MODULE *HermesRMWs*

EXTENDS *Hermes*

VARIABLES *Rmsgs*, *nodeFlagRMW*, *committedRMWs*, *committedWrites*

$HRMessage \triangleq$ Invalidation *msgs* exchanged by the *Hermes* Protocol w/ *RMWs*

$[type : \{\text{“RINV”}\},$ $flagRMW : \{0, 1\},$ *RMW* change
$epochID : 0 \,..\, (Cardinality(H\_NODES) - 1),$
$sender : H\_NODES,$
$version : 0 \,..\, H\_MAX\_VERSION,$
$tieBreaker : H\_NODES]$

$HRts \triangleq [version : 0 \,..\, H\_MAX\_VERSION,$
$tieBreaker : H\_NODES]$

$HRTypeOK \triangleq$ The type correctness invariant

$\land\ HTypeOK$
$\land\ Rmsgs \subseteq HRMessage$
$\land\ nodeFlagRMW \in [H\_NODES \to \{0, 1\}]$
$\land\ committedRMWs \subseteq HRts$
$\land\ committedWrites \subseteq HRts$

$HRSemanticsRMW \triangleq$ The invariant that an we cannot have two operations committed with same versions (*i.e.*, that read the same value unless they are both writes)

$\land \forall\, x \in committedRMWs :$
$\quad \forall\, y \in committedWrites : \land\, x.version \neq y.version$
$\qquad\qquad\qquad\qquad\qquad\ \land\, x.version \neq y.version - 1$
$\land \forall\, x,\, y \in committedRMWs : \lor\, x.version \neq y.version$
$\qquad\qquad\qquad\qquad\qquad\ \lor\, x.tieBreaker = y.tieBreaker$

$HRInit \triangleq$ The initial predicate

$\land\ HInit$
$\land\ Rmsgs = \{\}$
$\land\ committedRMWs = \{\}$
$\land\ committedWrites = \{\}$
$\land\ nodeFlagRMW = [n \in H\_NODES \mapsto 0]$ *RMW* change

---

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

$HRsend(m) \triangleq Rmsgs' = Rmsgs \cup \{m\}$

$smallerTS(v1,\, tb1,\, v2,\, tb2) \triangleq$
$\land \neg equalTS(v1,\, tb1,\, v2,\, tb2)$

$\land \neg greaterTS(v1,\ tb1,\ v2,\ tb2)$

$hr\_upd\_nothing \triangleq$
$\land$ UNCHANGED $\langle nodeFlagRMW,\ Rmsgs,\ committedRMWs,\ committedWrites\rangle$

$hr\_completeWrite(ver,\ tieB) \triangleq$
$\land\ committedWrites' = committedWrites \cup \{[version \mapsto ver,\ tieBreaker \mapsto tieB]\}$
$\land$ UNCHANGED $\langle Rmsgs,\ nodeFlagRMW,\ committedRMWs\rangle$

$hr\_completeRMW(ver,\ tieB) \triangleq$
$\land\ committedRMWs' = committedRMWs \cup \{[version \mapsto ver,\ tieBreaker \mapsto tieB]\}$
$\land$ UNCHANGED $\langle Rmsgs,\ nodeFlagRMW,\ committedWrites\rangle$

Helper functions

$hr\_upd\_state(n,\ newVersion,\ newTieBreaker,\ newState,\ newAcks,\ flagRMW) \triangleq$
$\land\ nodeFlagRMW' = [nodeFlagRMW \text{ EXCEPT } ![n] = flagRMW]$ *RMW* change
$\land\ h\_upd\_state(n,\ newVersion,\ newTieBreaker,\ newState,\ newAcks)$

$hr\_send\_inv(n,\ newVersion,\ newTieBreaker,\ flagRMW) \triangleq$
$\land\ HRsend([type \mapsto$ "RINV",
$epochID \mapsto epochID,$ we always use the latest *epochID*
$flagRMW \mapsto flagRMW,$ *RMW* change
$sender \mapsto n,$
$version \mapsto newVersion,$
$tieBreaker \mapsto newTieBreaker])$

$hr\_upd\_actions(n,\ newVersion,\ newTieBreaker,\ newState,\ newAcks,\ flagRMW) \triangleq$ Execute a write
$\land\ hr\_upd\_state(n,\ newVersion,\ newTieBreaker,\ newState,\ newAcks,\ flagRMW)$
$\land\ hr\_send\_inv(n,\ newVersion,\ newTieBreaker,\ flagRMW)$
$\land$ UNCHANGED $\langle aliveNodes,\ epochID,\ msgs,\ committedRMWs,\ committedWrites\rangle$

$hr\_upd\_replay\_actions(n,\ acks) \triangleq$ Apply a write-replay using same *TS* (version, Tie Breaker)
and either reset *acks* or keep already gathered *acks*
$\land\ hr\_upd\_actions(n,\ nodeTS[n].version,\ nodeTS[n].tieBreaker,$ "replay", $acks,\ nodeFlagRMW[n])$

