MODULE *CBCCasperSpec*

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

CONSTANTS

nodes,	set of validator ids
weights,	tuple of validator weights
threshold,	fault tolerance threshold
values,	set of consensus values
genesis	genesis message

VARIABLES

dags,	tuple of local *DAGs* for each validator (only contains parent pointers)
faulty,	tuple of sets of observed equivocating validators
scored_q,	tuple of records of scored messages with score
unscored_q,	tuple of tuples of messages which have not been scored
sent_msgs,	tuple of tuples of messages sent by each validator
equiv_msgs,	tuple of sets of tuples of equivocated messages
estimates,	tuple of sets of best current estimates
states	tuple of validator states (contains all justification pointers)

$vars \triangleq \langle faulty, scored\_q, unscored\_q, sent\_msgs, equiv\_msgs, estimates, states \rangle$

Messages

Unscored message = (estimate, sender, justification)

$Msg(est, from, just) \triangleq [estimate \mapsto est, sender \mapsto from, justification \mapsto just]$

Scored message.

$ScoredMsg(\_msg, \_score) \triangleq [msg \mapsto \_msg, score \mapsto \_score]$

Scored estimate.

$ScoredEst(\_est, \_score) \triangleq [est \mapsto \_est, score \mapsto \_score]$

Message decomposition functions.

$Estimate(msg) \triangleq msg.estimate$

$Sender(msg) \triangleq msg.sender$

$Justification(msg) \triangleq msg.justification$

LET $j \triangleq msg.justification$

IN $j.parents \cup j.nonparents$

$Just(p, n) \triangleq [\text{parents} \mapsto p, nonparents \mapsto n]$

$OnlyPar(p) \triangleq [\text{parents} \mapsto p, nonparents \mapsto \{\}]$

$Parents(msg) \triangleq msg.justification.parents$

The genesis message is abstract - it does not have estimate, sender, or justification fields.

Set of nodes who have sent at least one message.

$Senders \triangleq \{n \in nodes : sent\_msgs[n] \neq \langle \rangle\}$

$Observed(msgs) \triangleq \{Sender(m) : m \in (msgs \setminus \{genesis\})\}$

Estimator

TODO: make precise

GHOST fork choice rule - latest honest estimate driven

output should be ranked set of tips

$GHOST(state) \triangleq$
$\quad \text{IF } state = \{genesis\}$
$\quad\quad \text{THEN } values$
$\quad\quad \text{ELSE } \{Estimate(m) : m \in (state \setminus \{genesis\})\}$

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

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

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

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

$Max(S) \triangleq \text{CHOOSE } n \in S : \forall m \in S : m \leq n$

Auxiliary Functions & Definitions

Returns the tuple of *unscored* messages from tuple of scored messages.

$\text{RECURSIVE } Unscore(\_)$
$Unscore(seq) \triangleq$
$\quad \text{IF } seq = \langle\rangle$
$\quad\quad \text{THEN } \langle\rangle$
$\quad\quad \text{ELSE } \langle Head(seq).msg \rangle \circ Unscore(Tail(seq))$

Set of all messages received by a given validator.

$ReceivedMsgs(n) \triangleq ToSet(unscored\_q[n] \circ Unscore(scored\_q[n])) \setminus \{genesis\}$

Pick an arbitrary element from the given set.

$Pick(S) \triangleq \text{CHOOSE } s \in S : \text{TRUE}$

Set of nodes who have received at least one message (excludes genesis).

$Receivers \triangleq \{n \in nodes : ReceivedMsgs(n) \neq \{\}\}$

Broadcast given message to all other validators in given set.

arguments: message, sender, set of receivers (sender is excluded)

$Broadcast(msg, n, rec) \triangleq$
$\quad [i \in nodes \mapsto \text{IF } i \in (rec \setminus \{n\})$
$\quad\quad\quad\quad\quad\quad \text{THEN } \langle msg \rangle$
$\quad\quad\quad\quad\quad\quad \text{ELSE } \langle\rangle$
$\quad]$

Apply binary operation over entire set.

$\text{RECURSIVE } SetReduce(\_, \_, \_)$

$SetReduce(Op(\_, \_), S, value) \triangleq$
$\quad$ IF $S = \{\}$
$\quad\quad$ THEN $value$
$\quad\quad$ ELSE LET $s \triangleq Pick(S)$
$\quad\quad\quad\quad$ IN $SetReduce(Op, S \setminus \{s\}, Op(s, value))$

$SetSum(S) \triangleq$ LET $op(a, b) \triangleq a + b$ IN $SetReduce(op, S, 0)$
$SetAnd(S) \triangleq$ LET $op(a, b) \triangleq a \land b$ IN $SetReduce(op, S,$ TRUE$)$
$SetOr(S) \triangleq$ LET $op(a, b) \triangleq a \lor b$ IN $SetReduce(op, S,$ FALSE$)$

