTANT

Nodes,	set of participating nodes
	Nodes $\subseteq$ [*id* : 1 .. *n*, bootstrap : 1 .. *n*, status : *PossibleStatuses*]
None	hack: Options = {*Data*} $\cup$ {*None*}

VARIABLES

nodes,	set of Node *ids* $\leftarrow$ does not change
nStatus,	tuple of Node statuses
nInMsg,	tuple of incoming message tuples
nOutStreamMsg,	tuple of outgoing stream message tuples
nOutMsg	tuple of outgoing messages

$vars \triangleq \langle nodes, nStatus, nInMsg, nOutStreamMsg, nOutMsg \rangle$

Auxiliary functions from *SequencesExt* community module. See https://*github.com*/tlaplus/

$ToSet(s) \triangleq \{s[i] : i \in \text{DOMAIN } s\}$

$IsInjective(s) \triangleq \forall\, i, j \in \text{DOMAIN } s : (s[i] = s[j]) \Rightarrow (i = j)$

$SetToSeq(S) \triangleq \text{CHOOSE } f \in [1 \,..\, Cardinality(S) \to S] : IsInjective(f)$

Message *_msg* packaged with sender *_from* and receiver *_to*

$Pack(\_msg, \_from, \_to) \triangleq [msg \mapsto \_msg, from \mapsto \_from, to \mapsto \_to]$

_from, *_to* $\in$ *nodes*

$StreamMsg(\_msg, \_from, \_to) \triangleq$
$nOutStreamMsg' = [nOutStreamMsg \text{ EXCEPT } ![\_from] = Append(nOutStreamMsg[\_from], Pack(\_msg, \_from, \_to))]$

Given a set of recipients *_tos*, the sender creates a message pack for each and adds them to their list of outgoing messages.
_tos $\in$ SUBSET *nodes*
Used to complete genesis ceremony.

$BroadcastStream(\_msg, \_from, \_tos) \triangleq$
LET $CreatePacket(\_to) \triangleq Pack(\_msg, \_from, \_to)$
IN $nOutStreamMsg' = [nOutStreamMsg \text{ EXCEPT } ![\_from] = nOutStreamMsg[\_from] \circ SetToSeq(\{CreatePacket(to) : to \in \_tos\})]$

Enabled when a node *n* has nonempty *nOutStreamMsg* whose head has correct *msgType*.
The head *msg* is deleted from sender's outgoing msgs and added to recipient's incoming msgs.

$TransferStreamMsg(msgType) \triangleq \exists\, n \in nodes :$
$\land\ nOutStreamMsg[n] \neq \langle\rangle$
$\land\ Head(nOutStreamMsg[n]).msg.type = msgType$

$\land\ nOutStreamMsg' = [nOutStreamMsg \text{ EXCEPT } ![n] = Tail(nOutStreamMsg[n])]$
$\land\ nInMsg' = [nInMsg \text{ EXCEPT } ![Head(nOutStreamMsg[n]).to]$
$\qquad = Append(nInMsg[Head(nOutStreamMsg[n]).to],\ Head(nOutStreamMsg[n]))]$
$\land\ \text{UNCHANGED } \langle nodes,\ nStatus,\ nOutMsg\rangle$

Next *OutStream* message is dropped.

$LooseStreamMsg \triangleq \exists\, n \in nodes :$
$\land\ nOutStreamMsg[n] \neq \langle\rangle$
$\land\ nOutStreamMsg' = [nOutStreamMsg \text{ EXCEPT } ![n] = Tail(nOutStreamMsg[n])]$
$\land\ \text{UNCHANGED } \langle nodes,\ nStatus,\ nInMsg,\ nOutMsg\rangle$

$SendingStatus \triangleq \{\text{“init”},\ \text{“success”},\ \text{“failed”}\}$

$SentMsgStatus(\_status,\ \_packet) \triangleq [status \mapsto \_status,\ packet \mapsto \_packet]$

Initializes an outgoing message.

