MODULE *VDFConsensus*

EXTENDS *FiniteSets*, *Integers*

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

$MessageID \triangleq Nat$

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
    $Ss = \{\} \to \{\}$
  □ $\exists\, S \in Ss : Ss = \{S\} \to \text{CHOOSE } S \in Ss : Ss = \{S\}$
  □ OTHER $\to$
      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 sets of predecessors of each message is a strict majority of the predecessors 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)

$$Max(X,\ Leq(\_,\ \_)) \triangleq$$
$$\quad \text{CHOOSE } m \in X : \forall\, x \in X : Leq(x,\ m)$$

$$\text{RECURSIVE } ConsistentChain(\_)$$
$$ConsistentChain(M) \triangleq$$
$$\quad \text{IF } M = \{\}$$
$$\quad \text{THEN FALSE}$$
$$\quad \text{ELSE LET } r \triangleq Max(\{m.round : m \in M\},\ \leq) \text{IN}$$
$$\quad\quad \vee\ r = 0$$
$$\quad\quad \vee\ \text{LET } Tip \triangleq \{m \in M : m.round = r\}$$
$$\quad\quad\quad\quad Pred \triangleq \{m \in M : m.round = r - 1\}$$
$$\quad\quad\quad \text{IN} \quad \wedge\ \forall\, m \in Tip :$$
$$\quad\quad\quad\quad\quad\quad \wedge\ Pred \subseteq m.coffer$$
$$\quad\quad\quad\quad\quad\quad \wedge\ 2 * Cardinality(Pred) > Cardinality(m.coffer)$$
$$\quad\quad\quad\quad\quad \wedge\ ConsistentChain(M \setminus Tip)$$

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

$$HeaviestConsistentChain(M) \triangleq$$
$$\quad \text{LET } r \triangleq Max(\{m.round : m \in M\},\ \leq)$$
$$\quad\quad Cs \triangleq \{C \in \text{SUBSET } M : ConsistentChain(C)\}$$
$$\quad \text{IN}$$
$$\quad\quad \text{IF } Cs = \{\} \text{ THEN } \{\}$$
$$\quad\quad \text{ELSE } Max(Cs, \text{ LAMBDA } C1,\ C2 : Cardinality(C1) \leq Cardinality(C2))$$

Now we specify the algorithm

--algorithm *Algo* **{**
variables
$messages = \{\}$ **;**
$tick = 0$ **;**
$pendingMessage = [p \in P \mapsto \langle\rangle]$ **;**
$doneTick = [p \in P \mapsto -1]$ **;**
$messageCount = 0$ **;** used to generate unique message *IDs*
define {
$currentRound(t) \triangleq tick \div t$
$wellBehavedMessages \triangleq \{m \in messages : m.sender \in P \setminus B\}$
possible sets of messages received by a well-behaved process:
$receivedMsgsSets \triangleq \text{LET } msgs \triangleq \{m \in messages : m.round < tick\} \text{IN}$
$\{wellBehavedMessages \cup byzMsgs :$
$byzMsgs \in \text{SUBSET } (msgs \setminus wellBehavedMessages)\}$

```
    }
    macro sendMessage( m ) {
        messages := messages ∪ {m}
    }
    process ( clock ∈ {"clock"} ) {
tick:   while ( TRUE ) {
            wait for all processes to take their step before incrementing the tick
            await ∀ p ∈ P : doneTick[p] = tick ;
            tick := tick + 1 ;
        }
    }
    process ( proc ∈ P \ B )   a well-behaved process
    {
l1:     while ( TRUE ) {
            await tick > doneTick[self] ;
            if ( tick%tWB = 0 ) {
                Start the VDF computation for the next message:
                with ( msgs ∈ receivedMsgsSets )
                with ( predMsgs = {m ∈ msgs : m.round = currentRound(tWB) − 1} ) {
                    TODO: filter messages
                    pendingMessage[self] := [
                        sender ↦ self,
                        id ↦ messageCount + 1,
                        round ↦ currentRound(tWB),
                        coffer ↦ {m.id : m ∈ predMsgs}] ;
                    messageCount := messageCount + 1 ;
                }
            }
            else
            if ( tick%tWB = tWB − 1 )
                it's tWB − 1 because we want the message to be received by tick tWB
                sendMessage(pendingMessage[self]) ;
            else skip ;   busy computing the VDF
            doneTick[self] := tick ;
        }
    }
    process ( byz ∈ B )   a malicious process
    {
lb1:    while ( TRUE ) {
            await tick > doneTick[self] ;
            if ( tick%tAdv = 0 ) {
                Start the VDF computation for the next message:
                with ( msgs ∈ receivedMsgsSets )
                with ( rnd ∈ 0 .. currentRound(tAdv) )   can forge messages from any previous round
                with ( predMsgs = {m ∈ msgs : m.round = rnd − 1} ) {
```

```
                    pendingMessage[self] := [
                        sender ↦ self,
                        id ↦ messageCount + 1,
                        round ↦ rnd,
                        coffer ↦ {m.id : m ∈ predMsgs}] ;
                    messageCount := messageCount + 1 ;
                }
            }
            else
            if ( tick%tAdv = tAdv − 1 )
                sendMessage(pendingMessage[self]) ;
            else skip ;  busy computing the VDF
            doneTick[self] := tick ;
        } ;
    }
}
```