Apply binary operation over entire tuple.
RECURSIVE $SeqReduce(\_, \_, \_)$
$SeqReduce(Op(\_, \_), s, value) \triangleq$
$\quad$ IF $s = \langle\rangle$
$\quad\quad$ THEN $value$
$\quad\quad$ ELSE LET $h \triangleq Head(s)$
$\quad\quad\quad\quad$ IN $SeqReduce(Op, Tail(s), Op(h, value))$

$SeqSum(S) \triangleq$ LET $op(a, b) \triangleq a + b$ IN $SeqReduce(op, S, 0)$
$SeqAnd(S) \triangleq$ LET $op(a, b) \triangleq a \land b$ IN $SeqReduce(op, S,$ TRUE$)$
$SeqOr(S) \triangleq$ LET $op(a, b) \triangleq a \lor b$ IN $SeqReduce(op, S,$ FALSE$)$

Turns a set of 2-tuples into the set of individual elements.
RECURSIVE $UnSeqSet(\_)$
$UnSeqSet(S) \triangleq$
$\quad$ IF $S = \{\}$
$\quad\quad$ THEN $\{\}$
$\quad\quad$ ELSE LET $s \triangleq Pick(S)$
$\quad\quad\quad\quad$ IN $\{Head(s), Head(Tail(s))\} \cup UnSeqSet(S \setminus \{s\})$

Turns set of elements into the set of all possible 2-tuples.
$Pairs(S) \triangleq \{s \in Seq(S) : Len(s) = 2\}$

Global set of faulty validators.
$GlobalFaultySet \triangleq$ UNION $(ToSet(faulty))$

Initialize tuple with given value.
$Initialize(val) \triangleq [i \in 1 \,..\, Cardinality(nodes) \mapsto val]$

The dependencies of a message m are the messages in the justification
of m and in the justifications of the justifications of m and so on,
$i.e.$ justifications all the way down.
RECURSIVE $Dep(\_)$
$Dep(msg) \triangleq$
$\quad$ IF $msg = genesis$
$\quad\quad$ THEN $\{genesis\}$

$\quad$ ELSE IF $Cardinality(Justification(msg)) = 1$
$\quad\quad$ THEN $Justification(msg)$
$\quad\quad$ ELSE $Justification(msg)$
$\quad\quad\quad \cup$ UNION $\{Dep(m) : m \in (Justification(msg) \setminus \{genesis\})\}$

Gets the set of dependencies of all the messages in a set of messages.
$DepSet(msgs) \triangleq$ UNION $\{Dep(m) : m \in msgs\}$

Dependency depth of a message.
RECURSIVE $Depth(\_)$
$Depth(msg) \triangleq$
$\quad$ IF $msg = genesis$
$\quad$ THEN 0
$\quad$ ELSE $1 + Max(\{Depth(m) : m \in Dep(msg)\})$

Dependency depth of a set of messages.
$DepthSet(msgs) \triangleq$
$\quad$ IF $msgs = \{genesis\}$
$\quad$ THEN 0
$\quad$ ELSE $Max(\{Depth(m) : m \in msgs\})$

Latest *message*(*s*) from a validator in a given set of messages.
$LatestMsgs(n, msgs) \triangleq \{genesis\} \cup$
$\quad \{m \in (msgs \setminus \{genesis\}) :$
$\quad\quad \land Sender(m) = n$
$\quad\quad \land \neg \exists\, m0 \in (msgs \setminus \{genesis\}) :$
$\quad\quad\quad \land Sender(m0) = n$
$\quad\quad\quad \land m \neq m0$
$\quad\quad\quad \land m \in Dep(m0)$
$\quad \}$

Latest *estimate*(*s*) from a validator in a given set of messages.
$LatestEsts(n, msgs) \triangleq \{Estimate(m) : m \in (LatestMsgs(n, msgs) \setminus \{genesis\})\}$

Set of estimates in a state.
$Estimates(state) \triangleq \{Estimate(m) : m \in ((state \cup DepSet(state)) \setminus \{genesis\})\}$

Justifications of a set of messages.
$Justifications(msgs) \triangleq$ UNION $\{Justification(m) : m \in (msgs \setminus \{genesis\})\}$

Two messages are equivocating if they have the same sender, but do not justify each other.
$Equivocation(m1, m2) \triangleq$
$\quad \land m1 \neq m2$
$\quad \land Sender(m1) = Sender(m2)$
$\quad \land m1 \notin (Dep(m2) \setminus \{genesis\})$
$\quad \land m2 \notin (Dep(m1) \setminus \{genesis\})$

$CheckDepsForEquiv(msgs) \triangleq$

$\text{LET } deps \triangleq DepSet(msgs)$
$\text{IN } \land Cardinality(deps) > 1$
$\quad \land \exists m1, m2 \in (deps \setminus \{genesis\}) : Equivocation(m1, m2)$

