MODULE *Sailfish*

Specification of the *Sailfish* consensus algorithm at a high level of abstraction.

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 ∈ N ↦ {}];  no_vote messages sent by each node
  define {
    LeaderVertice(r) ≜ ⟨Leader(r), r⟩
    VerticeQuorums(r) ≜
      {VQ ∈ SUBSET vs :
          ∧ ∀ v ∈ VQ : Round(v) = r
          ∧ {Node(v) : v ∈ VQ} ∈ Quorum}
  }
  process ( correctNode ∈ N \ F )
    variables round = 0;  current round
  {
l0:  while ( TRUE )
     either with ( v = ⟨self, round⟩ ) {
         add a new vertex to the DAG and go to the next round
       vs := vs ∪ {v};
       if ( round > 0 )
       with ( vq ∈ VerticeQuorums(round − 1) ) {
           from GST onwards, each node receives all correct vertices of the previous round:
         when round ≥ GST ⇒ (N \ F) ⊆ {Node(v2) : v2 ∈ vq};
         if ( Leader(round) = self ) {
             we must either include the previous leader vertice,
             or a quorum of no_vote messages.
           when
             ∨ LeaderVertice(round − 1) ∈ vq
             ∨ ∃ Q ∈ Quorum : ∀ n ∈ Q \ {self} : LeaderVertice(round − 1) ∈ no_vote[n];
         };
         es := es ∪ {⟨v, pv⟩ : pv ∈ vq};  add the edges
         if ( LeaderVertice(round − 1) ∉ vq )  send no_vote if previous leader vertice not included
           no_vote[self] := no_vote[self] ∪ {LeaderVertice(round − 1)}
       };
       round := round + 1
     }
     or with ( r ∈ {r ∈ R : r > round} ) {
```

go to a higher round

when $r \leq GST$ **;** 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). Also note that creating a round-r vertice commutes to the left of actions of rounds greater than r and to the right of actions of rounds smaller than R, so we can, without loss of generality, schedule *Byzantine* nodes in the same "round-by-round" manner as other nodes.

process **(** $byzantineNode \in F$ **)**

variables $round\_ = 0$ **;**

{

$l0$: **while** **(** TRUE **)** **{**

maybe add a vertices to the DAG:

either with **(** $v = \langle self,\ round\_\rangle$ **)** **{**

$vs := vs \cup \{v\}$ **;**

if **(** $round\_ > 0$ **)**

with **(** $vq \in VerticeQuorums(round\_ - 1)$ **)** **{**

$es := es \cup \{\langle v,\ pv\rangle : pv \in vq\}$

}

} or skip ;

maybe send a no_vote messages:

if **(** $round\_ > 0$ **)**

either

$no\_vote[self] := no\_vote[self] \cup \{LeaderVertice(round\_ - 1)\}$

or skip ;

go to the next round:

$round\_ := round\_ + 1$

}

}

}

Next we define the safety and liveness properties

$$
\begin{aligned}
&Committed(v) \triangleq \\
&\quad \land\ v \in vs \\
&\quad \land\ Node(v) = Leader(Round(v)) \\
&\quad \land\ \exists\, Q \in Quorum : Q \subseteq \{Node(pv) : pv \in Parents(v,\ es)\} \\
&\quad \land\ \lor\ Round(v) = 0 \\
&\quad \ \lor\ LeaderVertice(Round(v) - 1) \in Children(v,\ es) \\
&\quad \ \lor\ \exists\, Q \in Quorum : \forall\, n \in Q : \\
&\quad \quad LeaderVertice(Round(v) - 1) \in no\_vote[n] \\
&Safety \triangleq \forall\, v1,\ v2 \in vs : \\
&\quad \land\ Committed(v1) \\
&\quad \land\ Committed(v2)
\end{aligned}
$$

$\land \ Round(v1) \leq Round(v2)$
$\Rightarrow Reachable(v2, \ v1, \ es)$

$Liveness \triangleq \forall r \in R :$
$\land \ r \geq GST$
$\land \ Leader(r) \notin F$
all correct $round - (r + 1)$ vertices are created:
$\land \ \forall n \in N \setminus F : round[n] > r + 1$
$\Rightarrow Committed(LeaderVertice(r))$

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

$$Quorum1 \triangleq \{Q \in \text{SUBSET } N : Cardinality(Q) \geq Cardinality(N) - Cardinality(F)\}$$
$$Blocking1 \triangleq \{Q \in \text{SUBSET } N : Cardinality(Q) > Cardinality(F)\}$$

The round of a node, whether $Byzantine$ or not
$Round\_(n) \triangleq \text{IF } n \in F \text{ THEN } round\_[n] \text{ ELSE } round[n]$

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

Sequentialization constraints, which enforce a particular ordering of the actions. Because of how actions commute, the set of reachable states remains unchanged. Essentially, we schedule all nodes "round-by-round" and in lock-steps, with the leader last. This speeds up model-checking a lot.

Note that we must always schedule the leader last because, due to its use of no_vote messages of other nodes, its action does not commute to the left of the actions of other nodes.

$SeqConstraints(n) \triangleq$
wait for all nodes to finish previous rounds:
$\land (Round\_(n) > 0 \Rightarrow \forall n2 \in N : Round\_(n2) \geq Round\_(n))$
wait for all nodes with lower index to leave the round (leader index is always last):
$\land \forall n2 \in N : NodeIndexLeaderLast(n2, \ Round\_(n)) < NodeIndexLeaderLast(n, \ Round\_(n)) \Rightarrow Round$

$SeqNext \triangleq (\exists self \in N \setminus F : SeqConstraints(self) \land correctNode(self))$
$\qquad \lor (\exists self \in F : SeqConstraints(self) \land byzantineNode(self))$
$SeqSpec \triangleq Init \land \Box[SeqNext]_{vars}$

Example assignment of leaders to rounds:
$ModLeader(r) \triangleq NodeSeq[(r\%Cardinality(N)) + 1]$

Constraint to stop the model checker:

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

Some properties we expect to be violated:

$Falsy1 \triangleq \neg($
$\quad \land\ Committed(\langle Leader(1),\ 1\rangle)$
$)$

$Falsy2 \triangleq \neg($
$\quad \land\ Committed(\langle Leader(0),\ 0\rangle)$
$\quad \land\ \neg Committed(\langle Leader(1),\ 1\rangle)$
$\quad \land\ \neg Committed(\langle Leader(2),\ 2\rangle)$
$\quad \land\ Committed(\langle Leader(3),\ 3\rangle)$
$)$

$Falsy3 \triangleq \neg($
$\quad \land\ Committed(LeaderVertice(0))$
$\quad \land\ \exists Q \in Quorum : \forall n \in Q : LeaderVertice(0) \in no\_vote[n]$
$\quad \land\ round[Leader(1)] > 1$
$\quad \land\ \langle LeaderVertice(1),\ LeaderVertice(0)\rangle \notin es$
$)$