―――――――――――――――――――― MODULE *XJupiterExtended* ――――――――――――――――――――

XJupiter extended with *Cop* with the *sctx* field.

EXTENDS *Integers*, *OT*, *TLCUtils*, *AdditionalFunctionOperators*, *AdditionalSequenceOperators*

├―――

CONSTANTS

    $Client,$ the set of client replicas

    $Server,$ the (unique) server replica

    $Char,$ set of characters allowed

    $InitState$ the initial state of each replica

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

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

$ClientNum \triangleq Cardinality(Client)$

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

direction flags

$Local \triangleq 0$

$Remote \triangleq 1$

├―――

ASSUME

    $\wedge Range(InitState) \cap Char = \{\}$ due to the uniqueness requirement

    $\wedge Priority \in [Client \rightarrow 1 .. ClientNum]$

├―――

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

$Op \triangleq Ins \cup Del$

├―――

Cop: operation of type *Op* with context

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

Cop with the *sctx* field (the extended part)

$Cop \triangleq [op : Op \cup \{Nop\}, oid : Oid, ctx :$ SUBSET $Oid, sctx :$ SUBSET $Oid]$

OT of two operations of type *Cop*.

$COT(lcop, rcop) \triangleq [lcop$ EXCEPT $!.op = Xform(lcop.op, rcop.op), !.ctx = @ \cup \{rcop.oid\}]$

├―――

VARIABLES

For the client replicas:

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

1

For the server replica (the extended part):

62     $soids$,   the set of operations the $Server$ has executed

The $2D$ state spaces (ss, for short). Each client maintains one $2D$ state space. The server maintains n $2D$ state spaces, one for each client.

68   css, $css[c]$: the $2D$ state space at client $c \in Client$
69     $ccur$, $cur[c]$: the current node of $css[c]$
70   sss, $sss[c]$: the $2D$ state space maintained by the $Server$ for client $c \in Client$
71     $scur$, $scur[c]$: the current node of $sss[c]$

For all replicas

75     $state$, $state[r]$: state (the list content) of replica $r \in Replica$

For communication between the $Server$ and the Clients:

79     $cincoming$, $cincoming[c]$: incoming channel at the client $c \in Client$
80     $sincoming$,   incoming channel at the $Server$

For model checking:

84     $chins$ a set of chars to insert

85 ⊢───

86 $comm \triangleq$ INSTANCE $CSComm$ WITH $Msg \leftarrow Cop$

87 ⊢───

88 $eVars \triangleq \langle chins \rangle$ variables for the environment
89 $cVars \triangleq \langle cseq \rangle$ variables for the clients
90 $cssVars \triangleq \langle css, ccur \rangle$ variables for $2D$ state spaces at clients
91 $sssVars \triangleq \langle sss, scur \rangle$ variables for $2D$ state spaces at the $Server$
92 $commVars \triangleq \langle cincoming, sincoming \rangle$ variables for communication
93 $vars \triangleq \langle eVars, cVars, commVars, cssVars, sssVars, state \rangle$ all variables

94 ⊢───

A $2D$ state space 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 and a direction flag indicating whether this edge is LOCAL or REMOTE. For clarity, we denote edges by records instead of tuples.

103 $IsSS(G) \triangleq$
104      $\land\ G = [node \mapsto G.node, edge \mapsto G.edge]$
105      $\land\ G.node \subseteq (\text{SUBSET } Oid)$
106      $\land\ G.edge \subseteq [from : G.node, to : G.node, cop : Cop, lr : \{Local, Remote\}]$

108 $TypeOK \triangleq$

For the client replicas:

112      $\land\ cseq \in [Client \rightarrow Nat]$

For the $2D$ state spaces:

116      $\land\ \forall c\ \in Client : IsSS(css[c]) \land IsSS(sss[c])$
117      $\land\ ccur \in [Client \rightarrow \text{SUBSET } Oid]$
118      $\land\ scur \in [Client \rightarrow \text{SUBSET } Oid]$
119      $\land\ state \in [Replica \rightarrow List]$

For communication between the server and the clients:

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

127          $\land\ chins \subseteq Char$

128 $\vdash$ ──

The *Init* predicate.

