MODULE *BlockGeneration*

Block generation specifies when and how braidpool miners generate blocks. The protocol to build current pool key and threshold signatures is assumed

EXTENDS
Sequences,
Integers,
DAG,
FiniteSets

CONSTANT
Miner, — Set of miners
ShareSeqNo, — Share seq numbers each miner generates
BlockReward, — Block reward in a difficulty period
MaxPathLen — Max path length to check for message stability. This helps constrain the search space in the *DAG*.

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*

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

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

Acks are the implicit acknowledgements sent with each share. These are the sequence number of shares receieved from each miner.

$Acks \triangleq \langle Share \rangle$

PublicKey is defined as sequence of miner identifier for now. The miners in a key sequence are the miners contributing to the key generated using *DKG*.

$PublicKey \triangleq Seq(Miner)$

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

$Coinbase \triangleq [key : PublicKey, value : BlockReward]$

$NoVal \triangleq 0$

$$
\begin{aligned}
&Init \triangleq \\
&\quad \land last\_sent = [m \in Miner \mapsto NoVal] \\
&\quad \land share\_dag = [node \mapsto \{\},\ edge \mapsto \{\}] \\
&\quad \land stable = \{\} \\
&\quad \land unpaid\_coinbases = \{\} \\
&\quad \land uhpo = [m \in Miner \mapsto NoVal] \\
&\quad \land pool\_key = \langle\rangle
\end{aligned}
$$

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

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

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.

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

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.

$$
\begin{aligned}
&StabiliseShare \triangleq \exists\, s \in share\_dag.node : \\
&\quad \land s \notin stable \\
&\quad \land \forall\, m \in Miner \setminus \{s.miner\} : \\
&\qquad \exists\, p \in Paths(MaxPathLen,\ share\_dag) : \land\, p[1].miner = m \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, p[2].miner = s.miner \\
&\quad \land stable' = stable \cup \{s\}
\end{aligned}
$$

$\land$ UNCHANGED $\langle last\_sent, share\_dag, unpaid\_coinbases, uhpo, pool\_key \rangle$

$DAGConstraint \triangleq Cardinality(share\_dag.node) \leq Cardinality(Miner) * Cardinality(ShareSeqNo)$

RecvBitcoinBlock

FindBitcoinBlock

UpdatePoolKey

$Next \triangleq$
$\quad \lor SendShare$
$\quad \lor StabiliseShare$

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