MODULE *AJupiter*

Model checking the *Jupiter* protocol presented by *Attiya* and others.

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

CONSTANTS

Client, the set of client replicas

Server, the (unique) server replica

Char, set of characters allowed

InitState the initial state of each replica

$List \triangleq Seq(Char \cup Range(InitState))$ 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.

$ClientNum \triangleq Cardinality(Client)$

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

ASSUME

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

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

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

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

VARIABLES

For the client replicas:

cbuf, *cbuf*[*c*]: buffer (of operations) at the client $c \in Client$

crec, *crec*[*c*]: the number of new messages have been received by the client $c \in Client$ since the last time a message was sent

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

For the server replica:

sbuf, *sbuf*[*c*]: buffer (of operations) at the *Server*, one per client $c \in Client$

srec, *srec*[*c*]: the number of new messages have been ..., one per client $c \in Client$

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

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:

```
58      list,      all list states across the system
59      chins      a set of chars to insert

61  ----------------------------------------------------------------
62  comm ≜ INSTANCE CSComm
63  ----------------------------------------------------------------
64  eVars ≜ ⟨chins⟩                                  variables for the environment
65  cVars ≜ ⟨cbuf, crec, cstate⟩                     variables for the clients
66  ecVars ≜ ⟨eVars, cVars⟩                          variables for the clients and the environment
67  sVars ≜ ⟨sbuf, srec, sstate⟩                     variables for the server
68  commVars ≜ ⟨cincoming, sincoming⟩                variables for communication
69  vars ≜ ⟨eVars, cVars, sVars, commVars, list⟩     all variables
70  ----------------------------------------------------------------
71  TypeOK ≜
        For the client replicas:
75      ∧ cbuf ∈ [Client → Seq(Op ∪ {Nop})]
76      ∧ crec ∈ [Client → Int]
77      ∧ cstate ∈ [Client → List]
        For the server replica:
81      ∧ sbuf ∈ [Client → Seq(Op ∪ {Nop})]
82      ∧ srec ∈ [Client → Int]
83      ∧ sstate ∈ List
        For communication between the server and the clients:
87      ∧ comm!TypeOK
        For model checking:
91      ∧ list ∈ SUBSET List
92      ∧ chins ∈ SUBSET Char
93  ----------------------------------------------------------------
    The Init predicate.
97  Init ≜
98      ∧ list = {InitState}
99      ∧ chins = Char
        For the client replicas:
103     ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
104     ∧ crec = [c ∈ Client ↦ 0]
105     ∧ cstate = [c ∈ Client ↦ InitState]
        For the server replica:
109     ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
110     ∧ srec = [c ∈ Client ↦ 0]
111     ∧ sstate = InitState
        For communication between the server and the clients:
115     ∧ comm!Init
116 ----------------------------------------------------------------
```

Client $c \in Client$ issues an operation op.

$DoOp(c,\ op) \triangleq$
  $\land\ cstate' = [cstate \text{ EXCEPT } ![c] = Apply(op,\ @)]$
  $\land\ list' = list \cup \{cstate'[c]\}$
  $\land\ cbuf' = [cbuf \text{ EXCEPT } ![c] = Append(@,\ op)]$
  $\land\ crec' = [crec \text{ EXCEPT } ![c] = 0]$
  $\land\ comm!CSend([c \mapsto c,\ ack \mapsto crec[c],\ op \mapsto op])$

$DoIns(c) \triangleq$
  $\exists\, ins \in Ins :$
    $\land\ ins.pos \in 1\ldots(Len(cstate[c]) + 1)$
    $\land\ ins.ch \in chins$
    $\land\ ins.pr = Priority[c]$
    $\land\ chins' = chins \setminus \{ins.ch\}$ We assume that all inserted elements are unique.
    $\land\ DoOp(c,\ ins)$
    $\land$ UNCHANGED $sVars$

$DoDel(c) \triangleq$
  $\exists\, del \in Del :$
    $\land\ del.pos \in 1\ldots Len(cstate[c])$
    $\land\ DoOp(c,\ del)$
    $\land$ UNCHANGED $\langle sVars,\ eVars\rangle$

