

```

1  |----- MODULE XJupiterExtended -----|
   | XJupiter extended with Cop with the sctx field. |
5  | EXTENDS Integers, OT, TLCUtils, 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  $f \in [Client \rightarrow 1 \dots ClientNum] : Injective(f)$ 

22  direction flags
23  Local  $\triangleq$  0
24  Remote  $\triangleq$  1

25  |-----|
26  ASSUME
27       $\wedge Range(InitState) \cap Char = \{\}$     due to the uniqueness requirement
28       $\wedge Priority \in [Client \rightarrow 1 \dots ClientNum]$ 

29  |-----|
   | The set of all operations. Note: The positions are indexed from 1. |
34  Rd  $\triangleq$  [type : { "Rd" }]
35  Del  $\triangleq$  [type : { "Del" }, pos : 1 .. MaxLen]
36  Ins  $\triangleq$  [type : { "Ins" }, pos : 1 .. (MaxLen + 1), ch : Char, pr : 1 .. ClientNum]    pr: priority

38  Op  $\triangleq$  Ins  $\cup$  Del

39  |-----|
   | Cop: operation of type Op with context |
43  Oid  $\triangleq$  [c : Client, seq : Nat]    operation identifier
   | Cop with the sctx field (the extended part) |
47  Cop  $\triangleq$  [op : Op  $\cup$  {Nop}, oid : Oid, ctx : SUBSET Oid, sctx : SUBSET Oid]

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

53  |-----|
54  VARIABLES
   | For the client replicas: |
58  cseq,    cseq[c]: local sequence number at client c  $\in$  Client

```

For the server replica (the extended part):

62 *soids*, the set of operations the *Server* has executed

The 2D state spaces (*ss*, for short). Each client maintains one 2D state space. The server maintains *n* 2D state spaces, one for each client.

68 *c2ss*, *c2ss*[*c*]: the 2D state space at client *c* ∈ *Client*

69 *ccur*, *cur*[*c*]: the current node of *c2ss*[*c*]

70 *s2ss*, *s2ss*[*c*]: the 2D state space maintained by the *Server* for client *c* ∈ *Client*

71 *scur*, *scur*[*c*]: the current node of *s2ss*[*c*]

For all replicas

75 *state*, *state*[*r*]: state (the list content) of replica *r* ∈ *Replica*

For communication between the *Server* and the *Clients*:

79 *cincoming*, *cincoming*[*c*]: incoming channel at the client *c* ∈ *Client*

80 *sincoming*, incoming channel at the *Server*

For model checking:

84 *chins* a set of chars to insert

85

86 *comm* \triangleq INSTANCE *CSComm* WITH *Msg* \leftarrow *Cop*

87

88 *eVars* \triangleq \langle *chins* \rangle variables for the environment

89 *cVars* \triangleq \langle *cseq* \rangle variables for the clients

90 *sVars* \triangleq \langle *soids* \rangle variables for the *Server*

91 *c2ssVars* \triangleq \langle *c2ss*, *ccur* \rangle variables for 2D state spaces at clients

92 *s2ssVars* \triangleq \langle *s2ss*, *scur* \rangle variables for 2D state spaces at the *Server*

93 *commVars* \triangleq \langle *cincoming*, *sincoming* \rangle variables for communication

94 *vars* \triangleq \langle *eVars*, *cVars*, *sVars*, *commVars*, *c2ssVars*, *s2ssVars*, *state* \rangle all variables

95

A 2D state space 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 and a direction flag indicating whether this edge is LOCAL or REMOTE. For clarity, we denote edges by records instead of tuples.

104 *IsSS*(*G*) \triangleq

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

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

107 $\wedge G.edge \subseteq [from : G.node, to : G.node, cop : Cop, lr : \{Local, Remote\}]$

109 *TypeOK* \triangleq

For the client replicas:

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

For the 2D state spaces:

117 $\wedge \forall c \in Client : IsSS(c2ss[c]) \wedge IsSS(s2ss[c])$

118 $\wedge ccur \in [Client \rightarrow \text{SUBSET } Oid]$

119 $\wedge scur \in [Client \rightarrow \text{SUBSET } Oid]$

120 $\wedge state \in [Replica \rightarrow List]$

For communication between the server and the clients:

```

124     $\wedge comm!TypeOK$ 
      For model checking:
128     $\wedge chins \subseteq Char$ 
129  |-----|
      The Init predicate.
133  Init  $\triangleq$ 
      For the client replicas:
137     $\wedge cseq = [c \in Client \mapsto 0]$ 
      For the Server replica (the extended part):
141     $\wedge soids = \{\}$ 
      For the 2D state spaces:
145     $\wedge c2ss = [c \in Client \mapsto [node \mapsto \{\{\}\}, edge \mapsto \{\}]]$ 
146     $\wedge ccur = [c \in Client \mapsto \{\}]$ 
147     $\wedge s2ss = [c \in Client \mapsto [node \mapsto \{\{\}\}, edge \mapsto \{\}]]$ 
148     $\wedge scur = [c \in Client \mapsto \{\}]$ 
      For all replicas:
152     $\wedge state = [r \in Replica \mapsto InitState]$ 
      For communication between the server and the clients:
156     $\wedge comm!Init$ 
      For model checking:
160     $\wedge chins = Char$ 
161  |-----|
      Locate the node in the 2D state space ss which matches the context ctx of cop.
165  Locate(cop, ss)  $\triangleq$  CHOOSE  $n \in (ss.node) : n = cop.ctx$ 

      xForm: iteratively transform cop with a path through the 2D state space ss at some client,
      following the edges with the direction flag d.
172  xForm(cop, ss, cur, d)  $\triangleq$ 
173    LET  $u \triangleq Locate(cop, ss)$ 
174     $v \triangleq u \cup \{cop.oid\}$ 
175    RECURSIVE xFormHelper( $-, -, -, -$ )
176    'h' stands for "helper"; xss: eXtra ss created during transformation
177    xFormHelper(uh, vh, coph, xss)  $\triangleq$ 
178    IF uh = cur
179    THEN xss
180    ELSE LET  $e \triangleq$  CHOOSE  $e \in ss.edge : e.from = uh \wedge e.lr = d$ 
181     $uprime \triangleq e.to$ 
182     $copprime \triangleq e.cop$ 
183     $coph2copprime \triangleq COT(coph, copprime)$ 
184     $copprime2coph \triangleq COT(copprime, coph)$ 
185     $vprime \triangleq vh \cup \{copprime.oid\}$ 
186    IN xFormHelper(uprime, vprime, coph2copprime,
187    [xss EXCEPT  $!.node = @ \circ \langle vprime \rangle$ ],

```

188 the order of recording edges here is important
189 so that the last one is labeled with the final transformed operation
190 $!.edge = @ \circ \langle [from \mapsto vh, to \mapsto vprime, cop \mapsto copprime2coph, lr \mapsto$
191 $[from \mapsto uprime, to \mapsto vprime, cop \mapsto coph2copprime,$
192 IN $xFormHelper(u, v, cop, [node \mapsto \langle v \rangle,$
193 $edge \mapsto \langle [from \mapsto u, to \mapsto v, cop \mapsto cop, lr \mapsto 1 - d] \rangle]$
194 \vdash Client $c \in Client$ perform operation cop guided by the direction flag d .

198 $ClientPerform(cop, c, d) \triangleq$
199 LET $xss \triangleq xForm(cop, c2ss[c], ccur[c], d)$
200 $xn \triangleq xss.node$
201 $xe \triangleq xss.edge$
202 $xcur \triangleq Last(xn)$
203 $xcop \triangleq Last(xe).cop$
204 IN $\wedge c2ss' = [c2ss \text{ EXCEPT } ![c].node = @ \cup Range(xn),$
205 $![c].edge = @ \cup Range(xe)]$
206 $\wedge ccur' = [ccur \text{ EXCEPT } ![c] = xcur]$
207 $\wedge state' = [state \text{ EXCEPT } ![c] = Apply(xcop.op, @)]$

Client $c \in Client$ issues an operation op .

