MODULE $AJupiter$

Model checking the *Jupiter* protocol presented by *Attiya* and others.

EXTENDS OT

CONSTANTS

$Client$, the set of client replicas

$Server$ the (unique) server replica

VARIABLES

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 $Server$

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

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

$commVars \triangleq \langle cincoming, sincoming \rangle$

$vars \triangleq cVars \circ sVars \circ commVars$

Messages between the $Server$ and the Clients. There are two kinds of messages according to their destinations.

$Msg \triangleq [c : Client, ack : Nat, op : Op]$ messages sent to the $Server$ from a client $c \in Client$

$\cup\, [ack : Nat, op : Op]$ messages broadcast to Clients from the $Server$

$TypeOK \triangleq$

For the client replicas:

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

$\land\ crec \in [Client \to Nat]$

$\land\ cstate \in [Client \to List]$

For the server replica:

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

$\land\ srec \in [Client \to Nat]$

$\land\ sstate \in [Client \to List]$

For communication between the server and the clients:

```
61      ∧ cincoming ∈ [Client → Seq(Msg)]
62      ∧ sincoming ∈ Seq(Msg)
63  ----------------------------------------------------------------
    The Init predicate.
67  Init ≜
        For the client replicas:
71      ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
72      ∧ crec = [c ∈ Client ↦ 0]
73      ∧ cstate = [c ∈ Client ↦ ⟨⟩]
        For the server replica:
77      ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
78      ∧ srec = [c ∈ Client ↦ 0]
79      ∧ sstate = [c ∈ Client ↦ ⟨⟩]
        For communication between the server and the clients:
83      ∧ cincoming = [c ∈ Client ↦ ⟨⟩]
84      ∧ sincoming = ⟨⟩
85  ----------------------------------------------------------------
    A client sends a message msg to the Server.
89  CSend(msg) ≜ ∧ sincoming' = Append(sincoming, msg)
90  ----------------------------------------------------------------
    Client c ∈ Client generates and performs an operation op.
94  Do(c, op) ≜
95      ∧ TRUE      no pre-condition
96      ∧ cstate' = [cstate EXCEPT ![c] = Apply(op, @)]
97      ∧ cbuf' = [cbuf EXCEPT ![c] = Append(@, op)]
98      ∧ CSend([c ↦ c, ack ↦ crec[c], op ↦ op])
99      ∧ crec' = [crec EXCEPT ![c] = 0]
100     ∧ UNCHANGED (sVars ∘ ⟨cincoming⟩)
101 ----------------------------------------------------------------
    Client c ∈ Client receives a message from the Server.
105 CRev(c) ≜
106     ∧ cincoming[c] ≠ ⟨⟩    there are messages to handle with
107     ∧ crec' = [crec EXCEPT ![c] = @ + 1]
108     ∧ LET m ≜ Head(cincoming[c])
109           cBuf ≜ cbuf[c]    the buffer at client c ∈ Client
110           cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))    buffer shifted
111           xop ≜ XformOpOps(m.op, cShiftedBuf)    transform op vs. shifted buffer
112           xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)    transform shifted buffer vs. op
113       IN  ∧ cbuf' = [cbuf EXCEPT ![c] = xcBuf]
114           ∧ cstate' = [cstate EXCEPT ![c] = Apply(xop, @)]    apply the transformed operation xop
115     ∧ cincoming' = [cincoming EXCEPT ![c] = Tail(@)]
116     ∧ UNCHANGED (sVars ∘ ⟨sincoming⟩)
117 ----------------------------------------------------------------
```

```
    The Server receives a message.
121 SRev ≜
122     ∧ sincoming ≠ ⟨⟩        there are messages for the Server to handle with
123     ∧ LET m ≜ Head(sincoming)   the message to handle with
124           c ≜ m.c               the client c ∈ Client that sends this message
125           cBuf ≜ sbuf[c]        the buffer at the Server for client c ∈ Client
126           cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
127           xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
128           xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
129       IN  ∧ srec′ = [cl ∈ Client ↦
130                       IF cl = c
131                       THEN srec[cl] + 1   receive one more operation from client c ∈ Client
132                       ELSE 0]   reset srec for other clients than c ∈ Client
133           ∧ sbuf′ = [cl ∈ Client ↦
134                       IF cl = c
135                       THEN xcBuf   transformed buffer for client c ∈ Client
136                       ELSE Append(sbuf[cl], xop)]   store transformed xop into other clients' bufs
137           ∧ cincoming′ = [cl ∈ Client ↦
138                       IF cl = c
139                       THEN cincoming[cl]
140                       broadcast the transformed operation to clients other than c ∈ Client
141                       ELSE Append(cincoming[cl], [ack ↦ srec[cl], op ↦ xop])]
142           ∧ sstate′ = Apply(xop, sstate)   apply the transformed operation
143     ∧ sincoming′ = Tail(sincoming)   consume a message
144     ∧ UNCHANGED cVars
145 ──────────────────────────────────────────────────────────────────────
    The Next state relation.
149 Next ≜
150     ∨ ∃ c ∈ Client, op ∈ Op : Do(c, op)
151     ∨ ∃ c ∈ Client : CRev(c)
152     ∨ SRev
    The Spec.
156 Spec ≜ Init ∧ □[Next]_vars
157 ──────────────────────────────────────────────────────────────────────
    \* Modification History
    \* Last modified Sun Jun 24 22:26:35 CST 2018 by hengxin
    \* Created Sat Jun 23 17:14:18 CST 2018 by hengxin
```