MODULE *BlockGeneration*

Block generation specifies when and how braidpool miners generate blocks. Block generation captures how coinbase and *UHPO* transactions are or updated. The protocol to build current pool key and threshold signatures is assumed

EXTENDS

TLC,
Sequences,
Integers,
DAG,
FiniteSets

CONSTANT

Miner,	Set of miners
ShareSeqNo,	Share seq numbers each miner generates
BlockReward,	Block reward in a difficulty period
GenesisShare	

VARIABLES

TODO: Replace these *last_*. ∗ *variables with operators on DAG*

last_sent,	*Function mapping miner to last sent share seq_no*
share_dag,	*A DAG with valid shares for now implemented as a set*
stable,	*Set of shares that are stable in the DAG, i.e.received by all other miners*
unpaid_coinbases,	*coinbases for braidpool blocks that haven* t been paid yet
uhpo,	Function mapping miner to unpaid balance
pool_key,	Current public key for *TS*
chain	chain of bitcoin blocks

Share is a record of miner and sequence number. All shares are assumed to be mined at same difficulty

$Share \triangleq [miner : Miner,\ seq\_no : ShareSeqNo]$

PublicKey is defined as the set of miner identifiers for now. As miners join/leave the network, the public key immediately changes The protocol to rotate the threshold signature public key is not speced here.

$PublicKey \triangleq Miner$

Coinbase is a payment to a *DKG* public key with an value.

$CoinbaseOutput \triangleq [scriptPubKey : Miner,\ value : BlockReward]$

$CoinbaseTx \triangleq [inputs : \langle\rangle,\ outputs : \langle CoinbaseOutput\rangle]$

$NoVal \triangleq 0$

$Init \triangleq$
$\land\ last\_sent = [m \in Miner \mapsto \text{IF } m = GenesisShare.miner \text{ THEN } 1 \text{ ELSE } NoVal]$
$\land\ share\_dag = [node \mapsto \{GenesisShare\},\ edge \mapsto \{\}]$
$\land\ stable = \{\}$
$\land\ unpaid\_coinbases = \{\}$
$\land\ uhpo = [m \in Miner \mapsto \{\}]$
$\land\ pool\_key = \{GenesisShare.miner\}$
$\land\ chain = \langle GenesisShare \rangle$

$TypeInvariant \triangleq$
$\land\ last\_sent \in [Miner \to Int \cup \{NoVal\}]$
$\land\ share\_dag.node \in \text{SUBSET } Share$
$\land\ share\_dag.edge \in \text{SUBSET } (Share \times Share)$
$\land\ stable \in \text{SUBSET } Share$
$\land\ unpaid\_coinbases \in \text{SUBSET } CoinbaseOutput$
$\land\ uhpo \in [Miner \to \text{SUBSET } Share]$
$\land\ pool\_key \in \text{SUBSET } Miner$
$\land\ chain \in Seq(Share)$

$vars \triangleq \langle last\_sent,\ share\_dag,\ stable,\ unpaid\_coinbases,\ uhpo,\ pool\_key,\ chain \rangle$

Send a share from a miner with a *seqno* = last share sent + 1 and in *ShareSeqNo*. The share is assumed to be successfully broadcast to all miners.

$SendShare(m,\ sno) \triangleq$
$\land\ sno = last\_sent[m] + 1$
$\land\ last\_sent' = [last\_sent \text{ EXCEPT } ![m] = @ + 1]$
$\land\ share\_dag' = [share\_dag \text{ EXCEPT}$
Add share to node list of graph
$!.node = @ \cup \{[miner \mapsto m,\ seq\_no \mapsto sno]\},$
Add edge from share to all non *NoVal last_sent*
This can be replaced by last share in *DAG* from others
$!.edge = @ \cup$
$\{[miner \mapsto m,\ seq\_no \mapsto sno]\}$
$\times$
$\{[miner \mapsto mo,\ seq\_no \mapsto last\_sent[mo]] :$
$mo \in \{mm \in Miner : last\_sent[mm] \neq NoVal\}\}]$
$\land\ \text{UNCHANGED } \langle stable,\ unpaid\_coinbases,\ uhpo,\ pool\_key,\ chain \rangle$

Stabilise a share if there is a path from the share to any share from all other miners.

How do we know all other miners? This comes from a separate protocol where a miner is dropped from the set of all other miners.

Miners are dropped from the list if they have not sent shares since the last bitcoin block was found. For now, we assume the list of to the group of miners is known.

$StabiliseShare(s) \triangleq$
$\land\ s \notin stable$

$$
\begin{aligned}
&\land \forall\, m \in Miner \setminus \{s.miner\} : \\
&\quad \exists\, p \in SimplePath(share\_dag), \\
&\qquad\quad i \in 1 \,..\, Cardinality(share\_dag.node), \\
&\qquad\quad j \in 1 \,..\, Cardinality(share\_dag.node) : \\
&\qquad\qquad \land Len(p) > 1 \\
&\qquad\qquad \land i < j \\
&\qquad\qquad \land j \leq Len(p) \\
&\qquad\qquad \land p[i].miner = s.miner \\
&\qquad\qquad \land p[j].miner = m \\
&\land stable' = stable \cup \{s\} \\
&\land \text{UNCHANGED } \langle last\_sent,\ share\_dag,\ unpaid\_coinbases,\ uhpo,\ pool\_key,\ chain \rangle
\end{aligned}
$$

On receiving a bitcoin block miners create a new new bitcoin block they are mining on.

Miners have to create a new coinbase transaction. However, the *UHPO* transaction remains the same.

$$ReceiveBitcoinBlock \triangleq$$

A miner on braidpool finds a new bitcoin block
1. Include the miner in the *pool_key*
2. Update *UHPO* payout miners and amount

Some miners can send shares with the old block

$$
\begin{aligned}
&FoundBitcoinBlock(share) \triangleq \\
&\qquad \land last\_sent[share.miner] = share.seq\_no \\
&\qquad \land \forall\, i \in 1 \,..\, Len(chain) : chain[i] \neq share \\
&\qquad \land chain' = Append(chain,\ share) \\
&\qquad \land pool\_key' = pool\_key \cup \{share.miner\} \\
&\qquad \land \forall\, ss \in NodesInSimplePath(share\_dag, \\
&\qquad\qquad\qquad\qquad chain[Len(chain)], \\
&\qquad\qquad\qquad\qquad chain[1]) : \\
&\qquad\qquad uhpo' = [uhpo \text{ EXCEPT } ![ss.miner] = @ \cup \{ss\}] \\
&\qquad \land \text{UNCHANGED } \langle stable,\ last\_sent,\ share\_dag,\ unpaid\_coinbases \rangle
\end{aligned}
$$

$$
\begin{aligned}
&Next \triangleq \\
&\qquad \lor \exists\, s \in Share : \\
&\qquad\qquad \lor SendShare(s.miner,\ s.seq\_no) \\
&\qquad\qquad \lor StabiliseShare(s) \\
&\qquad\qquad \lor FoundBitcoinBlock(s)
\end{aligned}
$$

Any share can be a bitcoin block.
We do not model difficulty or track valid bitcoin flag.

$$Liveness \triangleq \forall\, s \in share\_dag.node : \mathrm{WF}_{vars}(StabiliseShare(s) \lor FoundBitcoinBlock(s))$$

$$
\begin{aligned}
&Spec \triangleq \\
&\qquad \land Init \\
&\qquad \land \Box[Next]_{vars}
\end{aligned}
$$

$$FairSpec \triangleq Spec \\ \land Liveness$$