```
──────────────────────── MODULE CJupiter ────────────────────────
 Model of our own CJupiter protocol.

EXTENDS Integers, OT, TLC, AdditionalFunctionOperators
─────────────────────────────────────────────────────────────────
CONSTANTS
    Client,       the set of client replicas
    Server,       the (unique) server replica
    Char,         set of characters allowed
    InitState     the initial state of each replica

Replica ≜ Client ∪ {Server}

List ≜ Seq(Char ∪ Range(InitState))    all possible lists/strings
MaxLen ≜ Cardinality(Char) + Len(InitState)  the max length of lists in any states;
     We assume that all inserted elements are unique.

ClientNum ≜ Cardinality(Client)
Priority ≜ CHOOSE f ∈ [Client → 1 . . ClientNum] : Injective(f)
─────────────────────────────────────────────────────────────────
ASSUME
    ∧ Range(InitState) ∩ Char = {}
    ∧ Priority ∈ [Client → 1 . . ClientNum]
─────────────────────────────────────────────────────────────────
 The set of all operations. Note: The positions are indexed from 1.
Rd ≜ [type : {"Rd"}]
Del ≜ [type : {"Del"}, pos : 1 . . MaxLen]
Ins ≜ [type : {"Ins"}, pos : 1 . . (MaxLen + 1), ch : Char, pr : 1 . . ClientNum]  pr: priority

Op ≜ Ins ∪ Del   Now we don't consider Rd operations.
─────────────────────────────────────────────────────────────────
Oid ≜ [c : Client, seq : Nat]  operation identifier
Cop ≜ [op : Op, oid : Oid, ctx : SUBSET Oid, sctx : SUBSET Oid]  operation with context

cop1 ≺ cop2 ≜
    ∨ cop2.sctx = {}
    ∨ cop1.oid ∈ cop2.sctx

COT(lcop, rcop) ≜
      [op ↦ Xform(lcop.op, rcop.op), oid ↦ lcop.oid,
         ctx ↦ lcop.ctx ∪ {rcop.oid}, sctx ↦ lcop.stx]
─────────────────────────────────────────────────────────────────
VARIABLES
   For the client replicas:
    cseq,     cseq[c]: local sequence number at client c ∈ Client
    cstate,   cstate[c]: state (the list content) of the client c ∈ Client
    For the server replica:
    soids,       the set of operations the Server has executed
    sstate,      sstate: state (the list content) of the server Server
    For all replicas: the n-ary ordered state space
    css,         css[r]: the n-ary ordered state space at replica r
    cur,         cur[r]: the current node of css at replica r
    For communication between the Server and the Clients:
    cincoming,     cincoming[c]: incoming channel at the client c ∈ Client
    sincoming,     incoming channel at the Server
    For model checking:
    chins        a set of chars to insert

├──────────────────────────────────────────────────────────────────────┤
comm ≜ INSTANCE CSComm WITH Msg ← Cop
├──────────────────────────────────────────────────────────────────────┤
eVars ≜ ⟨chins⟩                                 variables for the environment
cVars ≜ ⟨cseq, cstate⟩               variables for the clients
ecVars ≜ ⟨eVars, cVars⟩                         variables for the clients and the environment
sVars ≜ ⟨soids, sstate⟩              variables for the server
commVars ≜ ⟨cincoming, sincoming⟩               variables for communication
vars ≜ ⟨eVars, cVars, sVars, commVars, css, cur⟩ all variables
├──────────────────────────────────────────────────────────────────────┤
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.
IsCSS(G) ≜
    ∧ G = [node ↦ G.node, edge ↦ G.edge]
    ∧ G.node ⊆ (SUBSET Oid)
    ∧ G.edge ⊆ [from : G.node, to : G.node, cop : Cop]

TypeOK ≜
    For the client replicas:
    ∧ cseq ∈ [Client → Nat]
    ∧ cstate ∈ [Client → List]
    For the server replica:
    ∧ soids ⊆ Oid
    ∧ sstate ∈ List
    For all replicas: the n-ary ordered state space
    ∧ ∀ r ∈ Replica : IsCSS(css[r])
    ∧ cur ∈ [Client → SUBSET Oid]
    For communication between the server and the clients:
```

```
    ∧ comm!TypeOK
    For model checking:
    ∧ chins ⊆ Char
────────────────────────────────────────────────────────────
The Init predicate.
Init ≜
    ∧ chins = Char
    For the client replicas:
    ∧ cseq = [c ∈ Client ↦ 0]
    ∧ cstate = [c ∈ Client ↦ InitState]
    For the server replica:
    ∧ soids = {}
    ∧ sstate = InitState
    For all replicas: the n-ary ordered state space
    ∧ css  = [r ∈ Replica ↦ [node ↦ {}, edge ↦ {}]]
    ∧ cur = [r ∈ Replica ↦ {}]
    For communication between the server and the clients:
    ∧ comm!Init
────────────────────────────────────────────────────────────
Client c ∈ Client issues an operation op.
DoOp(c, op) ≜    op: the raw operation generated by the client c ∈ Client
      ∧ cstate′ = [cstate EXCEPT ![c] = Apply(op, @)]
      ∧ cseq′ = [cseq EXCEPT ![c] = @ + 1]
      ∧ LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq′[c]],
           ctx ↦ cur[c], sctx ↦ {}]    cop: original operation with context
              v ≜ cur ∪ {cop.oid}
           IN   ∧ css′  = [css EXCEPT ![c].node = @ ∪ {v},
                                      ![c].edge  = @ ∪ {[from ↦ cur, to ↦ v, cop ↦ cop]}]
                ∧ cur′ = v
                ∧ comm!CSend(cop)

DoIns(c) ≜
    ∃ ins ∈ Ins :
       ∧ ins.pos ∈ 1 .. (Len(cstate[c]) + 1)
       ∧ ins.ch ∈ chins
       ∧ ins.pr = Priority[c]
       ∧ chins′ = chins \ {ins.ch}  We assume that all inserted elements are unique.
       ∧ DoOp(c, ins)
       ∧ UNCHANGED sVars

DoDel(c) ≜
    ∃ del ∈ Del :
       ∧ del.pos ∈ 1 .. Len(cstate[c])
       ∧ DoOp(c, del)
```

```
        ∧ UNCHANGED ⟨sVars, eVars⟩

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

Locate the node in *rcss* which matches the context *ctx* of cop.

rcss: the *css* at replica $r \in Replica$

