```
────────────────────── MODULE ChainRep ──────────────────────
CHAIN REPLICATION
EXTENDS Sequences, Naturals,  TLC
CONSTANT FirstReplica,  first replica, the one that's up at Init
         InitialChainLgth,
           Adr,
           Rep,
           Val,
           Object,
           Reply(_, _, _),
           NoReply

VARIABLES master,
          cache,
          data,
          channel,
          stat

├──────────────────────────────────────────────────────────┤
NoVal       ≜ CHOOSE v : v ∉ Val
InvalidVal  ≜ CHOOSE v : v ∉ Val ∧ v ≠ NoVal
NoRep       ≜ CHOOSE r : r ∉ Rep
InvalidRep  ≜ CHOOSE x : x ∉ (Rep ∪ {NoRep})

├──────────────────────────────────────────────────────────┤
WrReqMsg ≜  Write requests
   [type : {"wrReq", "cliWrReq"}, adr : Adr, w : Val]
Messages ≜
   WrReqMsg

TypeInvariant ≜
   ∧ master ∈
         [chains : [Object → Seq(Rep ∪ {NoRep})],
            health : [Rep → {"dead", "alive"}]]

   ∧ cache ∈ [Rep → [Object →
            [left : Rep ∪ {NoRep, InvalidRep}, right : Rep ∪ {NoRep, InvalidRep}, in_chain : {TRUE, FALSE}]]]
   ∧ data ∈ [Rep → [Object → [Adr → Val ∪ {NoVal, InvalidVal}]]]

   ∧ channel ∈ [Rep → [Object → [in : Seq(Messages), out : Seq(Messages)]]]

   ∧ stat ∈ [Rep → [phase : {"dead", "alive", "recover", "reconfig"}]]

Init ≜
   ∧ master = [chains ↦ [o ∈ Object ↦ ⟨NoRep, FirstReplica, NoRep⟩],
                                           health ↦ [r ∈ Rep ↦ IF r = FirstReplica THEN "alive" ELSE  "dead"]]
```

```
$\land$ $cache = [r \in Rep \mapsto [o \in Object \mapsto$
                 IF $r = FirstReplica$ THEN
                $[left \mapsto NoRep, right \mapsto NoRep, in\_chain \mapsto$ TRUE$]$
             ELSE $[left \mapsto InvalidRep, right \mapsto InvalidRep, in\_chain \mapsto$ FALSE$]]]$
$\land$ $data = [r \in Rep \mapsto [o \in Object \mapsto [a \in Adr \mapsto$
          IF $r = FirstReplica$ THEN $NoVal$ ELSE $InvalidVal]]]$

$\land$ $channel = [r \in Rep \mapsto [o \in Object \mapsto [in \mapsto \langle\rangle, out \mapsto \langle\rangle]]]$

$\land$ $stat = [r \in Rep \mapsto [phase \mapsto$ IF $r = FirstReplica$ THEN "alive" ELSE "dead"$]]$
```

Useful functions

```
$Hd(o) \triangleq$ IF $master.chains[o] = \langle NoRep, NoRep\rangle$ THEN
        $NoRep$
      ELSE
        $master.chains[o][2]$
$Tl(o) \triangleq$ IF $master.chains[o] = \langle NoRep, NoRep\rangle$ THEN
     $NoRep$
    ELSE
     $master.chains[o][Len(master.chains[o]) - 1]$
```

MASTER

```
$DelElem(seq, el) \triangleq$
  IF $(\exists\, i \in 2 \,..\, (Len(seq) - 1) : seq[i] = el)$ THEN
    LET $idx \triangleq$ CHOOSE $i \in 1 \,..\, Len(seq) : seq[i] = el$IN
      $SubSeq(seq, 1, idx - 1) \circ SubSeq(seq, idx + 1, Len(seq))$
  ELSE $seq$
$InSeq(seq, el) \triangleq$
  $\exists\, i \in 2 \,..\, (Len(seq) - 1) : seq[i] = el$

$\_RemoveRep(r) \triangleq$
 LET $alivepred(r1) \triangleq r1 \neq NoRep \land stat[r1].phase \neq$ "recover"
 IN
   $\land$ $master.health[r] \neq$ "dead"
   $\land$ $\forall\, o \in Object :$
       $InSeq(master.chains[o], r) \Rightarrow Len(SelectSeq(master.chains[o], alivepred)) > 1$ the two $NoRep$ plus at leas
   $\land$ $master' = [health \mapsto [master.health$ EXCEPT $![r] =$ "dead"$],$
                $chains \mapsto [o \in Object \mapsto$
                   $DelElem(master.chains[o], r)]]$
   $\land$ $stat' = [stat$ EXCEPT $![r].phase =$ "dead"$]$ ASSUMPTION : FAIL-STOP, or no partitions
                                  I do it in-line here, $s.t.$ I avoid transition
$RemoveRep \triangleq$
  $\exists\, r \in Rep :$
```

```
    $\land \_RemoveRep(r)$

Following some protocol, the master tells replica $r$ to restart for addition.
