MODULE *SailFish*

EXTENDS *DomainModel*, *TLC*

CONSTANT
GST the first synchronous round (all later rounds are synchronous)

--algorithm *SailFish* **{**
variables
$vs = \{\}$, the vertices of the *DAG*
$es = \{\}$, the edges of the *DAG*
$no\_vote = [n \in N \mapsto \{\}]$ **;** *no_vote* messages sent by each node
define {
$LeaderVertice(r) \triangleq \langle Leader(r),\ r\rangle$
$VerticeQuorums(r) \triangleq$
$\{VQ \in \text{SUBSET } vs :$
$\land\ \forall\, v \in VQ : Round(v) = r$
$\land\ \{Node(v) : v \in VQ\} \in Quorum\}$
}
process ($correctNode \in N \setminus B$ **)**
variables $round = 0$ **;** current round
{
*l*0: **while (** TRUE **)**
either with ($v = \langle self,\ round\rangle$ **) {**
add a new vertex to the *DAG* and go to the next round
$vs := vs \cup \{v\}$ **;**
if ($round > 0$ **)**
with ($vq \in VerticeQuorums(round - 1)$ **) {**
from *GST* onwards, each node receives all correct vertices of the previous round:
when $round \geq GST \Rightarrow (N \setminus B) \subseteq \{Node(v2) : v2 \in vq\}$ **;**
$es := es \cup \{\langle v,\ pv\rangle : pv \in vq\}$ **;**
if ($LeaderVertice(round - 1) \notin vq$ **)**
$no\_vote[self] := no\_vote[self] \cup \{LeaderVertice(round - 1)\}$
} ;
$round := round + 1$
}
or with ($r \in \{r \in R : r > round\}$ **) {**
go to a higher round
when $round < GST$ **;** from *GST* onwards, correct nodes do not skip rounds
$round := r$
}
}

Next comes our model of *Byzantine* nodes. Because the real protocol disseminates *DAG* vertices using reliable broadcast, *Byzantine* nodes cannot equivocate and cannot deviate much from the protocol (lest their messages be ignored).

```
    process ( byzantineNode ∈ B )
        variables round_ = 0 ;
    {
l0:     while ( TRUE ) {
              maybe add a vertices to the DAG:
            either with ( v = ⟨self, round_⟩ ) {
                vs := vs ∪ {v} ;
                if ( round_ > 0 )
                    with ( vq ∈ VerticeQuorums(round_ − 1) )
                        es := es ∪ {⟨v, pv⟩ : pv ∈ vq}
             } or skip ;
              maybe send a no_vote messages:
            if ( round_ > 0 )
            either
                no_vote[self] := no_vote[self] ∪ {LeaderVertice(round_ − 1)}
            or skip ;
              go to the next round:
            round_ := round_ + 1
         }
     }
 }
```

Next we define the safety and liveness properties

$$
\begin{aligned}
&\mathit{Committed}(v) \triangleq \\
&\quad \land\ v \in \mathit{vs} \\
&\quad \land\ \mathit{Node}(v) = \mathit{Leader}(\mathit{Round}(v)) \\
&\quad \land\ \{\mathit{Node}(pv) : pv \in \mathit{Parents}(v,\ \mathit{es})\} \in \mathit{Quorum} \\
&\quad \land\ \lor\ \mathit{Round}(v) = 0 \\
&\qquad \lor\ \mathit{LeaderVertice}(\mathit{Round}(v) - 1) \in \mathit{Children}(v,\ \mathit{es}) \\
&\qquad \lor\ \exists\, Q \in \mathit{Quorum} : \forall\, n \in Q : \\
&\qquad\quad \mathit{LeaderVertice}(\mathit{Round}(v) - 1) \in \mathit{no\_vote}[n] \\
&\mathit{Safety} \triangleq \forall\, v1,\ v2 \in \mathit{vs} : \\
&\quad \land\ \mathit{Committed}(v1) \\
&\quad \land\ \mathit{Committed}(v2) \\
&\quad \land\ \mathit{Round}(v1) \leq \mathit{Round}(v2) \\
&\quad \Rightarrow \mathit{Reachable}(v2,\ v1,\ \mathit{es}) \\
&\mathit{Liveness} \triangleq \forall\, r \in R : \\
&\quad \land\ r \geq \mathit{GST} \\
&\quad \land\ \mathit{Leader}(r) \notin B \\
&\quad \land\ \forall\, n \in N \setminus B : \mathit{round}[n] > r + 1 \\
&\quad \Rightarrow \mathit{Committed}(\mathit{LeaderVertice}(r))
\end{aligned}
$$

