

```

1  |----- MODULE CJupiter -----|
   | Model of our own CJupiter protocol. |
6  EXTENDS Integers, OT, TLC, AdditionalFunctionOperators, AdditionalSequenceOperators
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.

20  ClientNum  $\triangleq$  Cardinality(Client)
21  Priority  $\triangleq$  CHOOSE  $f \in [Client \rightarrow 1 \dots ClientNum] : \text{Injective}(f)$ 
22  |-----|
23  ASSUME
24       $\wedge$  Range(InitState)  $\cap$  Char = {}    due to the uniqueness requirement
25       $\wedge$  Priority  $\in [Client \rightarrow 1 \dots ClientNum]$ 
26  |-----|
   | The set of all operations. Note: The positions are indexed from 1. |
31  Rd  $\triangleq$  [type : {"Rd"}]
32  Del  $\triangleq$  [type : {"Del"}, pos : 1 .. MaxLen]
33  Ins  $\triangleq$  [type : {"Ins"}, pos : 1 .. (MaxLen + 1), ch : Char, pr : 1 .. ClientNum]    pr: priority
35  Op  $\triangleq$  Ins  $\cup$  Del
36  |-----|
   | Cop: operation of type Op with context |
40  Oid  $\triangleq$  [c : Client, seq : Nat]    operation identifier
41  Cop  $\triangleq$  [op : Op  $\cup$  {Nop}, oid : Oid, ctx : SUBSET Oid, sctx : SUBSET Oid]

   | tb: Is cop1 totally ordered before cop2? |
   | At a given replica r  $\in$  Replica, these can be determined in terms of sctx. |
48  tb(cop1, cop2, r)  $\triangleq$ 
49       $\vee$  cop1.oid  $\in$  cop2.sctx
50       $\vee$   $\wedge$  cop1.oid  $\notin$  cop2.sctx
51           $\wedge$  cop2.oid  $\notin$  cop1.sctx
52           $\wedge$  cop1.oid.c  $\neq$  r

   | OT of two operations of type Cop. |
57  COT(lcop, rcop)  $\triangleq$  [lcop EXCEPT !.op = Xform(lcop.op, rcop.op), !.ctx = @  $\cup$  {rcop.oid}]
58  |-----|

```

59 VARIABLES

For the client replicas:

63 $cseq$, $cseq[c]$: local sequence number at client $c \in Client$

For the server replica:

67 $soids$, the set of operations the *Server* has executed

For all replicas: the n -ary ordered state space

71 css , $css[r]$: the n -ary ordered state space at replica $r \in Replica$

72 cur , $cur[r]$: the current node of css at replica $r \in Replica$

73 $state$, $state[r]$: state (the list content) of replica $r \in Replica$

For communication between the *Server* and the Clients:

77 $cincoming$, $cincoming[c]$: incoming channel at the client $c \in Client$

78 $sincoming$, incoming channel at the *Server*

For model checking:

82 $chins$ a set of chars to insert

84 |

85 $comm \triangleq \text{INSTANCE } CSComm \text{ WITH } Msg \leftarrow Cop$

86 |

87 $eVars \triangleq \langle chins \rangle$ variables for the environment

88 $cVars \triangleq \langle cseq \rangle$ variables for the clients

89 $sVars \triangleq \langle soids \rangle$ variables for the server

90 $dsVars \triangleq \langle css, cur, state \rangle$ variables for the data structure: the n -ary ordered state space

91 $commVars \triangleq \langle cincoming, sincoming \rangle$ variables for communication

92 $vars \triangleq \langle eVars, cVars, sVars, commVars, dsVars \rangle$ all variables

93 |

An css is a directed graph with labeled edges.

It is represented by a record with node field and edge field.

Each node is characterized by its context, a set of operations.

Each edge is labeled with an operation. For clarity, we denote edges by records instead of tuples.

