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

$Replica \triangleq Client \cup \{Server\}$

$List \triangleq Seq(Char \cup Range(InitState))$ all possible lists/strings

$MaxLen \triangleq Cardinality(Char) + Len(InitState)$ the max length of lists in any states;

We assume that all inserted elements are unique.

$ClientNum \triangleq Cardinality(Client)$

$Priority \triangleq \text{CHOOSE } f \in [Client \to 1 \,..\, ClientNum] : Injective(f)$

---

ASSUME

$\land\ Range(InitState) \cap Char = \{\}$ due to the uniqueness requirement

$\land\ Priority \in [Client \to 1 \,..\, ClientNum]$

---

The set of all operations. Note: The positions are indexed from 1.

$Rd \triangleq [type : \{\text{“Rd”}\}]$

$Del \triangleq [type : \{\text{“Del”}\},\ pos : 1 \,..\, MaxLen]$

$Ins \triangleq [type : \{\text{“Ins”}\},\ pos : 1 \,..\, (MaxLen + 1),\ ch : Char,\ pr : 1 \,..\, ClientNum]$ $pr$: priority

$Op \triangleq Ins \cup Del$

---

Cop: operation of type $Op$ with context

$Oid \triangleq [c : Client,\ seq : Nat]$ operation identifier

$Cop \triangleq [op : Op \cup \{Nop\},\ oid : Oid,\ ctx : \text{SUBSET } Oid]$

$OT$ of two operations of type $Cop$.

$COT(lcop, rcop) \triangleq [lcop \text{ EXCEPT } !.op = Xform(lcop.op, rcop.op),\ !.ctx = @ \cup \{rcop.oid\}]$

---

VARIABLES

For the client replicas:

$cseq,$ $cseq[c]$: local sequence number at client $c \in Client$

For edge ordering

$serial,$ $serial[r]$: the serial order of operations from the view of each replica $r \in Replica$

$cincomingSerial,$

$sincomingSerial,$

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

*tb*: Is *cop*1 totally ordered before *cop*2? This is determined according to "serial" at replica $r \in Replica$

$$
\begin{array}{l}
tb(cop1, cop2, r) \triangleq \\
\quad \text{LET } pos1 \triangleq FirstIndexOfElementSafe(serial[r], cop1.oid) \\
\qquad\quad pos2 \triangleq FirstIndexOfElementSafe(serial[r], cop2.oid) \\
\quad\; \text{IN} \quad \text{IF } pos1 \neq 0 \land pos2 \neq 0 \\
\qquad\qquad \text{THEN } pos1 < pos2 \\
\qquad\qquad \text{ELSE } pos1 \neq 0
\end{array}
$$

---

$$
\begin{array}{l}
comm \triangleq \text{INSTANCE } CSComm \text{ WITH } Msg \leftarrow Cop \\
commSerial \triangleq \text{INSTANCE } CSComm \\
\qquad\qquad\qquad \text{WITH } Msg \leftarrow Seq(Oid),\ cincoming \leftarrow cincomingSerial,\ sincoming \leftarrow sincomingSerial
\end{array}
$$

---

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

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

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

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

$serialVars \triangleq \langle serial, cincomingSerial, sincomingSerial \rangle$

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

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

$TypeOK \triangleq$

For the client replicas:

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

For edge ordering:

$\land$ *serial* $\in [Replica \rightarrow Seq(Seq(Oid))]$

$\land$ *commSerial* ! *TypeOK*

For all replicas: the $n$-ary ordered state space

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

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

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

For communication between the server and the clients:

$\land$ *comm* ! *TypeOK*

For model checking:

$\land chins \subseteq Char$

---

The *Init* predicate.

$Init \triangleq$

For the client replicas:

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

For the server replica:

$\land serial = [r \in Replica \mapsto \langle\rangle]$

$\land$ *commSerial* ! *Init*

For all replicas: the $n$-ary ordered state space

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

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

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

For communication between the server and the clients:

$\land$ *comm* ! *Init*

For model checking:

$\land chins = Char$

---

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.

$xForm(cop, r) \triangleq$

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

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

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

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

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

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

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

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

$\quad\quad\quad \text{ELSE LET } \mathit{fedge} \triangleq \text{CHOOSE } e \in \mathit{rcss.edge} :$
$\quad\quad\quad\quad\quad \land\ e.\mathit{from} = \mathit{uh}$
$\quad\quad\quad\quad\quad \land\ \forall\, \mathit{uhe} \in \mathit{rcss.edge} :$
$\quad\quad\quad\quad\quad\quad (\mathit{uhe.from} = \mathit{uh} \land \mathit{uhe} \neq e) \Rightarrow \mathit{tb}(e.\mathit{cop}, \mathit{uhe.cop}, r)$
$\quad\quad\quad\quad \mathit{uprime} \triangleq \mathit{fedge.to}$
$\quad\quad\quad\quad \mathit{fcop} \triangleq \mathit{fedge.cop}$
$\quad\quad\quad\quad \mathit{coph2fcop} \triangleq \mathit{COT}(\mathit{coph}, \mathit{fcop})$
$\quad\quad\quad\quad \mathit{fcop2coph} \triangleq \mathit{COT}(\mathit{fcop}, \mathit{coph})$
$\quad\quad\quad\quad \mathit{vprime} \triangleq \mathit{vh} \cup \{\mathit{fcop.oid}\}$
$\quad\quad\quad \text{IN } \mathit{xFormHelper}(\mathit{uprime}, \mathit{vprime}, \mathit{coph2fcop},$
$\quad\quad\quad\quad [\mathit{xcss} \text{ EXCEPT } !.\mathit{node} = @ \circ \langle \mathit{vprime} \rangle,$

