MODULE *Hermes*

EXTENDS *Integers*

CONSTANTS $NODES$,
$MAX\_VERSION$

VARIABLES $msgs$,
$nodeTS$,
$nodeState$,
$nodeRcvedAcks$,
$nodeLastWriter$,
$nodeLastWriteTS$,
$aliveNodes$

The consistent invariant: all alive nodes in valid state should have the same value / *TS*

$$
\begin{array}{l}
HConsistent \triangleq \\
\quad \forall\, k,\, s \in aliveNodes : \begin{array}[t]{l} \lor\ nodeState[k] \neq \text{“valid”} \\ \lor\ nodeState[s] \neq \text{“valid”} \\ \lor\ nodeTS[k] = nodeTS[s] \end{array}
\end{array}
$$

HMessage $\triangleq$ Messages exchanged by the Protocol

$$
\begin{array}{l}
\quad [type : \{\text{“INV”}, \text{“ACK”}\},\ \begin{array}[t]{ll} sender & : NODES, \\ version & : 0 \,..\, MAX\_VERSION, \\ tieBreaker & : NODES] \end{array} \\
\qquad \cup \\
\quad [type : \{\text{“VAL”}\},\ \begin{array}[t]{ll} version & : 0 \,..\, MAX\_VERSION, \\ tieBreaker & : NODES] \end{array}
\end{array}
$$

HTypeOK $\triangleq$ The type correctness invariant

$$
\begin{array}{lll}
\land & msgs & \subseteq HMessage \\
\land & aliveNodes & \subseteq NODES \\
\land & \multicolumn{2}{l}{\forall\, n \in NODES : nodeRcvedAcks[n] \subseteq (NODES \setminus \{n\})} \\
\land & nodeLastWriter & \in [NODES \rightarrow NODES] \\
\land & nodeLastWriteTS & \in [NODES \rightarrow [\begin{array}[t]{ll} version & : 0 \,..\, MAX\_VERSION, \\ tieBreaker & : NODES \quad]] \end{array} \\
\land & nodeTS & \in [NODES \rightarrow [\begin{array}[t]{ll} version & : 0 \,..\, MAX\_VERSION, \\ tieBreaker & : NODES \quad]] \end{array} \\
\land & nodeState & \in [NODES \rightarrow \{\begin{array}[t]{l} \text{“valid”}, \text{“invalid”}, \text{“invalid\_write”}, \\ \text{“write”}, \text{“replay”}\}] \end{array}
\end{array}
$$

HInit $\triangleq$ The initial predicate

$$
\begin{array}{lll}
\land & msgs & = \{\} \\
\land & aliveNodes & = NODES \\
\land & nodeRcvedAcks & = [n \in NODES \mapsto \{\}] \\
\land & nodeState & = [n \in NODES \mapsto \text{“valid”}]
\end{array}
$$

```
    $\land\ nodeLastWriter \quad = [n \in NODES \mapsto \text{CHOOSE } k \in NODES :$
                                        $\forall m \in NODES : k \leq m]$
    $\land\ nodeTS \quad = [n \in NODES \mapsto [version \mapsto 0,$
                                         $tieBreaker \mapsto$
                                           $\text{CHOOSE } k \in NODES :$
                                             $\forall m \in NODES : k \leq m]]$
    $\land\ nodeLastWriteTS = [n \in NODES \mapsto [version \mapsto 0,$
                                            $tieBreaker \mapsto$
                                              $\text{CHOOSE } k \in NODES :$
                                                $\forall m \in NODES : k \leq m]]$
```

A buffer maintaining all network messages. Messages are only appended to this variable (not removed once delivered) intentionally to check protocols tolerance in dublicates and reorderings