$EquivPairsInDeps(msgs) \triangleq$
$\text{IF } \neg CheckDepsForEquiv(msgs)$
$\text{THEN } \{\}$
$\text{ELSE } \{\langle m1, m2 \rangle \in (DepSet(msgs) \setminus \{genesis\}) \times (DepSet(msgs) \setminus \{genesis\}) : Equivocation(m1, m2)\}$

A validator is faulty if it sends equivocating messages.
Checks if a validator equivicates in a given set of messages.
$FaultyNode(n, msgs) \triangleq$
$\land \exists m1 \in (DepSet(msgs) \setminus \{genesis\}) :$
$\quad \land \exists m2 \in (DepSet(msgs) \setminus \{genesis\}) :$
$\quad\quad \land Sender(m1) = n$
$\quad\quad \land Equivocation(m1, m2)$

Set of faulty validators in an observed set of messages.
$FaultyNodes(msgs) \triangleq \{n \in nodes : FaultyNode(n, msgs)\}$

Messages from equivocating validators in a given set of messages.
$EquivocatedMsgs(n, msgs) \triangleq DepSet(msgs) \cap UnSeqSet(equiv\_msgs[n])$

Checks existence of equivocated messages received by the given validator.
$EquivReceived(n) \triangleq \exists \langle m1, m2 \rangle \in Pairs(DepSet(ReceivedMsgs(n))) : Equivocation(m1, m2)$

Arbitrary node who has observed an equivocation.
$Equiv\_node \triangleq \text{CHOOSE } n \in nodes : EquivReceived(n)$

Set of messages later than a given message in a given set of messages.
$Later(msg, msgs) \triangleq \{m \in (msgs \setminus \{genesis\}) : msg \in Justification(m)\}$

Honest messages - messages from non-faulty validators.
$HonestMsgs(n, msgs) \triangleq DepSet(msgs) \setminus (EquivocatedMsgs(n, msgs) \cup \{genesis\})$

Set of latest honest messages received by a validator.
$LatestHonestMsgs(n, msgs) \triangleq \{m \in HonestMsgs(n, msgs) : m \in LatestMsgs(n, msgs)\}$

Set of latest honest estimates received by a validator.
$LatestHonestEsts(n, msgs) \triangleq \{Estimate(m) : m \in LatestHonestMsgs(n, msgs)\}$

Weights of subsets of validators.
$Weight(set) \triangleq$
$\quad SeqSum([n \in 1 \,..\, Cardinality(nodes) \mapsto \text{IF } n \in set \text{ THEN } weights[n] \text{ ELSE } 0])$

$TotalWeight \triangleq Weight(nodes)$
$FaultWeight(state) \triangleq Weight(FaultyNodes(state))$

Two validators are agreeing with each other on an estimate in a set of messages if:

$-$ $n1$ has exactly one latest message in the set
$-$ $n2$ has exactly one latest message in the justification of $n1$'s latest message
- the estimates of these latest messages agree with the given estimate
i.e. $n1$ is not equivocating in the set of messages and
$n2$ is not equivocating in the justification of $n1$'s latest message

$$
\begin{aligned}
& \mathit{Agreeing}(n1,\ n2,\ \mathit{estimate},\ \mathit{msgs}) \triangleq \\
& \quad \textsc{let}\ \mathit{n1\_latest\_msg} \triangleq \mathit{Pick}(\mathit{LatestMsgs}(n1,\ \mathit{msgs})) \\
& \qquad\quad \mathit{n2\_latest\_msg} \triangleq \mathit{Pick}(\mathit{LatestMsgs}(n2,\ \mathit{Justification}(\mathit{n1\_latest\_msg}))) \\
& \quad \textsc{in} \quad \land\ \mathit{Cardinality}(\mathit{LatestMsgs}(n1,\ \mathit{msgs})) = 1 \\
& \qquad\qquad \land\ \mathit{Cardinality}(\mathit{LatestMsgs}(n2,\ \mathit{Justification}(\mathit{n1\_latest\_msg}))) = 1 \\
& \qquad\qquad \land\ \mathit{Estimate}(\mathit{n1\_latest\_msg}) = \mathit{estimate} \\
& \qquad\qquad \land\ \mathit{Estimate}(\mathit{n2\_latest\_msg}) = \mathit{estimate}
\end{aligned}
$$