$SendMsg(\_msg,\ \_from,\ \_to) \triangleq$
$nOutMsg' = [nOutMsg \text{ EXCEPT } ![\_from]$
$\qquad = SentMsgStatus(\text{“init”},\ Pack(\_msg,\ \_from,\ \_to))]$

When an outgoing message is initialized, it can be successfully transmitted.

$TransferMsg \triangleq \exists\, n \in nodes :$
$\land\ nOutMsg[n] \neq None$
$\land\ nOutMsg[n].status = \text{“init”}$
$\land\ nOutMsg' = [nOutMsg \text{ EXCEPT } ![n].status = \text{“success”}]$
$\land\ nInMsg' = [nInMsg \text{ EXCEPT } ![nOutMsg[n].packet.to]$
$\qquad = Append(nInMsg[nOutMsg[n].packet.to],\ nOutMsg[n].packet)]$
$\land\ \text{UNCHANGED } \langle nodes,\ nStatus,\ nOutStreamMsg\rangle$

When an outgoing message is initialized, transmission can be unsuccessful.

$LooseMsg \triangleq \exists\, n \in nodes :$
$\land\ nOutMsg[n] \neq None$
$\land\ nOutMsg[n].status = \text{“init”}$
$\land\ nOutMsg' = [nOutMsg \text{ EXCEPT } ![n].status = \text{“failed”}]$
$\land\ \text{UNCHANGED } \langle nodes,\ nStatus,\ nInMsg,\ nOutStreamMsg\rangle$

Checks that given node's next In message has given type.

$NextInMsgIs(n,\ msgType) \triangleq$
$\land\ nInMsg[n] \neq \langle\rangle$ $Len(nInMsg[n]) > 0$
$\land\ Head(nInMsg[n]).msg.type = msgType$

Deletes the next In message from given node.

$PopNextInMsg(n) \triangleq nInMsg' = [nInMsg \text{ EXCEPT } ![n] = Tail(nInMsg[n])]$

$Handling(n,\ status,\ msgType) \triangleq$
$\land\ nStatus[n] = status$

$\land\ NextInMsgIs(n,\ msgType)$
$\land\ PopNextInMsg(n)$

$DoNotHandle(status,\ msgType) \triangleq \exists\, n \in nodes :$
$\land\ Handling(n,\ status,\ msgType)$
$\land$ UNCHANGED $\langle nodes,\ nStatus,\ nOutStreamMsg,\ nOutMsg\rangle$

$Sender(n) \triangleq Head(nInMsg[n]).from$

Messages definition – type, status, packet

$MessageType \triangleq$
$\{$"UnapprovedBlock", "BlockApproval", "ApprovedBlockRequest", "ApprovedBlock"$\}$

Set of all message types
$Messages \triangleq [type : MessageType]$
Set of packets
$Packets \triangleq [msg : Messages,\ from : nodes,\ to : nodes]$

$NewMessage(\_type) \triangleq [type \mapsto \_type]$
$NewUnapprovedBlock \triangleq NewMessage($"UnapprovedBlock"$)$
$NewBlockApproval \triangleq NewMessage($"BlockApproval"$)$
$NewApprovedBlockRequest \triangleq NewMessage($"ApprovedBlockRequest"$)$
$NewApprovedBlock \triangleq NewMessage($"ApprovedBlock"$)$

$PossibleStatuses \triangleq$
$\{$"new", "init", "running", "genesis_validator", "ceremony_master"$\}$

$HoldsMessageType(msgQueue) \triangleq ToSet(msgQueue) \subseteq Packets$

Verifies correctness of node status and formation of In, Out, and
OutStream queues for all nodes.
$TypeOK \triangleq \forall\, n \in nodes :$
$\land\ nStatus[n] \in PossibleStatuses$
$\land\ HoldsMessageType(nInMsg[n])$
$\land\ HoldsMessageType(nOutStreamMsg[n])$
$\land\ nOutMsg[n] \in ([status : SendingStatus,\ packet : Packets] \cup \{None\})$

Returns bootstrap node for given node
$Bootstrap(n) \triangleq$ LET $node \triangleq$ (CHOOSE $cn \in Nodes : cn.id = n$)
IN $node.bootstrap$

