

```

1  |----- MODULE AJupiter -----|
   | Model checking the Jupiter protocol presented by Attiya and others. |
5  | EXTENDS OT, TLC |
6  |-----|
7  CONSTANTS
8      Client,      the set of client replicas
9      Server,      the (unique) server replica
10     State,      the initial state of each replica
11     Cop         Cop[c]: operations issued by the client c ∈ Client

13 ASSUME
14     ∧ State ∈ List
15     ∧ Cop ∈ [Client → Seq(Op)]

17 VARIABLES
   | For model checking: |
21     cop,         cop[c]: operations issued by the client c ∈ Client

   | For the client replicas: |
26     cbuf,      cbuf[c]: buffer (of operations) at the client c ∈ Client
27     crec,      crec[c]: the number of new messages have been received by the client c ∈ Client
28                 since the last time a message was sent
29     cstate,     cstate[c]: state (the list content) of the client c ∈ Client

   | For the server replica: |
34     sbuf,      sbuf[c]: buffer (of operations) at the Server, one per client c ∈ Client
35     srec,      srec[c]: the number of new messages have been ... , one per client c ∈ Client
36     sstate,     sstate: state (the list content) of the server Server

   | For communication between the Server and the Clients: |
41     cincoming, cincoming[c]: incoming channel at the client c ∈ Client
42     sincoming  incoming channel at the Server

43 |-----|
44 comm ≜ INSTANCE CSComm
45 |-----|
46 cVars ≜ ⟨cop, cbuf, crec, cstate⟩
47 sVars ≜ ⟨sbuf, srec, sstate⟩
48 vars ≜ cVars ∘ sVars ∘ comm! vars
49 |-----|
50 TypeOK ≜
51     ∧ cop ∈ [Client → Seq(Op)]
   | For the client replicas: |
55     ∧ cbuf ∈ [Client → Seq(Op ∪ {Nop})]
56     ∧ crec ∈ [Client → Nat]
57     ∧ cstate ∈ [Client → List]

```

```

For the server replica:
61   $\wedge sbuf \in [Client \rightarrow Seq(Op \cup \{Nop\})]$ 
62   $\wedge srec \in [Client \rightarrow Nat]$ 
63   $\wedge sstate \in List$ 
For communication between the server and the clients:
67   $\wedge comm!TypeOK$ 
68 |-----|
The Init predicate.
72 Init  $\triangleq$ 
73   $\wedge cop = Cop$ 
For the client replicas:
77   $\wedge cbuf = [c \in Client \mapsto \langle \rangle]$ 
78   $\wedge crec = [c \in Client \mapsto 0]$ 
79   $\wedge cstate = [c \in Client \mapsto State]$ 
For the server replica:
83   $\wedge sbuf = [c \in Client \mapsto \langle \rangle]$ 
84   $\wedge srec = [c \in Client \mapsto 0]$ 
85   $\wedge sstate = State$ 
For communication between the server and the clients:
89   $\wedge comm!Init$ 
90 |-----|
Client  $c \in Client$  issues an operation  $op$ .
94 Do( $c$ )  $\triangleq$ 
95   $\wedge cop[c] \neq \langle \rangle$ 
96   $\wedge LET\ op \triangleq Head(cop[c])$ 
97    IN   $\wedge PrintT(c \circ ": Do " \circ ToString(op))$ 
98         $\wedge cstate' = [cstate\ EXCEPT\ ![c] = Apply(op, @)]$ 
99         $\wedge cbuf' = [cbuf\ EXCEPT\ ![c] = Append(@, op)]$ 
100         $\wedge comm!CSend([c \mapsto c, ack \mapsto crec[c], op \mapsto op])$ 
101   $\wedge crec' = [crec\ EXCEPT\ ![c] = 0]$ 
102   $\wedge cop' = [cop\ EXCEPT\ ![c] = Tail(@)]$ 
103   $\wedge UNCHANGED\ sVars$ 

Client  $c \in Client$  receives a message from the Server.
108 CRev( $c$ )  $\triangleq$ 
109   $\wedge comm!CRev(c)$ 
110   $\wedge crec' = [crec\ EXCEPT\ ![c] = @ + 1]$ 
111   $\wedge LET\ m \triangleq Head(cincoming[c])$ 
112     $cBuf \triangleq cbuf[c] \setminus *$  the buffer at client  $c \in Client$ 
113     $cShiftedBuf \triangleq SubSeq(cBuf, m.ack + 1, Len(cBuf)) \setminus *$  buffer shifted
114     $xop \triangleq XformOpOps(m.op, cShiftedBuf) \setminus *$  transform  $op$  vs. shifted buffer
115     $xcBuf \triangleq XformOpsOp(cShiftedBuf, m.op) \setminus *$  transform shifted buffer vs.  $op$ 
116  IN   $\wedge cbuf' = [cbuf\ EXCEPT\ ![c] = xcBuf]$ 

```

```

117    $\wedge cstate' = [cstate \text{ EXCEPT } ![c] = \text{Apply}(xop, @)] \setminus *$  apply the transformed operation  $xop$ 
118    $\wedge \text{UNCHANGED } (sVars \circ \langle cop \rangle)$ 
119 |-----|
    The Server receives a message.
123  $SRev \triangleq$ 
124    $\wedge comm!SRev$ 
125    $\wedge \text{LET } m \triangleq \text{Head}(sincoming)$  the message to handle with
126      $c \triangleq m.c$  the client  $c \in Client$  that sends this message
127      $cBuf \triangleq sbuf[c]$  the buffer at the Server for client  $c \in Client$ 
128      $cShiftedBuf \triangleq \text{SubSeq}(cBuf, m.ack + 1, Len(cBuf))$  buffer shifted
129      $xop \triangleq \text{XformOpOps}(m.op, cShiftedBuf)$  transform  $op$  vs. shifted buffer
130      $xcBuf \triangleq \text{XformOpsOp}(cShiftedBuf, m.op)$  transform shifted buffer vs.  $op$ 
131   IN  $\wedge srec' = [cl \in Client \mapsto$ 
132     IF  $cl = c$ 
133       THEN  $srec[cl] + 1$  receive one more operation from client  $c \in Client$ 
134       ELSE 0] reset  $srec$  for other clients than  $c \in Client$ 
135    $\wedge sbuf' = [cl \in Client \mapsto$ 
136     IF  $cl = c$ 
137       THEN  $xcBuf$  transformed buffer for client  $c \in Client$ 
138       ELSE  $\text{Append}(sbuf[cl], xop)$  store transformed  $xop$  into other clients' bufs
139    $\wedge sstate' = \text{Apply}(xop, sstate)$  apply the transformed operation
140    $\wedge comm!SSend(c, srec, xop)$ 
141    $\wedge \text{UNCHANGED } cVars$ 
142 |-----|
    The next-state relation.
146  $Next \triangleq$ 
147    $\vee \exists c \in Client : Do(c)$ 
148    $\vee SRev$ 
    The Spec.
152  $Spec \triangleq Init \wedge \Box [Next]_{vars}$ 
153 |-----|
    \ * Modification History
    \ * Last modified Tue Jul 03 13:57:27 CST 2018 by hengxin
    \ * Created Sat Jun 23 17:14:18 CST 2018 by hengxin

```