MODULE *Sailfish2*

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

Compared to the *Sailfish1* specification, we additionally model committing with just $f+1$ parents, as is possible in the *Sailfish* paper.

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 ≤ 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 ∈ F )
  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

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

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

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

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

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.

Compared to the *Sailfish*1 specification, 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:
$\quad \land (Round\_(n) > 0 \Rightarrow \forall\, n2 \in N : Round\_(n2) \geq Round\_(n))$
wait for all nodes with lower index to leave the round (leader index is always last):
$\quad \land \forall\, n2 \in N : NodeIndexLeaderLast(n2,\, Round\_(n)) < NodeIndexLeaderLast(n,\, Round\_(n)) \Rightarrow Round$

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

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

Constraint to stop the model checker:

$StateConstraint \triangleq$

$\quad \text{LET } Max(S) \triangleq \text{CHOOSE } x \in S : \forall y \in S : y \leq x \text{IN}$

$\qquad \forall n \in N : Round\_(n) \in 0 \, .. \, (Max(R) + 1)$

Some properties we expect to be violated:

$Falsy1 \triangleq \neg($

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

$)$

$Falsy2 \triangleq \neg($

$\quad \land\ Committed(\langle Leader(0), 0 \rangle)$

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

$\quad \land\ \neg Committed(\langle Leader(2), 2 \rangle)$

$\quad \land\ Committed(\langle Leader(3), 3 \rangle)$

$)$

$Falsy3 \triangleq \neg($

$\quad \land\ Committed(LeaderVertice(0))$

$\quad \land\ \exists Q \in Quorum : \forall n \in Q : LeaderVertice(0) \in no\_vote[n]$

$\quad \land\ round[Leader(1)] > 1$

$\quad \land\ \langle LeaderVertice(1), LeaderVertice(0) \rangle \notin es$

$)$