MODULE *AJupiter*

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

EXTENDS *Integers*, *OT*, *TLC*, *AdditionalFunctionOperators*

CONSTANTS

Client, the set of client replicas

Server, the (unique) server replica

Char, set of characters allowed

InitState the initial state of each replica

$List \triangleq Seq(Char \cup Range(InitState))$ all possible lists/strings

$MaxLen \triangleq Cardinality(Char) + Len(InitState)$ the max length of lists in any states; We assume that all inserted elements are unique.

$ClientNum \triangleq Cardinality(Client)$

$Priority \triangleq \text{CHOOSE } f \in [Client \to 1 \,..\, ClientNum] : Injective(f)$

ASSUME

$\land\ Range(InitState) \cap Char = \{\}$

$\land\ Priority \in [Client \to 1 \,..\, ClientNum]$

The set of all operations. Note: The positions are indexed from 1.

$Rd \triangleq [type : \{\text{"Rd"}\}]$

$Del \triangleq [type : \{\text{"Del"}\}, pos : 1 \,..\, MaxLen]$

$Ins \triangleq [type : \{\text{"Ins"}\}, pos : 1 \,..\, (MaxLen + 1), ch : Char, pr : 1 \,..\, ClientNum]$ *pr*: priority

$Op \triangleq Ins \cup Del$ Now we don't consider *Rd* operations.

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*

For model checking:

```
58      chins     a set of chars to insert

60  ----------------------------------------------------------------
61  comm ≜ INSTANCE CSComm
62  ----------------------------------------------------------------
63  eVars ≜ ⟨chins⟩                                  variables for the environment
64  cVars ≜ ⟨cbuf, crec, cstate⟩                     variables for the clients
65  ecVars ≜ ⟨eVars, cVars⟩                          variables for the clients and the environment
66  sVars ≜ ⟨sbuf, srec, sstate⟩                     variables for the server
67  commVars ≜ ⟨cincoming, sincoming⟩                variables for communication
68  vars ≜ ⟨eVars, cVars, sVars, commVars⟩           all variables
69  ----------------------------------------------------------------
70  TypeOK ≜
        For the client replicas:
74      ∧ cbuf ∈ [Client → Seq(Op ∪ {Nop})]
75      ∧ crec ∈ [Client → Int]
76      ∧ cstate ∈ [Client → List]
        For the server replica:
80      ∧ sbuf ∈ [Client → Seq(Op ∪ {Nop})]
81      ∧ srec ∈ [Client → Int]
82      ∧ sstate ∈ List
        For communication between the server and the clients:
86      ∧ comm!TypeOK
        For model checking:
90      ∧ chins ∈ SUBSET Char
91  ----------------------------------------------------------------
    The Init predicate.
95  Init ≜
96      ∧ chins = Char
        For the client replicas:
100     ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
101     ∧ crec = [c ∈ Client ↦ 0]
102     ∧ cstate = [c ∈ Client ↦ InitState]
        For the server replica:
106     ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
107     ∧ srec = [c ∈ Client ↦ 0]
108     ∧ sstate = InitState
        For communication between the server and the clients:
112     ∧ comm!Init
113 ----------------------------------------------------------------
    Client c ∈ Client issues an operation op.
117 DoOp(c, op) ≜
118         ∧ cstate′ = [cstate EXCEPT ![c] = Apply(op, @)]
```

```
119         ∧ cbuf' = [cbuf EXCEPT ![c] = Append(@, op)]
120         ∧ crec' = [crec EXCEPT ![c] = 0]
121         ∧ comm!CSend([c ↦ c, ack ↦ crec[c], op ↦ op])

123   DoIns(c) ≜
124       ∃ ins ∈ Ins :
125           ∧ ins.pos ∈ 1 .. (Len(cstate[c]) + 1)
126           ∧ ins.ch ∈ chins
127           ∧ ins.pr = Priority[c]
128           ∧ chins' = chins \ {ins.ch}  We assume that all inserted elements are unique.
129           ∧ DoOp(c, ins)
130           ∧ UNCHANGED sVars

132   DoDel(c) ≜
133       ∃ del ∈ Del :
134           ∧ del.pos ∈ 1 .. Len(cstate[c])
135           ∧ DoOp(c, del)
136           ∧ UNCHANGED ⟨sVars, eVars⟩

138   Do(c) ≜
139         ∨ DoIns(c)
140         ∨ DoDel(c)

      Client c ∈ Client receives a message from the Server.
145   Rev(c) ≜
146         ∧ comm!CRev(c)
147         ∧ crec' = [crec EXCEPT ![c] = @ + 1]
148         ∧ LET m ≜ Head(cincoming[c])
149               cBuf ≜ cbuf[c]  the buffer at client c ∈ Client
150               cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))  buffer shifted
151               xop ≜ XformOpOps(m.op, cShiftedBuf)  transform op vs. shifted buffer
152               xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)  transform shifted buffer vs. op
153           IN  ∧ cbuf' = [cbuf EXCEPT ![c] = xcBuf]
154               ∧ cstate' = [cstate EXCEPT ![c] = Apply(xop, @)]  apply the transformed operation xop
155         ∧ UNCHANGED ⟨sVars, eVars⟩
156 ├──────────────────────────────────────────────────────────────────────┤
      The Server receives a message.
160   SRev ≜
161       ∧ comm!SRev
162       ∧ LET m ≜ Head(sincoming)  the message to handle with
163             c ≜ m.c  the client c ∈ Client that sends this message
164             cBuf ≜ sbuf[c]  the buffer at the Server for client c ∈ Client
165             cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))  buffer shifted
166             xop ≜ XformOpOps(m.op, cShiftedBuf)  transform op vs. shifted buffer
167             xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)  transform shifted buffer vs. op
168         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 ecVars
```

The next-state relation.

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

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

```
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*)

```
\ * Modification History
\ * Last modified Thu Aug 30 21:44:10 CST 2018 by hengxin
\ * Created Sat Jun 23 17:14:18 CST 2018 by hengxin
```