$InitReplica(r, o) \triangleq$
  $\land stat' = [stat \text{ EXCEPT } ![r].phase = \text{"recover"}]$ revive the replica (just $s.t.$ I don't have one more transition)
  $\land channel' = [channel \text{ EXCEPT } ![r][o] = [in \mapsto \langle\rangle, out \mapsto \langle\rangle]]$
  $\land data' = [data \text{ EXCEPT } ![r][o] = [a \in Adr \mapsto InvalidVal]]$
  $\land cache' = [cache \text{ EXCEPT } ![r][o] = [cache[r][o] \text{ EXCEPT } !.in\_chain = \text{FALSE}]]$ make it

$\_AddRep(r, o) \triangleq$
  $\land \neg InSeq(master.chains[o], r)$ $r$ is not yet in chain for $o$
  $\land master' = [master \text{ EXCEPT } !.chains[o] = SubSeq(@, 1, Len(@) - 1) \circ \langle r, NoRep\rangle,$
                                        $!.health[r] = \text{"alive"}]$ append $r$ to the chain
  $\land InitReplica(r, o)$

$AddRep \triangleq$
  $\exists r \in Rep :$
    $\land master.health[r] = \text{"dead"}$ the replica is not dead or restarting

    Everybody have $ACKed$ that they know that $r$ is dead
    $\land \forall r1 \in Rep : \forall o \in Object :$
        $cache[r1][o].in\_chain \land stat[r1].phase \neq \text{"dead"}$
         $\Rightarrow cache[r1][o].right \neq r \land cache[r1][o].left \neq r$

    There is an object to which I can add it.
    $\land \exists o \in Object :$
       $\land stat[Tl(o)].phase \neq \text{"recover"}$ the tail finished recovery
       $\land \_AddRep(r, o)$

$MasterActions \triangleq$
  $\lor RemoveRep \land \text{UNCHANGED } \langle cache, channel, data\rangle$ either remove a replica (kill it)
  $\lor AddRep$  $\land \text{UNCHANGED } \langle cache, stat, channel, data\rangle$   or add it to some chain
```

REPLICA

```
$\_RecvUpdateConfig(r, o) \triangleq$  replica rreceives an update message on the configuration from master
  $\land InSeq(master.chains[o], r)$ $r$ should currently be in the chain for $o$
                                          late $updateConfig$ messages are not supported $TODO$
  $\land \text{LET } idx \triangleq \text{CHOOSE } i \in 2 \,..\, (Len(master.chains[o]) - 1) : master.chains[o][i] = r\text{IN}$
     $\land$   $\lor cache[r][o].in\_chain = \text{FALSE}$ $r$ doesn't know it's in chain
         OR $r$ doesn't know its right left/right neighbors
         $\lor cache[r][o].left \neq master.chains[o][idx - 1]$
         $\lor cache[r][o].right \neq master.chains[o][idx + 1]$

     $\land cache' = [cache \text{ EXCEPT } ![r][o].left = master.chains[o][idx - 1],$
                                     $![r][o].right = master.chains[o][idx + 1],$
                                      $! [r][o].in\_chain = \text{TRUE}]$
  $\land$ IF ( $\land$ $cache[r][o].right \neq cache'[r][o].right$  right neighbor has changed
             $\land$ $stat[r].phase \neq$ "recover" )  replica is not recovering
     THEN
         mark state as *reconfig*
        $stat' = [stat$ EXCEPT $![r].phase =$ "reconfig"$]$
     ELSE  UNCHANGED $stat$
  $\land$ UNCHANGED $\langle master,\ data,\ channel \rangle$

$RecvUpdateConfig \triangleq$
  $\exists\, r \in Rep$ :
     $\land$ $stat[r].phase \in \{$"alive", "reconfig", "recover"$\}$  $r$ is alive or reconfiguring, but not dead or recovering
     $\land$ $\exists\, o \in Object : \_RecvUpdateConfig(r,\ o)$

$IsNotClientReq(m) \triangleq$
    $m.type \neq$ "cliWrReq"


