```
┌──────────────────────────── MODULE CJupiter ────────────────────────────┐
 Model of our own CJupiter protocol.
EXTENDS JupiterInterface
├──────────────────────────────────────────────────────────────────────────┤
 Cop: operation of type Op with context
Oid ≜ [c : Client, seq : Nat]  operation identifier
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.
tb(cop1, cop2, sv) ≜
    LET pos1 ≜ FirstIndexOfElementSafe(sv, cop1.oid)
        pos2 ≜ FirstIndexOfElementSafe(sv, cop2.oid)
     IN  IF pos1 ≠ 0 ∧ pos2 ≠ 0  at the server or both are remote operations
          THEN pos1 < pos2       at a client: one is a remote operation and the other is a local operation
          ELSE pos1 ≠ 0
 OT of two operations of type Cop.
COT(lcop, rcop) ≜ [lcop EXCEPT !.op = Xform(lcop.op, rcop.op), !.ctx = @ ∪ {rcop.oid}]
├──────────────────────────────────────────────────────────────────────────┤
VARIABLES
     For the client replicas:
    cseq,         cseq[c]: local sequence number at client c ∈ Client
     For all replicas: the n-ary ordered state space
    css,          css[r]: the n-ary ordered state space at replica r ∈ Replica
    cur,          cur[r]: the current node of css at replica r ∈ Replica
     For edge ordering in CSS
    serial,       serial[r]: the serial view of replica r ∈ Replica about the server
    cincomingSerial,
    sincomingSerial

serialVars ≜ ⟨serial, cincomingSerial, sincomingSerial⟩
vars ≜ ⟨chins, cseq, css, cur, state, cincoming, sincoming, serialVars⟩
├──────────────────────────────────────────────────────────────────────────┤
commSerial ≜ INSTANCE CSComm WITH Msg ← Seq(Oid),
                   cincoming ← cincomingSerial, sincoming ← sincomingSerial
├──────────────────────────────────────────────────────────────────────────┤
 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.
IsCSS(G) ≜
    ∧ G = [node ↦ G.node, edge ↦ G.edge]
    ∧ G.node ⊆ (SUBSET Oid)
    ∧ G.edge ⊆ [from : G.node, to : G.node, cop : Cop]

EmptySS ≜ [node ↦ {{}}, edge ↦ {}]
```

```
$TypeOK \triangleq$
      $\land$   $TypeOKInt$
      $\land$   $Comm(Cop)!TypeOK$
      $\land$   $cseq \in [Client \rightarrow Nat]$
     For edge ordering in $CSS$:
      $\land serial \in [Replica \rightarrow Seq(Oid)]$
      $\land commSerial!TypeOK$
     For all replicas: the $n$-ary ordered state space
      $\land \forall r \in Replica : IsCSS(css[r])$
      $\land cur \in [Replica \rightarrow \text{SUBSET } Oid]$
--------------------------------------------------------------------------------
$Init \triangleq$
      $\land InitInt$
      $\land Comm(Cop)!Init$
      $\land cseq = [c \in Client \mapsto 0]$
     For the server replica:
      $\land serial = [r \in Replica \mapsto \langle\rangle]$
      $\land commSerial!Init$
     For all replicas: the $n$-ary ordered state space
      $\land css = [r \in Replica \mapsto EmptySS]$
      $\land cur = [r \in Replica \mapsto \{\}]$
--------------------------------------------------------------------------------
Locate the node in $rcss$ (the $css$ at replica $r \in Replica$) that matches the context $ctx$ of cop.
$Locate(cop, rcss) \triangleq \text{CHOOSE } n \in rcss.node : n = cop.ctx$
Take union of two state spaces $ss1$ and $ss2$.
$ss1 \oplus ss2 \triangleq [node \mapsto ss1.node \cup ss2.node, edge \mapsto ss1.edge \cup ss2.edge]$
$xForm$: Iteratively transform cop with a path through the $css$ at replica $r \in Replica$, following the first edges.
$xForm(cop, r) \triangleq$
    LET $rcss \triangleq css[r]$
        $u \triangleq Locate(cop, rcss)$
        $v \triangleq u \cup \{cop.oid\}$
        RECURSIVE $xFormHelper(\_, \_, \_, \_, \_, \_)$
         'h' stands for "helper"; $xcss$: $eXtra$ $css$ created during transformation
        $xFormHelper(uh, vh, coph, xcss, xcoph, xcurh) \triangleq$
            IF $uh = cur[r]$
             THEN $\langle xcss, xcoph, xcurh \rangle$
             ELSE LET $fedge \triangleq$ CHOOSE $e \in rcss.edge$ :
                                  $\land e.from = uh$
                                  $\land \forall uhe \in rcss.edge$ :
                                      $(uhe.from = uh \land uhe \neq e) \Rightarrow tb(e.cop, uhe.cop, serial[r])$
                      $uprime \triangleq fedge.to$
                      $fcop \triangleq fedge.cop$
```

```
                        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 ∈ 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 ∈ 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]} :
     ∧ DoOp(c, ins)
     ∧ chins′ = chins \ {ins.ch}   We assume that all inserted elements are unique.
     ∧ UNCHANGED ⟨serialVars⟩

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

Do(c) ≜
     ∨ DoIns(c)
     ∨ DoDel(c)

Client c ∈ Client receives a message from the Server.
Rev(c) ≜
     ∧ Comm(Cop)!CRev(c)
     ∧ Perform(Head(cincoming[c]), c)
     ∧ commSerial!CRev(c)
```

```
         ∧ serial′ = [serial EXCEPT ![c] = Head(cincomingSerial[c])]
         ∧ UNCHANGED ⟨chins, cseq⟩
├──────────────────────────────────────────────────────────────────────────┤
 The Server receives a message.
 SRev ≜
      ∧ Comm(Cop)!SRev
      ∧ LET cop ≜ Head(sincoming)
          IN   ∧ Perform(cop, Server)
               ∧ Comm(Cop)!SSendSame(cop.oid.c, cop)   broadcast the original operation
               ∧ serial′ = [serial EXCEPT ![Server] = Append(@, cop.oid)]
               ∧ commSerial!SSendSame(cop.oid.c, serial′[Server])
      ∧ UNCHANGED ⟨chins, cseq, sincomingSerial⟩
├──────────────────────────────────────────────────────────────────────────┤
 Next ≜
      ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
      ∨ SRev
 Fairness: There is no requirement that the clients ever generate operations.
 Fairness ≜
      WF_vars(SRev ∨ ∃ c ∈ Client : Rev(c))

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

 THEOREM Spec ⇒ Compactness
└──────────────────────────────────────────────────────────────────────────┘
 \* Modification History
 \* Last modified Tue Dec 04 21:10:17 CST 2018 by hengxin
 \* Created Sat Sep 01 11:08:00 CST 2018 by hengxin
```