MODULE $AJupiter$

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

EXTENDS $Integers$, OT, TLC

CONSTANTS

$Client$, the set of client replicas

$Server$, the (unique) server replica

$InitState$, the initial state of each replica

$Priority$ $Priority[c]$: the priority value of client $c \in Client$

Cop * $Cop[c]$: operations issued by the client $c \in Client$

ASSUME

$\land\ InitState \in List$

$\land\ Priority \in [Client \to PosInt]$

$\land$ Cop $\in [Client \to Seq(Op)]$

Generate operations for $AJupiter$ clients.

Note: Remember to overvide the definition of $PosInt$.

FIXME: $PosInt \Rightarrow MaxPos$; $MaxPr$ determined by the size of $Client$.

$$OpToIssue \triangleq \{opset \in \text{SUBSET } Op :$$
$$\land\ opset \neq \{\}$$
$$\land\ \forall\, op1 \in opset :$$
$$\forall\, op2 \in opset \setminus \{op1\} :$$
$$(op1.type = \text{“Ins”} \land op2.type = \text{“Ins”}) \Rightarrow op1.ch \neq op2.ch\}$$

VARIABLES

For model checking:

cop, * $cop[c]$: operations issued by the client $c \in Client$

cop, a set of operations for clients to issue

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 $Server$

```
60  comm ≜ INSTANCE CSComm
61  ----------------------------------------------------------------------
62  eVars ≜ ⟨cop⟩                                  variables for the environment
63  cVars ≜ ⟨cbuf, crec, cstate⟩                   variables for the clients
64  ecVars ≜ ⟨cop, cVars⟩                          variables for the clients and the environment
65  sVars ≜ ⟨sbuf, srec, sstate⟩                   variables for the server
66  commVars ≜ ⟨cincoming, sincoming⟩              variables for communication
67  jVars ≜ ⟨cVars, sVars, commVars⟩               variables for the Jupiter system
68  vars ≜ ⟨eVars, cVars, sVars, commVars⟩         all variables
69  ----------------------------------------------------------------------
70  TypeOK ≜
71     ∧ cop ∈ [Client → Seq(Op)]
72     ∧ cop ∈ SUBSET Op
       For the client replicas:
76     ∧ cbuf ∈ [Client → Seq(Op ∪ {Nop})]
77     ∧ crec ∈ [Client → Int]
78     ∧ cstate ∈ [Client → List]
       For the server replica:
82     ∧ sbuf ∈ [Client → Seq(Op ∪ {Nop})]
83     ∧ srec ∈ [Client → Int]
84     ∧ sstate ∈ List
       For communication between the server and the clients:
88     ∧ comm!TypeOK
89  ----------------------------------------------------------------------
    The Init predicate.
93  Init ≜
94     ∧ cop = Cop
95     ∧ cop ∈ OpToIssue
       For the client replicas:
99     ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
100    ∧ crec = [c ∈ Client ↦ 0]
101    ∧ cstate = [c ∈ Client ↦ InitState]
       For the server replica:
105    ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
106    ∧ srec = [c ∈ Client ↦ 0]
107    ∧ sstate = InitState
       For communication between the server and the clients:
111    ∧ comm!Init
112 ----------------------------------------------------------------------
113 LegalizeOp(op, c) ≜
114    LET len ≜ Len(cstate[c])
115    IN  CASE op.type = "Del" →
116              IF len = 0 THEN Nop ELSE [op EXCEPT !.pos = Min(@, len)]
```

```
117             □   op.type = "Ins" →
118                 [op EXCEPT !.pos = Min(@, len + 1), !.pr = Priority[c]]

    Client c ∈ Client issues an operation op.
123 Do(c) ≜
124   ∧ cop[c] ≠ ⟨⟩
125     ∧ cop ≠ {}
126     ∧ ∃ o ∈ cop :
127         LET op ≜ LegalizeOp(o, c)   preprocess an illegal operation
128         IN   ∨ ∧ op = Nop
129                ∧ cop' = cop \ {o}      consume one operation
130                ∧ UNCHANGED jVars
131              ∨ ∧ op ≠ Nop
132                ∧ PrintT(c ∘ " : Do" ∘ ToString(op))
133                ∧ cstate' = [cstate EXCEPT ![c] = Apply(op, @)]
134                ∧ cbuf' = [cbuf EXCEPT ![c] = Append(@, op)]
135                ∧ crec' = [crec EXCEPT ![c] = 0]
136                ∧ comm!CSend([c ↦ c, ack ↦ crec[c], op ↦ op])
137                ∧ cop' = cop \ {o}      consume one operation
138                ∧ UNCHANGED sVars
139   ∧ cop' = [cop EXCEPT ![c] = Tail(@)] \* consume one operation

    Client c ∈ Client receives a message from the Server.
144 Rev(c) ≜
145     ∧ comm!CRev(c)
146     ∧ crec' = [crec EXCEPT ![c] = @ + 1]
147     ∧ LET m ≜ Head(cincoming[c])
148           cBuf ≜ cbuf[c]   the buffer at client c ∈ Client
149           cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
150           xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
151           xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
152       IN   ∧ cbuf' = [cbuf EXCEPT ![c] = xcBuf]
153            ∧ cstate' = [cstate EXCEPT ![c] = Apply(xop, @)]   apply the transformed operation xop
154     ∧ UNCHANGED ⟨sbuf, srec, sstate, cop⟩   NOTE: sVars ∘ ⟨cop⟩ is wrong!
155 ├──────────────────────────────────────────────────────────────────────┤

    The Server receives a message.
159 SRev ≜
160     ∧ comm!SRev
161     ∧ LET m ≜ Head(sincoming)   the message to handle with
162           c ≜ m.c   the client c ∈ Client that sends this message
163           cBuf ≜ sbuf[c]   the buffer at the Server for client c ∈ Client
164           cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
165           xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
166           xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
167       IN   ∧ srec' = [cl ∈ Client ↦
```

```
                                IF cl = c
                                 THEN srec[cl] + 1 receive one more operation from client c ∈ Client
                                 ELSE 0] reset srec for other clients than c ∈ Client
          ∧ sbuf' = [cl ∈ Client ↦
                                IF cl = c
                                 THEN xcBuf  transformed buffer for client c ∈ Client
                                 ELSE Append(sbuf[cl], xop)] store transformed xop into other clients' bufs
          ∧ sstate' = Apply(xop, sstate)  apply the transformed operation
          ∧ comm!SSend(c, srec, xop)
    ∧ UNCHANGED ecVars
```

The next-state relation.

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

The *Spec*. (*TODO*: Check the fairness condition.)

```
Spec ≜ Init ∧ □[Next]_vars ∧ WF_vars(Next)
```

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*)

```
QConvergence ≜ ∀ c ∈ Client : cstate[c] = sstate
QC ≜ comm!EmptyChannel ⇒ QConvergence

THEOREM Spec ⇒ □QC
```

Strong Eventual Consistency (*SEC*)

Termination

```
Termination ≜
    ∧ cop = {}
    ∧ comm!EmptyChannel
```

Weak *List* Consistency (*WLSpec*)

Strong *List* Consistency (*SLSpec*)

```
\* Modification History
\* Last modified Sun Aug 12 22:22:32 CST 2018 by hengxin
\* Created Sat Jun 23 17:14:18 CST 2018 by hengxin
```