$send(m) \triangleq msgs' = msgs \cup \{m\}$

Check if all acknowledgments for a write have been received

$receivedAllAcks(n) \triangleq nodeRcvedAcks[n] = NODES \setminus \{n\}$

```
$equalTS(v1, tb1, v2, tb2) \triangleq$    Timestamp equality
    $\land\ v1 = v2$
    $\land\ tb1 = tb2$

$greaterTS(v1, tb1, v2, tb2) \triangleq$    Timestamp comparison
    $\lor\ v1 > v2$
    $\lor\ \land\ v1 = v2$
       $\land\ tb1 > tb2$

$isAlive(n) \triangleq n \in aliveNodes$

$nodeFailure(n) \triangleq$    Emulate a node failure
  Make sure that there are atleast 3 alive nodes before killing a node
    $\land\ \exists k, m \in aliveNodes : \land\ k \neq n$
                                  $\land\ m \neq n$
                                  $\land\ m \neq k$
    $\land\ aliveNodes' = aliveNodes \setminus \{n\}$
    $\land\ \text{UNCHANGED } \langle msgs, nodeState, nodeTS, nodeLastWriter,$
                      $nodeLastWriteTS, nodeRcvedAcks\rangle$
```

```
$HRead(n) \triangleq$    Execute a read
    $\land\ nodeState[n] = \text{"valid"}$
    $\land\ \text{UNCHANGED } \langle msgs, nodeTS, nodeState, nodeLastWriter,$
                      $aliveNodes, nodeLastWriteTS, nodeRcvedAcks\rangle$

$HWrite(n) \triangleq$    Execute a write
    $\land\ nodeState[n] \quad \in \{\text{"valid"}, \text{"invalid"}\}$
    $\land\ nodeTS[n].version < MAX\_VERSION$
    $\land\ nodeRcvedAcks' \quad = [nodeRcvedAcks \ \text{EXCEPT } ![n] = \{\}]$
    ∧ nodeLastWriter′    = [nodeLastWriter  EXCEPT ![n]  = n]
    ∧ nodeState′         = [nodeState      EXCEPT ![n]   = "write"]
    ∧ nodeTS′            = [nodeTS          EXCEPT ![n].version    =
                                                     nodeTS[n].version + 1,
                                                 ![n].tieBreaker = n]
    ∧ nodeLastWriteTS′ = [nodeLastWriteTS EXCEPT ![n].version =
                                                     nodeTS[n].version + 1,
                                                 ![n].tieBreaker = n]
    ∧ send([type          ↦ "INV",
            sender        ↦ n,
            version       ↦ nodeTS[n].version + 1,
            tieBreaker    ↦ n])
    ∧ UNCHANGED ⟨aliveNodes⟩

HReplayWrite(n) ≜   Execute a write-replay
    ∧ nodeState[n] = "invalid"
    ∧ ¬isAlive(nodeLastWriter[n])
    ∧ nodeLastWriter′   = [nodeLastWriter   EXCEPT ![n]  = n]
    ∧ nodeState′        = [nodeState        EXCEPT ![n]   = "replay"]
    ∧ nodeRcvedAcks′    = [nodeRcvedAcks    EXCEPT ![n]  = {}]
    ∧ nodeLastWriteTS′ = [nodeLastWriteTS  EXCEPT ![n] = nodeTS[n]]
    ∧ send([type          ↦ "INV",
            sender        ↦ n,
            version       ↦ nodeTS[n].version,
            tieBreaker ↦ nodeTS[n].tieBreaker])
    ∧ UNCHANGED ⟨nodeTS, aliveNodes⟩
```

```
HRcvAck(n) ≜      Process a received acknowledment
    ∃ m ∈ msgs :
        ∧ m.type = "ACK"
        ∧ m.sender ≠ n
        ∧ m.sender ∉ nodeRcvedAcks[n]
        ∧ equalTS(m.version,
                  m.tieBreaker,
                  nodeLastWriteTS[n].version,
                  nodeLastWriteTS[n].tieBreaker)
        ∧ nodeState[n] ∈ {"write", "invalid_write", "replay"}
