```
──────────────────── MODULE AJupiter ────────────────────
Model checking the Jupiter protocol presented by Attiya and others.
EXTENDS OT, TLC
─────────────────────────────────────────────────────────
CONSTANTS
    Client,       the set of client replicas
    Server,       the (unique) server replica
    InitState,    the initial state of each replica
    Cop           Cop[c]: operations issued by the client c ∈ Client

ASSUME
    ∧ InitState ∈ List
    ∧ Cop ∈ [Client → Seq(Op)]

VARIABLES
    For model checking:
    cop,          cop[c]: operations issued by the client c ∈ Client

    For the client replicas:
    cbuf,       cbuf[c]: buffer (of operations) at the client c ∈ Client
    crec,       crec[c]: the number of new messages have been received by the client c ∈ Client
                         since the last time a message was sent
    cstate,     cstate[c]: state (the list content) of the client c ∈ Client

    For the server replica:
    sbuf,       sbuf[c]: buffer (of operations) at the Server, one per client c ∈ Client
    srec,       srec[c]: the number of new messages have been ..., one per client c ∈ Client
    sstate,     sstate: state (the list content) of the server Server

    For communication between the Server and the Clients:
    cincoming,    cincoming[c]: incoming channel at the client c ∈ Client
    sincoming     incoming channel at the Server
─────────────────────────────────────────────────────────
comm ≜ INSTANCE CSComm
─────────────────────────────────────────────────────────
cVars ≜ ⟨cop, cbuf, crec, cstate⟩
sVars ≜ ⟨sbuf, srec, sstate⟩
  FIXME: subscript error (Don't know why yet!)
  vars ≜ cVars ∘ sVars ∘ ⟨cincoming, sincoming⟩
vars ≜ ⟨cop, cbuf, crec, cstate, sbuf, srec, sstate, cincoming, sincoming⟩
─────────────────────────────────────────────────────────
TypeOK ≜
    ∧ cop ∈ [Client → Seq(Op)]
    For the client replicas:
        $\land\ cbuf \in [Client \rightarrow Seq(Op \cup \{Nop\})]$
        $\land\ crec \in [Client \rightarrow Nat]$
        $\land\ cstate \in [Client \rightarrow List]$
        For the server replica:
        $\land\ sbuf \in [Client \rightarrow Seq(Op \cup \{Nop\})]$
        $\land\ srec \in [Client \rightarrow Nat]$
        $\land\ sstate \in List$
        For communication between the server and the clients:
        $\land\ comm!TypeOK$
--------------------------------------------------------------------------------
The $Init$ predicate.
$Init \triangleq$
        $\land\ cop = Cop$
        For the client replicas:
        $\land\ cbuf = [c \in Client \mapsto \langle\rangle]$
        $\land\ crec = [c \in Client \mapsto 0]$
        $\land\ cstate = [c \in Client \mapsto InitState]$
        For the server replica:
        $\land\ sbuf = [c \in Client \mapsto \langle\rangle]$
        $\land\ srec = [c \in Client \mapsto 0]$
        $\land\ sstate = InitState$
        For communication between the server and the clients:
        $\land\ comm!Init$
--------------------------------------------------------------------------------
Client $c \in Client$ issues an operation $op$.
$Do(c) \triangleq$
        $\land\ cop[c] \neq \langle\rangle$
        $\land$ LET $op \triangleq Head(cop[c])$
            IN    $\land\ PrintT(c \circ$ ": Do " $\circ\ ToString(op))$
                  $\land\ cstate' = [cstate$ EXCEPT $![c] = Apply(op, @)]$
                  $\land\ cbuf' = [cbuf$ EXCEPT $![c] = Append(@, op)]$
                  $\land\ comm!CSend([c \mapsto c,\ ack \mapsto crec[c],\ op \mapsto op])$