Finally we make a few auxiliary definitions used for model-checking with *TLC*

The round of a node, whether *Byzantine* or not

$$Round\_(n) \triangleq \text{IF } n \in B \text{ THEN } round\_[n] \text{ ELSE } round[n]$$

Basic typing invariant:

$$\begin{array}{l} TypeOK \triangleq \\ \quad \land \ \forall\, v \in vs : Node(v) \in N \land Round(v) \in Nat \\ \quad \land \ \forall\, e \in es : \\ \qquad \land \ e = \langle e[1],\, e[2] \rangle \\ \qquad \land \ \{e[1],\, e[2]\} \subseteq vs \\ \qquad \land \ Round(e[1]) > Round(e[2]) \\ \quad \land \ \forall\, n \in N : \\ \qquad \land \ Round\_(n) \in Nat \\ \qquad \land \ no\_vote[n] \subseteq \{\langle Leader(r),\, r \rangle : r \in R\} \end{array}$$

Sequentialization constraints, which enforce a particular ordering of the actions. Because of how actions commute, the set of reachable states remains unchanged. This speeds up model-checking a lot.

$$\begin{array}{l} SeqConstraints(n) \triangleq \\ \quad \text{wait for all nodes to finish previous rounds:} \\ \quad \land\, (Round\_(n) > 0 \Rightarrow \forall\, n2 \in N : Round\_(n2) \geq Round\_(n)) \\ \quad \text{wait for all nodes with lower index to leave the round:} \\ \quad \land \forall\, n2 \in N : NodeIndex(n2) < NodeIndex(n) \Rightarrow Round\_(n2) > Round\_(n) \end{array}$$

$$\begin{array}{l} SeqNext \triangleq \ (\exists\, self \in N \setminus B : SeqConstraints(self) \land correctNode(self)) \\ \qquad\qquad \lor\, (\exists\, self \in B : SeqConstraints(self) \land byzantineNode(self)) \\ SeqSpec \triangleq \ Init \land \Box[SeqNext]_{vars} \end{array}$$

Example assignment of leaders to rounds:

$$ModLeader(r) \triangleq NodeSeq[(r\%Cardinality(N)) + 1]$$

Constraint to stop the model checker:

$$\begin{array}{l} StateConstraint \triangleq \\ \quad \text{LET } Max(S) \triangleq \text{CHOOSE } x \in S : \forall\, y \in S : y \leq x \text{IN} \\ \qquad \forall\, n \in N : Round\_(n) \in 0\,..\,(Max(R) + 1) \end{array}$$

Some properties we expect to be violated:

$$\begin{array}{l} Falsy1 \triangleq \neg(\\ \quad \land\, Committed(\langle Leader(1),\, 1 \rangle) \\ ) \end{array}$$

$$\begin{array}{l} Falsy2 \triangleq \neg(\\ \quad \land\, Committed(\langle Leader(0),\, 0 \rangle) \\ \quad \land\, \neg Committed(\langle Leader(1),\, 1 \rangle) \\ \quad \land\, \neg Committed(\langle Leader(2),\, 2 \rangle) \\ \quad \land\, Committed(\langle Leader(3),\, 3 \rangle) \\ ) \end{array}$$