MODULE *AJupiter*

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

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

CONSTANTS

$\quad Client,$ the set of client replicas

$\quad Server,$ the (unique) server replica

$\quad Char,$ set of characters allowed

$\quad InitState$ 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.

$ClientNum \triangleq Cardinality(Client)$

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

ASSUME

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

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

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.

Messages between the *Server* and the Clients. There are two kinds of messages according to their destinations.

$Msg \triangleq [c : Client, ack : Int, op : Op \cup \{Nop\}] \cup$ messages sent to the *Server* from a client $c \in Client$

$\qquad [ack : Int, op : Op \cup \{Nop\}]$ messages broadcast to Clients from the *Server*

VARIABLES

For the client replicas:

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

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

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

For the server replica:

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

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

$\quad sstate,$ *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 ∈ Client
        sincoming,   incoming channel at the Server
        For model checking:
        chins    a set of chars to insert

├──────────────────────────────────────────────────────────────────────┤
comm ≜ INSTANCE CSComm WITH Msg ← Msg
├──────────────────────────────────────────────────────────────────────┤
eVars ≜ ⟨chins⟩                                  variables for the environment
cVars ≜ ⟨cbuf, crec, cstate⟩                     variables for the clients
ecVars ≜ ⟨eVars, cVars⟩                          variables for the clients and the environment
sVars ≜ ⟨sbuf, srec, sstate⟩                     variables for the server
commVars ≜ ⟨cincoming, sincoming⟩                variables for communication
vars ≜ ⟨eVars, cVars, sVars, commVars⟩           all variables
├──────────────────────────────────────────────────────────────────────┤
TypeOK ≜
    For the client replicas:
    ∧ cbuf  ∈ [Client → Seq(Op ∪ {Nop})]
    ∧ crec  ∈ [Client → Int]
    ∧ cstate ∈ [Client → List]
    For the server replica:
    ∧ sbuf  ∈ [Client → Seq(Op ∪ {Nop})]
    ∧ srec  ∈ [Client → Int]
    ∧ sstate ∈ List
    For communication between the server and the clients:
    ∧ comm!TypeOK
    For model checking:
    ∧ chins ∈ SUBSET Char
├──────────────────────────────────────────────────────────────────────┤
The Init predicate.
Init ≜
    ∧ chins = Char
    For the client replicas:
    ∧ cbuf  = [c ∈ Client ↦ ⟨⟩]
    ∧ crec  = [c ∈ Client ↦ 0]
    ∧ cstate = [c ∈ Client ↦ InitState]
    For the server replica:
    ∧ sbuf  = [c ∈ Client ↦ ⟨⟩]
    ∧ srec  = [c ∈ Client ↦ 0]
    ∧ sstate = InitState
    For communication between the server and the clients:
    ∧ comm!Init
├──────────────────────────────────────────────────────────────────────┤
```

```
Client $c \in Client$ issues an operation $op$.
$DoOp(c, op) \triangleq$
        $\land cstate' = [cstate \text{ EXCEPT } ![c] = Apply(op, @)]$
        $\land cbuf' = [cbuf \text{ EXCEPT } ![c] = Append(@, op)]$
        $\land crec' = [crec \text{ EXCEPT } ![c] = 0]$
        $\land comm!CSend([c \mapsto c, ack \mapsto crec[c], op \mapsto op])$

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

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

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

Client $c \in Client$ receives a message from the $Server$.
$Rev(c) \triangleq$
        $\land comm!CRev(c)$
        $\land crec' = [crec \text{ EXCEPT } ![c] = @ + 1]$
        $\land \text{LET } m \triangleq Head(cincoming[c])$
                $cBuf \triangleq cbuf[c]$   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. $op$
            $\text{IN}$  $\land cbuf' = [cbuf \text{ EXCEPT } ![c] = xcBuf]$
                 $\land cstate' = [cstate \text{ EXCEPT } ![c] = Apply(xop, @)]$   apply the transformed operation $xop$
        $\land \text{UNCHANGED } \langle sVars, eVars \rangle$
├──────────────────────────────────────────────────────────┤
The $Server$ receives a message.
$SRev \triangleq$
        $\land comm!SRev$
        $\land \text{LET } m \triangleq Head(sincoming)$   the message to handle with
                $c \triangleq m.c$   the client $c \in Client$ that sends this message
                $cBuf \triangleq sbuf[c]$   the buffer at the $Server$ for 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    $\land srec' = [cl \in Client \mapsto$
                            IF $cl = c$
                             THEN $srec[cl] + 1$ receive one more operation from client $c \in Client$
                             ELSE $0]$ reset $srec$ for other clients than $c \in Client$
             $\land sbuf' = [cl \in Client \mapsto$
                            IF $cl = c$
                             THEN $xcBuf$ transformed buffer for client $c \in Client$
                             ELSE $Append(sbuf[cl], xop)]$ store transformed $xop$ into other clients' bufs
             $\land sstate' = Apply(xop, sstate)$ apply the transformed operation
       $\land comm!SSend(c, [cl \in Client \mapsto [ack \mapsto srec[cl], op \mapsto xop]])$
            $\land comm!SSend2(c, srec, xop)$
     $\land$ UNCHANGED $ecVars$
────────────────────────────────────────────────────────────
The next-state relation.
$Next \triangleq$
     $\lor \exists c \in Client : Do(c) \lor Rev(c)$
     $\lor SRev$
The $Spec$. ($TODO$: Check the fairness condition.)
$Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)$
────────────────────────────────────────────────────────────
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 ($QC$)
$QConvergence \triangleq \forall c \in Client : cstate[c] = sstate$
$QC \triangleq comm!EmptyChannel \Rightarrow QConvergence$

THEOREM $Spec \Rightarrow \Box QC$

Strong Eventual Consistency ($SEC$)
════════════════════════════════════════════════════════════
\* Modification History
\* Last modified Sun $Sep$ 02 12:54:18 $CST$ 2018 by $hengxin$
\* Created Sat $Jun$ 23 17:14:18 $CST$ 2018 by $hengxin$
```