104 $IsCSS(G) \triangleq$

105 $\wedge G = [node \mapsto G.node, edge \mapsto G.edge]$

106 $\wedge G.node \subseteq (\text{SUBSET } Oid)$

107 $\wedge G.edge \subseteq [from : G.node, to : G.node, cop : Cop]$

109 $TypeOK \triangleq$

For the client replicas:

113 $\wedge cseq \in [Client \rightarrow Nat]$

For the server replica:

117 $\wedge soids \subseteq Oid$

For all replicas: the n -ary ordered state space

121 $\wedge \forall r \in Replica : IsCSS(css[r])$

122 $\wedge cur \in [Replica \rightarrow \text{SUBSET } Oid]$

```

123     $\wedge state \in [Replica \rightarrow List]$ 
      For communication between the server and the clients:
127     $\wedge comm!TypeOK$ 
      For model checking:
131     $\wedge chins \subseteq Char$ 
132  |-----|
      The Init predicate.
136  Init  $\triangleq$ 
      For the client replicas:
140     $\wedge cseq = [c \in Client \mapsto 0]$ 
      For the server replica:
144     $\wedge soids = \{\}$ 
      For all replicas: the n-ary ordered state space
148     $\wedge css = [r \in Replica \mapsto [node \mapsto \{\{\}\}, edge \mapsto \{\}]]$ 
149     $\wedge cur = [r \in Replica \mapsto \{\}]$ 
150     $\wedge state = [r \in Replica \mapsto InitState]$ 
      For communication between the server and the clients:
154     $\wedge comm!Init$ 
      For model checking:
158     $\wedge chins = Char$ 
159  |-----|
      Locate the node in rcss which matches the context ctx of cop.
      rcss: the css at replica  $r \in Replica$ 
165  Locate(cop, rcss)  $\triangleq$  CHOOSE  $n \in (rcss.node) : n = cop.ctx$ 

      xForm: iteratively transform cop with a path through the css at replica  $r \in Replica$ , following
      the first edges.
171  xForm(cop, r)  $\triangleq$ 
172    LET rcss  $\triangleq$  css[r]
173    u  $\triangleq$  Locate(cop, rcss)
174    v  $\triangleq$   $u \cup \{cop.oid\}$ 
175    RECURSIVE xFormHelper(u, u, u, u)
176    'h' stands for "helper"; xcss: eXtra css created during transformation
177    xFormHelper(uh, vh, coph, xcss)  $\triangleq$ 
178      IF  $uh = cur[r]$ 
179      THEN xcss
180      ELSE LET fedge  $\triangleq$  CHOOSE  $e \in rcss.edge :$ 
181         $\wedge e.from = uh$ 
182         $\wedge \forall uhe \in rcss.edge :$ 
183           $(uhe.from = uh \wedge uhe \neq e) \Rightarrow tb(e.cop, uhe.cop, r)$ 
184        uprime  $\triangleq$  fedge.to
185        fcop  $\triangleq$  fedge.cop

