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 *JupiterCtx*

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

$Local \triangleq 0$

$Remote \triangleq 1$

VARIABLES

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 $r \in Replica$

$vars \triangleq \langle intVars,\ ctxVars,\ cur,\ c2ss,\ s2ss \rangle$

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.

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

$EmptySS \triangleq [node \mapsto \{\{\}\},\ edge \mapsto \{\}]$

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

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

$$
\begin{aligned}
&TypeOK \triangleq \\
&\quad \land\quad TypeOKInt \\
&\quad \land\quad TypeOKCtx \\
&\quad \land\quad Comm(Cop)!TypeOK \\
&\quad \land\quad \forall\, c \in Client : IsSS(c2ss[c]) \land IsSS(s2ss[c]) \\
&\quad \land\quad cur \in [Replica \to \text{SUBSET } Oid]
\end{aligned}
$$

$$
\begin{aligned}
&Init \triangleq \\
&\quad \land\ InitInt \\
&\quad \land\ InitCtx \\
&\quad \land\ Comm(Cop)!Init \\
&\quad \land\ c2ss = [c \in Client \mapsto EmptySS] \\
&\quad \land\ s2ss = [c \in Client \mapsto EmptySS] \\
&\quad \land\ cur\ \ = [r \in Replica \mapsto \{\}]
\end{aligned}
$$

```
    Locate the node in the 2D state space ss which matches the context ctx of cop.
64  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.
70  xForm(cop, ss, current, d) ≜
71      LET u ≜ Locate(cop, ss)
72          v ≜ u ∪ {cop.oid}
73          RECURSIVE xFormHelper(_, _, _, _, _, _)
74           'h' stands for "helper"; xss: eXtra ss created during transformation
75          xFormHelper(uh, vh, coph, xss, xcoph, xcurh) ≜
76              IF uh = current
77               THEN ⟨xss, xcoph, xcurh⟩
78               ELSE LET e ≜ CHOOSE e ∈ ss.edge : e.from = uh ∧ e.lr = d
79                        uprime ≜ e.to
80                        copprime ≜ e.cop
81                        coph2copprime ≜ COT(coph, copprime)
82                        copprime2coph ≜ COT(copprime, coph)
83                         vprime ≜ vh ∪ {copprime.oid}
84                    IN   xFormHelper(uprime, vprime, coph2copprime,
85                           [node ↦ xss.node ∪ {vprime},
86                            edge ↦ xss.edge ∪ {[from ↦ vh, to ↦ vprime, cop ↦ copprime2coph, lr ↦ d],
87                                               [from ↦ uprime, to ↦ vprime, cop ↦ coph2copprime, lr ↦ 1 − d]}],
88                                          coph2copprime, vprime)
89      IN  xFormHelper(u, v, cop, [node ↦ {v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop, lr ↦ 1 − d]}], cop, v)
90  ⊢⊣
    Client c ∈ Client perform operation cop guided by the direction flag d.
94  ClientPerform(cop, c, d) ≜
95      LET xform ≜ xForm(cop, c2ss[c], cur[c], d)  xform: ⟨xss, xcop, xcur⟩
96          xss ≜ xform[1]
97          xcop ≜ xform[2]
98          xcur ≜ xform[3]
99      IN  ∧ c2ss′ = [c2ss EXCEPT ![c] = @ ⊕ xss]
100         ∧ cur′ = [cur EXCEPT ![c] = xcur]
101         ∧ state′ = [state EXCEPT ![c] = Apply(xcop.op, @)]
    Client c ∈ Client generates an operation op.
105 DoOp(c, op) ≜
106       ∧ LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq′[c]], ctx ↦ cur[c]]
107           IN  ∧ ClientPerform(cop, c, Remote)
108               ∧ Comm(Cop)!CSend(cop)

110 DoIns(c) ≜
111     ∃ ins ∈ {op ∈ Ins : op.pos ∈ 1 .. (Len(state[c]) + 1) ∧ op.ch ∈ chins ∧ op.pr = Priority[c]} :
112         ∧ DoOp(c, ins)
113         ∧ 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 chins

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

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

```
Rev(c) ≜
      ∧ Comm(Cop)!CRev(c)
      ∧ LET cop ≜ Head(cincoming[c])   the received (transformed) operation
        IN  ClientPerform(cop, c, Local)
      ∧ RevCtx(c)
      ∧ UNCHANGED ⟨chins, 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]}]
                  ]
        ∧ cur' = [cur EXCEPT ![Server] = xcur]
        ∧ state' = [state EXCEPT ![Server] = Apply(xcop.op, @)]
        ∧ Comm(Cop)!SSendSame(c, xcop)   broadcast the transformed operation
```

The *Server* receives a message.

```
SRev ≜
      ∧ Comm(Cop)!SRev
      ∧ LET cop ≜ Head(sincoming)
        IN  ServerPerform(cop)
      ∧ SRevCtx
      ∧ UNCHANGED ⟨chins, c2ss⟩
```

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

$$Fairness \triangleq$$
$$\quad \mathrm{WF}_{vars}(SRev \lor \exists\, c \in Client : Rev(c))$$

$$Spec \triangleq Init \land \Box[Next]_{vars} \land Fairness$$

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

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

* Modification History
* Last modified Sat *Dec* 15 17:39:52 *CST* 2018 by *hengxin*
* Created *Tue Oct* 09 16:33:18 *CST* 2018 by *hengxin*