Coordinator functions

$HRWrite(n) \triangleq$ Execute a write
$\land\ nodeState[n] \in \{$"valid", "invalid"$\}$
writes in invalid state are also supported as an optimization
$\land\ nodeState[n] =$ "valid"
$\land\ nodeTS[n].version + 2 \leq H\_MAX\_VERSION$ condition to bound execution
$\land\ hr\_upd\_actions(n,\ nodeTS[n].version + 2,\ n,$ "write", $\{\},\ 0)$

$HRRMW(n) \triangleq$ Execute an *RMW*
- $\wedge$ $nodeState[n] = \text{“valid”}$
- $\wedge$ $nodeTS[n].version + 1 \leq H\_MAX\_VERSION$ condition to bound execution
- $\wedge$ $hr\_upd\_actions(n,\ nodeTS[n].version + 1,\ n,\ \text{“write”},\ \{\},\ 1)$

$HRWriteReplay(n) \triangleq$ Execute a write-replay
- $\wedge$ $nodeState[n] \in \{\text{“write”},\ \text{“replay”}\}$
- $\wedge$ $nodeWriteEpochID[n] < epochID$
- $\wedge$ $\neg receivedAllAcks(n)$ optimization to not replay when we have gathered *acks* from all alive
- $\wedge$ $nodeFlagRMW[n] = 0$
- $\wedge$ $hr\_upd\_replay\_actions(n,\ nodeRcvedAcks[n])$

$HRRMWReplay(n) \triangleq$ Execute an RMW-replay
- $\wedge$ $nodeState[n] \in \{\text{“write”},\ \text{“replay”}\}$
- $\wedge$ $nodeWriteEpochID[n] < epochID$
- $\wedge$ $\neg receivedAllAcks(n)$ optimization to not replay when we have gathered *acks* from all alive
- $\wedge$ $nodeFlagRMW[n] = 1$
- $\wedge$ $hr\_upd\_replay\_actions(n,\ \{\})$

Keep the *HRead*, *HRcvAck* and *HSendVals* the same as *Hermes* w/o *RMWs*

$HRRead(n) \triangleq$
- $\wedge$ $HRead(n)$
- $\wedge$ $hr\_upd\_nothing$

$HRRcvAck(n) \triangleq$
- $\wedge$ $HRcvAck(n)$
- $\wedge$ $hr\_upd\_nothing$

$HRSendValsRMW(n) \triangleq$
- $\wedge$ $nodeFlagRMW[n] = 1$
- $\wedge$ $HSendVals(n)$
- $\wedge$ $hr\_completeRMW(nodeTS[n].version,\ nodeTS[n].tieBreaker)$

$HRSendValsWrite(n) \triangleq$
- $\wedge$ $nodeFlagRMW[n] = 0$
- $\wedge$ $HSendVals(n)$
- $\wedge$ $hr\_completeWrite(nodeTS[n].version,\ nodeTS[n].tieBreaker)$

$HRSendVals(n) \triangleq$
- $\vee$ $HRSendValsRMW(n)$
- $\vee$ $HRSendValsWrite(n)$

$HRCoordinatorActions(n) \triangleq$ Actions of a read/write/*RMW* coordinator
- $\vee$ $HRRead(n)$
- $\vee$ $HRRMWReplay(n)$
- $\vee$ $HRWriteReplay(n)$
- $\vee$ $HRWrite(n)$
- $\vee$ $HRRMW(n)$

$\lor\ HRRcvAck(n)$
$\lor\ HRSendVals(n)$

Follower functions

$HRRcvWriteInv(n) \triangleq$ Process a received invalidation for a write

$\exists\, m \in Rmsgs :$
$\land\ m.type = \text{“RINV”}$
$\land\ m.epochID\ = epochID$
$\land\ m.sender \neq n$
$\land\ m.flagRMW = 0$ *RMW* change
always acknowledge a received invalidation (irrelevant to the timestamp)
$\land\ h\_send\_inv\_or\_ack(n,\ m.version,\ m.tieBreaker,\ \text{“ACK”})$
$\land\ \lor\ \land\ greaterTS(m.version,$
$m.tieBreaker,$
$nodeTS[n].version,$
$nodeTS[n].tieBreaker)$
$\land\ nodeFlagRMW' = [nodeFlagRMW\ \text{EXCEPT}\ ![n] = m.flagRMW]$ *RMW* change
$\land\ nodeLastWriter' = [nodeLastWriter\ \text{EXCEPT}\ ![n] = m.sender]$
$\land\ nodeTS' = [nodeTS\ \text{EXCEPT}\ ![n].version\ = m.version,$
$![n].tieBreaker = m.tieBreaker]$
$\land\ \lor\ \land\ nodeState[n] \in \{\text{“valid”},\ \text{“invalid”},\ \text{“replay”}\}$
$\land\ nodeState' = [nodeState\ \text{EXCEPT}\ ![n] = \text{“invalid”}]$
$\lor\ \land\ nodeState[n] \in \{\text{“write”},\ \text{“invalid\_write”}\}$
$\land\ nodeFlagRMW[n] = 0$ *RMW* change
$\land\ nodeState' = [nodeState\ \text{EXCEPT}\ ![n] = \text{“invalid\_write”}]$
$\lor\ \land\ nodeState[n] = \text{“write”}$ *RMW* change
$\land\ nodeFlagRMW[n] = 1$ *RMW* change
$\land\ nodeState' = [nodeState\ \text{EXCEPT}\ ![n] = \text{“invalid”}]$ *RMW* change
$\lor\ \land\ \neg greaterTS(m.version,\ m.tieBreaker,$
$nodeTS[n].version,\ nodeTS[n].tieBreaker)$
$\land\ \text{UNCHANGED}\ \langle nodeState,\ nodeTS,\ nodeLastWriter,\ nodeFlagRMW\rangle$
$\land\ \text{UNCHANGED}\ \langle nodeLastWriteTS,\ aliveNodes,\ nodeRcvedAcks,\ Rmsgs,$
$epochID,\ nodeWriteEpochID,\ committedRMWs,\ committedWrites\rangle$

