MODULE *Sailfish*

This is a high-level specification of the *Sailfish* and *Sailfish*++ (also called signature-free *Sailfish*) algorithms. At the level of abstraction of this specification, the differences between the two algorithms are not visible.

EXTENDS $Integers$, $FiniteSets$, $Sequences$

CONSTANTS
N The set of all nodes
, F The set of *Byzantine* nodes
, R The set of rounds
, $IsQuorum(\_)$ Whether a set is a quorum (*i.e.* cardinality $\geq n$-f)
, $IsBlocking(\_)$ Whether a set is a blocking set (*i.e.* cardinality $\geq f+1$)
, $Leader(\_)$ operator mapping each round to its leader
, GST the first round in which the system is synchronous

ASSUME $\exists n \in R : R = 1 \,..\, n$ rounds start at 1; 0 is used as default placeholder

INSTANCE $BlockDag$ Import definitions related to *DAGs* of blocks

Now we specify the algorithm in the *PlusCal* language.

--algorithm *Sailfish*{
variables
$vs = \{Genesis\}$, the vertices of the *DAG*
$es = \{\}$ **;** the edges of the *DAG*
define {
$dag \triangleq \langle vs, es\rangle$
$NoLeaderVoteQuorum(r, vertices, add) \triangleq$
LET $NoLeaderVote \triangleq \{v \in vertices : LeaderVertex(r-1) \notin Children(dag, v)\}$
IN $IsQuorum(\{Node(v) : v \in NoLeaderVote\} \cup add)$
}
process ($correctNode \in N \setminus F$ **)**
variables
$round = 0$, current round; 0 means the node has not started execution
$log = \langle\rangle$ **;** delivered *log*
{
$l0$: **while (** TRUE **) {**
if ($round = 0$ **) {** start the first round $r = 1$
$round := 1$ **;**
$vs := vs \cup \{\langle self, 1\rangle\}$ **;**
$es := es \cup \{\langle\langle self, 1\rangle, Genesis\rangle\}$
}
else { start a round $r > 1$
with ($r \in \{r \in R : r > round\}$ **)**
with ($deliveredVertices \in$ SUBSET $\{v \in vs : Round(v) = r-1\}$ **) {**
we enter a round only if we have a quorum of vertices:

```
            await IsQuorum({Node(v) : v ∈ deliveredVertices}) ;
             if this is after GST, we must have all correct vertices:
            await r ≥ GST ⇒ (N \ F) ⊆ {Node(v) : v ∈ deliveredVertices} ;
             enter round r:
            round := r ;
             if the r − 1 leader does not reference the r − 2 leader,
             then we must be sure the r − 2 leader cannot commit:
            await LeaderVertex(r − 1) ∈ deliveredVertices ⇒
                    ∨ LeaderVertex(r − 2) ∈ Children(dag, LeaderVertex(r − 1))
                    ∨ NoLeaderVoteQuorum(r − 1, deliveredVertices, {}) ;
             if we are the leader, then we must include the r − 1 leader or
             have evidence it cannot commit:
            if ( Leader(r) = self )
                await   ∨ LeaderVertex(r − 1) ∈ deliveredVertices
                        ∨ NoLeaderVoteQuorum(r, {v ∈ vs : Round(v) = r}, {self}) ;
             create a new vertex:
            with ( newV = ⟨self, r⟩ ) {
                vs := vs ∪ {newV} ;
                es := es ∪ {⟨newV, pv⟩ : pv ∈ deliveredVertices} ;
             } ;
             commit if there is a quorum of votes for the leader of r − 2:
            if ( r > 2 )
                with ( votesForLeader = {pv ∈ deliveredVertices : ⟨pv, LeaderVertex(r − 2)⟩ ∈ es}
                if ( IsQuorum({Node(pv) : pv ∈ votesForLeader}) )
                    log := Linearize(dag, LeaderVertex(r − 2))
          }
        }
      }
    }
```

