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

$list$, all list states across the system

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 ├──────────────────────────────────────────────────────────────────────┤
61  comm ≜ INSTANCE CSComm
62 ├──────────────────────────────────────────────────────────────────────┤
63  eVars ≜ ⟨cop⟩                                  variables for the environment
64  cVars ≜ ⟨cbuf, crec, cstate⟩                   variables for the clients
65  ecVars ≜ ⟨cop, cVars⟩                          variables for the clients and the environment
66  sVars ≜ ⟨sbuf, srec, sstate⟩                   variables for the server
67  commVars ≜ ⟨cincoming, sincoming⟩              variables for communication
68  jVars ≜ ⟨cVars, sVars, commVars⟩               variables for the Jupiter system
69  vars ≜ ⟨eVars, cVars, sVars, commVars, list⟩   all variables
70 ├──────────────────────────────────────────────────────────────────────┤
71  TypeOK ≜
72     ∧ cop ∈ [Client → Seq(Op)]
73     ∧ cop ∈ SUBSET Op
74     ∧ list ∈ SUBSET List
       For the client replicas:
78     ∧ cbuf ∈ [Client → Seq(Op ∪ {Nop})]
79     ∧ crec ∈ [Client → Int]
80     ∧ cstate ∈ [Client → List]
       For the server replica:
84     ∧ sbuf ∈ [Client → Seq(Op ∪ {Nop})]
85     ∧ srec ∈ [Client → Int]
86     ∧ sstate ∈ List
       For communication between the server and the clients:
90     ∧ comm!TypeOK
91 ├──────────────────────────────────────────────────────────────────────┤
   The Init predicate.
95  Init ≜
96     ∧ cop = Cop
97     ∧ PrintT(Cardinality(OpToIssue))
98     ∧ cop ∈ OpToIssue
99     ∧ list = {InitState}
       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 ├──────────────────────────────────────────────────────────────────────┤
```

```
LegalizeOp(op, c) ≜
    LET len ≜ Len(cstate[c])
    IN  CASE op.type = "Del" →
                IF len = 0 THEN Nop ELSE [op EXCEPT !.pos = Min(@, len)]
          □     op.type = "Ins" →
                [op EXCEPT !.pos = Min(@, len + 1), !.pr = Priority[c]]

Client c ∈ Client issues an operation op.
Do(c) ≜
  ∧ cop[c] ≠ ⟨⟩
    ∧ cop ≠ {}
    ∧ ∃ o ∈ cop :
        LET op ≜ LegalizeOp(o, c)   preprocess an illegal operation
        IN  ∨ ∧ op = Nop
              ∧ cop′ = cop \ {o}      consume one operation
              ∧ UNCHANGED ⟨jVars, list⟩
            ∨ ∧ op ≠ Nop
              ∧ PrintT(c ∘ " : Do" ∘ ToString(op))
              ∧ cstate′ = [cstate EXCEPT ![c] = Apply(op, @)]
              ∧ list′   = list ∪ {cstate′[c]}
              ∧ cbuf′ = [cbuf EXCEPT ![c] = Append(@, op)]
              ∧ crec′ = [crec EXCEPT ![c] = 0]
              ∧ comm!CSend([c ↦ c, ack ↦ crec[c], op ↦ op])
              ∧ cop′ = cop \ {o}     consume one operation
              ∧ UNCHANGED sVars
  ∧ cop′ = [cop EXCEPT ![c] = Tail(@)] \* consume one operation

Client c ∈ Client receives a message from the Server.
Rev(c) ≜
    ∧ comm!CRev(c)
    ∧ crec′ = [crec EXCEPT ![c] = @ + 1]
    ∧ LET m ≜ Head(cincoming[c])
          cBuf ≜ cbuf[c]   the buffer at client c ∈ Client
          cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
          xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
          xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
      IN  ∧ cbuf′ = [cbuf EXCEPT ![c] = xcBuf]
          ∧ cstate′ = [cstate EXCEPT ![c] = Apply(xop, @)]   apply the transformed operation xop
          ∧ list′ = list ∪ {cstate′[c]}
    ∧ UNCHANGED ⟨sbuf, srec, sstate, cop⟩   NOTE: sVars ∘ ⟨cop⟩ is wrong!
-----------------------------------------------------------------------------
The Server receives a message.
SRev ≜
    ∧ comm!SRev
    ∧ LET m ≜ 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     ∧ cop = {}
225     ∧ comm!EmptyChannel

    Weak List Consistency (WLSpec) ∧ Termination ⇒ ∀ l1, l2 ∈ list: Compatible(l1, l2)
```

231 $WLSpec \triangleq$

232 $\quad \wedge \quad Termination \Rightarrow \forall\, l1,\, l2 \in list : Compatible(l1,\, l2)$

234 THEOREM $Spec \Rightarrow WLSpec$

Strong *List* Consistency (*SLSpec*)

238

* Modification History

* Last modified *Thu Aug* 16 23:18:05 *CST* 2018 by *hengxin*

* Created Sat *Jun* 23 17:14:18 *CST* 2018 by *hengxin*