MODULE $VDFConsensus$

EXTENDS $FiniteSets$, $Integers$, $Utils$

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

A message consists of a unique ID, a round number, and a coffer containing the IDs of a set of predecessor messages:
$MessageID \triangleq P \times Nat$
$Message \triangleq [id : MessageID, round : Round, coffer : \text{SUBSET } MessageID]$
$sender(m) \triangleq m.id[1]$

A strongly 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 strongly consistent chain.

* A set of messages C with some non-zero rounds and maximal round r is a strongly 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.

The max round of a set of messages is the maximal round of its messages:
$MaxRound(M) \triangleq MaxInteger(\{m.round : m \in M\})$

$StronglyConsistentChain(M) \triangleq$
$\quad \land\ M \neq \{\}$
$\quad \land\ \lor\ MaxRound(M) = 0$
$\quad \quad \lor\ \forall r \in 1 \,..\, MaxRound(M) :$
$\quad \quad \quad \text{LET } Tip \triangleq \{m \in M : m.round = r\}$
$\quad \quad \quad \quad \quad Pred \triangleq \{m \in M : m.round = r - 1\}$
$\quad \quad \quad \text{IN} \quad \land\ Tip \neq \{\}$
$\quad \quad \quad \quad \quad \land\ \forall m \in Tip :$
$\quad \quad \quad \quad \quad \quad \land\ \{m2.id : m2 \in Pred\} \subseteq m.coffer$

$$\land \;\; 2 * Cardinality(Pred) > Cardinality(m.coffer)$$

A weaker version of the above:

$$
\begin{array}{l}
ConsistentChain(M) \;\triangleq \\
\quad \land \;\; M \neq \{\} \\
\quad \land \;\; \lor \;\; MaxRound(M) = 0 \\
\quad \;\; \lor \;\; \forall\, r \in 1 \,..\, MaxRound(M) : \\
\qquad\qquad \text{LET } Tip \;\triangleq\; \{m \in M : m.round = r\} \\
\qquad\qquad\phantom{\text{LET }} Pred \;\triangleq\; \{m \in M : m.round = r - 1\} \\
\qquad\qquad \text{IN} \quad \land \;\; Tip \neq \{\} \\
\qquad\qquad\phantom{\text{IN}} \quad \land \;\; \forall\, m \in Tip : \exists\, Maj \in \text{SUBSET } Pred : \\
\qquad\qquad\qquad\quad \land \{m2.id : m2 \in Maj\} \subseteq m.coffer \\
\qquad\qquad\qquad\quad \land \;\; 2 * Cardinality(Maj) > Cardinality(m.coffer)
\end{array}
$$

$$
\begin{array}{l}
\text{RECURSIVE } Chains(\_,\, \_) \\
Chains(M,\, r) \;\triangleq \\
\quad \text{LET } MM \;\triangleq\; \{m \in M : m.round = r\} \text{IN} \\
\qquad \text{IF } r = 0 \\
\qquad\; \text{THEN } \{M1 : M1 \in (\text{SUBSET } MM) \setminus \{\}\} \\
\qquad\; \text{ELSE } \;\{M1 \cup M2 : M1 \in (\text{SUBSET } MM) \setminus \{\},\, M2 \in Chains(M \setminus MM,\, r - 1)\}
\end{array}
$$

The set of all consistent chains that can be found in M:

$$
\begin{array}{l}
ConsistentChains(M,\, r) \;\triangleq \\
\quad \{C \in Chains(M,\, r) : ConsistentChain(C)\}
\end{array}
$$

The set of all strongly consistent chains that can be found in M:

$$
\begin{array}{l}
StronglyConsistentChains(M,\, r) \;\triangleq \\
\quad \{C \in Chains(M,\, r) : StronglyConsistentChain(C)\}
\end{array}
$$

We can rank the chains by wheight, $i.e.$ just their cardinality, or we can consider the maximal or minimal one in the subset order:

$$
\begin{array}{l}
HeaviestConsistentChains(M,\, r) \;\triangleq\; MaxCardinalitySets(ConsistentChains(M,\, r)) \\
HeaviestStronglyConsistentChains(M,\, r) \;\triangleq\; MaxCardinalitySets(StronglyConsistentChains(M,\, r)) \\
MinimalStronglyConsistentChains(M,\, r) \;\triangleq\; MinimalSets(StronglyConsistentChains(M,\, r)) \\
MaximalStronglyConsistentChains(M,\, r) \;\triangleq\; MaximalSets(StronglyConsistentChains(M,\, r))
\end{array}
$$

