

```

1  |----- MODULE CJupiter -----|
   | Model of our own CJupiter protocol. |
5  | EXTENDS Integers, OT, TLC, AdditionalFunctionOperators, AdditionalSequenceOperators |
6  |-----|
7  | CONSTANTS
8      Client,      the set of client replicas
9      Server,      the (unique) server replica
10     Char,        set of characters allowed
11     InitState    the initial state of each replica

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

15  List  $\triangleq$  Seq(Char  $\cup$  Range(InitState))    all possible lists/strings
16  MaxLen  $\triangleq$  Cardinality(Char) + Len(InitState)    the max length of lists in any states;
17      We assume that all inserted elements are unique.

19  ClientNum  $\triangleq$  Cardinality(Client)
20  Priority  $\triangleq$  CHOOSE  $f \in [Client \rightarrow 1 \dots ClientNum] : Injective(f)$ 
21  |-----|
22  | ASSUME
23       $\wedge$  Range(InitState)  $\cap$  Char = {}    due to the uniqueness requirement
24       $\wedge$  Priority  $\in [Client \rightarrow 1 \dots ClientNum]$ 
25  |-----|
   | The set of all operations. Note: The positions are indexed from 1. |
30  Rd  $\triangleq$  [type : {"Rd"}]
31  Del  $\triangleq$  [type : {"Del"}, pos : 1 .. MaxLen]
32  Ins  $\triangleq$  [type : {"Ins"}, pos : 1 .. (MaxLen + 1), ch : Char, pr : 1 .. ClientNum]    pr: priority
34  Op  $\triangleq$  Ins  $\cup$  Del
35  |-----|
   | Cop: operation of type Op with context |
39  Oid  $\triangleq$  [c : Client, seq : Nat]    operation identifier
40  Cop  $\triangleq$  [op : Op  $\cup$  {Nop}, oid : Oid, ctx : SUBSET Oid]

   | tb: Is cop1 totally ordered before cop2? |
   | This can be determined according to the serial view (sv) of any replica. |
47  tb(cop1, cop2, sv)  $\triangleq$ 
48      LET pos1  $\triangleq$  FirstIndexOfElementSafe(sv, cop1.oid)
49          pos2  $\triangleq$  FirstIndexOfElementSafe(sv, cop2.oid)
50      IN  IF pos1  $\neq$  0  $\wedge$  pos2  $\neq$  0    at the server or both are remote operations
51          THEN pos1 < pos2    at a client: one is a remote operation and the other is a local operation
52          ELSE pos1  $\neq$  0

   | OT of two operations of type Cop. |
56  COT(lcop, rcop)  $\triangleq$  [lcop EXCEPT !.op = Xform(lcop.op, rcop.op), !.ctx = @  $\cup$  {rcop.oid}]
57  |-----|

```

```

58  VARIABLES
    For the client replicas:
62    cseq,      cseq[c]: local sequence number at client  $c \in Client$ 
    For all replicas: the  $n$ -ary ordered state space
66    css,      css[r]: the  $n$ -ary ordered state space at replica  $r \in Replica$ 
67    cur,      cur[r]: the current node of css at replica  $r \in Replica$ 
68    state,    state[r]: state (the list content) of replica  $r \in Replica$ 
    For edge ordering in CSS
72    serial,   serial[r]: the serial view of replica  $r \in Replica$  about the server
73    cincomingSerial,
74    sincomingSerial,
    For communication between the Server and the Clients:
78    cincoming, cincoming[c]: incoming channel at the client  $c \in Client$ 
79    sincoming, sincoming: incoming channel at the Server
    For model checking:
83    chins    a set of chars to insert
84  |
85  serialVars  $\triangleq \langle serial, cincomingSerial, sincomingSerial \rangle$ 
86  vars  $\triangleq \langle chins, cseq, css, cur, state, cincoming, sincoming, serialVars \rangle$ 
87  |
88  comm  $\triangleq$  INSTANCE CSComm WITH Msg  $\leftarrow Cop$ 
89  commSerial  $\triangleq$  INSTANCE CSComm WITH Msg  $\leftarrow Seq(Oid)$ ,
90    cincoming  $\leftarrow cincomingSerial$ , sincoming  $\leftarrow sincomingSerial$ 
91  |
    A css is a directed graph with labeled edges, represented by a record with node field and edge field.
    Each node is characterized by its context, a set of oids. Each edge is labeled with an operation.
98  IsCSS(G)  $\triangleq$ 
99     $\wedge G = [node \mapsto G.node, edge \mapsto G.edge]$ 
100     $\wedge G.node \subseteq (SUBSET\ Oid)$ 
101     $\wedge G.edge \subseteq [from : G.node, to : G.node, cop : Cop]$ 
103  TypeOK  $\triangleq$ 
    For the client replicas:
107     $\wedge cseq \in [Client \rightarrow Nat]$ 
    For edge ordering in CSS:
111     $\wedge serial \in [Replica \rightarrow Seq(Oid)]$ 
112     $\wedge commSerial!TypeOK$ 
    For all replicas: the  $n$ -ary ordered state space
116     $\wedge \forall r \in Replica : IsCSS(css[r])$ 
117     $\wedge cur \in [Replica \rightarrow SUBSET\ Oid]$ 
118     $\wedge state \in [Replica \rightarrow List]$ 
    For communication between the server and the clients:

```

```

122     $\wedge comm!TypeOK$ 
      For model checking:
126     $\wedge chins \subseteq Char$ 
127  |-----|
      The Init predicate.
131  Init  $\triangleq$ 
      For the client replicas:
135     $\wedge cseq = [c \in Client \mapsto 0]$ 
      For the server replica:
139     $\wedge serial = [r \in Replica \mapsto \langle \rangle]$ 
140     $\wedge commSerial!Init$ 
      For all replicas: the n-ary ordered state space
144     $\wedge css = [r \in Replica \mapsto [node \mapsto \{\{\}\}, edge \mapsto \{\}]]$ 
145     $\wedge cur = [r \in Replica \mapsto \{\}]$ 
146     $\wedge state = [r \in Replica \mapsto InitState]$ 
      For communication between the server and the clients:
150     $\wedge comm!Init$ 
      For model checking:
154     $\wedge chins = Char$ 
155  |-----|
      Locate the node in rcss (the css at replica  $r \in Replica$ ) that matches the context ctx of cop.
159  Locate(cop, rcss)  $\triangleq$  CHOOSE  $n \in rcss.node : n = cop.ctx$ 
      Take union of two state spaces ss1 and ss2.
163  ss1  $\oplus$  ss2  $\triangleq$ 
164    [ss1 EXCEPT  $!.node = @ \cup ss2.node$ ,
165      $!.edge = @ \cup ss2.edge$ ]
      xForm: Iteratively transform cop with a path through the css at replica  $r \in Replica$ , following
      the first edges.
170  xForm(cop, r)  $\triangleq$ 
171    LET rcss  $\triangleq$  css[r]
172    u  $\triangleq$  Locate(cop, rcss)
173    v  $\triangleq$  u  $\cup$  {cop.oid}
174    RECURSIVE xFormHelper(u, u, u, u, u, u)
175    'h' stands for "helper"; xcss: eXtra css created during transformation
176    xFormHelper(uh, vh, coph, xcss, xcoph, xcurh)  $\triangleq$ 
177    IF uh = cur[r]
178    THEN  $\langle xcss, xcoph, xcurh \rangle$ 
179    ELSE LET fedge  $\triangleq$  CHOOSE  $e \in rcss.edge :$ 
180       $\wedge e.from = uh$ 
181       $\wedge \forall uhe \in rcss.edge :$ 
182         $(uhe.from = uh \wedge uhe \neq e) \Rightarrow tb(e.cop, uhe.cop, serial[r])$ 
183      uprime  $\triangleq$  fedge.to
184      fcop  $\triangleq$  fedge.cop

