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

EXTENDS Integers, OT, TLC, AdditionalFunctionOperators, AdditionalSequenceOperators
─────────────────────────────────────────────────────────
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 = {}    due to the uniqueness requirement
    ∧ 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
─────────────────────────────────────────────────────────
Cop: operation of type Op with context
Oid ≜ [c : Client, seq : Nat]   operation identifier
Cop ≜ [op : Op ∪ {Nop}, oid : Oid, ctx : SUBSET Oid, sctx : SUBSET Oid]

tb: Is cop1 totally ordered before cop2?

At a given replica r ∈ Replica, these can be determined in terms of sctx.
tb(cop1, cop2, r) ≜
    ∨ cop1.oid ∈ cop2.sctx
    ∨ ∧ cop1.oid ∉ cop2.sctx
      ∧ cop2.oid ∉ cop1.sctx
      ∧ cop1.oid.c ≠ r

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 $c \in Client$

For the server replica:

$soids$, — the set of operations the *Server* has executed

For all replicas: the n-ary ordered state space

css, — $css[r]$: the n-ary ordered state space at replica $r \in Replica$

cur, — $cur[r]$: the current node of css at replica $r \in Replica$

$state$, — $state[r]$: state (the list content) of replica $r \in Replica$

For communication between the *Server* and the Clients:

$cincoming$, — $cincoming[c]$: incoming channel at the client $c \in Client$

$sincoming$, — incoming channel at the *Server*

For model checking:

$chins$ — a set of chars to insert

$comm \triangleq \text{INSTANCE } CSComm \text{ WITH } Msg \leftarrow Cop$

$eVars \triangleq \langle chins \rangle$ — variables for the environment

$cVars \triangleq \langle cseq \rangle$ — variables for the clients

$ecVars \triangleq \langle eVars, cVars \rangle$ — variables for the clients and the environment

$sVars \triangleq \langle soids \rangle$ — variables for the server

$dsVars \triangleq \langle css, cur, state \rangle$ — variables for the data structure: the n-ary ordered state space

$commVars \triangleq \langle cincoming, sincoming \rangle$ — variables for communication

$vars \triangleq \langle eVars, cVars, sVars, commVars, dsVars \rangle$ — 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.

$$
\begin{aligned}
&IsCSS(G) \triangleq \\
&\quad \land G = [node \mapsto G.node, edge \mapsto G.edge] \\
&\quad \land G.node \subseteq (\text{SUBSET } Oid) \\
&\quad \land G.edge \subseteq [from : G.node, to : G.node, cop : Cop]
\end{aligned}
$$

$TypeOK \triangleq$

For the client replicas:

$\quad \land cseq \in [Client \rightarrow Nat]$

For the server replica:

$\quad \land soids \subseteq Oid$

For all replicas: the n-ary ordered state space

$\quad \land \forall r \in Replica : IsCSS(css[r])$

$\quad \land\ cur \in [Replica \rightarrow \text{SUBSET } Oid]$

$\quad \land\ state \in [Replica \rightarrow List]$

For communication between the server and the clients:

$\quad \land\ comm!TypeOK$

For model checking:

$\quad \land\ chins \subseteq Char$

The *Init* predicate.

$Init \triangleq$

$\quad \land\ chins = Char$

For the client replicas:

$\quad \land\ cseq = [c \in Client \mapsto 0]$

For the server replica:

$\quad \land\ soids = \{\}$

For all replicas: the n-ary ordered state space

$\quad \land\ css = [r \in Replica \mapsto [node \mapsto \{\{\}\},\ edge \mapsto \{\}]]$

$\quad \land\ cur = [r \in Replica \mapsto \{\}]$

$\quad \land\ state = [r \in Replica \mapsto InitState]$

For communication between the server and the clients:

$\quad \land\ comm!Init$

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

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

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

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

$xForm(cop,\ r) \triangleq$

$\quad \text{LET } rcss \triangleq css[r]$

$\qquad u \triangleq Locate(cop,\ rcss)$

$\qquad v \triangleq u \cup \{cop.oid\}$

$\qquad \text{RECURSIVE } xFormHelper(\_,\ \_,\ \_,\ \_)$

'h' stands for "helper"; *xcss*: *eXtra css* created during transformation

$\qquad xFormHelper(uh,\ vh,\ coph,\ xcss) \triangleq$

$\qquad\quad \text{IF } uh = cur[r]$

$\qquad\quad\ \text{THEN } xcss$

$\qquad\quad\ \text{ELSE LET } fedge \triangleq \text{CHOOSE } e \in rcss.edge :$