the order of recording edges here is important

$\quad\quad\quad\quad\quad !.\mathit{edge} = @ \circ \langle [\mathit{from} \mapsto \mathit{vh}, \mathit{to} \mapsto \mathit{vprime}, \mathit{cop} \mapsto \mathit{fcop2coph}],$
$\quad\quad\quad\quad\quad\quad [\mathit{from} \mapsto \mathit{uprime}, \mathit{to} \mapsto \mathit{vprime}, \mathit{cop} \mapsto \mathit{coph2fcop}] \rangle])$
$\quad \text{IN } \mathit{xFormHelper}(u, v, \mathit{cop}, [\mathit{node} \mapsto \langle v \rangle,$
$\quad\quad\quad\quad \mathit{edge} \mapsto \langle [\mathit{from} \mapsto u, \mathit{to} \mapsto v, \mathit{cop} \mapsto \mathit{cop}] \rangle])$

Perform cop at replica $r \in \mathit{Replica}$.

$\mathit{Perform}(\mathit{cop}, r) \triangleq$
$\quad \text{LET } \mathit{xcss} \triangleq \mathit{xForm}(\mathit{cop}, r)$
$\quad\quad \mathit{xn} \triangleq \mathit{xcss.node}$
$\quad\quad \mathit{xe} \triangleq \mathit{xcss.edge}$
$\quad\quad \mathit{xcur} \triangleq \mathit{Last}(\mathit{xn})$
$\quad\quad \mathit{xcop} \triangleq \mathit{Last}(\mathit{xe}).\mathit{cop}$
$\quad \text{IN } \land\ \mathit{css}' = [\mathit{css} \text{ EXCEPT } ![r].\mathit{node} = @ \cup \mathit{Range}(\mathit{xn}),$
$\quad\quad\quad\quad ![r].\mathit{edge} = @ \cup \mathit{Range}(\mathit{xe})]$
$\quad\quad \land\ \mathit{cur}' = [\mathit{cur} \text{ EXCEPT } ![r] = \mathit{xcur}]$
$\quad\quad \land\ \mathit{state}' = [\mathit{state} \text{ EXCEPT } ![r] = \mathit{Apply}(\mathit{xcop.op}, @)]$

---

Client $c \in \mathit{Client}$ issues an operation $\mathit{op}$.

$\mathit{DoOp}(c, \mathit{op}) \triangleq$ $\mathit{op}$: the raw operation generated by the client $c \in \mathit{Client}$
$\quad \land\ \mathit{cseq}' = [\mathit{cseq} \text{ EXCEPT } ![c] = @ + 1]$
$\quad \land\ \text{LET } \mathit{cop} \triangleq [\mathit{op} \mapsto \mathit{op}, \mathit{oid} \mapsto [c \mapsto c, \mathit{seq} \mapsto \mathit{cseq}'[c]], \mathit{ctx} \mapsto \mathit{cur}[c]]$
$\quad\quad \text{IN } \land\ \mathit{Perform}(\mathit{cop}, c)$
$\quad\quad\quad \land\ \mathit{comm}!\mathit{CSend}(\mathit{cop})$

$\mathit{DoIns}(c) \triangleq$
$\quad \exists\, \mathit{ins} \in \mathit{Ins} :$
$\quad\quad \land\ \mathit{ins.pos} \in 1 \,..\, (\mathit{Len}(\mathit{state}[c]) + 1)$
$\quad\quad \land\ \mathit{ins.ch} \in \mathit{chins}$
$\quad\quad \land\ \mathit{ins.pr} = \mathit{Priority}[c]$
$\quad\quad \land\ \mathit{chins}' = \mathit{chins} \setminus \{\mathit{ins.ch}\}$ We assume that all inserted elements are unique.
$\quad\quad \land\ \mathit{DoOp}(c, \mathit{ins})$
$\quad\quad \land\ \text{UNCHANGED } \langle \mathit{serialVars} \rangle$

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

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

Client c ∈ Client receives a message from the Server.
Rev(c) ≜
      ∧ comm!CRev(c)
      ∧ LET cop ≜ Head(cincoming[c])  the received original operation
         IN  Perform(cop, c)
      ∧ commSerial!CRev(c)
      ∧ serial′ = [serial EXCEPT ![c] = Head(cincomingSerial[c])]
      ∧ UNCHANGED ⟨eVars, cVars⟩
--------------------------------------------------------------------------------
The Server receives a message.
SRev ≜
     ∧ comm!SRev
     ∧ LET cop ≜ Head(sincoming)
        IN  ∧ Perform(cop, Server)
            ∧ comm!SSendSame(cop.oid.c, cop)  broadcast the original operation
            ∧ serial′ = [serial EXCEPT ![Server] = Append(@, cop.oid)]
            ∧ commSerial!SSendSame(cop.oid.c, serial′)
     ∧ UNCHANGED ⟨eVars, cVars, sincomingSerial⟩
--------------------------------------------------------------------------------
The next-state relation.
Next ≜
     ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
     ∨ SRev
The Spec. There is no requirement that the clients ever generate operations.
Spec ≜ Init ∧ □[Next]_vars ∧ WF_vars(SRev ∨ ∃ c ∈ Client : Rev(c))
--------------------------------------------------------------------------------
The compactness of CJupiter: the css at all replicas are essentially the same.
Compactness ≜
    comm!EmptyChannel ⇒ Cardinality({css[r] : r ∈ Replica}) = 1

THEOREM Spec ⇒ Compactness
================================================================================
\* Modification History
\* Last modified Mon Nov 05 21:24:49 CST 2018 by hengxin
\* Created Sat Sep 01 11:08:00 CST 2018 by hengxin
```