211 $DoOp(c, op) \triangleq$ op : the raw operation generated by the client $c \in Client$
212 $\wedge cseq' = [cseq \text{ EXCEPT } ![c] = @ + 1]$
213 op with the $sctx$ field (the extended part)
214 $\wedge \text{LET } cop \triangleq [op \mapsto op, oid \mapsto [c \mapsto c, seq \mapsto cseq'[c]], ctx \mapsto ccur[c], sctx \mapsto \{\}]$
215 IN $\wedge ClientPerform(cop, c, Remote)$
216 $\wedge comm!CSend(cop)$

218 $DoIns(c) \triangleq$
219 $\exists ins \in Ins :$
220 $\wedge ins.pos \in 1 \dots (Len(state[c]) + 1)$
221 $\wedge ins.ch \in chins$
222 $\wedge ins.pr = Priority[c]$
223 $\wedge chins' = chins \setminus \{ins.ch\}$ We assume that all inserted elements are unique.
224 $\wedge DoOp(c, ins)$
225 $\wedge \text{UNCHANGED } \langle sVars, s2ssVars \rangle$

227 $DoDel(c) \triangleq$
228 $\exists del \in Del :$
229 $\wedge del.pos \in 1 \dots Len(state[c])$
230 $\wedge DoOp(c, del)$
231 $\wedge \text{UNCHANGED } \langle sVars, s2ssVars, eVars \rangle$

233 $Do(c) \triangleq$
234 $\vee DoIns(c)$
235 $\vee DoDel(c)$

Client $c \in Client$ receives a message from the *Server*.

```

239  $Rev(c) \triangleq$ 
240    $\wedge comm!CRev(c)$ 
241    $\wedge LET\ cop \triangleq Head(cincoming[c])$  the received (transformed) operation
242      $IN\ ClientPerform(cop, c, Local)$ 
243    $\wedge UNCHANGED \langle eVars, cVars, sVars, s2ssVars \rangle$ 
244 |-----|
245 | The Server performs operation cop.
246 |
247  $ServerPerform(cop) \triangleq$ 
248    $LET\ c \triangleq cop.oid.c$ 
249    $xss \triangleq xForm(cop, s2ss[c], scur[c], Remote)$ 
250    $xn \triangleq xss.node$ 
251    $xe \triangleq xss.edge$ 
252    $xcur \triangleq Last(xn)$ 
253    $xcop \triangleq Last(xe).cop$ 
254    $IN\ \wedge s2ss' = [cl \in Client \mapsto$ 
255      $IF\ cl = c$ 
256        $THEN\ [s2ss[cl]\ EXCEPT\ !.node = @ \cup Range(xn),$ 
257          $!\ .edge = @ \cup Range(xe)]$ 
258        $ELSE\ LET\ scurcl \triangleq scur[cl]$ 
259          $scurclprime \triangleq scurcl \cup \{cop.oid\}$ 
260          $IN\ [s2ss[cl]\ EXCEPT\ !.node = @ \cup \{scurclprime\},$ 
261            $!\ .edge = @ \cup \{[from \mapsto scurcl, to \mapsto scurclprime,$ 
262              $cop \mapsto xcop, lr \mapsto Remote]\}]$ 
263        $]$ 
264    $\wedge scur' = [cl \in Client \mapsto$ 
265      $IF\ cl = c\ THEN\ xcur\ ELSE\ scur[cl] \cup \{cop.oid\}]$ 
266    $\wedge state' = [state\ EXCEPT\ ![Server] = Apply(xcop.op, @)]$ 
267    $\wedge comm!SSendSame(c, xcop)$  broadcast the transformed operation
268 |-----|
269 | The Server receives a message.
270 |
271  $SRev \triangleq$ 
272    $\wedge comm!SRev$ 
273    $\wedge LET\ cop \triangleq [Head(sincoming)\ EXCEPT\ !.sctx = soids]$ 
274     cop with the sctx field (the extended part)
275    $IN\ \wedge soids' = soids \cup \{cop.oid\}$ 
276      $\wedge ServerPerform(cop)$ 
277    $\wedge UNCHANGED \langle eVars, cVars, c2ssVars \rangle$ 
278 |-----|
279 | The next-state relation.
280 |
281  $Next \triangleq$ 
282    $\vee \exists c \in Client : Do(c) \vee Rev(c)$ 
283    $\vee SRev$ 
284 |-----|
285 | The Spec. (TODO: Check the fairness condition.)
286 |
287  $Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge WF_{vars}(Next)$ 
288 |-----|
289
290

```

* Modification History
* *Last* modified *Wed Oct 31 19:02:50 CST 2018* by *hengxin*
* Created *Tue Oct 30 20:32:27 CST 2018* by *hengxin*