1 MODULE *AJupiter*

Specification of the *Jupiter* protocol presented by *Attiya* and others.

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

7 ---

8 CONSTANTS

9 *Client*, the set of client replicas

10 *Server*, the (unique) server replica

11 *Char*, set of characters allowed

12 *InitState* the initial state of each replica

14 $Replica \triangleq Client \cup \{Server\}$

16 $List \triangleq Seq(Char \cup Range(InitState))$ all possible lists/strings

17 $MaxLen \triangleq Cardinality(Char) + Len(InitState)$ the max length of lists in any states;

18 We assume that all inserted elements are unique.

19 $ClientNum \triangleq Cardinality(Client)$

20 $Priority \triangleq \text{CHOOSE } f \in [Client \to 1 \,..\, ClientNum] : Injective(f)$

21 ---

22 ASSUME

23 $\land\ Range(InitState) \cap Char = \{\}$

24 $\land\ Priority \in [Client \to 1 \,..\, ClientNum]$

25 ---

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

30 $Rd \triangleq [type : \{\text{“Rd”}\}]$

31 $Del \triangleq [type : \{\text{“Del”}\}, pos : 1 \,..\, MaxLen]$

32 $Ins \triangleq [type : \{\text{“Ins”}\}, pos : 1 \,..\, (MaxLen + 1), ch : Char, pr : 1 \,..\, ClientNum]$ *pr*: priority

34 $Op \triangleq Ins \cup Del$ Now we don't consider *Rd* operations.

35 ---

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

40 $Msg \triangleq [c : Client, ack : Int, op : Op \cup \{Nop\}] \cup$ messages sent to the *Server* from a client $c \in Client$

41 $[ack : Int, op : Op \cup \{Nop\}]$ messages broadcast to Clients from the *Server*

42 ---

43 VARIABLES

For the client replicas:

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

48 *crec*, *crec*[*c*]: the number of new messages have been received by the client $c \in Client$

49 since the last time a message was sent

For the server replica:

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

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

For all replicas.

