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.

$ClientNum \triangleq Cardinality(Client)$

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

direction flags

$Local \triangleq 0$

$Remote \triangleq 1$

ASSUME

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

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

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

$Rd \triangleq [type : \{\text{"Rd"}\}]$

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

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

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

$ccur,$ $cur[c]$: the current node of $css[c]$

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

$scur,$ $scur[c]$: the current node of $sss[c]$

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

$cssVars \triangleq \langle css, ccur \rangle$ variables for $2D$ state spaces at clients

$sssVars \triangleq \langle sss, scur \rangle$ variables for $2D$ state spaces at the $Server$

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

$vars \triangleq \langle eVars, cVars, commVars, cssVars, sssVars, state \rangle$ 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.

$IsSS(G) \triangleq$

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

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

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

$TypeOK \triangleq$

For the client replicas:

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

For the $2D$ state spaces:

$\quad \land \forall c \in Client : IsSS(css[c]) \land IsSS(sss[c])$

$\quad \land ccur \in [Client \rightarrow \text{SUBSET } Oid]$

$\quad \land scur \in [Client \rightarrow \text{SUBSET } Oid]$

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

For communication between the server and the clients:

$\quad \land comm!TypeOK$

For model checking:

```
    ∧ chins ⊆ Char
────────────────────────────────────────────────────────────────
```

The *Init* predicate.

```
Init ≜
```

For the client replicas:

```
    ∧ cseq = [c ∈ Client ↦ 0]
```

For the $2D$ state spaces:

```
    ∧ css = [c ∈ Client ↦ [node ↦ {{}}, edge ↦ {}]]
    ∧ ccur = [c ∈ Client ↦ {}]
    ∧ sss = [c ∈ Client ↦ [node ↦ {{}}, edge ↦ {}]]
    ∧ scur = [c ∈ Client ↦ {}]
```

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, cur, 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) ≜
            IF uh = cur
             THEN xss
             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,
                         [xss EXCEPT !.node = @ ∘ ⟨vprime⟩,
                                      the order of recording edges here is important
                                      so that the last one is labeled with the final transformed operation
                                     !.edge = @ ∘ ⟨[from ↦ vh, to ↦ vprime, cop ↦ copprime2coph, lr ↦
                                                  [from ↦ uprime, to ↦ vprime, cop ↦ coph2copprime,
```

```
        IN  xFormHelper(u, v, cop, [node ↦ ⟨v⟩,
                                    edge ↦ ⟨[from ↦ u, to ↦ v, cop ↦ cop, lr ↦ 1 − d]⟩])
├──────────────────────────────────────────────────────────────────────────────┤
 Client c ∈ Client perform operation cop guided by the direction flag d.
ClientPerform(cop, c, d) ≜
    LET xss ≜ xForm(cop, css[c], ccur[c], d)
        xn ≜ xss.node
        xe ≜ xss.edge
        xcur ≜ Last(xn)
        xcop ≜ Last(xe).cop
    IN  ∧ css′ = [css EXCEPT ![c].node = @ ∪ Range(xn),
                             ![c].edge = @ ∪ Range(xe)]
        ∧ ccur′ = [ccur EXCEPT ![c] = xcur]
        ∧ state′ = [state EXCEPT ![c] = 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 ↦ ccur[c]]
           IN  ∧ ClientPerform(cop, c, Remote)
               ∧ 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 ⟨sssVars⟩

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

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 (transformed) operation
          IN  ClientPerform(cop, c, Local)
      ∧ UNCHANGED ⟨eVars, cVars, sssVars⟩
```

The *Server* performs operation cop.

```
ServerPerform(cop) ≜
    LET c ≜ cop.oid.c
        xss ≜ xForm(cop, sss[c], scur[c], Remote)
        xn ≜ xss.node
        xe ≜ xss.edge
        xcur ≜ Last(xn)
        xcop ≜ Last(xe).cop
    IN  ∧ sss' = [cl ∈ Client ↦
                    IF cl = c
                     THEN [sss[cl] EXCEPT !.node = @ ∪ Range(xn),
                                          !.edge = @ ∪ Range(xe)]
                     ELSE LET scurcl ≜ scur[cl]
                              scurclprime ≜ scurcl ∪ {cop.oid}
                          IN  [sss[cl] EXCEPT !.node = @ ∪ {scurclprime},
                                              !.edge = @ ∪ {[from ↦ scurcl, to ↦ scurclprime,
                                                             cop ↦ xcop, lr ↦ Remote]}]
                  ]
        ∧ scur' = [cl ∈ Client ↦
                    IF cl = c THEN xcur ELSE scur[cl] ∪ {cop.oid}]
        ∧ state' = [state EXCEPT ![Server] = Apply(xcop.op, @)]
        ∧ comm!SSendSame(c, xcop)  \* broadcast the transformed operation
```

The *Server* receives a message.

```
SRev ≜
    ∧ comm!SRev
    ∧ LET cop ≜ Head(sincoming)
        IN ServerPerform(cop)
    ∧ UNCHANGED ⟨eVars, cVars, cssVars⟩
```

The next-state relation.

```
Next ≜
    ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
    ∨ SRev
```

The *Spec*.

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

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

```
ASSUME (TLCSet(1, ⟨"SameOids", 0⟩))
CSSync ≜
    ∀ c ∈ Client :
        (ccur[c] = scur[c] ∧ TLCCnt(1, 100)) ⇒ css[c] = sss[c]
```

\ * Modification History
\ * *Last* modified *Thu Nov* 01 10:07:53 *CST* 2018 by *hengxin*
\ * Created *Tue Oct* 09 16:33:18 *CST* 2018 by *hengxin*