```
1  ---------- MODULE CJupiter ----------
   Model of our own CJupiter protocol.
5  EXTENDS JupiterInterface
6  ----------
   Cop: operation of type Op with context
10 Oid ≜ [c : Client, seq : Nat]   operation identifier
11 Cop ≜ [op : Op ∪ {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.
18 tb(cop1, cop2, sv) ≜
19     LET pos1 ≜ FirstIndexOfElementSafe(sv, cop1.oid)
20         pos2 ≜ FirstIndexOfElementSafe(sv, cop2.oid)
21      IN IF pos1 ≠ 0 ∧ pos2 ≠ 0   at the server or both are remote operations
22         THEN pos1 < pos2         at a client: one is a remote operation and the other is a local operation
23         ELSE pos1 ≠ 0
   OT of two operations of type Cop.
27 COT(lcop, rcop) ≜ [lcop EXCEPT !.op = Xform(lcop.op, rcop.op), !.ctx = @ ∪ {rcop.oid}]
28 ----------
29 VARIABLES
      For the client replicas:
33    cseq,     cseq[c]: local sequence number at client c ∈ Client
      For all replicas: the n-ary ordered state space
37    css,      css[r]: the n-ary ordered state space at replica r ∈ Replica
38    cur,      cur[r]: the current node of css at replica r ∈ Replica
39    state,    state[r]: state (the list content) of replica r ∈ Replica
      For edge ordering in CSS
43    serial,   serial[r]: the serial view of replica r ∈ Replica about the server
44    cincomingSerial,
45    sincomingSerial,
      For communication between the Server and the Clients:
49    cincoming,   cincoming[c]: incoming channel at the client c ∈ Client
50    sincoming,   incoming channel at the Server
      For model checking:
54    chins     a set of chars to insert
55 ----------
56 serialVars ≜ ⟨serial, cincomingSerial, sincomingSerial⟩
57 vars ≜ ⟨chins, cseq, css, cur, state, cincoming, sincoming, serialVars⟩
58 ----------
59 comm ≜ INSTANCE CSComm WITH Msg ← Cop
60 commSerial ≜ INSTANCE CSComm WITH Msg ← Seq(Oid),
61                  cincoming ← cincomingSerial, sincoming ← sincomingSerial
62 ----------
```

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.

69 $IsCSS(G) \triangleq$
70 $\quad \land\ G = [node \mapsto G.node,\ edge \mapsto G.edge]$
71 $\quad \land\ G.node \subseteq (\text{SUBSET}\ Oid)$
72 $\quad \land\ G.edge \subseteq [from : G.node,\ to : G.node,\ cop : Cop]$

74 $EmptySS \triangleq [node \mapsto \{\{\}\},\ edge \mapsto \{\}]$

76 $TypeOK \triangleq$
For the client replicas:
80 $\quad \land\ cseq \in [Client \to Nat]$
For edge ordering in *CSS*:
84 $\quad \land\ serial \in [Replica \to Seq(Oid)]$
85 $\quad \land\ commSerial\,!\,TypeOK$
For all replicas: the n-ary ordered state space
89 $\quad \land\ \forall\, r \in Replica : IsCSS(css[r])$
90 $\quad \land\ cur \in [Replica \to \text{SUBSET}\ Oid]$
91 $\quad \land\ state \in [Replica \to List]$
For communication between the server and the clients:
95 $\quad \land\ comm\,!\,TypeOK$
For model checking:
99 $\quad \land\ chins \subseteq Char$

100 ⊢———————————————————————————————————⊣

101 $Init \triangleq$
For the client replicas:
105 $\quad \land\ cseq = [c \in Client \mapsto 0]$
For the server replica:
109 $\quad \land\ serial = [r \in Replica \mapsto \langle\rangle]$
110 $\quad \land\ commSerial\,!\,Init$
For all replicas: the n-ary ordered state space
114 $\quad \land\ css\ = [r \in Replica \mapsto EmptySS]$
115 $\quad \land\ cur = [r \in Replica \mapsto \{\}]$
116 $\quad \land\ state = [r \in Replica \mapsto InitState]$
For communication between the server and the clients:
120 $\quad \land\ comm\,!\,Init$
For model checking:
124 $\quad \land\ chins = Char$

125 ⊢———————————————————————————————————⊣

Locate the node in *rcss* (the *css* at replica $r \in Replica$) that matches the context *ctx* of cop.