58 *state*, *state*[*r*]: state (the list content) of replica $r \in Replica$

```
                For communication between the Server and the Clients:
62              cincoming,      cincoming[c]: incoming channel at the client c ∈ Client
63              sincoming,      incoming channel at the Server
                For model checking:
67              chins           a set of chars to insert
68  ----------------------------------------------------------------------------
69  comm ≜ INSTANCE CSComm WITH Msg ← Msg
70  ----------------------------------------------------------------------------
71  eVars ≜ ⟨chins⟩                          variables for the environment
72  cVars ≜ ⟨cbuf, crec⟩                     variables for the clients
73  ecVars ≜ ⟨eVars, cVars⟩                  variables for the clients and the environment
74  sVars ≜ ⟨sbuf, srec⟩                     variables for the server
75  commVars ≜ ⟨cincoming, sincoming⟩        variables for communication
76  vars ≜ ⟨eVars, cVars, sVars, commVars, state⟩  all variables
77  ----------------------------------------------------------------------------
78  TypeOK ≜
                For the client replicas:
82              ∧ cbuf ∈ [Client → Seq(Op ∪ {Nop})]
83              ∧ crec ∈ [Client → Int]
                For the server replica:
87              ∧ sbuf ∈ [Client → Seq(Op ∪ {Nop})]
88              ∧ srec ∈ [Client → Int]
                For all replicas.
92              ∧ state ∈ [Replica → List]
                For communication between the server and the clients:
96              ∧ comm!TypeOK
                For model checking:
100             ∧ chins ∈ SUBSET Char
101 ----------------------------------------------------------------------------
    The Init predicate.
105 Init ≜
106             ∧ chins = Char
                For the client replicas:
110             ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
111             ∧ crec = [c ∈ Client ↦ 0]
                For the server replica:
115             ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
116             ∧ srec = [c ∈ Client ↦ 0]
                For all replicas.
120             ∧ state = [r ∈ Replica ↦ InitState]
                For communication between the server and the clients:
```

```
124     ∧ comm!Init
125 ├──────────────────────────────────────────────────────────────────────┤
    Client c ∈ Client issues an operation op.
129 DoOp(c, op) ≜
130     ∧ state' = [state EXCEPT ![c] = Apply(op, @)]
131     ∧ cbuf' = [cbuf EXCEPT ![c] = Append(@, op)]
132     ∧ crec' = [crec EXCEPT ![c] = 0]
133     ∧ comm!CSend([c ↦ c, ack ↦ crec[c], op ↦ op])

135 DoIns(c) ≜
136   ∃ ins ∈ Ins :
137     ∧ ins.pos ∈ 1 .. (Len(state[c]) + 1)
138     ∧ ins.ch ∈ chins
139     ∧ ins.pr = Priority[c]
140     ∧ chins' = chins \ {ins.ch}  We assume that all inserted elements are unique.
141     ∧ DoOp(c, ins)
142     ∧ UNCHANGED sVars

144 DoDel(c) ≜
145   ∃ del ∈ Del :
146     ∧ del.pos ∈ 1 .. Len(state[c])
147     ∧ DoOp(c, del)
148     ∧ UNCHANGED ⟨sVars, eVars⟩

150 Do(c) ≜
151     ∨ DoIns(c)
152     ∨ DoDel(c)

    Client c ∈ Client receives a message from the Server.
157 Rev(c) ≜
158     ∧ comm!CRev(c)
159     ∧ crec' = [crec EXCEPT ![c] = @ + 1]
160     ∧ LET m ≜ Head(cincoming[c])
161           cBuf ≜ cbuf[c]  the buffer at client c ∈ Client
162           cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))  buffer shifted
163           xop ≜ XformOpOps(m.op, cShiftedBuf)  transform op vs. shifted buffer
164           xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)  transform shifted buffer vs. op
165       IN  ∧ cbuf' = [cbuf EXCEPT ![c] = xcBuf]
166           ∧ state' = [state EXCEPT ![c] = Apply(xop, @)]  apply the transformed operation xop
167     ∧ UNCHANGED ⟨sVars, eVars⟩
168 ├──────────────────────────────────────────────────────────────────────┤
    The Server receives a message.
172 SRev ≜
173     ∧ comm!SRev
174     ∧ LET m ≜ Head(sincoming)  the message to handle with
```

```
175             c ≜ m.c                    the client c ∈ Client that sends this message
176             cBuf ≜ sbuf[c]             the buffer at the Server for client c ∈ Client
177             cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))  buffer shifted
178             xop ≜ XformOpOps(m.op, cShiftedBuf)  transform op vs. shifted buffer
179             xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)  transform shifted buffer vs. op
180         IN  ∧ srec' = [cl ∈ Client ↦
181                         IF cl = c
182                          THEN srec[cl] + 1  receive one more operation from client c ∈ Client
183                          ELSE 0]  reset srec for other clients than c ∈ Client
184             ∧ sbuf' = [cl ∈ Client ↦
185                         IF cl = c
186                          THEN xcBuf  transformed buffer for client c ∈ Client
187                          ELSE Append(sbuf[cl], xop)]  store transformed xop into other clients' bufs
188             ∧ state' = [state EXCEPT ![Server] = Apply(xop, @)]  apply the transformed operation
189             ∧ comm!SSend(c, [cl ∈ Client ↦ [ack ↦ srec[cl], op ↦ xop]])
190     ∧ UNCHANGED ecVars
191 ├──────────────────────────────────────────────────────────────────────┤
    The next-state relation.
195 Next ≜
196     ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
197     ∨ SRev
    The Spec. (TODO: Check the fairness condition.)
201 Spec ≜ Init ∧ □[Next]_vars ∧ WF_vars(Next)
202 ├──────────────────────────────────────────────────────────────────────┤
    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)
217 QC ≜ comm!EmptyChannel ⇒ Cardinality(Range(state)) = 1

219 THEOREM Spec ⇒ □QC

    Strong Eventual Consistency (SEC)
224 └──────────────────────────────────────────────────────────────────────┘
    \* Modification History
    \* Last modified Sun Sep 09 10:05:29 CST 2018 by hengxin
    \* Created Sat Jun 23 17:14:18 CST 2018 by hengxin
```