Two validators are disagreeing with each other on an estimate in a set of messages if:
$-$ $n1$ has exactly one latest message in messages
$-$ $n2$ has exactly one latest message in the justification of $n1$'s latest message
$-$ $n2$ has a new latest message that doens't agree with the estimate

$$
\begin{aligned}
& \mathit{Disagreeing}(n1,\ n2,\ \mathit{estimate},\ \mathit{msgs}) \triangleq \\
& \quad \land\ \mathit{Cardinality}(\mathit{LatestMsgs}(n1,\ \mathit{msgs})) = 1 \\
& \quad \land\ \textsc{let}\ \mathit{n1\_latest\_msg} \triangleq \mathit{Pick}(\mathit{LatestMsgs}(n1,\ \mathit{msgs})) \\
& \qquad \textsc{in} \quad \land\ \mathit{Cardinality}(\mathit{LatestMsgs}(n2,\ \mathit{Justification}(\mathit{n1\_latest\_msg}))) = 1 \\
& \qquad\qquad \land\ \textsc{let}\ \mathit{n2\_latest\_msg} \triangleq \mathit{Pick}(\mathit{LatestMsgs}(n2,\ \mathit{Justification}(\mathit{n1\_latest\_msg}))) \\
& \qquad\qquad\quad \textsc{in} \quad \exists\, m \in \mathit{msgs} : \land\ \mathit{n2\_latest\_msg} \in \mathit{Dep}(m) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ \mathit{estimate} \neq \mathit{Estimate}(m)
\end{aligned}
$$

An e-clique is a group of non-faulty nodes in a set of observed messages such that:
- they mutually see each other agreeing with the given estimate in the given set of messages, and
- they mutually cannot see each other disagreeing with the given estimate in the given set of messages.
If nodes in an e-clique see each other agreeing on e and can't see each other disagreeing on e,
then there does not exist any new message from inside the clique that will cause them to assign
lower scores to e. Further, if the clique has more than half of the validators by weight,
then no messages external to the clique can raise the scores these validators assign to
a competing estimate to cause it to become larger than the score they assign to e.

$$
\begin{aligned}
& \mathit{Eclique}(\mathit{estimate},\ \mathit{state}) \triangleq \\
& \quad \{\mathit{sub} \in \textsc{subset}\ (\mathit{nodes}) : \\
& \qquad \land\ \mathit{Cardinality}(\mathit{sub}) > 1 \\
& \qquad \land\ \forall\, n1 \in \mathit{sub} : \\
& \qquad\qquad \forall\, n2 \in (\mathit{sub} \setminus \{n1\}) : \\
& \qquad\qquad\quad \land\ \mathit{Agreeing}(n1,\ n2,\ \mathit{estimate},\ \mathit{state}) \\
& \qquad\qquad\quad \land\ \neg \mathit{Disagreeing}(n1,\ n2,\ \mathit{estimate},\ \mathit{state}) \\
& \qquad\qquad\quad \land\ \neg \mathit{FaultyNode}(n1,\ \mathit{state}) \\
& \qquad\qquad\quad \land\ \neg \mathit{FaultyNode}(n2,\ \mathit{state}) \\
& \quad \}
\end{aligned}
$$

Checks for existence of an e-clique with cumulative weight $> 50\%$ of total validator weight.

$$
\mathit{EcliqueEstimateSafety}(\mathit{estimate},\ \mathit{state}) \triangleq
$$

state is valid

$\exists\, ec \in Eclique(estimate, state) :$
$\quad 2 * SeqSum(Weight(ec)) > TotalWeight + threshold - FaultWeight(state)$

Set of messges received from honest validators by a particular valiadtor.

$$HonestReceivedMsgs(n) \triangleq \{m \in ReceivedMsgs(n) : m \notin UnSeqSet(equiv\_msgs[n])\}$$

A temporal property checking that finality can eventually be reached.

$$\begin{aligned}
&CheckSafetyOracle \triangleq \\
&\quad \text{LET } n \triangleq RandomElement(nodes \setminus GlobalFaultySet) \\
&\quad \text{IN } \quad \Diamond(\exists\, v \in values : EcliqueEstimateSafety(v, HonestReceivedMsgs(n)))
\end{aligned}$$

Protocol Messages & States

Protocol messages have an estimate given by the estimator applied to the justification.

$$\begin{aligned}
&ValidMsg(msg) \triangleq \\
&\quad \lor\ msg = genesis && \text{genesis is a valid message} \\
&\quad \lor\ \land\ msg \neq genesis && \text{non-genesis message is valid if sender and estimate are valid} \\
&\qquad \land\ Sender(msg) \in nodes \\
&\qquad \land\ Estimate(msg) \in GHOST(Justification(msg))
\end{aligned}$$

$$ProtocolMsgs \triangleq \{m \in \text{UNION } (\{ToSet(sent\_msgs[n]) : n \in nodes\}) : ValidMsg(m)\}$$

Protocol states are finite sets of protocol messages which contain
their justifications and have fault weight less than the theshold.