$FwdWrite(r,\ o,\ q) \triangleq$   while in *reconfig* state, fwd a part of the writes already processed at $r$
                                   to $q$,. who's $r$'s right neighbor
 LET $index\_unknown \triangleq Len(SelectSeq(channel[q][o].in,\ IsNotClientReq)) + Len(channel[q][o].out) + 1$IN
   $\land$  $Len(channel[r][o].out) \geq index\_unknown$   there are still writes to fwd
   $\land$  $channel' = [channel$ EXCEPT $![q][o].in = @ \circ \langle channel[r][o].out[index\_unknown] \rangle]$
   $\land$  $NoReply$

$SendAcks(r,\ o) \triangleq$   while in *reconfig* state, *ACK* ALL unacked committed writes to the client
  $\land$ $Len(channel[r][o].out) > 0$
  $\land$ LET $wmsg \triangleq Head(channel[r][o].out)$IN
        $Reply(r,$ "wr", $[object \mapsto o,\ adr \mapsto wmsg.adr,\ val \mapsto wmsg.w,\ ack \mapsto$ "ok"$])$
  $\land$ $channel' = [r1 \in Rep \mapsto [channel[r1]$ EXCEPT $![o] =$
                              $[in \mapsto channel[r1][o].in,$
                               $out \mapsto$ IF $Len(channel[r1][o].out) > 0$ THEN $Tail(channel[r1][o].out)$
                                      ELSE $\langle\rangle]]]$

$ResendNext(r,\ o) \triangleq$   replica $r$ reconciles with its right neighbor by re-sending to it the "sent" writes
  $\land$ $stat[r].phase =$ "reconfig"  reconfiguring state
  $\land$ $(cache[r][o].right = NoRep \Rightarrow SendAcks(r,\ o))$  send acks to client
  $\land$ $(cache[r][o].right \neq NoRep \Rightarrow FwdWrite(r,\ o,\ cache[r][o].right))$
  $\land$ UNCHANGED $\langle stat,\ data,\ master,\ cache \rangle$

$FinishReconfig(r,\ o) \triangleq$   finish a reconfiguration state
  $\land$ $stat[r].phase =$ "reconfig"
   no more things to fwd
  $\land$ $cache[r][o].right = NoRep \Rightarrow Len(channel[r][o].out) = 0$
  $\land$ $cache[r][o].right \neq NoRep \Rightarrow Len(channel[r][o].out) =$
        $Len(SelectSeq(channel[cache[r][o].right][o].in,\ IsNotClientReq))$
      $+ Len(channel[cache[r][o].right][o].out)$
  $\land\ stat' = [stat\ \text{EXCEPT}\ ![r].phase = \text{"alive"}]$
  $\land\ \text{UNCHANGED}\ \langle data, channel, master, cache \rangle$

$Reconfiguration \triangleq$   sends reply
  $\lor\ \exists\, r \in Rep,\ o \in Object :$
        $\lor\ ResendNext(r, o)$   either advance reconfiguration
        $\lor\ FinishReconfig(r, o) \land NoReply$   or finish reconfiguration
  $\lor\ RecvUpdateConfig \land NoReply$


$Reconcile(r, o, a) \triangleq$   Replica $r$ helps its recovering right neighbor to reconcile
  $TODO$: missing some transitions in the real system!
 $\text{LET}\ IsForAddress(m) \triangleq (m.adr = a)\ \text{IN}$
  $\land\ \ stat[r].phase = \text{"alive"}$
  $\land\ \ cache[r][o].right \neq NoRep$
  $\land\ \ stat[cache[r][o].right].phase = \text{"recover"}$   the right neighbor is in recovery state
  $\land\ \ data[cache[r][o].right][o][a] \neq data[r][o][a]$   hasn't yet reconciled for this address

  $\land\ \ SelectSeq(channel[r][o].out, IsForAddress) = \langle\rangle$   no messages already in the "sent" that await fwding
                                                            or have already been fwded to the right neighbor
                                                            Such updates already contain the value that I would send now
                                                            and some other updates, as well.

