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 $c \in Client$

ASSUME

$\land\ InitState \in List$

$\land\ Cop \in [Client \to Seq(Op)]$

VARIABLES

For model checking:

cop, $cop[c]$: operations issued by the client $c \in Client$

For the client replicas:

$cbuf$, $cbuf[c]$: buffer (of operations) at the client $c \in Client$

$crec$, $crec[c]$: the number of new messages have been received by the client $c \in Client$ since the last time a message was sent

$cstate$, $cstate[c]$: state (the list content) of the client $c \in Client$

For the server replica:

$sbuf$, $sbuf[c]$: buffer (of operations) at the $Server$, one per client $c \in Client$

$srec$, $srec[c]$: the number of new messages have been ..., one per client $c \in Client$

$sstate$, $sstate$: state (the list content) of the server $Server$

For communication between the $Server$ and the Clients:

$cincoming$, $cincoming[c]$: incoming channel at the client $c \in Client$

$sincoming$ incoming channel at the $Server$

$comm \triangleq$ INSTANCE $CSComm$

$cVars \triangleq \langle cop, cbuf, crec, cstate \rangle$

$sVars \triangleq \langle sbuf, srec, sstate \rangle$

FIXME: subscript error (Don't know why yet!)

$vars \triangleq cVars \circ sVars \circ \langle cincoming, sincoming \rangle$

$vars \triangleq \langle cop, cbuf, crec, cstate, sbuf, srec, sstate, cincoming, sincoming \rangle$

$TypeOK \triangleq$

$\land\ cop \in [Client \to Seq(Op)]$

For the client replicas:

```
57      ∧ cbuf ∈ [Client → Seq(Op ∪ {Nop})]
58      ∧ crec ∈ [Client → Nat]
59      ∧ cstate ∈ [Client → List]
        For the server replica:
63      ∧ sbuf ∈ [Client → Seq(Op ∪ {Nop})]
64      ∧ srec ∈ [Client → Nat]
65      ∧ sstate ∈ List
        For communication between the server and the clients:
69      ∧ comm ! TypeOK
70  ├──────────────────────────────────────────────────────────────┤
    The Init predicate.
74  Init ≜
75      ∧ cop = Cop
        For the client replicas:
79      ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
80      ∧ crec = [c ∈ Client ↦ 0]
81      ∧ cstate = [c ∈ Client ↦ InitState]
        For the server replica:
85      ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
86      ∧ srec = [c ∈ Client ↦ 0]
87      ∧ sstate = InitState
        For communication between the server and the clients:
91      ∧ comm ! Init
92  ├──────────────────────────────────────────────────────────────┤
    Client c ∈ Client issues an operation op.
96  Do(c) ≜
97      ∧ cop[c] ≠ ⟨⟩
98      ∧ LET op ≜ Head(cop[c])
99          IN  ∧ PrintT(c ∘ ": Do " ∘ ToString(op))
100             ∧ cstate′ = [cstate EXCEPT ![c] = Apply(op, @)]
101             ∧ cbuf′ = [cbuf EXCEPT ![c] = Append(@, op)]
102             ∧ comm ! CSend([c ↦ c, ack ↦ crec[c], op ↦ op])
103     ∧ crec′ = [crec EXCEPT ![c] = 0]
104     ∧ cop′ = [cop EXCEPT ![c] = Tail(@)]
105     ∧ UNCHANGED sVars

    Client c ∈ Client receives a message from the Server.
110 Rev(c) ≜
111     ∧ comm ! CRev(c)
112     ∧ crec′ = [crec EXCEPT ![c] = @ + 1]
113     ∧ LET m ≜ Head(cincoming[c])
114           cBuf ≜ cbuf[c]   the buffer at client c ∈ Client
115           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
```

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 ≜ 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  ∧ srec′ = [cl ∈ Client ↦
                      IF cl = c
                      THEN srec[cl] + 1  receive one more operation from client c ∈ Client
                      ELSE 0]  reset srec for other clients than c ∈ Client
          ∧ sbuf′ = [cl ∈ Client ↦
                      IF cl = c
                      THEN xcBuf  transformed buffer for client c ∈ Client
                      ELSE Append(sbuf[cl], xop)]  store transformed xop into other clients' bufs
          ∧ sstate′ = Apply(xop, sstate)  apply the transformed operation
          ∧ comm!SSend(c, srec, xop)
    ∧ UNCHANGED cVars
```

The next-state relation.

```
Next ≜
    ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
    ∨ SRev
```

The *Spec*.

```
Spec ≜ Init ∧ □[Next]_vars
```

```
\* Modification History
\* Last modified Sat Jul 07 14:51:40 CST 2018 by hengxin
\* Created Sat Jun 23 17:14:18 CST 2018 by hengxin
```