Transistions the given node to the given status.
$TransitionTo(n,\ status) \triangleq nStatus' = [nStatus$ EXCEPT $![n] = status]$

If node has no Out messages and "new" status, it can change status to "init"
while sending a *NewApprovedBlockRequest* to its bootstrap.

$$
\begin{aligned}
&\mathit{LaunchFromNewToInit} \triangleq \exists\, n \in \mathit{nodes} : \\
&\quad \land\ \mathit{nStatus}[n] = \text{"new"} \\
&\quad \land\ \mathit{nOutMsg}[n] = \mathit{None} \\
&\quad \land\ \mathit{TransitionTo}(n,\ \text{"init"}) \\
&\quad \land\ \mathit{SendMsg}(\mathit{NewApprovedBlockRequest},\ n,\ \mathit{Bootstrap}(n)) \\
&\quad \land\ \text{UNCHANGED}\ \langle \mathit{nodes},\ \mathit{nInMsg},\ \mathit{nOutStreamMsg} \rangle
\end{aligned}
$$

If the node's bootstrapped *NewApprovedBlockRequest* is successful, the success
message can be removed from the node's Out message queue.

$$
\begin{aligned}
&\mathit{SuccessFromNewToInit} \triangleq \exists\, n \in \mathit{nodes} : \\
&\quad \land\ \mathit{nStatus}[n] = \text{"init"} \\
&\quad \land\ \mathit{nOutMsg}[n] = \mathit{SentMsgStatus}(\text{"success"},\ \mathit{Pack}(\mathit{NewApprovedBlockRequest},\ n,\ \mathit{Bootstrap}(n))) \\
&\quad \land\ \mathit{nOutMsg}' = [\mathit{nOutMsg}\ \text{EXCEPT}\ ![n] = \mathit{None}] \\
&\quad \land\ \text{UNCHANGED}\ \langle \mathit{nodes},\ \mathit{nInMsg},\ \mathit{nOutStreamMsg},\ \mathit{nStatus} \rangle
\end{aligned}
$$

If the node's bootstrapped *NewApprovedBlockRequest* is unsuccessful, another
attempt can be made.
Does this allow for inifinite loops of failing/retrying *NewApprovedBlockRequests*?

$$
\begin{aligned}
&\mathit{FailedFromNewToInit} \triangleq \exists\, n \in \mathit{nodes} : \\
&\quad \land\ \mathit{nStatus}[n] = \text{"init"} \\
&\quad \land\ \mathit{nOutMsg}[n] = \mathit{SentMsgStatus}(\text{"failed"},\ \mathit{Pack}(\mathit{NewApprovedBlockRequest},\ n,\ \mathit{Bootstrap}(n))) \\
&\quad \land\ \mathit{SendMsg}(\mathit{NewApprovedBlockRequest},\ n,\ \mathit{Bootstrap}(n)) \\
&\quad \land\ \text{UNCHANGED}\ \langle \mathit{nodes},\ \mathit{nInMsg},\ \mathit{nOutStreamMsg},\ \mathit{nStatus} \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{ResendWhileInit} \triangleq \exists\, n \in \mathit{nodes} : \\
&\quad \land\ \mathit{nStatus}[n] = \text{"init"} \\
&\quad \land\ \mathit{nOutMsg}[n] = \mathit{None} \\
&\quad \land\ \neg(\mathit{Pack}(\mathit{NewApprovedBlockRequest},\ n,\ \mathit{Bootstrap}(n)) \in \mathit{ToSet}(\mathit{nInMsg}[\mathit{Bootstrap}(n)])) \\
&\quad \land\ \neg(\mathit{Pack}(\mathit{NewApprovedBlock},\ \mathit{Bootstrap}(n),\ n) \in \mathit{ToSet}(\mathit{nOutStreamMsg}[\mathit{Bootstrap}(n)])) \\
&\quad \land\ \mathit{SendMsg}(\mathit{NewApprovedBlockRequest},\ n,\ \mathit{Bootstrap}(n)) \\
&\quad \land\ \text{UNCHANGED}\ \langle \mathit{nodes},\ \mathit{nInMsg},\ \mathit{nOutStreamMsg},\ \mathit{nStatus} \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{FromNewToInit} \triangleq \\
&\quad \lor\ \mathit{LaunchFromNewToInit} \\
&\quad \lor\ \mathit{SuccessFromNewToInit} \\
&\quad \lor\ \mathit{FailedFromNewToInit} \\
&\quad \lor\ \mathit{ResendWhileInit}
\end{aligned}
$$