        ∧ nodeRcvedAcks′ = [nodeRcvedAcks EXCEPT ![n] =
                                         nodeRcvedAcks[n] ∪ {m.sender}]
        ∧ UNCHANGED ⟨msgs, nodeLastWriter, nodeLastWriteTS,
                     aliveNodes, nodeTS, nodeState⟩

HSendVals(n) ≜    Send validations once received acknowledments from all alive nodes
    ∧ nodeState[n] ∈ {"write", "replay"}
    ∧ receivedAllAcks(n)
```

```
   ∧ nodeState'           = [nodeState EXCEPT ![n] = "valid"]
   ∧ send([type           ↦ "VAL",
            version       ↦ nodeTS[n].version,
            tieBreaker    ↦ nodeTS[n].tieBreaker])
   ∧ UNCHANGED ⟨nodeTS, nodeLastWriter, nodeLastWriteTS,
                  aliveNodes, nodeRcvedAcks⟩

HCoordinatorActions(n) ≜      Actions of a read/write coordinator
   ∨ HRead(n)
   ∨ HReplayWrite(n)
   ∨ HWrite(n)
   ∨ HRcvAck(n)
   ∨ HSendVals(n)
├──────────────────────────────────────────────────────────────────┤
HRcvInv(n) ≜    Process a received invalidation
   ∃ m ∈ msgs :
      ∧ m.type = "INV"
      ∧ m.sender ≠ n
      always acknowledge a received invalidation (irrelevant to the timestamp)
      ∧ send([type          ↦ "ACK",
               sender       ↦ n,
               version      ↦ m.version,
               tieBreaker   ↦ m.tieBreaker])
      ∧ ∨ ∧ greaterTS(m.version,
                       m.tieBreaker,
                       nodeTS[n].version,
                       nodeTS[n].tieBreaker)
          ∧ nodeLastWriter' = [nodeLastWriter EXCEPT ![n] = m.sender]
          ∧ nodeTS' = [nodeTS EXCEPT ![n].version    = m.version,
                                     ![n].tieBreaker = m.tieBreaker]
          ∧ ∨ ∧ nodeState[n] ∈ {"valid", "invalid", "replay"}
              ∧ nodeState' = [nodeState EXCEPT ![n] = "invalid"]
            ∨ ∧ nodeState[n] ∈ {"write", "invalid_write"}
              ∧ nodeState' = [nodeState EXCEPT ![n] = "invalid_write"]
        ∨ ∧ ¬greaterTS(m.version,
                        m.tieBreaker,
                        nodeTS[n].version,
                        nodeTS[n].tieBreaker)
          ∧ UNCHANGED ⟨nodeState, nodeTS, nodeLastWriter⟩
      ∧ UNCHANGED ⟨nodeLastWriteTS, aliveNodes, nodeRcvedAcks⟩

HRcvVal(n) ≜    Process a received validation
   ∃ m ∈ msgs :
      ∧ nodeState[n] ≠ "valid"
      ∧ m.type = "VAL"
```

```
        $\land\ equalTS(m.version,$
                        $m.tieBreaker,$
                        $nodeTS[n].version,$
                        $nodeTS[n].tieBreaker)$
        $\land\ nodeState' = [nodeState\ \textsc{except}\ ![n] = \text{“valid”}]$
        $\land\ \textsc{unchanged}\ \langle msgs,\ nodeTS,\ nodeLastWriter,\ nodeLastWriteTS,$
                            $aliveNodes,\ nodeRcvedAcks \rangle$

$HFollowerActions(n) \triangleq$    Actions of a write follower
    $\lor\ HRcvInv(n)$
    $\lor\ HRcvVal(n)$

$HNext \triangleq$  Modeling *Hermes* protocol (Coordinator and Follower actions while emulating failures)
    $\exists\, n \in aliveNodes :$
        $\lor\ HFollowerActions(n)$
        $\lor\ HCoordinatorActions(n)$
        $\lor\ nodeFailure(n)$  emulate node failures
```