```

```

186       $coph2fcop \triangleq COT(coph, fcop)$ 
187       $fcop2coph \triangleq COT(fcop, coph)$ 
188       $vprime \triangleq vh \cup \{fcop.oid\}$ 
189      IN  $xFormHelper(uprime, vprime, coph2fcop,$ 
190           $[xcss \text{ EXCEPT } !.node = @ \circ \langle vprime \rangle,$ 
191               $\text{the order of recording edges here is important}$ 
192               $!.edge = @ \circ \langle [from \mapsto vh, to \mapsto vprime, cop \mapsto fcop2coph],$ 
193                   $[from \mapsto uprime, to \mapsto vprime, cop \mapsto coph2fcop] \rangle])$ 
194      IN  $xFormHelper(u, v, cop, [node \mapsto \langle v \rangle,$ 
195           $edge \mapsto \langle [from \mapsto u, to \mapsto v, cop \mapsto cop] \rangle])$ 

Perform cop at replica  $r \in Replica$ .
200  $Perform(cop, r) \triangleq$ 
201     LET  $xcss \triangleq xForm(cop, r)$ 
202      $xn \triangleq xcsc.node$ 
203      $xe \triangleq xcsc.edge$ 
204      $xcur \triangleq Last(xn)$ 
205      $xcop \triangleq Last(xe).cop$ 
206     IN  $\wedge css' = [css \text{ EXCEPT } ![r].node = @ \cup Range(xn),$ 
207          $![r].edge = @ \cup Range(xe)]$ 
208          $\wedge cur' = [cur \text{ EXCEPT } ![r] = xcur]$ 
209          $\wedge state' = [state \text{ EXCEPT } ![r] = Apply(xcop.op, @)]$ 

210 |-----|
Client  $c \in Client$  issues an operation  $op$ .
214  $DoOp(c, op) \triangleq$   $op$ : the raw operation generated by the client  $c \in Client$ 
215      $\wedge cseq' = [cseq \text{ EXCEPT } ![c] = @ + 1]$ 
216      $\wedge \text{LET } cop \triangleq [op \mapsto op, oid \mapsto [c \mapsto c, seq \mapsto cseq'[c]], ctx \mapsto cur[c], scctx \mapsto \{\}]$ 
217     IN  $\wedge Perform(cop, c)$ 
218      $\wedge comm!CSend(cop)$ 

220  $DoIns(c) \triangleq$ 
221      $\exists ins \in Ins :$ 
222      $\wedge ins.pos \in 1 \dots (Len(state[c]) + 1)$ 
223      $\wedge ins.ch \in chins$ 
224      $\wedge ins.pr = Priority[c]$ 
225      $\wedge chins' = chins \setminus \{ins.ch\}$   $\text{We assume that all inserted elements are unique.}$ 
226      $\wedge DoOp(c, ins)$ 
227      $\wedge \text{UNCHANGED } sVars$ 

229  $DoDel(c) \triangleq$ 
230      $\exists del \in Del :$ 
231      $\wedge del.pos \in 1 \dots Len(state[c])$ 
232      $\wedge DoOp(c, del)$ 
233      $\wedge \text{UNCHANGED } \langle sVars, eVars \rangle$ 

235  $Do(c) \triangleq$ 

```

```

236       $\vee DoIns(c)$ 
237       $\vee DoDel(c)$ 
    Client  $c \in Client$  receives a message from the Server.
241   $Rev(c) \triangleq$ 
242       $\wedge comm!CRev(c)$ 
243       $\wedge LET\ cop \triangleq Head(cincoming[c])$  the received original operation
244      IN  $Perform(cop, c)$ 
245       $\wedge UNCHANGED \langle eVars, cVars, sVars \rangle$ 
246 |-----|
    The Server receives a message.
250   $SRev \triangleq$ 
251       $\wedge comm!SRev$ 
252       $\wedge LET\ cop \triangleq [Head(sincoming) EXCEPT !.sctx = soids]$  set its sctx field
253      IN  $\wedge soids' = soids \cup \{cop.oid\}$ 
254       $\wedge Perform(cop, Server)$ 
255       $\wedge comm!SSendSame(cop.oid.c, cop)$  broadcast the original operation
256       $\wedge UNCHANGED \langle eVars, cVars \rangle$ 
257 |-----|
    The next-state relation.
261   $Next \triangleq$ 
262       $\vee \exists c \in Client : Do(c) \vee Rev(c)$ 
263       $\vee SRev$ 
    The Spec. (TODO: Check the fairness condition.)
267   $Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge WF_{vars}(Next)$ 
268 |-----|
    The compactness of CJupiter: the css at all replicas are essentially the same.
273   $IgnoreSctx(rcss) \triangleq$ 
274       $[rcss EXCEPT !.edge = \{[e EXCEPT !.cop.sctx = \{\}]: e \in @\}]$ 
276   $Compactness \triangleq$ 
277       $comm!EmptyChannel \Rightarrow Cardinality(\{IgnoreSctx(css[r]): r \in Replica\}) = 1$ 
279  THEOREM  $Spec \Rightarrow Compactness$ 
280 |-----|
    \ * Modification History
    \ * Last modified Wed Oct 24 10:41:24 CST 2018 by hengxin
    \ * Created Sat Sep 01 11:08:00 CST 2018 by hengxin

```