MODULE *CJupiter*

Model of our own *CJupiter* protocol.

EXTENDS *JupiterSerial*

VARIABLES

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 $r \in Replica$

$vars \triangleq \langle intVars, ctxVars, serialVars, css, cur \rangle$

A *css* is a directed graph with labeled edges, represented by a record with node field and edge field. Each node is characterized by its context, a set of oids. Each edge is labeled with an operation.

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

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

$$
\begin{aligned}
&TypeOK \triangleq \\
&\quad \land \quad TypeOKInt \\
&\quad \land \quad TypeOKCtx \\
&\quad \land \quad TypeOKSerial \\
&\quad \land \quad Comm(Cop)!\,TypeOK \\
&\quad \land \quad \forall\, r \in Replica : IsCSS(css[r]) \\
&\quad \land \quad cur \in [Replica \to \text{SUBSET } Oid]
\end{aligned}
$$

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

Locate the node in *rcss* (the *css* at replica $r \in Replica$) that matches the context *ctx* of cop.

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

Take union of two state spaces *ss*1 and *ss*2.

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

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

$$
\begin{aligned}
&xForm(cop, r) \triangleq \\
&\quad \text{LET } rcss \triangleq css[r] \\
&\qquad\ \ u \triangleq Locate(cop, rcss)
\end{aligned}
$$

```
60          v ≜ u ∪ {cop.oid}
61          RECURSIVE xFormHelper(_, _, _, _, _, _)
62           'h' stands for "helper"; xcss: eXtra css created during transformation
63          xFormHelper(uh, vh, coph, xcss, xcoph, xcurh) ≜
64              IF uh = cur[r]
65               THEN ⟨xcss, xcoph, xcurh⟩
66               ELSE LET fedge ≜ CHOOSE e ∈ rcss.edge :
67                                   ∧ e.from = uh
68                                   ∧ ∀ uhe ∈ rcss.edge :
69                                       (uhe.from = uh ∧ uhe ≠ e) ⇒ tb(e.cop.oid, uhe.cop.oid, serial[r])
70                        uprime ≜ fedge.to
71                        fcop ≜ fedge.cop
72                        coph2fcop ≜ COT(coph, fcop)
73                        fcop2coph ≜ COT(fcop, coph)
74                         vprime ≜ vh ∪ {fcop.oid}
75                    IN    xFormHelper(uprime, vprime, coph2fcop,
76                             [xcss EXCEPT !.node = @ ∪ {vprime},
77                               !.edge = @ ∪ {[from ↦ vh, to ↦ vprime, cop ↦ fcop2coph],
78                                             [from ↦ uprime, to ↦ vprime, cop ↦ coph2fcop]}],
79                             coph2fcop, vprime)
80      IN    xFormHelper(u, v, cop, [node ↦ {v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop]}], cop, v)
    Perform cop at replica r ∈ Replica.
84  Perform(cop, r) ≜
85      LET xform ≜ xForm(cop, r)   xform: ⟨xcss, xcop, xcur⟩
86          xcss ≜ xform[1]
87          xcop ≜ xform[2]
88          xcur ≜ xform[3]
89      IN  ∧ css′ = [css EXCEPT ![r] = @ ⊕ xcss]
90          ∧ cur′ = [cur EXCEPT ![r] = xcur]
91          ∧ state′ = [state EXCEPT ![r] = Apply(xcop.op, @)]
92  ──────────────────────────────────────────────────────────────────────
    Client c ∈ Client issues an operation op.
96  DoOp(c, op) ≜   op: the raw operation generated by the client c ∈ Client
97        ∧ LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq′[c]], ctx ↦ cur[c]]
98            IN  ∧ Perform(cop, c)
99                ∧ Comm(Cop)!CSend(cop)

101 DoIns(c) ≜
102     ∃ ins ∈ {op ∈ Ins : op.pos ∈ 1 .. (Len(state[c]) + 1) ∧ op.ch ∈ chins ∧ op.pr = Priority[c]} :
103         ∧ DoOp(c, ins)
104         ∧ chins′ = chins \ {ins.ch}   We assume that all inserted elements are unique.

106 DoDel(c) ≜
107     ∃ del ∈ {op ∈ Del : op.pos ∈ 1 .. Len(state[c])} :
108         ∧ DoOp(c, del)
```

```
109         ∧ UNCHANGED chins

111 Do(c) ≜
112         ∧ DoCtx(c)
113         ∧ DoSerial(c)
114         ∧ ∨ DoIns(c)
115           ∨ DoDel(c)
    Client c ∈ Client receives a message from the Server.
119 Rev(c) ≜
120         ∧ Comm(Cop)!CRev(c)
121         ∧ Perform(Head(cincoming[c]), c)
122         ∧ RevSerial(c)
123         ∧ RevCtx(c)
124         ∧ UNCHANGED chins
125 ├──────────────────────────────────────────────────────────────────────┤
    The Server receives a message.
129 SRev ≜
130       ∧ Comm(Cop)!SRev
131       ∧ LET cop ≜ Head(sincoming)
132           IN  ∧ Perform(cop, Server)
133               ∧ Comm(Cop)!SSendSame(cop.oid.c, cop)   broadcast the original operation
134       ∧ SRevSerial
135       ∧ SRevCtx
136       ∧ UNCHANGED chins
137 ├──────────────────────────────────────────────────────────────────────┤
138 Next ≜
139       ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
140       ∨ SRev
    Fairness: There is no requirement that the clients ever generate operations.
144 Fairness ≜
145       WF_vars(SRev ∨ ∃ c ∈ Client : Rev(c))

147 Spec ≜ Init ∧ □[Next]_vars   ∧ Fairness (We care more about safety.)
148 ├──────────────────────────────────────────────────────────────────────┤
    The compactness of CJupiter: the CSSes at all replicas are the same.
152 Compactness ≜
153       Comm(Cop)!EmptyChannel ⇒ Cardinality(Range(css)) = 1

155 THEOREM Spec ⇒ Compactness
156 └──────────────────────────────────────────────────────────────────────┘
    \* Modification History
    \* Last modified Sat Dec 15 17:35:17 CST 2018 by hengxin
    \* Created Sat Sep 01 11:08:00 CST 2018 by hengxin
```