```
1  ---------- MODULE CJupiter ----------
   Model of our own CJupiter protocol.
5  EXTENDS JupiterSerial
6  ----------
7  VARIABLES
8      cseq,     cseq[c]: local sequence number at client c ∈ Client
       For all replicas: the n-ary ordered state space
12     css,      css[r]: the n-ary ordered state space at replica r ∈ Replica
13     cur       cur[r]: the current node of css at replica r ∈ Replica

15 vars ≜ ⟨chins, cseq, css, cur, state, cincoming, sincoming, serialVars⟩
16 ----------
   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.
23 IsCSS(G) ≜
24     ∧ G = [node ↦ G.node, edge ↦ G.edge]
25     ∧ G.node ⊆ (SUBSET Oid)
26     ∧ G.edge ⊆ [from : G.node, to : G.node, cop : Cop]

28 EmptySS ≜ [node ↦ {{}}, edge ↦ {}]

30 TypeOK ≜
31     ∧  TypeOKInt
32     ∧  TypeOKSerial
33     ∧  Comm(Cop)!TypeOK
34     ∧  cseq ∈ [Client → Nat]
       For all replicas: the n-ary ordered state space
38     ∧ ∀ r ∈ Replica : IsCSS(css[r])
39     ∧ cur ∈ [Replica → SUBSET Oid]
40 ----------
41 Init ≜
42     ∧ InitInt
43     ∧ InitSerial
44     ∧ Comm(Cop)!Init
45     ∧ cseq = [c ∈ Client ↦ 0]
       For all replicas: the n-ary ordered state space
49     ∧ css = [r ∈ Replica ↦ EmptySS]
50     ∧ cur = [r ∈ Replica ↦ {}]
51 ----------
   Locate the node in rcss (the css at replica r ∈ Replica) that matches the context ctx of cop.
55 Locate(cop, rcss) ≜ CHOOSE n ∈ rcss.node : n = cop.ctx
   Take union of two state spaces ss1 and ss2.
59 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(Cop)!CSend(cop)

DoIns(c) ≜
    ∃ ins ∈ {op ∈ Ins : op.pos ∈ 1 .. (Len(state[c]) + 1) ∧ op.ch ∈ chins ∧ op.pr = Priority[c]} :
```

```
111         ∧ DoOp(c, ins)
112         ∧ chins′ = chins \ {ins.ch}  We assume that all inserted elements are unique.

114 DoDel(c) ≜
115     ∃ del ∈ {op ∈ Del : op.pos ∈ 1 .. Len(state[c])} :
116         ∧ DoOp(c, del)
117         ∧ UNCHANGED chins

119 Do(c) ≜
120       ∧ DoSerial(c)
121       ∧ ∨ DoIns(c)
122         ∨ DoDel(c)

    Client c ∈ Client receives a message from the Server.
126 Rev(c) ≜
127       ∧ Comm(Cop)!CRev(c)
128       ∧ Perform(Head(cincoming[c]), c)
129       ∧ RevSerial(c)
130       ∧ UNCHANGED ⟨chins, cseq⟩
131 ├──────────────────────────────────────────────────────────────────────┤

    The Server receives a message.
135 SRev ≜
136     ∧ Comm(Cop)!SRev
137     ∧ LET cop ≜ Head(sincoming)
138       IN  ∧ Perform(cop, Server)
139           ∧ Comm(Cop)!SSendSame(cop.oid.c, cop)   broadcast the original operation
140     ∧ SRevSerial
141     ∧ UNCHANGED ⟨chins, cseq⟩
142 ├──────────────────────────────────────────────────────────────────────┤
143 Next ≜
144     ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
145     ∨ SRev

    Fairness: There is no requirement that the clients ever generate operations.
149 Fairness ≜
150     WF_vars(SRev ∨ ∃ c ∈ Client : Rev(c))

152 Spec ≜ Init ∧ □[Next]_vars   ∧ Fairness (We care more about safety.)
153 ├──────────────────────────────────────────────────────────────────────┤

    The compactness of CJupiter: the CSSes at all replicas are the same.
157 Compactness ≜
158     Comm(Cop)!EmptyChannel ⇒ Cardinality(Range(css)) = 1

160 THEOREM Spec ⇒ Compactness
161 └──────────────────────────────────────────────────────────────────────┘

    \* Modification History
    \* Last modified Wed Dec 12 20:17:44 CST 2018 by hengxin
    \* Created Sat Sep 01 11:08:00 CST 2018 by hengxin
```