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

Priority *Priority*[*c*]: the priority value of client $c \in Client$

Cop * *Cop*[*c*]: operations issued by the client $c \in Client$

ASSUME

$\land\ InitState \in List$

$\land\ Priority \in [Client \to PosInt]$

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

Generate operations for *AJupiter* clients.

Note: Remember to overvide the definition of *PosInt*.

FIXME: *PosInt* $\Rightarrow$ *MaxPos*; *MaxPr* determined by the size of *Client*.

$$OpToIssue \triangleq \{opset \in \text{SUBSET } Op : \forall\, op1,\, op2 \in opset : (op1.type = \text{“Ins”} \land op2.type = \text{“Ins”}) \Rightarrow op1.ch \neq op2.ch\}$$

VARIABLES

For model checking:

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

cop, a set of operations for clients to issue

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*

$$comm \triangleq \text{INSTANCE } CSComm$$

```
60  cVars ≜ ⟨cop, cbuf, crec, cstate⟩
61  sVars ≜ ⟨sbuf, srec, sstate⟩
62   FIXME: subscript error (Don't know why yet!)
63   vars ≜ cVars ∘ sVars ∘ ⟨cincoming, sincoming⟩
64  jVars ≜ ⟨cbuf, crec, cstate, sbuf, srec, sstate, cincoming, sincoming⟩   all variables
65  vars ≜ ⟨cop, cbuf, crec, cstate, sbuf, srec, sstate, cincoming, sincoming⟩   all variables
66 ├──────────────────────────────────────────────────────────────────────────┤
67  TypeOK ≜
68     ∧ cop ∈ [Client → Seq(Op)]
69     ∧ cop ∈ SUBSET Op
       For the client replicas:
73     ∧ cbuf ∈ [Client → Seq(Op ∪ {Nop})]
74     ∧ crec ∈ [Client → Nat]
75     ∧ cstate ∈ [Client → List]
       For the server replica:
79     ∧ sbuf ∈ [Client → Seq(Op ∪ {Nop})]
80     ∧ srec ∈ [Client → Nat]
81     ∧ sstate ∈ List
       For communication between the server and the clients:
85     ∧ comm!TypeOK
86 ├──────────────────────────────────────────────────────────────────────────┤
   The Init predicate.
90  Init ≜
91    ∧ cop = Cop
92     ∧ cop ∈ OpToIssue
       For the client replicas:
96     ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
97     ∧ crec = [c ∈ Client ↦ 0]
98     ∧ cstate = [c ∈ Client ↦ InitState]
       For the server replica:
102    ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
103    ∧ srec = [c ∈ Client ↦ 0]
104    ∧ sstate = InitState
       For communication between the server and the clients:
108    ∧ comm!Init
109 ├──────────────────────────────────────────────────────────────────────────┤
110 LegalizeOp(op, c) ≜
111    LET len ≜ Len(cstate[c])
112    IN  CASE op.type = "Del" →
113                IF len = 0 THEN Nop ELSE [op EXCEPT !.pos = Min(@, len)]
114          □     op.type = "Ins" →
115                [op EXCEPT !.pos = Min(@, len + 1), !.pr = Priority[c]]
   Client c ∈ Client issues an operation op.
```

```
119 Do(c) ≜
120   ∧ cop[c] ≠ ⟨⟩
121     ∧ cop ≠ {}
122     ∧ LET o ≜ CHOOSE x ∈ cop : TRUE
123           op ≜ LegalizeOp(o, c)   preprocess an illegal operation
124       IN    ∨ ∧ op = Nop
125                 ∧ cop′ = cop \ {o}    consume one operation
126                 ∧ UNCHANGED jVars
127               ∨ ∧ op ≠ Nop
128                 ∧ PrintT(c ∘ " : Do" ∘ ToString(op))
129                 ∧ cstate′ = [cstate EXCEPT ![c] = Apply(op, @)]
130                 ∧ cbuf′ = [cbuf EXCEPT ![c] = Append(@, op)]
131                 ∧ crec′ = [crec EXCEPT ![c] = 0]
132                 ∧ comm!CSend([c ↦ c, ack ↦ crec[c], op ↦ op])
133                 ∧ cop′ = cop \ {o}    consume one operation
134                 ∧ UNCHANGED sVars
135   ∧ cop′ = [cop EXCEPT ![c] = Tail(@)] \* consume one operation

    Client c ∈ Client receives a message from the Server.
140 Rev(c) ≜
141     ∧ comm!CRev(c)
142     ∧ crec′ = [crec EXCEPT ![c] = @ + 1]
143     ∧ LET m ≜ Head(cincoming[c])
144           cBuf ≜ cbuf[c]   the buffer at client c ∈ Client
145           cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
146           xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
147           xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
148       IN   ∧ cbuf′ = [cbuf EXCEPT ![c] = xcBuf]
149            ∧ cstate′ = [cstate EXCEPT ![c] = Apply(xop, @)]   apply the transformed operation xop
150     ∧ UNCHANGED ⟨sbuf, srec, sstate, cop⟩   NOTE: sVars ∘ ⟨cop⟩ is wrong!
151 ├──────────────────────────────────────────────────────────────────────┤
    The Server receives a message.
155 SRev ≜
156     ∧ comm!SRev
157     ∧ LET m ≜ Head(sincoming)   the message to handle with
158           c ≜ m.c   the client c ∈ Client that sends this message
159           cBuf ≜ sbuf[c]   the buffer at the Server for client c ∈ Client
160           cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
161           xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
162           xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
163       IN   ∧ srec′ = [cl ∈ Client ↦
164                         IF cl = c
165                         THEN srec[cl] + 1   receive one more operation from client c ∈ Client
166                         ELSE 0]   reset srec for other clients than c ∈ Client
167            ∧ 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 ∧ 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 ≜ ∀ c ∈ Client : cstate[c] = sstate
QC ≜ comm!EmptyChannel ⇒ QConvergence

THEOREM Spec ⇒ □QC
```

Strong Eventual Consistency (*SEC*)

Weak *List* Consistency (*WLSpec*)

Strong *List* Consistency (*SLSpec*)

```
\* Modification History
\* Last modified Sun Jul 15 14:57:35 CST 2018 by hengxin
\* Created Sat Jun 23 17:14:18 CST 2018 by hengxin
```