129 $Locate(cop,\ rcss) \triangleq \text{CHOOSE}\ n \in rcss.node : n = cop.ctx$

Take union of two state spaces *ss*1 and *ss*2.

```
ss1 ⊕ ss2 ≜ [node ↦ ss1.node ∪ ss2.node, edge ↦ ss1.edge ∪ ss2.edge]
```

xForm: Iteratively transform cop with a path through the *css* at replica $r \in Replica$, following the first edges.

```
xForm(cop, r) ≜
    LET rcss ≜ css[r]
        u ≜ Locate(cop, rcss)
        v ≜ u ∪ {cop.oid}
        RECURSIVE xFormHelper(_, _, _, _, _, _)
         'h' stands for "helper"; xcss: eXtra css created during transformation
        xFormHelper(uh, vh, coph, xcss, xcoph, xcurh) ≜
            IF uh = cur[r]
             THEN ⟨xcss, xcoph, xcurh⟩
             ELSE LET fedge ≜ CHOOSE e ∈ rcss.edge :
                                  ∧ e.from = uh
                                  ∧ ∀ uhe ∈ rcss.edge :
                                      (uhe.from = uh ∧ uhe ≠ e) ⇒ tb(e.cop, uhe.cop, serial[r])
                      uprime ≜ fedge.to
                      fcop ≜ fedge.cop
                      coph2fcop ≜ COT(coph, fcop)
                      fcop2coph ≜ COT(fcop, coph)
                       vprime ≜ vh ∪ {fcop.oid}
                  IN   xFormHelper(uprime, vprime, coph2fcop,
                         [xcss EXCEPT !.node = @ ∪ {vprime},
                           !.edge = @ ∪ {[from ↦ vh, to ↦ vprime, cop ↦ fcop2coph],
                                         [from ↦ uprime, to ↦ vprime, cop ↦ coph2fcop]}],
                         coph2fcop, vprime)
    IN   xFormHelper(u, v, cop, [node ↦ {v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop]}], cop, v)
```

Perform cop at replica $r \in Replica$.

```
Perform(cop, r) ≜
    LET xform ≜ xForm(cop, r)   xform: ⟨xcss, xcop, xcur⟩
        xcss ≜ xform[1]
        xcop ≜ xform[2]
        xcur ≜ xform[3]
    IN  ∧ css' = [css EXCEPT ![r] = @ ⊕ xcss]
        ∧ cur' = [cur EXCEPT ![r] = xcur]
        ∧ state' = [state EXCEPT ![r] = Apply(xcop.op, @)]
```

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

```
DoOp(c, op) ≜   op: the raw operation generated by the client c ∈ Client
    ∧ cseq' = [cseq EXCEPT ![c] = @ + 1]
    ∧ LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq'[c]], ctx ↦ cur[c]]
      IN  ∧ Perform(cop, c)
          ∧ comm!CSend(cop)

DoIns(c) ≜
```

```
184     ∃ ins ∈ {op ∈ Ins : op.pos ∈ 1 .. (Len(state[c]) + 1) ∧ op.ch ∈ chins ∧ op.pr = Priority[c]} :
185         ∧ DoOp(c, ins)
186         ∧ chins′ = chins \ {ins.ch}  We assume that all inserted elements are unique.
187         ∧ UNCHANGED ⟨serialVars⟩

189 DoDel(c) ≜
190     ∃ del ∈ {op ∈ Del : op.pos ∈ 1 .. Len(state[c])} :
191         ∧ DoOp(c, del)
192         ∧ UNCHANGED ⟨chins, serialVars⟩

194 Do(c) ≜
195         ∨ DoIns(c)
196         ∨ DoDel(c)
    Client c ∈ Client receives a message from the Server.
200 Rev(c) ≜
201         ∧ comm!CRev(c)
202         ∧ Perform(Head(cincoming[c]), c)
203         ∧ commSerial!CRev(c)
204         ∧ serial′ = [serial EXCEPT ![c] = Head(cincomingSerial[c])]
205         ∧ UNCHANGED ⟨chins, cseq⟩
206 ──────────────────────────────────────────────────────────────────
    The Server receives a message.
210 SRev ≜
211     ∧ comm!SRev
212     ∧ LET cop ≜ Head(sincoming)
213       IN  ∧ Perform(cop, Server)
214           ∧ comm!SSendSame(cop.oid.c, cop)  broadcast the original operation
215           ∧ serial′ = [serial EXCEPT ![Server] = Append(@, cop.oid)]
216           ∧ commSerial!SSendSame(cop.oid.c, serial′[Server])
217     ∧ UNCHANGED ⟨chins, cseq, sincomingSerial⟩
218 ──────────────────────────────────────────────────────────────────
219 Next ≜
220     ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
221     ∨ SRev
    Fairness: There is no requirement that the clients ever generate operations.
225 Fairness ≜
226     WF_vars(SRev ∨ ∃ c ∈ Client : Rev(c))

228 Spec ≜ Init ∧ □[Next]_vars  ∧ Fairness (We care more about safety.)
229 ──────────────────────────────────────────────────────────────────
    The compactness of CJupiter: the CSSes at all replicas are the same.
233 Compactness ≜
234     comm!EmptyChannel ⇒ Cardinality(Range(css)) = 1

236 THEOREM Spec ⇒ Compactness
```

```
237 └──────────────────────────────────────────────────────────────────────┘
    \ * Modification History
    \ * Last modified Tue Dec 04 19:34:24 CST 2018 by hengxin
    \ * Created Sat Sep 01 11:08:00 CST 2018 by hengxin
```