MODULE *HermesRMWs*

EXTENDS *Hermes*

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

all *Hermes*(+ environment, + *RMW*) variables

$hrvars \triangleq \langle msgs,\ nodeTS,\ nodeState,\ nodeRcvedAcks,\ nodeLastWriter,\ nodeLastWriteTS,\ nodeWriteEpochID,\ aliveNodes,\ epochID,\ Rmsgs,\ nodeFlagRMW,\ committedRMWs,\ committedWrites\rangle$

$HRMessage \triangleq$ 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

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

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

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

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

Helper functions

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

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

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

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

Coordinator functions

$HRWrite(n) \triangleq$ Execute a write
$\quad \wedge nodeState[n] \in \{$"valid", "invalid"$\}$
writes in invalid state are also supported as an optimization
$\quad \wedge\ nodeState[n] =$ "valid"
$\quad \wedge\ nodeTS[n].version + 2 \leq H\_MAX\_VERSION$ Only to configurably terminate the model checking

$\land$ $hr\_actions\_for\_upd(n,\ nodeTS[n].version + 2,\ n,\ \text{“write”},\ \{\},\ 0)$

$HRRMW(n) \triangleq$ Execute an *RMW*
$\land$ $nodeState[n] = \text{“valid”}$
$\land$ $nodeTS[n].version + 1 \leq H\_MAX\_VERSION$ Only to configurably terminate the model checking
$\land$ $hr\_actions\_for\_upd(n,\ nodeTS[n].version + 1,\ n,\ \text{“write”},\ \{\},\ 1)$

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

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

Keep the *HRead*, *HRcvAck* and *HSendVals* the same as *Hermes* w/o *RMWs*
$HRRead(n) \triangleq$
$\land$ $HRead(n)$
$\land$ $hr\_upd\_nothing$

$HRRcvAck(n) \triangleq$
$\land$ $HRcvAck(n)$
$\land$ $hr\_upd\_nothing$

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

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

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

$\lor\ HRSendValsRMW(n)$
$\lor\ HRSendValsWrite(n)$

Follower functions

$hr\_upd\_state\_greater\_inv(n) \triangleq$
IF $nodeState[n] \in \{\text{“valid”}, \text{“invalid”}, \text{“replay”}\}$
THEN
$nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{“invalid”}]$
ELSE IF $nodeState[n] \in \{\text{“write”}, \text{“invalid\_write”}\} \land nodeFlagRMW[n] = 0$
THEN
$nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{“invalid\_write”}]$
ELSE $nodeState[n] \in \{\text{“write”}\} \land nodeFlagRMW[n] = 1$
$nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{“invalid”}]$

$HRRcvWriteInv(n) \triangleq$ 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)
$\land\ h\_send\_inv\_or\_ack(n,\ m.version,\ m.tieBreaker,\ \text{“ACK”})$
$\land$ IF $greaterTS(m.version,\ m.tieBreaker,$
$nodeTS[n].version,\ nodeTS[n].tieBreaker)$
THEN
$\land\ nodeLastWriter' = [nodeLastWriter \text{ EXCEPT } ![n] = m.sender]$
$\land\ nodeFlagRMW' = [nodeFlagRMW \text{ EXCEPT } ![n] = m.flagRMW]$ *RMW* change
$\land\ nodeTS' = [nodeTS \text{ EXCEPT } ![n].version = m.version,$
$![n].tieBreaker = m.tieBreaker]$
$\land\ hr\_upd\_state\_greater\_inv(n)$
ELSE
$\land$ UNCHANGED $\langle nodeState,\ nodeTS,\ nodeLastWriter,\ nodeFlagRMW\rangle$
$\land$ UNCHANGED $\langle nodeLastWriteTS,\ aliveNodes,\ nodeRcvedAcks,\ Rmsgs,$
$epochID,\ nodeWriteEpochID,\ committedRMWs,\ committedWrites\rangle$

$HRRcvRMWInv(n) \triangleq$ Process a received invalidation for a write
$\exists\, m \in Rmsgs :$
$\land\ m.type = \text{“RINV”}$
$\land\ m.epochID = epochID$
$\land\ m.sender \neq n$
$\land\ m.flagRMW = 1$
$\land$ IF $greaterTS(m.version,\ m.tieBreaker,$
$nodeTS[n].version,\ nodeTS[n].tieBreaker)$
THEN

$\land\ nodeLastWriter' = [nodeLastWriter \text{ EXCEPT } ![n] \quad = m.sender]$
$\land\ nodeFlagRMW' = [nodeFlagRMW \quad \text{EXCEPT } ![n] = m.flagRMW]$ *RMW* change
$\land\ nodeTS' = [nodeTS \text{ EXCEPT } ![n].version \quad = m.version,$
$\qquad\qquad ![n].tieBreaker = m.tieBreaker]$
acknowledge a received invalidation (w/ greater timestamp)
$\land\ h\_send\_inv\_or\_ack(n,\ m.version,\ m.tieBreaker,\ \text{“ACK”})$
$\land\ hr\_upd\_state\_greater\_inv(n)$
$\land$ UNCHANGED $\langle Rmsgs \rangle$
ELSE IF $equalTS(m.version,\ m.tieBreaker,$
$\qquad nodeTS[n].version,\ nodeTS[n].tieBreaker)$
THEN
acknowledge a received invalidation (w/ equal timestamp)
$\land\ h\_send\_inv\_or\_ack(n,\ m.version,\ m.tieBreaker,\ \text{“ACK”})$
$\land$ UNCHANGED $\langle nodeState,\ nodeTS,\ nodeLastWriter,\ nodeFlagRMW,\ Rmsgs \rangle$
ELSE smaller *TS*
$\land\ hr\_send\_inv(n,\ nodeTS[n].version,\ nodeTS[n].tieBreaker,\ nodeFlagRMW[n])$
$\land$ UNCHANGED $\langle nodeState,\ nodeTS,\ nodeLastWriter,\ nodeFlagRMW,\ msgs \rangle$
$\land$ UNCHANGED $\langle nodeLastWriteTS,\ aliveNodes,\ nodeRcvedAcks,\ epochID,$
$\qquad nodeWriteEpochID,\ committedRMWs,\ committedWrites \rangle$

Keep the *HRcvVals* the same as *Hermes* w/o *RMWs*
$HRRcvVal(n) \triangleq$
$\land\ HRcvVal(n)$
$\land\ hr\_upd\_nothing$

$HRFollowerWriteReplay(n) \triangleq$ Execute a write-replay when coordinator failed
$\land\ nodeState[n] = \text{“invalid”}$
$\land\ \neg isAlive(nodeLastWriter[n])$
$\land\ hr\_actions\_for\_upd\_replay(n,\ \{\})$

$HRFollowerActions(n) \triangleq$ Actions of a write follower
$\lor\ HRFollowerWriteReplay(n)$
$\lor\ HRRcvWriteInv(n)$
$\lor\ HRRcvRMWInv(n)$
$\lor\ HRRcvVal(n)$

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

$HRNext \triangleq$ *Hermes* (read,write *RMWs*) protocol (Coordinator and Follower actions) + failures
$\exists\, n \in aliveNodes :$
$\lor\ HRFollowerActions(n)$

$\lor\ HRCoordinatorActions(n)$
$\lor\ HRNodeFailure(n)$

$HRSpec \triangleq HRInit \land \Box[HRNext]_{hrvars}$

THEOREM $HRSpec \Rightarrow (\Box HRTypeOK) \land (\Box HConsistent) \land (\Box HRSemanticsRMW)$