132   $Init\ \triangleq$

For the client replicas:

136          $\land\ cseq = [c \in Client \mapsto 0]$

For the *Server* replica (the extended part):

140          $\land\ soids = \{\}$

For the $2D$ state spaces:

144          $\land\ css = [c \in Client \mapsto [node \mapsto \{\{\}\},\ edge \mapsto \{\}]]$
145          $\land\ ccur = [c \in Client \mapsto \{\}]$
146          $\land\ sss = [c \in Client \mapsto [node \mapsto \{\{\}\},\ edge \mapsto \{\}]]$
147          $\land\ scur = [c \in Client \mapsto \{\}]$

For all replicas:

151          $\land\ state = [r \in Replica \mapsto InitState]$

For communication between the server and the clients:

155          $\land\ comm\,!\,Init$

159          $\land\ chins = Char$

160 $\vdash$ ──

Locate the node in the $2D$ state space *ss* which matches the context *ctx* of cop.

164   $Locate(cop,\ ss)\ \triangleq\ \text{CHOOSE}\ n \in (ss.node) : n = cop.ctx$

xForm: iteratively transform cop with a path through the $2D$ state space *ss* at some client, following the edges with the direction flag *d*.

171   $xForm(cop,\ ss,\ cur,\ d)\ \triangleq$
172       $\text{LET}\ u\ \triangleq\ Locate(cop,\ ss)$
173            $v\ \triangleq\ u \cup \{cop.oid\}$
174            $\text{RECURSIVE}\ xFormHelper(\_,\ \_,\ \_,\ \_)$
175   'h' stands for "helper"; *xss*: *eXtra ss* created during transformation
176            $xFormHelper(uh,\ vh,\ coph,\ xss)\ \triangleq$
177                $\text{IF}\ uh = cur$
178                 $\text{THEN}\ xss$
179                 $\text{ELSE}\ \ \text{LET}\ e\ \triangleq\ \text{CHOOSE}\ e \in ss.edge : e.from = uh \land e.lr = d$
180                         $uprime\ \triangleq\ e.to$
181                         $copprime\ \triangleq\ e.cop$
182                         $coph2copprime\ \triangleq\ COT(coph,\ copprime)$
183                         $copprime2coph\ \triangleq\ COT(copprime,\ coph)$
184                         $vprime\ \triangleq\ vh \cup \{copprime.oid\}$
185                    $\text{IN}\ \ \ xFormHelper(uprime,\ vprime,\ coph2copprime,$
186                          $[xss\ \text{EXCEPT}\ !.node = @ \circ \langle vprime \rangle,$

3

```
187                                          the order of recording edges here is important
188                                          so that the last one is labeled with the final transformed operation
189                                          !.edge = @ ∘ ⟨[from ↦ vh, to ↦ vprime, cop ↦ copprime2coph, lr ↦
190                                                    [from ↦ uprime, to ↦ vprime, cop ↦ coph2copprime,
191      IN   xFormHelper(u, v, cop, [node ↦ ⟨v⟩,
192                                   edge ↦ ⟨[from ↦ u, to ↦ v, cop ↦ cop, lr ↦ 1 − d]⟩])
193 ├───────────────────────────────────────────────────────────────────────────────────────┤
```

Client $c \in Client$ perform operation cop guided by the direction flag d.

```
197  ClientPerform(cop, c, d)  ≜
198       LET xss  ≜  xForm(cop, css[c], ccur[c], d)
199           xn   ≜  xss.node
200           xe   ≜  xss.edge
201           xcur ≜  Last(xn)
202           xcop ≜  Last(xe).cop
203       IN   ∧ css' = [css EXCEPT ![c].node = @ ∪ Range(xn),
204                                  ![c].edge = @ ∪ Range(xe)]
205            ∧ ccur' = [ccur EXCEPT ![c] = xcur]
206            ∧ state' = [state EXCEPT ![c] = Apply(xcop.op, @)]
```

