MODULE *VDFConsensus*

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

CONSTANTS
P the set of processes
, B the set of malicious processes
, $tAdv$ the time it takes for a malicious process to produce a message
, tWB the time it takes for a well-behaved process to produce a message

ASSUME $B \subseteq P$ malicious processes are a subset of all processes
$W \triangleq P \setminus B$ the set of well-behaved processes

$Tick \triangleq Nat$ a tick is a real-time clock tick
$Round \triangleq Nat$ a round is just a tag on a message

Processes build a *DAG* of messages. The message-production rate of well-behaved processes is of 1 message per *tWB* ticks, and that of malicious processes is of 1 message per *tAdv* ticks. We require that, collectively, well-behaved processes produce messages at a rate strictly higher than that of malicious processes.

ASSUME $Cardinality(W) * tAdv > Cardinality(B) * tWB$

TODO: *I* think we're going to need $Cardinality(W) * tAdv > 2 * Cardinality(B) * tWB$

$MessageID \triangleq Nat$

A message consists of a unique *ID*, a round number, and a coffer containing the *IDs* of a set of predecessor messages:
$Message \triangleq [sender : P,\ id : MessageID,\ round : Round,\ coffer : \text{SUBSET } MessageID]$

We will need the intersection of a set of sets:
RECURSIVE $Intersection(\_)$
$Intersection(Ss) \triangleq$
CASE
$Ss = \{\} \rightarrow \{\}$
$\Box\ \exists S \in Ss : Ss = \{S\} \rightarrow \text{CHOOSE } S \in Ss : Ss = \{S\}$
$\Box$ OTHER $\rightarrow$
LET $S \triangleq (\text{CHOOSE } S \in Ss : \text{TRUE})$
IN $S \cap Intersection(Ss \setminus \{S\})$

A set of messages is consistent when the intersection of the coffers of each message is a strict majority of the coffer of each message.

$ConsistentSet(M) \triangleq$
LET $I \triangleq Intersection(\{m.coffer : m \in M\})$
IN $\forall m \in M : 2 * Cardinality(I) > Cardinality(m.coffer)$

A consistent chain is a subset of the messages in the *DAG* that potentially has some dangling pointers (*i.e.* messages that have predecessors not in the chain) and that satisfies the following recursive predicate:

* Any set of messages which all have a round of 0 is a consistent chain.

* A set of messages C with some non-zero rounds and maximal round r is a consistent chain when, with *Tip* being the set of messages in the chain that have round r and *Pred* being the set of messages in the chain with round $r-1$, *Pred* is a strict majority of the set of predecessors of each message in *Tip* and $C \setminus Tip$ is a consistent chain. (Note that this implies that *Tip* is a consistent set)

$$\begin{array}{l}
\mathit{Max}(X,\ \mathit{Leq}(_,\ _)) \ \triangleq \\
\qquad \textsc{choose}\ m \in X : \forall x \in X : \mathit{Leq}(x,\ m)
\end{array}$$

TODO this might be too restrictive: maybe we should only require that a subset of *Pred* be a strict majority of the set of predecessors of each message in *Tip*

$$\begin{array}{l}
\textsc{recursive}\ \mathit{ConsistentChain}(_) \\
\mathit{ConsistentChain}(M) \ \triangleq \\
\quad \land\ M \neq \{\} \\
\quad \land\ \textsc{let}\ r \ \triangleq\ \mathit{Max}(\{m.\mathit{round} : m \in M\},\ \leq)\textsc{in} \\
\qquad \lor\ \ r = 0 \\
\qquad \lor\ \ \textsc{let}\ \mathit{Tip} \ \triangleq\ \{m \in M : m.\mathit{round} = r\} \\
\qquad\qquad\qquad \mathit{Pred} \ \triangleq\ \{m \in M : m.\mathit{round} = r - 1\} \\
\qquad\quad\ \textsc{in} \quad \land\ \mathit{Tip} \neq \{\} \\
\qquad\qquad\qquad \land\ \exists\, \mathit{Maj} \in \textsc{subset}\ \mathit{Pred} : \\
\qquad\qquad\qquad\quad \land\ \mathit{Maj} \neq \{\} \\
\qquad\qquad\qquad\quad \land\ \forall\, m \in \mathit{Tip} : \\
\qquad\qquad\qquad\qquad \land \forall\, m2 \in \mathit{Maj} : m2.\mathit{id} \in m.\mathit{coffer} \\
\qquad\qquad\qquad\qquad \land\ 2 * \mathit{Cardinality}(\mathit{Maj}) > \mathit{Cardinality}(m.\mathit{coffer}) \\
\qquad\qquad\qquad \land\ \mathit{ConsistentChain}(M \setminus \mathit{Tip})
\end{array}$$

Given a message DAG, the heaviest consistent chain is a consistent chain in the DAG that has a maximal number of messages.

$$\begin{array}{l}
\mathit{HeaviestConsistentChain}(M) \ \triangleq \\
\quad \textsc{let}\ r \ \triangleq\ \mathit{Max}(\{m.\mathit{round} : m \in M\},\ \leq) \\
\qquad\quad \mathit{Cs} \ \triangleq\ \{C \in \textsc{subset}\ M : (\exists\, m \in C : m.\mathit{round} = r) \land \mathit{ConsistentChain}(C)\} \\
\quad \textsc{in} \\
\qquad \textsc{if}\ \mathit{Cs} = \{\}\ \textsc{then}\ \{\} \\
\qquad\ \ \textsc{else}\ \ \mathit{Max}(\mathit{Cs},\ \textsc{lambda}\ C1,\ C2 : \mathit{Cardinality}(C1) \leq \mathit{Cardinality}(C2))
\end{array}$$