```

```

185          $coph2fcop \triangleq COT(coph, fcop)$ 
186          $fcop2coph \triangleq COT(fcop, coph)$ 
187          $vprime \triangleq vh \cup \{fcop.oid\}$ 
188     IN     $xFormHelper(uprime, vprime, coph2fcop,$ 
189            $[xcss \text{ EXCEPT } !.node = @ \cup \{vprime\},$ 
190            $!.edge = @ \cup \{[from \mapsto vh, to \mapsto vprime, cop \mapsto fcop2coph],$ 
191            $[from \mapsto uprime, to \mapsto vprime, cop \mapsto coph2fcop]\},$ 
192            $coph2fcop, vprime)$ 
193     IN     $xFormHelper(u, v, cop, [node \mapsto \{v\}, edge \mapsto \{[from \mapsto u, to \mapsto v, cop \mapsto cop]\}], cop, v)$ 
194     Perform cop at replica  $r \in Replica$ .
195
196      $Perform(cop, r) \triangleq$ 
197     LET  $xform \triangleq xForm(cop, r)$ 
198     LET  $xcss \triangleq xform[1]$ 
199     LET  $xcop \triangleq xform[2]$ 
200     LET  $xcur \triangleq xform[3]$ 
201     IN     $\wedge css' = [css \text{ EXCEPT } ![r] = @ \oplus xcss]$ 
202      $\wedge cur' = [cur \text{ EXCEPT } ![r] = xcur]$ 
203      $\wedge state' = [state \text{ EXCEPT } ![r] = Apply(xcop.op, @)]$ 
204
205 |-----|
206     Client  $c \in Client$  issues an operation  $op$ .
207
208      $DoOp(c, op) \triangleq$   $op$ : the raw operation generated by the client  $c \in Client$ 
209      $\wedge cseq' = [cseq \text{ EXCEPT } ![c] = @ + 1]$ 
210      $\wedge \text{LET } cop \triangleq [op \mapsto op, oid \mapsto [c \mapsto c, seq \mapsto cseq'[c]], ctx \mapsto cur[c]]$ 
211     IN     $\wedge Perform(cop, c)$ 
212      $\wedge comm! CSend(cop)$ 
213
214
215      $DoIns(c) \triangleq$ 
216      $\exists ins \in \{op \in Ins : op.pos \in 1 \dots (Len(state[c]) + 1) \wedge op.ch \in chins \wedge op.pr = Priority[c]\} :$ 
217      $\wedge DoOp(c, ins)$ 
218      $\wedge chins' = chins \setminus \{ins.ch\}$  We assume that all inserted elements are unique.
219      $\wedge \text{UNCHANGED } \langle serialVars \rangle$ 
220
221      $DoDel(c) \triangleq$ 
222      $\exists del \in \{op \in Del : op.pos \in 1 \dots Len(state[c])\} :$ 
223      $\wedge DoOp(c, del)$ 
224      $\wedge \text{UNCHANGED } \langle chins, serialVars \rangle$ 
225
226      $Do(c) \triangleq$ 
227      $\vee DoIns(c)$ 
228      $\vee DoDel(c)$ 
229
230     Client  $c \in Client$  receives a message from the Server.
231
232      $Rev(c) \triangleq$ 
233      $\wedge comm! CRev(c)$ 
234      $\wedge Perform(Head(cincoming[c]), c)$ 
235      $\wedge commSerial! CRev(c)$ 

```

```

236       $\wedge serial' = [serial \text{ EXCEPT } ![c] = Head(cincomingSerial[c])]$ 
237       $\wedge \text{UNCHANGED } \langle chins, cseq \rangle$ 
238 |-----|
      The Server receives a message.
242  $SRev \triangleq$ 
243    $\wedge comm!SRev$ 
244    $\wedge \text{LET } cop \triangleq Head(sincoming)$ 
245    $\text{IN } \wedge Perform(cop, Server)$ 
246    $\wedge comm!SSendSame(cop.oid.c, cop)$  broadcast the original operation
247    $\wedge serial' = [serial \text{ EXCEPT } ![Server] = Append(@, cop.oid)]$ 
248    $\wedge commSerial!SSendSame(cop.oid.c, serial'[Server])$ 
249    $\wedge \text{UNCHANGED } \langle chins, cseq, sincomingSerial \rangle$ 
250 |-----|
      The next-state relation.
254  $Next \triangleq$ 
255    $\vee \exists c \in Client : Do(c) \vee Rev(c)$ 
256    $\vee SRev$ 
      The Spec. There is no requirement that the clients ever generate operations.
261  $Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge WF_{vars}(SRev \vee \exists c \in Client : Rev(c))$ 
262 |-----|
      The compactness of CJupiter: the CSSes at all replicas are the same.
266  $Compactness \triangleq$ 
267    $comm!EmptyChannel \Rightarrow Cardinality(\{css[r] : r \in Replica\}) = 1$ 
269 THEOREM  $Spec \Rightarrow Compactness$ 
270 |-----|
      \ * Modification History
      \ * Last modified Sat Nov 10 22:46:01 CST 2018 by hengxin
      \ * Created Sat Sep 01 11:08:00 CST 2018 by hengxin

```