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

```
210  DoOp(c, op)  ≜     op: the raw operation generated by the client c ∈ Client
211       ∧ cseq' = [cseq EXCEPT ![c] = @ + 1]
212            op with the sctx field (the extended part)
213       ∧ LET  cop  ≜  [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq'[c]], ctx ↦ ccur[c], sctx ↦ {}]
214         IN   ∧ ClientPerform(cop, c, Remote)
215              ∧ comm!CSend(cop)

217  DoIns(c)  ≜
218       ∃ ins ∈ Ins :
219           ∧ ins.pos ∈ 1 .. (Len(state[c]) + 1)
220           ∧ ins.ch ∈ chins
221           ∧ ins.pr = Priority[c]
222           ∧ chins' = chins \ {ins.ch}  We assume that all inserted elements are unique.
223           ∧ DoOp(c, ins)
224           ∧ UNCHANGED ⟨sssVars⟩

226  DoDel(c)  ≜
227       ∃ del ∈ Del :
228           ∧ del.pos ∈ 1 .. Len(state[c])
229           ∧ DoOp(c, del)
230           ∧ UNCHANGED ⟨sssVars, eVars⟩

232  Do(c)  ≜
233           ∨ DoIns(c)
234           ∨ DoDel(c)
```

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

4

$238 \quad Rev(c) \triangleq$

$239 \qquad \wedge comm!CRev(c)$

$240 \qquad \wedge \text{LET } cop \triangleq Head(cincoming[c])$ the received (transformed) operation

$241 \qquad\qquad \text{IN } ClientPerform(cop, c, Local)$

$242 \qquad \wedge \text{UNCHANGED } \langle eVars, cVars, sssVars \rangle$

The *Server* performs operation cop.

$247 \quad ServerPerform(cop) \triangleq$

$248 \qquad \text{LET } c \triangleq cop.oid.c$

$249 \qquad\quad xss \triangleq xForm(cop, sss[c], scur[c], Remote)$

$250 \qquad\quad xn \triangleq xss.node$

$251 \qquad\quad xe \triangleq xss.edge$

$252 \qquad\quad xcur \triangleq Last(xn)$

$253 \qquad\quad xcop \triangleq Last(xe).cop$

$254 \qquad \text{IN} \quad \wedge sss' = [cl \in Client \mapsto$

$255 \qquad\qquad\qquad\qquad \text{IF } cl = c$

$256 \qquad\qquad\qquad\qquad \text{THEN } [sss[cl] \text{ EXCEPT } !.node = @ \cup Range(xn),$

$257 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad !.edge = @ \cup Range(xe)]$

$258 \qquad\qquad\qquad\qquad \text{ELSE } \text{LET } scurcl \triangleq scur[cl]$

$259 \qquad\qquad\qquad\qquad\qquad\qquad\qquad scurclprime \triangleq scurcl \cup \{cop.oid\}$

$260 \qquad\qquad\qquad\qquad\qquad\qquad \text{IN } [sss[cl] \text{ EXCEPT } !.node = @ \cup \{scurclprime\},$

$261 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad !.edge = @ \cup \{[from \mapsto scurcl, to \mapsto scurclprime,$

$262 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad cop \mapsto xcop, lr \mapsto Remote]\}]$

$263 \qquad\qquad\qquad\qquad]$

$264 \qquad\qquad \wedge scur' = [cl \in Client \mapsto$

$265 \qquad\qquad\qquad\qquad \text{IF } cl = c \text{ THEN } xcur \text{ ELSE } scur[cl] \cup \{cop.oid\}]$

$266 \qquad\qquad \wedge state' = [state \text{ EXCEPT } ![Server] = Apply(xcop.op, @)]$

$267 \qquad\qquad \wedge comm!SSendSame(c, xcop)$ broadcast the transformed operation

The *Server* receives a message.

$271 \quad SRev \triangleq$

$272 \qquad \wedge comm!SRev$

$273 \qquad \wedge \text{LET } cop \triangleq [Head(sincoming) \text{ EXCEPT } !.sctx = soids]$

$274 \qquad\qquad cop$ with the *sctx* field (the extended part)

$275 \qquad\qquad \text{IN } ServerPerform(cop)$

$276 \qquad \wedge \text{UNCHANGED } \langle eVars, cVars, cssVars \rangle$

The next-state relation.

$281 \quad Next \triangleq$

$282 \qquad \vee \exists c \in Client : Do(c) \vee Rev(c)$

$283 \qquad \vee SRev$

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

$287 \quad Spec \triangleq Init \wedge \square[Next]_{vars} \wedge \text{WF}_{vars}(Next)$

\ * Modification History

\ * *Last* modified *Tue Oct* 30 20:52:50 *CST* 2018 by *hengxin*
\ * Created *Tue Oct* 30 20:32:27 *CST* 2018 by *hengxin*