$$\begin{aligned}
&ValidState(state) \triangleq \\
&\quad \lor\ state = \{genesis\} \\
&\quad \lor\ \forall\, m \in (state \setminus \{genesis\}) : \\
&\qquad \land\ Justification(m) \subseteq state \\
&\qquad \land\ FaultWeight(state) < threshold
\end{aligned}$$

$$\begin{aligned}
&ProtocolStates \triangleq \\
&\quad \{s \in \text{SUBSET } (ProtocolMsgs) : \\
&\qquad \land\ ValidState(s) \\
&\qquad \land\ IsFiniteSet(s) \\
&\quad \}
\end{aligned}$$

$$SentSet \triangleq \text{UNION } (\{ToSet(sent\_msgs[n]) : n \in nodes\})$$
$$StateSet \triangleq \text{UNION } (\{states[n] : n \in nodes\})$$

Decisions & Consistency

Futures of a given state.

$$Futures(state) \triangleq \{s \in ProtocolStates : state \subseteq s\}$$

Check whether a given property is decided in a given state.

$$Decided(prop, state) \triangleq \forall\, s \in Futures(state) : prop[s]$$

Decisions in a given state: set of properties which are decided in the state.

$Decisions(state) \triangleq$
$\quad \{prop \in [ProtocolStates \rightarrow \{\text{FALSE}, \text{TRUE}\}] : Decided(prop, state)\}$

Previous messages.

$PrevMsg(msg) \triangleq$
$\quad \text{IF } Justification(msg) = \{genesis\}$
$\quad\quad \text{THEN } \{genesis\}$
$\quad\quad \text{ELSE } \{genesis\} \cup$
$\quad\quad\quad\quad \text{UNION } \{LatestMsgs(n, Justification(msg)) : n \in Observed(Justification(msg))\}$

Previous estimates.

$PrevEst(msg) \triangleq$
$\quad \text{IF } Justification(msg) = \{genesis\}$
$\quad\quad \text{THEN } \{\}$
$\quad\quad \text{ELSE } \text{UNION } \{LatestEsts(n, Justification(msg)) : n \in Observed(Justification(msg))\}$

Message ancestry.

$\text{RECURSIVE } n\_cestorMsg(\_, \_)$
$n\_cestorMsg(msg, n) \triangleq$
$\quad \text{IF } n = 0 \lor msg = genesis$
$\quad\quad \text{THEN } msg$
$\quad\quad \text{ELSE } \text{UNION } (n\_cestorMsg(PrevMsg(msg), n - 1))$

Estimate ancestry.

$\text{RECURSIVE } n\_cestorEst(\_, \_)$
$n\_cestorEst(msg, n) \triangleq$
$\quad \text{IF } msg = genesis$
$\quad\quad \text{THEN } \{\}$
$\quad\quad \text{ELSE } \text{IF } n = 0$
$\quad\quad\quad\quad \text{THEN } msg$
$\quad\quad\quad\quad \text{ELSE } \text{UNION } (n\_cestorMsg(PrevEst(msg), n - 1))$

Block membership: $b1$ is conatined in $b2$'s chain/dag.
$Membership(b1, b2) \triangleq \exists\, n \in Nat : b1 = n\_cestor(b2, n)$

$Membership(m1, m2) \triangleq$
$\quad \lor m1 = genesis$
$\quad \lor m1 = m2$
$\quad \lor \land m1 \neq genesis$
$\quad\quad \land Estimate(m1) \in Estimates(\{m2\} \cup Dep(m2))$

Set of validators supporting a given estimate in a dag.

$\text{RECURSIVE } Supporters(\_, \_)$
$Supporters(est, state) \triangleq$
$\quad \text{IF } state = \{genesis\} \lor est \notin Estimates(state)$
$\quad\quad \text{THEN } \{\}$

```
    ELSE LET m ≜ Pick(state \ {genesis})
         IN  IF est ∈ Estimates({m})
               THEN {Sender(m)} ∪ Supporters(est, Justification(m)) ∪ Supporters(est, (state \ {m}))
               ELSE Supporters(est, (state \ {m}))
```

Score of a block (estimate) in a given state.

```
Score(msg, state) ≜
  LET S ≜ {n ∈ nodes : ∃ m ∈ LatestHonestEsts(n, state) : Membership(msg, m)}
  IN  SeqSum([n ∈ S ↦ weights[n]])
```

```
Score(est, state) ≜ Weight(Supporters(est, state) \ GlobalFaultySet)
```

Children: a child of a block has that block as (one of) its *Prev* blocks.

```
Children(msg, state) ≜ {m ∈ state : msg ∈ PrevMsg(m)}
```

Updates scored message scores in current state.

```
RECURSIVE UpdateScores(_, _)
UpdateScores(n, scored) ≜
  IF scored = ⟨⟩
   THEN ⟨⟩
   ELSE LET hd ≜ Head(scored)
            tl ≜ Tail(scored)
        IN  IF hd = genesis
              THEN ⟨ScoredMsg(genesis, TotalWeight)⟩ ∘ UpdateScores(n, tl)
              ELSE ⟨ScoredMsg(hd.msg, Score(Estimate(hd.msg), states[n]))⟩
                     ∘ UpdateScores(n, tl)
```