```
Locate(cop, rcss) ≜ CHOOSE n ∈ (rcss.node) : n = cop.ctx
```

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

```
RECURSIVE xForm(_, _)
xForm(cop, r) ≜
    LET rcss ≜ css[r]
        u ≜ Locate(cop, rcss)
        v ≜ u ∪ {cop.oid}
        RECURSIVE xFormHelper(_, _, _)
         xFormHelper(uh, vh, coph) ≜
            IF uh = cur[r]
             THEN css' = [css EXCEPT ![r].node = @ ∪ {vh},
                                     ![r].edge = @ ∪ {[from ↦ uh, to ↦ vh, cop ↦ coph]}]
             ELSE LET fedge ≜ CHOOSE e ∈ rcss.edge :
                                    ∧ e.from = uh
                                    ∧ ∀ ue ∈ rcss.edge :
                                        (ue.from = uh ∧ ue ≠ e) ⇒ (e.cop ≺ ue.cop)
                      uprime ≜ fedge.to
                      fcop   ≜ fedge.cop
                      cop2fcop ≜ COT(cop, fcop)
                      fcop2cop ≜ COT(fcop, cop)
                       vprime ≜ v.oids ∪ {fcop.oid}
                 IN    ∧ css' = [css EXCEPT ![r].node = @ ∪ {vh},
                                            ![r].edge = @ ∪ {[from ↦ uh, to ↦ vh, cop ↦ coph],
                                                              [from ↦ vh, to ↦ vprime, cop ↦ fcop2cop]}
                       ∧ xFormHelper(uprime, vprime, cop2fcop)
    IN     xFormHelper(u, v, cop)
```

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

```
Rev(c) ≜
      ∧ comm!CRev(c)
      ∧ LET cop ≜ Head(cincoming[c])   the received original operation
              xcop ≜ xForm(cop, c)      the transformed operation
          IN    cstate' = [cstate EXCEPT ![c] = Apply(xcop.op, @)]
      ∧ UNCHANGED ⟨sVars, eVars⟩
```

The *Server* receives a message.

```
SRev ≜
    ∧ comm!SRev
    ∧ LET org ≜ Head(sincoming)   the received operation
              cop ≜ [org EXCEPT !.sctx = soids]        set its sctx field
               xcop ≜ xForm(cop, Server)
          IN   ∧ soids' = soids ∪ {cop.oid}
               ∧ sstate' = Apply(xcop.op, sstate)       apply the transformed operation
               ∧ comm!SSendSame(cop.oid.cid, cop)       broadcast the original operation
    ∧ UNCHANGED ecVars
--------------------------------------------------------------------------------
The next-state relation.
Next ≜
    ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
    ∨ SRev
The Spec. (TODO: Check the fairness condition.)
Spec ≜ Init ∧ □[Next]_vars ∧ WF_vars(Next)
================================================================================
\* Modification History
\* Last modified Mon Sep 03 14:03:58 CST 2018 by hengxin
\* Created Sat Sep 01 11:08:00 CST 2018 by hengxin
```