Genesis ceremony begins

node with *ceremony_master* status and without *NewUnapprovedBlock* in their In
or *OutStream* queues, can broadcast *NewUnapprovedBlock* to all other nodes.

$$
\begin{aligned}
&\mathit{CMBroadcastUnapprovedBlock} \triangleq \exists\, n \in \mathit{nodes} : \\
&\quad \land\ \mathit{nStatus}[n] = \text{"ceremony\_master"}
\end{aligned}
$$

$\land \neg(\exists\, p \in \mathit{ToSet}(\mathit{nOutStreamMsg}[n]) : p.\mathit{msg}.\mathit{type} = \mathit{NewUnapprovedBlock}.\mathit{type})$
$\land \neg(\exists\, p \in \mathit{ToSet}(\mathit{nInMsg}[n]) : p.\mathit{msg}.\mathit{type} = \mathit{NewUnapprovedBlock}.\mathit{type})$
$\land\ \mathit{BroadcastStream}(\mathit{NewUnapprovedBlock},\ n,\ \mathit{nodes} \setminus \{n\})$
$\land$ UNCHANGED $\langle \mathit{nodes},\ \mathit{nStatus},\ \mathit{nInMsg},\ \mathit{nOutMsg} \rangle$

$\mathit{LaunchCeremonyMaster} \triangleq \mathit{CMBroadcastUnapprovedBlock}$

Ceremony master message handling

Ceremony master node turns a *BlockApproval* into a *NewApprovedBlock* and broadcasts the *NewApprovedBlock* to all other nodes and changes to running status.

$\mathit{CeremonyMasterHandlesBlockApproval} \triangleq \exists\, n \in \mathit{nodes} :$
$\land\ \mathit{Handling}(n,$ "ceremony_master", "BlockApproval"$)$
$\land\ \mathit{TransitionTo}(n,$ "running"$)$
$\land\ \mathit{BroadcastStream}(\mathit{NewApprovedBlock},\ n,\ \mathit{nodes} \setminus \{n\})$
$\land$ UNCHANGED $\langle \mathit{nodes},\ \mathit{nOutMsg} \rangle$

Ceremony master *node*(s) only handle *BlockApproval*

$\mathit{CeremonyMasterHandlesApprovedBlock} \triangleq \mathit{DoNotHandle}($"ceremony_master", "ApprovedBlock"$)$
$\mathit{CeremonyMasterHandlesApprovedBlockRequest} \triangleq \mathit{DoNotHandle}($"ceremony_master", "ApprovedBlockRequest"
$\mathit{CeremonyMasterHandlesUnapprovedBlock} \triangleq \mathit{DoNotHandle}($"ceremony_master", "UnapprovedBlock"$)$

Genesis *validator* message handling

Genesis *validator* node turns an *UnapprovedBlock* in their In queue into a *NewBlockApproval* in their *OutStream* queue and $\rightsquigarrow$ *init* status.

$\mathit{GenesisValidatorHandlesUnapprovedBlock} \triangleq \exists\, n \in \mathit{nodes} :$
$\land\ \mathit{Handling}(n,$ "genesis_validator", "UnapprovedBlock"$)$
$\land\ \mathit{TransitionTo}(n,$ "init"$)$
$\land\ \mathit{StreamMsg}(\mathit{NewBlockApproval},\ n,\ \mathit{Sender}(n))$
$\land$ UNCHANGED $\langle \mathit{nodes},\ \mathit{nOutMsg} \rangle$