  send it the value at address a
  $\land\ data' = [data\ \text{EXCEPT}\ ![cache[r][o].right][o][a] = data[r][o][a]]$

  $\land\ \text{UNCHANGED}\ \langle master, cache, stat, channel \rangle$

$FinishReconcile(r, o) \triangleq$   $r$ knows it has finished reconciliation and thus it can now start
                             serving requests when its data is up-to-date and
                             it knows its correct neighbors.
  $\land\ stat[r].phase = \text{"recover"}$   $r$ is recovering

  Reconciliation is finished when the left neighbor has sent $r$ all the data
  this simulates a FINISH RECONCILE message received from left neighbor
  $\land\ \forall\, a \in Adr : data[r][o][a] \neq InvalidVal$   the data was fully reconciled

  By the time I finish reconciliation, the new replica must have read the config from the master
  at least once. Otherwise, it can't start working, b/c it doesn't know its place in the chain
  $\land\ cache[r][o].in\_chain$

  $\land\ stat' = [stat\ \text{EXCEPT}\ ![r].phase = \text{"alive"}]$   replica will start to respond to queries
  $\land\ \text{UNCHANGED}\ \langle master, cache, data, channel \rangle$

$Recovery \triangleq$   sends reply
 $\exists\, r \in Rep,\ o \in Object :$
    $\lor\ \exists\, a \in Adr : Reconcile(r, o, a)$
    $\lor\ FinishReconcile(r, o)$
```

```
$ReplicaDeath \triangleq \text{FALSE}$   avoid this transition b/c it's fail-stop anyway

$HdlWrite(r, o, wmsg, ch) \triangleq$
  $\land cache[r][o].right \neq NoRep$   not tail
  fwd the write to the right neighbor
  $\land channel' = [ch \text{ EXCEPT } ![r][o].out = @ \circ \langle wmsg \rangle,$
                                  $![cache[r][o].right][o].in = @ \circ \langle wmsg \rangle]$

$FinishWrite(r, o, wmsg, ch) \triangleq$   sends a reply
  $\land cache[r][o].right = NoRep$   no right neighbor, so $r$ believes it's tail

  reply to client
  $\land Reply(r, \text{"wr"}, [object \mapsto o, adr \mapsto wmsg.adr, val \mapsto wmsg.w, ack \mapsto \text{"ok"}])$
  $ACK$ to all replicas, to remove their – ATOMICALLY, b/c it's only for performance
  $\land channel' = [r1 \in Rep \mapsto [channel[r1] \text{ EXCEPT } ![o] =$
                      $[in \mapsto ch[r1][o].in,$
                       $out \mapsto \text{IF } Len(ch[r1][o].out) > 0 \text{ THEN } Tail(ch[r1][o].out)$
                                $\text{ELSE } \langle\rangle]]]$

$ProcessWrite(r, o, wmsg, ch) \triangleq$   replica $r$ processes a message in its incoming $FIFO$ – Sends reply
  $\land stat[r].phase \in \{\text{"alive"}, \text{"recover"}\}$   not reconfiguring, or dead
  $\land cache[r][o].in\_chain$   $r$ believes itself in the chain for $o$

  $\land data' = [data \text{ EXCEPT } ![r][o][wmsg.adr] = wmsg.w]$   commit to disk
  $\land \lor HdlWrite\ (r, o, wmsg, ch) \land NoReply$
     $\lor FinishWrite(r, o, wmsg, ch)$
  $\land \text{UNCHANGED } \langle master, cache, stat \rangle$

$\_ProcessMsg(r, o) \triangleq$   sends reply
  $\land channel[r][o].in \neq \langle\rangle$   the channel is not empty
  $\land stat[r].phase \in \{\text{"alive"}, \text{"recover"}\}$

  $\land \text{LET } wmsg \triangleq Head(channel[r][o].in)$
         $ch \triangleq [channel \text{ EXCEPT } ![r][o].in = Tail(@)]$
    $\text{IN}$
        $\land wmsg.type = \text{"wrReq"} \Rightarrow ProcessWrite(r, o, wmsg, ch)$