$\qquad\qquad\qquad\qquad \land\ e.from = uh$

$\qquad\qquad\qquad\qquad \land\ \forall\, uhe \in rcss.edge :$

$\qquad\qquad\qquad\qquad\quad (uhe.from = uh \land uhe \neq e) \Rightarrow tb(e.cop,\ uhe.cop,\ r)$

$\qquad\qquad\qquad uprime \triangleq fedge.to$

$\qquad\qquad\qquad fcop \triangleq fedge.cop$

$\qquad\qquad\qquad coph2fcop \triangleq COT(coph,\ fcop)$

```
                              fcop2coph ≜ COT(fcop, coph)
                               vprime ≜ vh ∪ {fcop.oid}
                         IN   xFormHelper(uprime, vprime, coph2fcop,
                                 [xcss EXCEPT !.node = @ ∘ ⟨vprime⟩,
                                                 the order of recording edges here is important
                                                !.edge = @ ∘ ⟨[from ↦ vh, to ↦ vprime, cop ↦ fcop2coph],
                                                        [from ↦ uprime, to ↦ vprime, cop ↦ coph2fcop]⟩])
    IN  xFormHelper(u, v, cop, [node ↦ ⟨v⟩, edge ↦ ⟨[from ↦ u, to      ↦ v, cop ↦ cop]⟩])
```

Perform cop at replica $r \in Replica$.

```
Perform(cop, r) ≜
    LET xcss ≜ xForm(cop, r)
        xn ≜ xcss.node
        xe ≜ xcss.edge
        xcur ≜ Last(xn)
        xcop ≜ Last(xe).cop
    IN  ∧ css′ = [css EXCEPT ![r].node = @ ∪ Range(xn),
                             ![r].edge = @ ∪ Range(xe)]
        ∧ 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], sctx ↦ {}]
         IN  ∧ Perform(cop, c)
             ∧ comm!CSend(cop)

DoIns(c) ≜
    ∃ ins ∈ Ins :
       ∧ ins.pos ∈ 1 .. (Len(state[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(state[c])
       ∧ DoOp(c, del)
       ∧ UNCHANGED ⟨sVars, eVars⟩

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

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

$$
\begin{aligned}
&Rev(c) \triangleq \\
&\quad \land\ comm!CRev(c) \\
&\quad \land\ \text{LET } cop \triangleq Head(cincoming[c]) \quad \text{the received original operation} \\
&\quad \quad \text{IN } \quad Perform(cop,\ c) \\
&\quad \land\ \text{UNCHANGED } \langle ecVars,\ sVars \rangle
\end{aligned}
$$

The $Server$ receives a message.

$$
\begin{aligned}
&SRev \triangleq \\
&\quad \land\ comm!SRev \\
&\quad \land\ \text{LET } cop \triangleq [Head(sincoming) \text{ EXCEPT } !.sctx = soids] \quad \text{set its } sctx \text{ field} \\
&\quad \quad \text{IN } \quad \land\ soids' = soids \cup \{cop.oid\} \\
&\quad \quad \quad \quad \land\ Perform(cop,\ Server) \\
&\quad \quad \quad \quad \land\ comm!SSendSame(cop.oid.c,\ cop) \quad \text{broadcast the original operation} \\
&\quad \land\ \text{UNCHANGED } ecVars
\end{aligned}
$$

The next-state relation.

$$
\begin{aligned}
&Next \triangleq \\
&\quad \lor\ \exists\, c \in Client : Do(c) \lor Rev(c) \\
&\quad \lor\ SRev
\end{aligned}
$$

The *Spec*. (*TODO*: Check the fairness condition.)

$$
Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)
$$

The compactness of *CJupiter*: the *css* at all replicas are essentially the same.

$$
\begin{aligned}
&IgnoreSctx(rcss) \triangleq \\
&\quad [rcss \text{ EXCEPT } !.edge = \{[e \text{ EXCEPT } !.cop.sctx = \{\}] : e \in @\}]
\end{aligned}
$$

$$
\begin{aligned}
&Compactness \triangleq \\
&\quad comm!EmptyChannel \Rightarrow Cardinality(\{IgnoreSctx(css[r]) : r \in Replica\}) = 1
\end{aligned}
$$

$$
\text{THEOREM } Spec \Rightarrow Compactness
$$

---

* Modification History

* *Last* modified *Thu Sep* 06 21:18:42 *CST* 2018 by *hengxin*

* Created Sat *Sep* 01 11:08:00 *CST* 2018 by *hengxin*