$HRRcvRMWInv(n) \triangleq$ Process a received invalidation for a write

$\exists\, m \in Rmsgs :$
$\land\ m.type = \text{“RINV”}$
$\land\ m.epochID\ = epochID$
$\land\ m.sender \neq n$
$\land\ m.flagRMW = 1$
$\land\ \lor\ \land\ greaterTS(m.version,$
$m.tieBreaker,$
$nodeTS[n].version,$

```
                    nodeTS[n].tieBreaker)
        ∧ nodeFlagRMW' = [nodeFlagRMW EXCEPT ![n] = m.flagRMW]   RMW change
        ∧ nodeLastWriter' = [nodeLastWriter EXCEPT ![n] = m.sender]
        ∧ nodeTS' = [nodeTS EXCEPT ![n].version     = m.version,
                                   ![n].tieBreaker = m.tieBreaker]
        acknowledge a received invalidation (w/ greater timestamp)
        ∧ h_send_inv_or_ack(n, m.version, m.tieBreaker, "ACK")
        ∧ ∨ ∧ nodeState[n] ∈ {"valid", "invalid", "replay"}
            ∧ nodeState' = [nodeState EXCEPT ![n] = "invalid"]
          ∨ ∧ nodeState[n] ∈ {"write", "invalid_write"}
            ∧ nodeFlagRMW[n] = 0        RMW change
            ∧ nodeState' = [nodeState EXCEPT ![n] = "invalid_write"]
          ∨ ∧ nodeState[n] = "write"    RMW change
            ∧ nodeFlagRMW[n] = 1        RMW change
            ∧ nodeState' = [nodeState EXCEPT ![n] = "invalid"]      RMW change
        ∧ UNCHANGED ⟨Rmsgs⟩
    ∨ ∧ equalTS(m.version, m.tieBreaker,
                nodeTS[n].version, nodeTS[n].tieBreaker)
      acknowledge a received invalidation (w/ equal timestamp)
      ∧ h_send_inv_or_ack(n, m.version, m.tieBreaker, "ACK")
      ∧ UNCHANGED ⟨nodeState, nodeTS, nodeLastWriter, nodeFlagRMW, Rmsgs⟩
    ∨ ∧ smallerTS(m.version, m.tieBreaker,
                  nodeTS[n].version, nodeTS[n].tieBreaker)
      ∧ hr_send_inv(n, nodeTS[n].version, nodeTS[n].tieBreaker, nodeFlagRMW[n])
      ∧ UNCHANGED ⟨nodeState, nodeTS, nodeLastWriter, nodeFlagRMW, msgs⟩
∧ UNCHANGED ⟨nodeLastWriteTS, aliveNodes, nodeRcvedAcks, epochID,
             nodeWriteEpochID, committedRMWs, committedWrites⟩

 Keep the HRcvVals the same as Hermes w/o RMWs
HRRcvVal(n) ≜
    ∧ HRcvVal(n)
    ∧ hr_upd_nothing

HRFollowerWriteReplay(n) ≜   Execute a write-replay when coordinator failed
    ∧  nodeState[n] = "invalid"
    ∧  ¬isAlive(nodeLastWriter[n])
    ∧  hr_upd_replay_actions(n, {})

HRFollowerActions(n) ≜   Actions of a write follower
    ∨ HRFollowerWriteReplay(n)
    ∨ HRRcvWriteInv(n)
    ∨ HRRcvRMWInv(n)
    ∨ HRRcvVal(n)
```

$HRNodeFailure(n) \triangleq$
$\quad \land\ nodeFailure(n)$
$\quad \land\ hr\_upd\_nothing$

$HRNext \triangleq$ Modeling *Hermes* protocol (Coordinator and Follower actions while emulating failures)
$\quad \exists\, n \in aliveNodes :$
$\qquad \lor\ HRFollowerActions(n)$
$\qquad \lor\ HRCoordinatorActions(n)$
$\qquad \lor\ HRNodeFailure(n)$ emulate node failures