        $\land\ crec' = [crec$ EXCEPT $![c] = 0]$
        $\land\ cop' = [cop$ EXCEPT $![c] = Tail(@)]$
        $\land$ UNCHANGED $sVars$

Client $c \in Client$ receives a message from the $Server$.
$Rev(c) \triangleq$
        $\land\ comm!CRev(c)$
        $\land\ crec' = [crec$ EXCEPT $![c] = @ + 1]$
        $\land$ LET $m \triangleq Head(cincoming[c])$
                $cBuf \triangleq cbuf[c]$    the buffer at client $c \in Client$
                $cShiftedBuf \triangleq SubSeq(cBuf,\ m.ack + 1,\ Len(cBuf))$    buffer shifted
```

```
              $xop \triangleq XformOpOps(m.op, cShiftedBuf)$  transform $op$ vs. shifted buffer
               $xcBuf \triangleq XformOpsOp(cShiftedBuf, m.op)$  transform shifted buffer vs. $op$
          IN   $\land\ cbuf' = [cbuf$ EXCEPT $![c] = xcBuf]$
               $\land\ cstate' = [cstate$ EXCEPT $![c] = Apply(xop, @)]$  apply the transformed operation $xop$
    $\land$ UNCHANGED $\langle sbuf, srec, sstate, cop \rangle$   NOTE: $sVars \circ \langle cop \rangle$ is wrong!
|-----------------------------------------------------------------------|
The $Server$ receives a message.
$SRev \triangleq$
    $\land\ comm!SRev$
    $\land$ LET $m \triangleq Head(sincoming)$   the message to handle with
            $c \triangleq m.c$                the client $c \in Client$ that sends this message
            $cBuf \triangleq sbuf[c]$          the buffer at the $Server$ for client $c \in Client$
            $cShiftedBuf \triangleq SubSeq(cBuf, m.ack + 1, Len(cBuf))$  buffer shifted
            $xop \triangleq XformOpOps(m.op, cShiftedBuf)$  transform $op$ vs. shifted buffer
             $xcBuf \triangleq XformOpsOp(cShiftedBuf, m.op)$  transform shifted buffer vs. $op$
        IN  $\land\ srec' = [cl \in Client \mapsto$
                          IF $cl = c$
                           THEN $srec[cl] + 1$  receive one more operation from client $c \in Client$
                           ELSE $0]$  reset $srec$ for other clients than $c \in Client$
            $\land\ sbuf' = [cl \in Client \mapsto$
                          IF $cl = c$
                           THEN $xcBuf$  transformed buffer for client $c \in Client$
                           ELSE $Append(sbuf[cl], xop)]$  store transformed $xop$ into other clients' bufs
            $\land\ sstate' = Apply(xop, sstate)$  apply the transformed operation
            $\land\ comm!SSend(c, srec, xop)$
    $\land$ UNCHANGED $cVars$
|-----------------------------------------------------------------------|
The next-state relation.
$Next \triangleq$
    $\lor\ \exists c \in Client : Do(c) \lor Rev(c)$
    $\lor\ SRev$
The $Spec$.
$Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)$
|-----------------------------------------------------------------------|
The safety properties to check: Eventual Convergence ($EC$), Quiescent Consistency ($QC$), Strong
Eventual Convergence ($SEC$), Weak $List$ Specification, ($WLSpec$), and Strong $List$ Specification,
($SLSpec$).

Eventual Consistency ($EC$)

Quiescent Consistency ($QC$)
$QConvergence \triangleq \forall c \in Client : cstate[c] = sstate$
$QC \triangleq comm!EmptyChannel \Rightarrow QConvergence$

THEOREM $Spec \Rightarrow \Box QC$
```

Strong Eventual Consistency (*SEC*)

Weak *List* Consistency (*WLSpec*)

Strong *List* Consistency (*SLSpec*)

\ $*$ Modification History
\ $*$ Last modified Sat *Jul* 07 16:01:04 *CST* 2018 by *hengxin*
\ $*$ Created Sat *Jun* 23 17:14:18 *CST* 2018 by *hengxin*