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

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

 Generate operations for AJupiter clients.

 Note: Remember to overvide the definition of PosInt.

 FIXME: PosInt ⇒ MaxPos; MaxPr determined by the size of Client.
OpToIssue ≜ {opset ∈ SUBSET Op :
                 ∧ opset ≠ {}
                 ∧ ∀ op1 ∈ opset :
                      ∀ op2 ∈ opset \ {op1} :
                         (op1.type = "Ins" ∧ op2.type = "Ins") ⇒ op1.ch ≠ op2.ch}

VARIABLES
    For model checking:
    cop,       \* cop[c]: operations issued by the client c ∈ Client
    cop,        a set of operations for clients to issue
    list,       all list states across the system

    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
────────────────────────────────────────────────────────────────
eVars ≜ ⟨cop⟩                                        variables for the environment
cVars ≜ ⟨cbuf, crec, cstate⟩                         variables for the clients
ecVars ≜ ⟨cop, cVars⟩                                variables for the clients and the environment
sVars ≜ ⟨sbuf, srec, sstate⟩                         variables for the server
commVars ≜ ⟨cincoming, sincoming⟩                    variables for communication
jVars ≜ ⟨cVars, sVars, commVars⟩                     variables for the Jupiter system
vars ≜ ⟨eVars, cVars, sVars, commVars⟩                all variables
────────────────────────────────────────────────────────────────
TypeOK ≜
   ∧ cop ∈  [Client → Seq(Op)]
    ∧ cop ∈ SUBSET Op
    ∧ list ∈ SUBSET List
    For the client replicas:
    ∧ cbuf ∈ [Client → Seq(Op ∪ {Nop})]
    ∧ crec  ∈ [Client → Int]
    ∧ cstate ∈ [Client → List]
    For the server replica:
    ∧ sbuf ∈ [Client → Seq(Op ∪ {Nop})]
    ∧ srec  ∈ [Client → Int]
    ∧ sstate ∈ List
    For communication between the server and the clients:
    ∧ comm!TypeOK
────────────────────────────────────────────────────────────────
The Init predicate.
Init ≜
   ∧ cop =  Cop
    ∧ cop ∈ OpToIssue
    For the client replicas:
    ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
    ∧ crec  = [c ∈ Client ↦ 0]
    ∧ cstate = [c ∈ Client ↦ InitState]
    For the server replica:
    ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
    ∧ srec  = [c ∈ Client ↦ 0]
    ∧ sstate = InitState
    For communication between the server and the clients:
    ∧ comm!Init
────────────────────────────────────────────────────────────────
LegalizeOp(op, c) ≜
    LET len ≜ Len(cstate[c])
```

```
     IN   CASE op.type = "Del" →
               IF len = 0 THEN Nop ELSE [op EXCEPT !.pos = Min(@, len)]
          □    op.type = "Ins" →
               [op EXCEPT !.pos = Min(@, len + 1), !.pr = Priority[c]]

Client c ∈ Client issues an operation op.
Do(c) ≜
  ∧ cop[c] ≠ ⟨⟩
   ∧ cop ≠ {}
   ∧ ∃ o ∈ cop :
       LET op ≜ LegalizeOp(o, c)   preprocess an illegal operation
       IN   ∨ ∧ op = Nop
                ∧ cop′ = cop \ {o}       consume one operation
                ∧ UNCHANGED jVars
              ∨ ∧ op ≠ Nop
              ∧ PrintT(c ∘ " : Do" ∘ ToString(op))
                ∧ cstate′ = [cstate EXCEPT ![c] = Apply(op, @)]
                ∧ list′   = list ∪ {cstate′[c]}
                ∧ cbuf′ = [cbuf EXCEPT ![c] = Append(@, op)]
                ∧ crec′  = [crec EXCEPT ![c]  = 0]
                ∧ comm!CSend([c ↦ c, ack ↦ crec[c], op ↦ op])
                ∧ cop′ = cop \ {o}      consume one operation
                ∧ UNCHANGED sVars
  ∧ cop′ =  [cop  EXCEPT ![c] = Tail(@)]  \* consume one operation

Client c ∈ Client receives a message from the Server.
Rev(c) ≜
   ∧ comm!CRev(c)
   ∧ crec′ = [crec EXCEPT ![c] = @ + 1]
   ∧ LET m ≜ Head(cincoming[c])
          cBuf ≜ cbuf[c]   the buffer at client c ∈ Client
          cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
          xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
           xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
      IN   ∧ cbuf′ = [cbuf EXCEPT ![c] = xcBuf]
           ∧ cstate′ = [cstate EXCEPT ![c] = Apply(xop, @)]   apply the transformed operation xop
           ∧ list′ = list ∪ {cstate′[c]}
   ∧ UNCHANGED ⟨sbuf, srec, sstate, cop⟩       NOTE: sVars ∘ ⟨cop⟩ is wrong!
├──────────────────────────────────────────────────────────────────────┤
The Server receives a message.
SRev ≜
   ∧ comm!SRev
   ∧ LET m ≜ Head(sincoming)   the message to handle with
          c  ≜ m.c               the client c ∈ Client that sends this message
          cBuf ≜ sbuf[c]          the buffer at the Server for client c ∈ 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
                 $\land list' = list \cup \{sstate'\}$
                 $\land comm!SSend(c, srec, xop)$
        $\land$ UNCHANGED $ecVars$
```

The next-state relation.

```
$Next \triangleq$
     $\lor \exists c \in Client : Do(c) \lor Rev(c)$
     $\lor SRev$
```

The *Spec*. (*TODO*: Check the fairness condition.)

```
$Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)$
```

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*)

Termination

```
$Termination \triangleq$
     $\land cop = \{\}$
     $\land comm!EmptyChannel$
```

Weak *List* Consistency (*WLSpec*)

```
$WLSpec \triangleq$
     $Termination \Rightarrow \forall l1, l2 \in list : Compatible(l1, l2)$
```

THEOREM $Spec \Rightarrow WLSpec$

Strong *List* Consistency (*SLSpec*)

\* Modification History
\* *Last* modified Sun *Aug* 12 23:13:41 *CST* 2018 by *hengxin*
\* Created Sat *Jun* 23 17:14:18 *CST* 2018 by *hengxin*