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 )
    {
l0:   while ( TRUE ) {
        with ( r ∈ R )
        with ( newV = ⟨self, r⟩ ) {
          when newV ∉ vs ;   no equivocation
          if ( r = 1 ) {
              vs := vs ∪ {newV} ;
              es := es ∪ {⟨newV, Genesis⟩}
           }
          else
          with ( delivered ∈ SUBSET {v ∈ vs : Round(v) = r − 1} ) {
            await IsQuorum({Node(v) : v ∈ delivered}) ;   ignored otherwise
```

$vs := vs \cup \{newV\}$ **;**
$es := es \cup \{\langle newV,\ pv \rangle : pv \in delivered\}$
}
}
}
}
}

BEGIN TRANSLATION ($chksum(pcal) =$ "$c16dfa43$" $\land\ chksum(tla) =$ "$9cdbd4f5$")

Label $l0$ of process *correctNode* at line 42 col 9 changed to $l0\_$

VARIABLES $vs,\ es$

define statement

$dag \triangleq \langle vs,\ es \rangle$

$NoLeaderVoteQuorum(r,\ vertices,\ add) \triangleq$
$\quad$ LET $NoLeaderVote \triangleq \{v \in vertices : LeaderVertex(r-1) \notin Children(dag,\ v)\}$
$\quad$ IN $\quad IsQuorum(\{Node(v) : v \in NoLeaderVote\} \cup add)$

VARIABLES $round,\ log$

$vars \triangleq \langle vs,\ es,\ round,\ log \rangle$

$ProcSet \triangleq (N \setminus F) \cup (F)$

$Init \triangleq$ Global variables
$\quad \land\ vs = \{Genesis\}$
$\quad \land\ es = \{\}$
$\quad$ Process *correctNode*
$\quad \land\ round = [self \in N \setminus F \mapsto 0]$
$\quad \land\ log = [self \in N \setminus F \mapsto \langle\rangle]$

$correctNode(self) \triangleq$ IF $round[self] = 0$
$\quad$ THEN $\land\ round' = [round$ EXCEPT $![self] = 1]$
$\qquad \land\ vs' = (vs \cup \{\langle self,\ 1 \rangle\})$
$\qquad \land\ es' = (es \cup \{\langle \langle self,\ 1 \rangle,\ Genesis \rangle\})$
$\qquad \land\ log' = log$
$\quad$ ELSE $\land\ \exists\, r \in \{r \in R : r > round[self]\} :$
$\qquad \exists\, deliveredVertices \in$ SUBSET $\{v \in vs : Round(v) = r-1\} :$
$\qquad\quad \land\ IsQuorum(\{Node(v) : v \in deliveredVertices\})$
$\qquad\quad \land\ r \geq GST \Rightarrow (N \setminus F) \subseteq \{Node(v) : v \in deliveredVertices\}$
$\qquad\quad \land\ round' = [round$ EXCEPT $![self] = r]$
$\qquad\quad \land\ LeaderVertex(r-1) \in deliveredVertices \Rightarrow$
$\qquad\qquad \lor\ LeaderVertex(r-2) \in Children(dag,\ LeaderVertex(r-1))$
$\qquad\qquad \lor\ NoLeaderVoteQuorum(r-1,\ deliveredVertices,\ \{\})$
$\qquad\quad \land$ IF $Leader(r) = self$
$\qquad\qquad$ THEN $\land\ \lor\ LeaderVertex(r-1) \in deliveredVertices$
$\qquad\qquad\qquad \lor\ NoLeaderVoteQuorum(r,\ \{v \in vs : Round(v) = r\}$
$\qquad\qquad$ ELSE $\land$ TRUE

$\land$ LET $newV \triangleq \langle self, r \rangle$ IN
$\quad \land vs' = (vs \cup \{newV\})$
$\quad \land es' = (es \cup \{\langle newV, pv \rangle : pv \in deliveredVertices\})$
$\land$ IF $r > 2$
THEN $\land$ LET $votesForLeader \triangleq \{pv \in deliveredVertices : \langle pv$
IF $IsQuorum(\{Node(pv) : pv \in votesForLeader\})$
THEN $\land log' = [log$ EXCEPT $![self] = Linearize($
ELSE $\land$ TRUE
$\land log' = log$
ELSE $\land$ TRUE
$\land log' = log$

$byzantineNode(self) \triangleq \land \exists r \in R :$
LET $newV \triangleq \langle self, r \rangle$ IN
$\land newV \notin vs$
$\land$ IF $r = 1$
THEN $\land vs' = (vs \cup \{newV\})$
$\land es' = (es \cup \{\langle newV, Genesis \rangle\})$
ELSE $\land \exists delivered \in$ SUBSET $\{v \in vs : Round(v) = r - 1\} :$
$\land IsQuorum(\{Node(v) : v \in delivered\})$
$\land vs' = (vs \cup \{newV\})$
$\land es' = (es \cup \{\langle newV, pv \rangle : pv \in delivered\})$
$\land$ UNCHANGED $\langle round, log \rangle$

$Next \triangleq (\exists self \in N \setminus F : correctNode(self))$
$\quad \lor (\exists self \in F : byzantineNode(self))$

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

END TRANSLATION

Basic type invariant:

$TypeOK \triangleq$
$\land \forall v \in vs \setminus \{\langle\rangle\} :$
$\quad \land Node(v) \in N \land Round(v) \in Nat \setminus \{0\}$
$\quad \land \forall c \in Children(dag, v) : Round(c) = Round(v) - 1$
$\land \forall e \in es :$
$\quad \land e = \langle e[1], e[2] \rangle$
$\quad \land \{e[1], e[2]\} \subseteq vs$
$\land \forall n \in N \setminus F : round[n] \in Nat$

Next we define the safety and liveness properties

$Agreement \triangleq \forall n1, n2 \in N \setminus F : Compatible(log[n1], log[n2])$

$Liveness \triangleq \forall r \in R : r \geq GST \land Leader(r) \notin F \Rightarrow$
$\quad \forall n \in N \setminus F : round[n] \geq r + 2 \Rightarrow$

$\exists\, i \in \textsc{domain}\ log[n] : log[n][i] = LeaderVertex(r)$