Scores all *unscored* messages in current state.

```
RECURSIVE ScoreUnscored(_, _)
ScoreUnscored(n, unscored) ≜
  IF unscored = ⟨⟩
   THEN ⟨⟩
   ELSE LET hd ≜ Head(unscored)
            tl ≜ Tail(unscored)
        IN  IF hd ∈ EquivReceived(n)
              THEN ScoreUnscored(n, tl)
              ELSE ⟨ScoredMsg(hd, Score(hd, states[n]))⟩ ∘ ScoreUnscored(n, tl)
```

Local *DAG* views
– *dags*[*n*] consists of a set of nested sets of estimates
- what is the exact relation between *dags*[*n*] and *states*[*n*]? refinement?

Set of estimates present in a *DAG*.

```
RECURSIVE DagEstimateSet(_)
DagEstimateSet(dag) ≜
  LET l ≜ Len(dag)
```

```
  IN   IF dag = ⟨genesis⟩
         THEN {}
         ELSE  ToSet(SubSeq(dag, 1, l − 1)) ∪ DagEstimateSet(dag[l])
```

DAG height.

```
RECURSIVE DagHeight(_)
DagHeight(dag) ≜
  IF Len(dag) ≤ 1
   THEN 0
   ELSE  1 + DagHeight(dag[Len(dag)])
```

Depth of estimate in DAG.

```
RECURSIVE DagDepth(_, _)
DagDepth(est, dag) ≜
  LET l   ≜ Len(dag)
      d   ≜ DagDepth(est, dag[l])
  IN    IF est = genesis
          THEN DagHeight(dag)
          ELSE  IF est ∉ DagEstimateSet(dag)
                  THEN  − 1
                  ELSE  IF l ≤ 1
                          THEN 0
                          ELSE  1 + d
```

Set of DAG tips.

```
Tips(dag) ≜ ToSet(SubSeq(dag, 1, Len(dag) − 1))
```

Add scored estimate at level.

```
AddAtLevel(est, dag) ≜
  IF dag = ⟨⟩
   THEN ⟨est⟩
   ELSE  IF Depth(⟨⟩)  finish
           THEN ⟨⟩        finish
           ELSE  ⟨⟩       finish
```

Add estimate to dag.

```
AddEstimateToDag(n, est) ≜
  LET e   ≜ ⟨est⟩
      d   ≜ dags[n]
  IN    IF dags[n] = ⟨⟩
          THEN e
          ELSE  IF Depth(est.est) > DagHeight(d)
                  THEN e ∘ ⟨d⟩
                  ELSE  ⟨⟩  finish
```

Preliminary conditions

$ThresholdCheck \triangleq threshold \geq 0 \land threshold < TotalWeight$
$NodeWeightLen \triangleq Len(weights) = Cardinality(nodes)$
$AllSendsValid \triangleq SentSet = \{m \in SentSet : ValidMsg(m)\}$
$AllStatesValid \triangleq StateSet = \{s \in StateSet : ValidState(s)\}$

Must hold in all reachable states.

$TypeOK \triangleq$
$\quad \land AllSendsValid$
$\quad \land AllStatesValid$

Initial state conditions

All validators start with scored genesis block only.

$Init \triangleq$
$\quad \land ThresholdCheck$
$\quad \land NodeWeightLen$
$\quad \land dags = Initialize(\langle genesis \rangle)$
$\quad \land faulty = Initialize(\{\})$
$\quad \land scored\_q = Initialize(\langle ScoredMsg(genesis, TotalWeight) \rangle)$
$\quad \land unscored\_q = Initialize(\langle\rangle)$
$\quad \land sent\_msgs = Initialize(\langle\rangle)$
$\quad \land equiv\_msgs = Initialize(\{\})$
$\quad \land estimates = Initialize(values)$
$\quad \land states = Initialize(\{genesis\})$

Updates

A validator can update their set of valid estimates.

$Update\_Estimates(n) \triangleq$
$\quad \land estimates' = [estimates \text{ EXCEPT } ![n] = GHOST(states[n])]$
$\quad \land \text{UNCHANGED } \langle dags, faulty, scored\_q, unscored\_q, sent\_msgs, equiv\_msgs, states \rangle$

A validator can score *unscored* estimates and update their scores.