        $\land wmsg.type = \text{"cliWrReq"} \Rightarrow$
              $(\text{IF } (cache[r][o].in\_chain \land cache[r][o].left = NoRep)$   check $r$ is head
                $\text{THEN}$
                    $ProcessWrite(r, o, [wmsg \text{ EXCEPT } !.type = \text{"wrReq"}], ch)$   go ahead $w/$ the write
                $\text{ELSE}$   $r$ is not head, so just drop the writ
                      $\land channel' = ch$
                      $\land \text{UNCHANGED } \langle master, cache, stat, data \rangle \land NoReply)$
$ProcessMsg \triangleq$
  $\exists r \in Rep, o \in Object : \_ProcessMsg(r, o)$

$HeadWrite(r, o, a, v) \triangleq$   sends reply
```

```
  $\land\ cache[r][o].left = NoRep$  $r$ believes itself head
  The line below is a hacky solution to having a 0-phase write and still being mappable to $SS$
  It fails a write that goes to a head when the head is the only one in the chain
  $\land\ cache[r][o].right \neq NoRep$

  $\land\ ProcessWrite(r,\ o,\ [type \mapsto \text{"wrReq"},\ adr \mapsto a,\ w \mapsto v],\ channel)$

$TailRead(r,\ o,\ a)\ \triangleq$                 sends reply
  $\land\ stat[r].phase \in \{\text{"alive"}$  $, \text{"reconfig"}\}$  not recovering, reconfiguring, or dead
  $\land\ cache[r][o].in\_chain$  $r$ believes itself in the chain for $o$
  $\land\ cache[r][o].right = NoRep$

  $\land\ Reply(r,\ \text{"rd"},\ [object \mapsto o,\ adr \mapsto a,\ val \mapsto data[r][o][a],\ ack \mapsto \text{"ok"}])$
  $\land\ \text{UNCHANGED}\ \langle data,\ master,\ cache,\ channel,\ stat\rangle$

$ReplicaActions\ \triangleq$
  $\lor\ Reconfiguration$
  $\lor\ Recovery$
  $\lor\ ReplicaDeath$

  $\lor\ ProcessMsg$
```

CLIENT

```
$\_CliRead(o,\ a)\ \triangleq$
  $\exists\, r \in Rep :$
      $TailRead(r,\ o,\ a)$
$CliRead\ \triangleq$
  $\exists\, o \in Object,\ a \in Adr : \_CliRead(o,\ a)$

$\_CliWrite(o,\ a,\ v)\ \triangleq$
   $\land\ \exists\, r \in Rep :$
        the line below is more of a hack $s.t.$ I don't add a huge # of states
        $HeadWrite(r,\ o,\ a,\ v)$
        Uncomment below and comment above to be most fair
       $channel' = [channel\ \text{EXCEPT}\ ![r][o].in = @ \circ \langle[type \mapsto \text{"cliWrReq"},\ adr \mapsto a,\ w \mapsto v]\rangle]$
  $\land\ \text{UNCHANGED}\ \langle master,\ cache,\ data,\ stat\rangle$
$CliWrite\ \triangleq$
  $\exists\, o \in Object,\ a \in Adr,\ v \in Val :$
     $\_CliWrite(o,\ a,\ v)$

$ClientActions\ \triangleq$
  $\lor\ CliRead$
  $\lor\ CliWrite \land NoReply$
```

Full specification of the Chain replication system

```
$Next \triangleq$
    $\lor MasterActions$
    $\lor ReplicaActions$
    $\lor ClientActions$
```

Full spec:

```
$chainvars \triangleq \langle master, cache, data, channel, stat \rangle$
$Spec \triangleq Init \land \Box[Next]_{chainvars}$
```

Invariants

```
$includedInSeq(smallseq, bigseq) \triangleq$
  $\forall i \in 1 \,..\, Len(smallseq) :$
        $smallseq[i] = bigseq[i]$
$UpdatePropagation \triangleq$    Update propagation invariant from paper
   $\forall o \in Object :$
     $\forall i \in (3 \,..\, (Len(master.chains[o]) - 1)) :$
       $includedInSeq(channel[master.chains[o][i]][o].out, channel[master.chains[o][i-1]][o].out)$

$AllInvariants \triangleq$
     $\land TypeInvariant$
     $\land UpdatePropagation$
```

Theorem

```
THEOREM $Spec \Rightarrow \Box AllInvariants$
```