1 MODULE *CJupiter*

Model of our own *CJupiter* protocol.

6 EXTENDS *Integers*, *OT*, *TLC*, *AdditionalFunctionOperators*

7

8 CONSTANTS

9 *Client*, the set of client replicas

10 *Server*, the (unique) server replica

11 *Char*, set of characters allowed

12 *InitState* the initial state of each replica

14 $Replica \triangleq Client \cup \{Server\}$

16 $List \triangleq Seq(Char \cup Range(InitState))$ all possible lists/strings

17 $MaxLen \triangleq Cardinality(Char) + Len(InitState)$ the max length of lists in any states;

18 We assume that all inserted elements are unique.

20 $ClientNum \triangleq Cardinality(Client)$

21 $Priority \triangleq \text{CHOOSE } f \in [Client \to 1 \,..\, ClientNum] : Injective(f)$

22

23 ASSUME

24 $\land\ Range(InitState) \cap Char = \{\}$

25 $\land\ Priority \in [Client \to 1 \,..\, ClientNum]$

26

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

31 $Rd \triangleq [type : \{\text{“Rd”}\}]$

32 $Del \triangleq [type : \{\text{“Del”}\}, pos : 1 \,..\, MaxLen]$

33 $Ins \triangleq [type : \{\text{“Ins”}\}, pos : 1 \,..\, (MaxLen + 1), ch : Char, pr : 1 \,..\, ClientNum]$ *pr*: priority

35 $Op \triangleq Ins \cup Del$ Now we don't consider *Rd* operations.

36

41 $Oid \triangleq [c : Client, seq : Nat]$ operation identifier

42 $Cop \triangleq [op : Op, oid : Oid, ctx : \text{SUBSET } Oid, sctx : \text{SUBSET } Oid]$ operation with context

44 $cop1 \prec cop2 \triangleq$

45 $\lor\ cop2.sctx = \{\}$

46 $\lor\ cop1.oid \in cop2.sctx$

48 $COT(lcop, rcop) \triangleq$

49 $[op \mapsto Xform(lcop.op, rcop.op), oid \mapsto lcop.oid,$

50 $ctx \mapsto lcop.ctx \cup \{rcop.oid\}, sctx \mapsto lcop.stx]$

51

52 VARIABLES

For the client replicas:

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

57 *cstate*, *cstate*[*c*]: state (the list content) of the client $c \in Client$

For the server replica:

sstate, *sstate*: state (the list content) of the server *Server*

For all replicas: the n-ary ordered state space

css, *css*[r]: the n-ary ordered state space at replica r

cur, *cur*[r]: the current node of *css* at replica r

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

$comm \triangleq$ INSTANCE $CSComm$

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

$cVars \triangleq \langle cseq, cstate \rangle$ variables for the clients

$ecVars \triangleq \langle eVars, cVars \rangle$ variables for the clients and the environment

$sVars \triangleq \langle sstate \rangle$ variables for the server

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

$vars \triangleq \langle eVars, cVars, sVars, commVars, css, cur \rangle$ 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.

$IsCSS(G) \triangleq$

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

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

$\land\ G.edge \subseteq [from : G.node,\ to : G.node,\ cop : Cop]$

$TypeOK \triangleq$

For the client replicas:

$\land\ cseq \in [Client \to Nat]$

$\land\ cstate \in [Client \to List]$

For the server replica:

$\land\ sstate \in List$

For all replicas: the n-ary ordered state space

$\land\ \forall\, r \in Replica : IsCSS(r)$

$\land\ cur \in [Client \to \text{SUBSET } Oid]$

For communication between the server and the clients:

$\land\ comm\,!\,TypeOK$

For model checking:

```
124     ∧ chins ⊆ Char
125 ├──────────────────────────────────────────────────────────────────────┤
    The Init predicate.
129 Init ≜
130     ∧ chins = Char
        For the client replicas:
134     ∧ cseq = [c ∈ Client ↦ 0]
135     ∧ cstate = [c ∈ Client ↦ InitState]
        For the server replica:
139     ∧ sstate = InitState
        For all replicas: the n-ary ordered state space
143     ∧ css = [c ∈ Client ↦ [node ↦ {}, edge ↦ {}]]
144     ∧ cur = {}
        For communication between the server and the clients:
148     ∧ comm ! Init
149 ├──────────────────────────────────────────────────────────────────────┤
    Client c ∈ Client issues an operation op.
153 DoOp(c, op) ≜
154         ∧ cstate′ = [cstate EXCEPT ![c] = Apply(op, @)]
155         ∧ cseq′ = [cseq EXCEPT ![c] = @ + 1]
156         ∧ LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq′[c]],
157               ctx ↦ cur[c], sctx ↦ {}]
158                 v ≜ cur ∪ {cop.oid}
159             IN  ∧ css′ = [css EXCEPT ![c].node = @ ∪ {v},
160                                      ![c].edge = @ ∪ {[from ↦ cur, to ↦ v, cop ↦ cop]}]
161                 ∧ cur′ = v
162                 ∧ comm ! CSend([c ↦ c, op ↦ cop])

164 DoIns(c) ≜
165     ∃ ins ∈ Ins :
166         ∧ ins.pos ∈ 1 .. (Len(cstate[c]) + 1)
167         ∧ ins.ch ∈ chins
168         ∧ ins.pr = Priority[c]
169         ∧ chins′ = chins \ {ins.ch}   We assume that all inserted elements are unique.
170         ∧ DoOp(c, ins)
171         ∧ UNCHANGED sVars

173 DoDel(c) ≜
174     ∃ del ∈ Del :
175         ∧ del.pos ∈ 1 .. Len(cstate[c])
176         ∧ DoOp(c, del)
177         ∧ UNCHANGED ⟨sVars, eVars⟩

179 Do(c) ≜
```

```
180         ∨ DoIns(c)
181         ∨ DoDel(c)
    Locate the node in rcss which matches the context ctx of cop.
    rcss: the css at replica r ∈ Replica
187 Locate(cop, rcss) ≜ CHOOSE n ∈ (rcss.node) : n = cop.ctx
    xForm: iteratively transform cop with a path through the css at replica r ∈ Replica, following
    the first edges.
192 RECURSIVE xForm(_, _)
193 xForm(cop, r) ≜
194     LET rcss ≜ css[r]
195         u ≜ Locate(cop, rcss)
196         v ≜ u ∪ {cop.oid}
197         RECURSIVE xFormHelper(_, _, _)
198          xFormHelper(uh, vh, coph) ≜
199              IF uh = cur[r]
200               THEN css′ = [css EXCEPT ![r].node = @ ∪ {vh},
201                                       ![r].edge = @ ∪ {[from ↦ uh, to ↦ vh, cop ↦ coph]}]
202               ELSE LET fedge ≜ CHOOSE e ∈ rcss.edge :
203                                  ∧ e.from = uh
204                                  ∧ ∀ ue ∈ rcss.edge :
205                                      (ue.from = uh ∧ ue ≠ e) ⇒ (e.cop ≺ ue.cop)
206                        uprime ≜ fedge.to
207                        fcop ≜ fedge.cop
208                        cop2fcop ≜ COT(cop, fcop)
209                        fcop2cop ≜ COT(fcop, cop)
210                         vprime ≜ v.oids ∪ {fcop.oid}
211                    IN    ∧ css′ = [css EXCEPT ![r].node = @ ∪ {vh},
212                                               ![r].edge = @ ∪ {[from ↦ uh, to ↦ vh, cop ↦ coph],
213                                                               [from ↦ vh, to ↦ vprime, cop ↦ fcop2cop]}]
214                          ∧ xFormHelper(uprime, vprime, cop2fcop)
215     IN    xFormHelper(u, v, cop)

    Client c ∈ Client receives a message from the Server.
220 Rev(c) ≜
221     ∧ comm!CRev(c)
222     ∧ LET m ≜ Head(cincoming[c])
223        IN   ∧ TRUE
224          ∧ cstate′ = [cstate EXCEPT ![c] = Apply(xop, @)] \* apply the transformed operation xop
225     ∧ UNCHANGED ⟨sVars, eVars⟩
226 ├──────────────────────────────────────────────────────────────────────────┤
    The Server receives a message.
230 SRev ≜
231     ∧ comm!SRev
232     ∧ LET m ≜ Head(sincoming)  the message to handle with
```

```
233             IN    ∧ TRUE
234          ∧ sstate′ = Apply(xop, sstate) \* apply the transformed operation
235          ∧ comm!SSend(c, srec, xop)
236       ∧ UNCHANGED ecVars
237 ----------------------------------------------------------------------
    The next-state relation.
241 Next ≜
242       ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
243       ∨ SRev
    The Spec. (TODO: Check the fairness condition.)
247 Spec ≜ Init ∧ □[Next]_vars ∧ WF_vars(Next)
248 ======================================================================
    \* Modification History
    \* Last modified Sun Sep 02 10:59:51 CST 2018 by hengxin
    \* Created Sat Sep 01 11:08:00 CST 2018 by hengxin
```