Two chains are disjoint when there is a round in which they have no messages in common:

$$\begin{array}{l}
\mathit{DisjointChains}(C1,\ C2) \ \triangleq \\
\quad \textsc{let}\ r1 \ \triangleq\ \mathit{Max}(\{m.\mathit{round} : m \in C1\},\ \leq) \\
\qquad\quad r2 \ \triangleq\ \mathit{Max}(\{m.\mathit{round} : m \in C2\},\ \leq) \\
\quad \textsc{in}\quad \exists\, r \in \mathit{Round} : \\
\qquad\qquad \land\ r \leq r1 \\
\qquad\qquad \land\ r \leq r2 \\
\qquad\qquad \land\ \{m \in C1 : m.\mathit{round} = r\} \cap \{m \in C2 : m.\mathit{round} = r\} = \{\}
\end{array}$$

Now we specify the algorithm

TODO: let me be a function from *ID* to message; will help reading counter-examples

```
--algorithm Algo{
  variables
      messages = {} ;
      tick = 0 ;
      phase = "start" ;   each tick has two phases: "start" and "end"
      donePhase = [p ∈ P ↦ "end"] ;
      pendingMessage = [p ∈ P ↦ ⟨⟩] ;
      messageCount = 0 ;   used to generate unique message IDs
  define {
      currentRound ≜ tick ÷ tWB   round of well-behaved processes
      wellBehavedMessages ≜ {m ∈ messages : m.sender ∈ P \ B}
        possible sets of messages received by a well-behaved process:
      receivedMsgsSets ≜
            ignore messages from future rounds:
          LET msgs ≜ {m ∈ messages : m.round < currentRound} IN
          {wellBehavedMessages ∪ byzMsgs :
              byzMsgs ∈ SUBSET (msgs \ wellBehavedMessages)}
   }
  macro sendMessage( m ) {
      messages := messages ∪ {m}
   }
  process ( clock ∈ {"clock"} ) {
tick:  while ( TRUE ) {
          await ∀ p ∈ P : donePhase[p] = phase ;
          if ( phase = "start" )
              phase := "end"
          else {
              phase := "start" ;
              tick := tick + 1
           }
        }
   }
  process ( proc ∈ P \ B )   a well-behaved process
  {
l1:    while ( TRUE ) {
          await phase = "start" ;
          if ( tick%tWB = 0 ) {
                Start the VDF computation for the next message:
              with ( msgs ∈ receivedMsgsSets )
              with ( hCC = HeaviestConsistentChain(msgs) )
              with ( predMsgs = {m ∈ hCC : m.round = currentRound − 1} ) {
                  pendingMessage[self] := [
                      sender ↦ self,
                      id ↦ messageCount + 1,
                      round ↦ currentRound,
```

```
                    coffer ↦ {m.id : m ∈ predMsgs}] ;
                messageCount := messageCount + 1 ;
              }
          } ;
          donePhase[self] := "start" ;
l2:       await phase = "end" ;
          if ( tick%tWB = tWB − 1 )
              it's the end of the tWB period, the VDF has been computed
              sendMessage(pendingMessage[self]) ;
          donePhase[self] := "end" ;
        }
    }
    process ( byz ∈ B )   a malicious process
    {
lb1:  while ( TRUE ) {
          await phase = "start" ;
          if ( tick%tAdv = 0 ) {
              Start the VDF computation for the next message:
              with ( maxRound = Max({m.round : m ∈ messages} ∪ {0}, ≤ ) )
              with ( rnd ∈ {maxRound, maxRound + 1} )
              with ( predMsgs ∈ SUBSET {m ∈ messages : m.round = rnd − 1} ) {
                  when rnd > 0 ⇒ predMsgs ≠ {} ;
                  pendingMessage[self] := [
                      sender ↦ self,
                      id ↦ messageCount + 1,
                      round ↦ rnd,
                      coffer ↦ {m.id : m ∈ predMsgs}] ;
                  messageCount := messageCount + 1 ;
              }
          } ;
          donePhase[self] := "start" ;
lb2:      await phase = "end" ;
          if ( tick%tAdv = tAdv − 1 )
              sendMessage(pendingMessage[self]) ;
          donePhase[self] := "end" ;
      } ;
    }
}
TypeOK ≜
    ∧ messages ∈ SUBSET Message
    ∧ pendingMessage ∈ [P → Message ∪ {⟨⟩}]
    ∧ tick ∈ Tick
    ∧ phase ∈ {"start", "end"}
    ∧ donePhase ∈ [P → {"start", "end"}]
    ∧ messageCount ∈ Nat
```

$$messageWithID(id) \triangleq \text{CHOOSE } m \in messages : m.id = id$$

The main property we want to establish is that, each round, for each message m of a well-behaved process, the messages of well-behaved processes from the previous round are all in m's coffer and consist of a strict majority of m's coffer.

$$
\begin{aligned}
&Safety \triangleq \forall\, m \in messages : m.round > 0 \land m.sender \notin B \Rightarrow \\
&\quad \land\ \forall\, m2 \in wellBehavedMessages : m2.round = m.round - 1 \Rightarrow m2.id \in m.coffer \\
&\quad \land\ \text{LET } M \triangleq \{m2 \in wellBehavedMessages : m2.round = m.round - 1\} \\
&\quad\quad \text{IN}\quad 2 * Cardinality(M) > Cardinality(m.coffer)
\end{aligned}
$$

A basic well-formedness property:

$$
\begin{aligned}
&Inv1 \triangleq \forall\, m \in messages : \forall\, id \in m.coffer : \\
&\quad \land\ \exists\, m2 \in messages : m2.id = id \\
&\quad \land\ messageWithID(id).round = m.round - 1
\end{aligned}
$$