BEGIN TRANSLATION ($chksum(pcal) =$ "5b9b3a07" $\wedge\ chksum(tla) =$ "9de7770e")

Label tick of process clock at line 112 col 9 changed to *tick_*

VARIABLES $messages$, $tick$, $pendingMessage$, $doneTick$, $messageCount$

define statement

$currentRound(t) \triangleq tick \div t$

$wellBehavedMessages \triangleq \{m \in messages : m.sender \in P \setminus B\}$

$receivedMsgsSets \triangleq$ LET $msgs \triangleq \{m \in messages : m.round < tick\}$IN
$\{wellBehavedMessages \cup byzMsgs :$
$byzMsgs \in$ SUBSET $(msgs \setminus wellBehavedMessages)\}$

$vars \triangleq \langle messages, tick, pendingMessage, doneTick, messageCount\rangle$

$ProcSet \triangleq (\{$"clock"$\}) \cup (P \setminus B) \cup (B)$

$Init \triangleq$ Global variables
$\wedge\ messages = \{\}$
$\wedge\ tick = 0$
$\wedge\ pendingMessage = [p \in P \mapsto \langle\rangle]$
$\wedge\ doneTick = [p \in P \mapsto -1]$
$\wedge\ messageCount = 0$

$clock(self) \triangleq \wedge \forall\, p \in P : doneTick[p] = tick$
$\wedge\ tick' = tick + 1$
$\wedge$ UNCHANGED $\langle messages, pendingMessage, doneTick,$
$messageCount\rangle$

$proc(self) \triangleq \wedge\ tick > doneTick[self]$
$\wedge$ IF $tick\%tWB = 0$

```
                THEN ∧ ∃ msgs ∈ receivedMsgsSets :
                        LET predMsgs ≜ {m ∈ msgs : m.round = currentRound(tWB) − 1} IN
                          ∧ pendingMessage′ = [pendingMessage EXCEPT ![self] =
                                                sender ↦ self,
                                                id ↦ messageCo
                                                round ↦ current
                                                coffer ↦ {m.id :
                             ∧ messageCount′ = messageCount + 1
                     ∧ UNCHANGED messages
                ELSE ∧ IF tick%tWB = tWB − 1
                          THEN ∧ messages′ = (messages ∪ {(pendingMessage[self])})
                          ELSE ∧ TRUE
                               ∧ UNCHANGED messages
                     ∧ UNCHANGED ⟨pendingMessage, messageCount⟩
           ∧ doneTick′ = [doneTick EXCEPT ![self] = tick]
           ∧ tick′ = tick

byz(self) ≜ ∧ tick > doneTick[self]
            ∧ IF tick%tAdv = 0
                THEN ∧ ∃ msgs ∈ receivedMsgsSets :
                          ∃ rnd ∈ 0 .. currentRound(tAdv) :
                            LET predMsgs ≜ {m ∈ msgs : m.round = rnd − 1} IN
                              ∧ pendingMessage′ = [pendingMessage EXCEPT ![self] =
                                                    sender ↦ self,
                                                    id ↦ messageC
                                                    round ↦ rnd,
                                                    coffer ↦ {m.id
                                 ∧ messageCount′ = messageCount + 1
                         ∧ UNCHANGED messages
                ELSE ∧ IF tick%tAdv = tAdv − 1
                          THEN ∧ messages′ = (messages ∪ {(pendingMessage[self])})
                          ELSE ∧ TRUE
                               ∧ UNCHANGED messages
                     ∧ UNCHANGED ⟨pendingMessage, messageCount⟩
           ∧ doneTick′ = [doneTick EXCEPT ![self] = tick]
           ∧ tick′ = tick

Next ≜ (∃ self ∈ {"clock"} : clock(self))
       ∨ (∃ self ∈ P \ B : proc(self))
       ∨ (∃ self ∈ B : byz(self))

Spec ≜ Init ∧ □[Next]_vars

END TRANSLATION

TypeOK ≜
```

$\land$ $messages \in \text{SUBSET } Message$
$\land$ $pendingMessage \in [P \to Message \cup \{\langle\rangle\}]$
$\land$ $tick \in Tick$
$\land$ $doneTick \in [P \to Tick \cup \{-1\}]$

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

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.

$Safety \triangleq \forall\, m \in messages : m.round > 0 \land m.sender \notin B \Rightarrow$
$\land$ $\forall\, m2 \in wellBehavedMessages : m2.round = m.round \Rightarrow m2.id \in m.coffer$
$\land$ $2 * Cardinality(\{m2 \in wellBehavedMessages : m2.round = m.round\}) > Cardinality(m.coffer)$

A basic well-formedness property:

$Inv1 \triangleq \forall\, m \in messages : \forall\, id \in m.coffer :$
$\land$ $\exists\, m2 \in messages : m2.id = id$
$\land$ $messageWithID(id).round = m.round - 1$