Genesis *validator node*(s) only handle *UnapprovedBlock*

$\mathit{GenesisValidatorHandlesApprovedBlock} \triangleq \mathit{DoNotHandle}($"genesis_validator", "ApprovedBlock"$)$
$\mathit{GenesisValidatorHandlesApprovedBlockRequest} \triangleq \mathit{DoNotHandle}($"genesis_validator", "ApprovedBlockRequest"
$\mathit{GenesisValidatorHandlesBlockApproval} \triangleq \mathit{DoNotHandle}($"genesis_validator", "BlockApproval"$)$

Initializing message handling

init node + *ApprovedBlock* in In queue $\rightsquigarrow$ *running* status

$\mathit{InitHandlesApprovedBlock} \triangleq \exists\, n \in \mathit{nodes} :$
$\land\ \mathit{Handling}(n,$ "init", "ApprovedBlock"$)$
$\land\ \mathit{TransitionTo}(n,$ "running"$)$
$\land$ UNCHANGED $\langle \mathit{nodes},\ \mathit{nOutMsg},\ \mathit{nOutStreamMsg} \rangle$

init node with *UnapprovedBlock* in In queue, sends a *NewBlockApproval* to the sender of that *UnapprovedBlock*.

$\mathit{InitHandlesUnapprovedBlock} \triangleq \exists\, n \in \mathit{nodes} :$

$\land$ $Handling(n,$ "init", "UnapprovedBlock"$)$
$\land$ $StreamMsg(NewBlockApproval,\ n,\ Sender(n))$
$\land$ UNCHANGED $\langle nodes,\ nStatus,\ nOutMsg\rangle$

init nodes only handle *ApprovedBlock* or *UnapprovedBlock*

$InitHandlesApprovedBlockRequest \triangleq DoNotHandle($"init", "ApprovedBlockRequest"$)$
$InitHandlesBlockApproval \triangleq DoNotHandle($"init", "BlockApproval"$)$

Running message handling

running node with *ApprovedBlockRequest* in In queue, sends a *NewApprovedBlock* to the sender of that *ApprovedBlockRequest*.

$RunningHandlesApprovedBlockRequest \triangleq \exists\, n \in nodes :$
$\land$ $Handling(n,$ "running", "ApprovedBlockRequest"$)$
$\land$ $StreamMsg(NewApprovedBlock,\ n,\ Sender(n))$
$\land$ UNCHANGED $\langle nodes,\ nOutMsg,\ nStatus\rangle$

running nodes only handle *ApprovedBlockRequest*

$RunningHandlesApprovedBlock \triangleq DoNotHandle($"running", "ApprovedBlock"$)$
$RunningHandlesUnapprovedBlock \triangleq DoNotHandle($"running", "UnapprovedBlock"$)$
$RunningHandlesBlockApproval \triangleq DoNotHandle($"running", "BlockApproval"$)$

At some point, in all later states, all nodes have running status.

$EventuallyAllNodesAreRunning \triangleq \Diamond\Box[\forall\, n \in nodes : nStatus[n] =$ "running"$]_{vars}$

Status of Node with given *id*

$InitNodeStatus(n) \triangleq$ (CHOOSE $cn \in Nodes : cn.id = n).status$

Never used ...

$InitNodeState(n) \triangleq [bootstrap \mapsto Bootstrap(n)]$

LET $node \triangleq$ (CHOOSE $cn \in Nodes : cn.id = n$) IN $[bootstrap \mapsto node.bootstrap]$

nodes are the set of Node ids, nodes are given initial statuses,
In and *OutStream* queues are empty, and Out queues are set to $\langle\rangle$.