$Update\_Scores(msg, n, rec) \triangleq$
$\quad \land scored\_q' = [scored\_q \text{ EXCEPT } ![n] =$
$\qquad UpdateScores(scored\_q[n], states[n]) \circ ScoreUnscored(unscored\_q[n], states[n])]$
$\quad \land unscored\_q' = [unscored\_q \text{ EXCEPT } ![n] = \langle\rangle]$
$\quad \land \text{UNCHANGED } \langle dags, faulty, sent\_msgs, equiv\_msgs, estimates, states \rangle$

$Update(n) \triangleq$
$\quad \land Update\_Scores(n)$
$\quad \land Update\_Estimates(n)$

Transitions

Sending/Receiving/Dropping messages

Given validator sends given message to given set of validators.

$SendMsg(msg, n, rec) \triangleq$
$\quad \land\ unscored\_q' = unscored\_q \circ Broadcast(msg, n, rec)$
$\quad \land\ sent\_msgs' = [sent\_msgs \text{ EXCEPT } ![n] = sent\_msgs[n] \circ \langle msg \rangle]$
$\quad \land\ scored\_q' = [scored\_q \text{ EXCEPT } ![n] = scored\_q[n] \circ \langle ScoredMsg(msg, weights[n]) \rangle]$
$\quad \land\ states' = [states \text{ EXCEPT } ![n] = states[n] \cup \{msg\}]$

Honest validator sends honest message.

$Send\_Honest \triangleq$
$\quad \land\ \exists\, n \in (nodes \setminus GlobalFaultySet) : estimates[n] \neq \{\}$ enabling condition: honest node with valid estimates
$\quad \land\ \text{LET } v \triangleq RandomElement(\{n \in (nodes \setminus GlobalFaultySet) : estimates[n] \neq \{\}\})$ honest validator with valid
$\qquad\qquad e \triangleq RandomElement(estimates[v])$
$\qquad \text{IN } \land\ SendMsg(Msg(e, v, states[v]), v, nodes)$
$\qquad\qquad \land\ \text{UNCHANGED } \langle dags, faulty, equiv\_msgs, estimates \rangle$

Dropped message.

$Send\_Drop \triangleq$
$\quad \land\ \exists\, n \in nodes : estimates[n] \neq \{\}$ enabling condition: node with valid estimates
$\quad \land\ \text{LET } v \triangleq RandomElement(\{n \in nodes : estimates[n] \neq \{\}\})$
$\qquad\qquad e \triangleq RandomElement(estimates[v])$
$\qquad \text{IN } \land\ sent\_msgs' = [sent\_msgs \text{ EXCEPT } ![v] = sent\_msgs[v] \circ \langle Msg(e, v, states[v]) \rangle]$
$\qquad\qquad \land\ scored\_q' = [scored\_q \text{ EXCEPT } ![v] = scored\_q[v] \circ \langle ScoredMsg(Msg(e, v, states[v]), weights[$
$\qquad\qquad \land\ states' = [states \text{ EXCEPT } ![v] = states[v] \cup \{Msg(e, v, states[v])\}]$
$\qquad\qquad \land\ \text{UNCHANGED } \langle dags, faulty, unscored\_q, equiv\_msgs, estimates \rangle$

Equivocations.

Send messages with different estimates to disjoint sets of validators.

$Send\_Equiv\_Est \triangleq$
$\quad \land\ \exists\, n \in nodes : Cardinality(estimates[n]) > 1$
$\quad \land\ \text{LET } v \triangleq RandomElement(\{n \in nodes : Cardinality(estimates[n]) > 1\})$
$\qquad\quad N1 \triangleq RandomElement(\{sub1 \in \text{SUBSET } (nodes \setminus \{v\}) : sub1 \neq \{\}\})$
$\qquad\quad N2 \triangleq RandomElement(\{sub2 \in \text{SUBSET } (nodes \setminus (N1 \cup \{v\})) : sub2 \neq \{\}\})$
$\qquad\quad e1 \triangleq RandomElement(estimates[v])$
$\qquad\quad e2 \triangleq RandomElement(estimates[v] \setminus \{e1\})$
$\qquad \text{IN } \land\ SendMsg(Msg(e1, v, states[v]), v, N1)$
$\qquad\qquad \land\ SendMsg(Msg(e2, v, states[v]), v, N2)$
$\qquad\qquad \land\ \text{UNCHANGED } \langle dags, faulty, equiv\_msgs \rangle$

Send messages with different justifications to disjoint sets of validators.

