MODULE *XJupiter*

Specification of the *Jupiter* protocol described in *CSCW′*2014 by *Yi Xu*, *Chengzheng* Sun, and *Mo Li*. We call it *XJupiter*, with 'X' for "*Xu*".

EXTENDS *Integers*, *OT*, *TLCUtils*, *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.

$ClientNum \triangleq Cardinality(Client)$

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

Direction flags for edges in $2D$ state spaces and OT.

$Local \triangleq 0$

$Remote \triangleq 1$

ASSUME

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

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

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 $c \in Client$

The $2D$ state spaces (ss, for short). Each client maintains one $2D$ state space. The server maintains n $2D$ state spaces, one for each client.

$c2ss,$ $c2ss[c]$: the $2D$ state space at client $c \in Client$

$s2ss,$ $s2ss[c]$: the $2D$ state space maintained by the *Server* for client $c \in Client$

$cur,$ $cur[r]$: the current node of the $2D$ state space at replica $r \in Replica$

For all replicas

$state,$ $state[r]$: 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

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

$vars \triangleq \langle eVars, cVars, cur, commVars, c2ss, s2ss, state \rangle$ all variables

---

A $2D$ state space 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 and a direction flag indicating whether this edge is LOCAL or REMOTE. For clarity, we denote edges by records instead of tuples.

$IsSS(G) \triangleq$

$\quad \land G = [node \mapsto G.node, edge \mapsto G.edge]$

$\quad \land G.node \subseteq (\text{SUBSET } Oid)$

$\quad \land G.edge \subseteq [from : G.node, to : G.node, cop : Cop, lr : \{Local, Remote\}]$

Take union of two state spaces $ss1$ and $ss2$.

$ss1 \oplus ss2 \triangleq [node \mapsto ss1.node \cup ss2.node, edge \mapsto ss1.edge \cup ss2.edge]$

$TypeOK \triangleq$

For the client replicas:

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

For the $2D$ state spaces:

$\quad \land \forall c \in Client : IsSS(c2ss[c]) \land IsSS(s2ss[c])$

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

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

For communication between the server and the clients:

$\quad \land comm!TypeOK$

For model checking:

$\quad \land chins \subseteq Char$

$Init \triangleq$

For the client replicas:

```
    ∧ cseq = [c ∈ Client ↦ 0]
   For the 2D state spaces:
    ∧ c2ss = [c ∈ Client ↦ [node ↦ {{}}, edge ↦ {}]]
    ∧ s2ss = [c ∈ Client ↦ [node ↦ {{}}, edge ↦ {}]]
    ∧ cur  = [r ∈ Replica ↦ {}]
   For all replicas:
    ∧ state = [r ∈ Replica ↦ InitState]
   For communication between the server and the clients:
    ∧ comm!Init
   For model checking:
    ∧ chins = Char
├──────────────────────────────────────────────────────────────────────┤
Locate the node in the 2D state space ss which matches the context ctx of cop.
Locate(cop, ss) ≜ CHOOSE n ∈ ss.node : n = cop.ctx
xForm: iteratively transform cop with a path through the 2D state space ss at some client,
following the edges with the direction flag d.
xForm(cop, ss, current, d) ≜
    LET u ≜ Locate(cop, ss)
        v ≜ u ∪ {cop.oid}
        RECURSIVE xFormHelper(_, _, _, _, _, _)
         'h' stands for "helper"; xss: eXtra ss created during transformation
        xFormHelper(uh, vh, coph, xss, xcoph, xcurh) ≜
            IF uh = current
             THEN ⟨xss, xcoph, xcurh⟩
             ELSE LET e ≜ CHOOSE e ∈ ss.edge : e.from = uh ∧ e.lr = d
                      uprime ≜ e.to
                      copprime ≜ e.cop
                      coph2copprime ≜ COT(coph, copprime)
                      copprime2coph ≜ COT(copprime, coph)
                       vprime ≜ vh ∪ {copprime.oid}
                  IN  xFormHelper(uprime, vprime, coph2copprime,
                        [node ↦ xss.node ∪ {vprime},
                         edge ↦ xss.edge ∪ {[from ↦ vh, to ↦ vprime, cop ↦ copprime2coph, lr ↦ d],
                                     [from ↦ uprime, to ↦ vprime, cop ↦ coph2copprime, lr ↦ 1 − d]}],
                                coph2copprime, vprime)
    IN  xFormHelper(u, v, cop, [node ↦ {v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop, lr ↦ 1 − d]}], cop, v)
├──────────────────────────────────────────────────────────────────────┤
Client c ∈ Client perform operation cop guided by the direction flag d.
ClientPerform(cop, c, d) ≜
    LET xform ≜ xForm(cop, c2ss[c], cur[c], d)   xform: ⟨xss, xcop, xcur⟩
          xss ≜ xform[1]
         xcop ≜ xform[2]
         xcur ≜ xform[3]
```

```
        IN   ∧ c2ss′ = [c2ss EXCEPT ![c] = @ ⊕ xss]
             ∧ cur′ = [cur EXCEPT ![c] = xcur]
             ∧ state′ = [state EXCEPT ![c] = Apply(xcop.op, @)]
```