$Init \triangleq$
$\land$ $nodes = \{n.id : n \in Nodes\}$
$\land$ $nStatus = [n \in nodes \mapsto InitNodeStatus(n)]$
$\land$ $nInMsg = [n \in nodes \mapsto \langle\rangle]$
$\land$ $nOutStreamMsg = [n \in nodes \mapsto \langle\rangle]$
$\land$ $nOutMsg = [n \in nodes \mapsto None]$
$\land \exists\, n \in nodes : nStatus[n] =$ "ceremony_master"
$\land \exists\, n \in nodes : nStatus[n] =$ "genesis_validator"

$Next \triangleq$
Message drops/transfers
$\lor \exists\, mt \in MessageType : TransferStreamMsg(mt)$

$\lor$ *TransferMsg*
$\lor$ *LooseStreamMsg*
$\lor$ *LooseMsg*
Message/node status changes
$\lor$ *FromNewToInit*
Genesis ceremony
$\lor$ *LaunchCeremonyMaster*
$\lor$ *CeremonyMasterHandlesApprovedBlock*
$\lor$ *CeremonyMasterHandlesApprovedBlockRequest*
$\lor$ *CeremonyMasterHandlesUnapprovedBlock*
$\lor$ *CeremonyMasterHandlesBlockApproval*
$\lor$ *GenesisValidatorHandlesApprovedBlock*
$\lor$ *GenesisValidatorHandlesApprovedBlockRequest*
$\lor$ *GenesisValidatorHandlesUnapprovedBlock*
$\lor$ *GenesisValidatorHandlesBlockApproval*
$\lor$ *InitHandlesApprovedBlock*
$\lor$ *InitHandlesApprovedBlockRequest*
$\lor$ *InitHandlesUnapprovedBlock*
$\lor$ *InitHandlesBlockApproval*
$\lor$ *RunningHandlesApprovedBlock*
$\lor$ *RunningHandlesApprovedBlockRequest*
$\lor$ *RunningHandlesUnapprovedBlock*
$\lor$ *RunningHandlesBlockApproval*

Spec $\triangleq$
safety
$\land$ *Init* $\land \Box([Next]_{vars})$
liveness
$\land \mathrm{SF}_{vars}(\mathit{TransferMsg})$
$\land \forall\, mt \in \mathit{MessageType} : \mathrm{SF}_{vars}(\mathit{TransferStreamMsg}(mt))$
$\land \mathrm{WF}_{vars}(\mathit{FromNewToInit})$
$\land \mathrm{WF}_{vars}(\mathit{LaunchCeremonyMaster})$
$\land \mathrm{WF}_{vars}(\mathit{CeremonyMasterHandlesApprovedBlock})$
$\land \mathrm{WF}_{vars}(\mathit{CeremonyMasterHandlesApprovedBlockRequest})$
$\land \mathrm{WF}_{vars}(\mathit{CeremonyMasterHandlesUnapprovedBlock})$
$\land \mathrm{WF}_{vars}(\mathit{CeremonyMasterHandlesBlockApproval})$
$\land \mathrm{WF}_{vars}(\mathit{GenesisValidatorHandlesApprovedBlock})$
$\land \mathrm{WF}_{vars}(\mathit{GenesisValidatorHandlesApprovedBlockRequest})$
$\land \mathrm{WF}_{vars}(\mathit{GenesisValidatorHandlesUnapprovedBlock})$
$\land \mathrm{WF}_{vars}(\mathit{GenesisValidatorHandlesBlockApproval})$
$\land \mathrm{WF}_{vars}(\mathit{InitHandlesApprovedBlock})$
$\land \mathrm{WF}_{vars}(\mathit{InitHandlesApprovedBlockRequest})$
$\land \mathrm{WF}_{vars}(\mathit{InitHandlesUnapprovedBlock})$
$\land \mathrm{WF}_{vars}(\mathit{InitHandlesBlockApproval})$
$\land \mathrm{WF}_{vars}(\mathit{RunningHandlesApprovedBlock})$

$\land \text{WF}_{vars}(RunningHandlesApprovedBlockRequest)$
$\land \text{WF}_{vars}(RunningHandlesUnapprovedBlock)$
$\land \text{WF}_{vars}(RunningHandlesBlockApproval)$

THEOREM $Spec \Rightarrow EventuallyAllNodesAreRunning$

\ * Modification History
\ * Last modified *Tue Nov* 19 10:18:30 *EST* 2019 by *isaac*
\ * Created *Fri Nov* 15 13:26:32 *EST* 2019 by *isaac*