$Do(c) \triangleq$
  $\lor\ DoIns(c)$
  $\lor\ DoDel(c)$

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

$Rev(c) \triangleq$
  $\land\ comm!CRev(c)$
  $\land\ crec' = [crec \text{ EXCEPT } ![c] = @ + 1]$
  $\land$ LET $m \triangleq Head(cincoming[c])$
      $cBuf \triangleq cbuf[c]$ the buffer at client $c \in Client$
      $cShiftedBuf \triangleq SubSeq(cBuf,\ m.ack + 1,\ Len(cBuf))$ buffer shifted
      $xop \triangleq XformOpOps(m.op,\ cShiftedBuf)$ transform op vs. shifted buffer
      $xcBuf \triangleq XformOpsOp(cShiftedBuf,\ m.op)$ transform shifted buffer vs. op
    IN $\land\ cbuf' = [cbuf \text{ EXCEPT } ![c] = xcBuf]$
       $\land\ cstate' = [cstate \text{ EXCEPT } ![c] = Apply(xop,\ @)]$ apply the transformed operation xop
       $\land\ list' = list \cup \{cstate'[c]\}$
  $\land$ UNCHANGED $\langle sVars,\ eVars\rangle$

The $Server$ receives a message.

$SRev \triangleq$
  $\land\ comm!SRev$
  $\land$ LET $m \triangleq Head(sincoming)$ the message to handle with

```
168             c ≜ m.c                          the client c ∈ Client that sends this message
169             cBuf ≜ sbuf[c]                   the buffer at the Server for client c ∈ Client
170             cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
171             xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
172             xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
173         IN  ∧ srec' = [cl ∈ Client ↦
174                          IF cl = c
175                           THEN srec[cl] + 1   receive one more operation from client c ∈ Client
176                           ELSE 0]   reset srec for other clients than c ∈ Client
177             ∧ sbuf' = [cl ∈ Client ↦
178                          IF cl = c
179                           THEN xcBuf   transformed buffer for client c ∈ Client
180                           ELSE Append(sbuf[cl], xop)]   store transformed xop into other clients' bufs
181             ∧ sstate' = Apply(xop, sstate)   apply the transformed operation
182             ∧ list' = list ∪ {sstate'}
183             ∧ comm!SSend(c, srec, xop)
184     ∧ UNCHANGED ecVars
185 ----------------------------------------------------------------------------
    The next-state relation.
189 Next ≜
190     ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
191     ∨ SRev
    The Spec. (TODO: Check the fairness condition.)
195 Spec ≜ Init ∧ □[Next]_vars ∧ WF_vars(Next)
196 ----------------------------------------------------------------------------
    The safety properties to check: Eventual Convergence (EC), Quiescent Consistency (QC), Strong
    Eventual Convergence (SEC), Weak List Specification, (WLSpec), and Strong List Specification,
    (SLSpec).

    Eventual Consistency (EC)

    Quiescent Consistency (QC)
211 QConvergence ≜ ∀ c ∈ Client : cstate[c] = sstate
212 QC ≜ comm!EmptyChannel ⇒ QConvergence

214 THEOREM Spec ⇒ □QC

    Strong Eventual Consistency (SEC)

    Termination
223 Termination ≜
224     ∧ comm!EmptyChannel

    Weak List Consistency (WLSpec)
229 WLSpec ≜
```

$$\wedge \quad Termination \Rightarrow \forall\, l1,\, l2 \in list :$$
$$\wedge\, Injective(l1)$$
$$\wedge\, Injective(l2)$$
$$\wedge\, Compatible(l1,\, l2)$$

THEOREM $Spec \Rightarrow WLSpec$

Strong *List* Consistency (*SLSpec*)

* Modification History
* Last modified *Tue Aug* 28 19:34:29 *CST* 2018 by *hengxin*
* Created Sat *Jun* 23 17:14:18 *CST* 2018 by *hengxin*