Client $c \in$ *Client* generates an operation *op*.

```
DoOp(c, op) ≜
    ∧ cseq′ = [cseq EXCEPT ![c] = @ + 1]
    ∧ LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq′[c]], ctx ↦ cur[c]]
        IN  ∧ ClientPerform(cop, c, Remote)
            ∧ comm!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.

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

Do(c) ≜
    ∧ ∨ DoIns(c)
      ∨ DoDel(c)
    ∧ UNCHANGED ⟨s2ss⟩
```

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

```
Rev(c) ≜
    ∧ comm!CRev(c)
    ∧ LET cop ≜ Head(cincoming[c])   the received (transformed) operation
        IN  ClientPerform(cop, c, Local)
    ∧ UNCHANGED ⟨eVars, cVars, s2ss⟩
```

The *Server* performs operation cop.

```
ServerPerform(cop) ≜
    LET c     ≜ cop.oid.c
        scur  ≜ cur[Server]
        xform ≜ xForm(cop, s2ss[c], scur, Remote)   xform: ⟨xss, xcop, xcur⟩
        xss   ≜ xform[1]
        xcop  ≜ xform[2]
        xcur  ≜ xform[3]
    IN  ∧ s2ss′ = [cl ∈ Client ↦
                     IF cl = c
                      THEN s2ss[cl] ⊕ xss
                      ELSE s2ss[cl] ⊕ [node ↦ {xcur},
                                       edge ↦ {[from ↦ scur, to ↦ xcur,
                                                cop ↦ xcop, lr ↦ Remote]}]
```

$$]$$
$$\land\ cur' = [cur \ \text{EXCEPT}\ ![Server] = xcur]$$
$$\land\ state' = [state \ \text{EXCEPT}\ ![Server] = Apply(xcop.op, @)]$$
$$\land\ comm!SSendSame(c, xcop)$$ broadcast the transformed operation

The *Server* receives a message.

$$SRev \triangleq$$
$$\land\ comm!SRev$$
$$\land\ \text{LET}\ cop \triangleq Head(sincoming)$$
$$\quad \text{IN}\quad ServerPerform(cop)$$
$$\land\ \text{UNCHANGED}\ \langle eVars, cVars, c2ss\rangle$$

$$Next \triangleq$$
$$\lor\ \exists\, c \in Client : Do(c) \lor Rev(c)$$
$$\lor\ SRev$$

$$Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(SRev \lor \exists\, c \in Client : Rev(c))$$

In *Jupiter* (not limited to *XJupiter*), each client synchronizes with the server. In *XJupiter*, this is expressed as the following *CSSync* property.

$$CSSync \triangleq$$
$$\forall\, c \in Client : (cur[c] = cur[Server]) \Rightarrow c2ss[c] = s2ss[c]$$

---

* Modification History

* *Last* modified *Fri Nov* 16 13:57:18 *CST* 2018 by *hengxin*

* Created *Tue Oct* 09 16:33:18 *CST* 2018 by *hengxin*