$Send\_Equiv\_Just \triangleq$
$\quad \land\ \exists\, n \in nodes : Cardinality(states[n]) > 1 \land estimates[n] \neq \{\}$
$\quad \land\ \text{LET } v \triangleq RandomElement(\{n \in nodes : Cardinality(states[n]) > 1\})$
$\qquad\quad e \triangleq RandomElement(estimates[v])$
$\qquad\quad N1 \triangleq RandomElement(\{sub1 \in \text{SUBSET } (nodes \setminus \{v\}) : sub1 \neq \{\}\})$
$\qquad\quad N2 \triangleq RandomElement(\{sub2 \in \text{SUBSET } (nodes \setminus (N1 \cup \{v\})) : sub2 \neq \{\}\})$

$$
\begin{array}{l}
\quad\quad j1 \quad \triangleq \ RandomElement(\text{SUBSET } (states[v])) \\
\quad\quad\ \ j2 \ \triangleq \ RandomElement(\{j \in \text{SUBSET } (states[v]) : j \neq j1\}) \\
\quad \text{IN} \quad \land SendMsg(Msg(e, v, j1), v, N1) \\
\quad\quad\quad \land SendMsg(Msg(e, v, j2), v, N2) \\
\quad\quad\quad \land \text{UNCHANGED } \langle dags, faulty, equiv\_msgs \rangle
\end{array}
$$

Send messages with different estimates and different justifications to disjoint sets of validators.

$$
\begin{array}{l}
Send\_Equiv\_Both \triangleq \\
\quad \land \exists n \in nodes : Cardinality(states[n]) > 1 \land Cardinality(estimates[n]) > 1 \\
\quad \land \text{LET } v \ \triangleq \ RandomElement(\{n \in nodes : Cardinality(states[n]) > 1\}) \\
\quad\quad\quad e1 \ \triangleq \ RandomElement(estimates[v]) \\
\quad\quad\quad e2 \ \triangleq \ RandomElement(estimates[v] \setminus \{e1\}) \\
\quad\quad\quad N1 \ \triangleq \ RandomElement(\{sub1 \in \text{SUBSET } (nodes \setminus \{v\}) : sub1 \neq \{\}\}) \\
\quad\quad\quad N2 \ \triangleq \ RandomElement(\{sub2 \in \text{SUBSET } (nodes \setminus (N1 \cup \{v\})) : sub2 \neq \{\}\}) \\
\quad\quad\quad j1 \ \triangleq \ RandomElement(\text{SUBSET } (states[v])) \\
\quad\quad\quad j2 \ \triangleq \ RandomElement(\{j \in \text{SUBSET } (states[v]) : j \neq j1\}) \\
\quad\quad \text{IN} \quad \land SendMsg(Msg(e1, v, j1), v, N1) \\
\quad\quad\quad\quad \land SendMsg(Msg(e2, v, j2), v, N2) \\
\quad\quad\quad\quad \land \text{UNCHANGED } \langle dags, faulty, equiv\_msgs \rangle
\end{array}
$$

$$
\begin{array}{l}
Send\_Success \triangleq \\
\quad \lor Send\_Honest \\
\quad \lor Send\_Equiv\_Est \\
\quad \lor Send\_Equiv\_Just \\
\quad \lor Send\_Equiv\_Both
\end{array}
$$

$$
\begin{array}{l}
Send \triangleq \\
\quad \lor Send\_Success \\
\quad \lor Send\_Drop
\end{array}
$$

$vars \triangleq \langle faulty, scored\_q, unscored\_q, sent\_msgs, equiv\_msgs, estimates, states \rangle$

TODO

Upon detection of an equivocation, all validators except the equivocator add equivicator to faulty set

- check dependencies of all received messages for equivocations

- put equivocated message pairs in $equiv\_msgs$

$$
\begin{array}{l}
HandleEquiv \triangleq \\
\quad \land \exists n \in nodes : CheckDepsForEquiv(ReceivedMsgs(n)) \\
\quad \land \text{LET } n \ \triangleq \ RandomElement(\{v \in nodes : CheckDepsForEquiv(ReceivedMsgs(v))\}) \\
\quad\quad\quad\quad E \ \triangleq \ EquivPairsInDeps(ReceivedMsgs(n)) \\
\quad\quad\quad\quad p \ \triangleq \ Pick(E) \\
\quad\quad \text{IN} \quad \land faulty' = [faulty \text{ EXCEPT } ![n] = faulty[n] \cup \{Sender(Head(p))\}] \\
\quad\quad\quad\quad \land equiv\_msgs' = [equiv\_msgs \text{ EXCEPT } ![n] = equiv\_msgs[n] \cup E] \\
\quad\quad\quad\quad \land \text{UNCHANGED } \langle dags, scored\_q, unscored\_q, sent\_msgs, estimates, states \rangle
\end{array}
$$

$Next \triangleq$

$\lor$ *Send*
$\lor$ *HandleEquiv*

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

$LivenessSpec \triangleq$
$\land \mathrm{WF}_{vars}(Send)$
$\land \mathrm{SF}_{vars}(\exists\, n \in nodes : Update(n))$
$\land \mathrm{SF}_{vars}(HandleEquiv)$

$Spec \triangleq$
$\land$ *SafetySpec*
$\land$ *LivenessSpec*

\ * Modification History
\ * Last modified Sat *Dec* 14 13:00:41 *EST* 2019 by *isaac*
\ * Created *Wed Nov* 27 17:00:08 *EST* 2019 by *isaac*