Two chains are disjoint when there is a round smaller than their max round in which they have no messages in common.

$$
\begin{array}{l}
DisjointChains(C1,\, C2) \;\triangleq \\
\quad \text{LET } rmax \;\triangleq\; MaxRound(C1 \cup C2) \\
\quad \text{IN} \quad \exists\, r \in 0 \,..\, (rmax - 1) : \\
\qquad\qquad \{m \in C1 : m.round = r\} \cap \{m \in C2 : m.round = r\} = \{\}
\end{array}
$$

Acceptance rule

$$
\begin{array}{l}
AcceptedMessages(M,\, r) \;\triangleq\; \{m \in M : \\
\quad \land \;\; m.round = r - 1
\end{array}
$$

$\land$ LET $CCs \triangleq MaximalStronglyConsistentChains(M, r-1)$ IN   This looks promising!
$\quad\land\ \exists C \in CCs : m \in C$
$\quad\land\ \forall C1, C2 \in CCs :$
$\qquad\land\ m \in C1$
$\qquad\land\ m \notin C2$
$\qquad\land\ DisjointChains(C1, C2)$
$\qquad\Rightarrow Cardinality(C2) \leq Cardinality(C1)\}$

M does not have dangling edges:
$Closed(M) \triangleq \forall m \in M : \forall i \in m.coffer : \exists m2 \in M : m2.id = i$

Now we specify the algorithm

```
--algorithm Algo{
  variables
      messages = {} ;
      tick = 0 ;
      phase = "start" ;   each tick has two phases: "start" and "end"
      donePhase = [p ∈ P ↦ "end"] ;
      pendingMessage = [p ∈ P ↦ ⟨⟩] ;   message we're computing the VDF on
      messageCount = [p ∈ P ↦ 0] ;   used to generate unique message IDs
  define {
      currentRound ≜ tick ÷ tWB   round of well-behaved processes
      wellBehavedMessages ≜ {m ∈ messages : sender(m) ∈ 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
        {M ∈ SUBSET msgs :
             don't use a set of messages that has dangling edges (messages in coffers that are missing):
           ∧ Closed(M)
           ∧ wellBehavedMessages ⊆ M}
  }
  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 ( predMsgs = AcceptedMessages(msgs, currentRound) ) {
              pendingMessage[self] := [
                id ↦ ⟨self, messageCount[self] + 1⟩,
                round ↦ currentRound,
                coffer ↦ {m.id : m ∈ predMsgs}] ;
              messageCount[self] := messageCount[self] + 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] := [
                id ↦ ⟨self, messageCount[self] + 1⟩,
                round ↦ rnd,
                coffer ↦ {m.id : m ∈ predMsgs}] ;
              messageCount[self] := messageCount[self] + 1
            }
          } ;
          donePhase[self] := "start" ;
lb2:      await phase = "end" ;
          if ( tick%tAdv = tAdv − 1 )
```

$\quad\quad\quad\quad sendMessage(pendingMessage[self])$ **;**
$\quad\quad\quad donePhase[self] :=$ "end" **;**
$\quad\quad$ **}** **;**
$\quad$ **}**
}

Invariant describing the type of the variables:

$$
\begin{aligned}
&TypeOK \triangleq \\
&\quad \land\ messages \in \text{SUBSET } Message \\
&\quad \land\ pendingMessage \in [P \to Message \cup \{\langle\rangle\}] \\
&\quad \land\ tick \in Tick \\
&\quad \land\ phase \in \{\text{"start"}, \text{"end"}\} \\
&\quad \land\ donePhase \in [P \to \{\text{"start"}, \text{"end"}\}] \\
&\quad \land\ messageCount \in [P \to Nat]
\end{aligned}
$$

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.

$$
\begin{aligned}
&Safety \triangleq \forall p \in P \setminus B : \text{LET } m \triangleq pendingMessage[p] \text{IN} \\
&\quad \land\ m \neq \langle\rangle \\
&\quad \land\ m.round > 0 \\
&\quad \Rightarrow \\
&\quad \land\ \forall m2 \in wellBehavedMessages : m2.round = m.round - 1 \Rightarrow m2.id \in m.coffer \\
&\quad \land\ \text{LET } M \triangleq \{m2 \in wellBehavedMessages : m2.round = m.round - 1\} \\
&\quad\quad \text{IN} \quad 2 * Cardinality(M) > Cardinality(m.coffer)
\end{aligned}
$$

helper definition:

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

Basic well-formedness properties:

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