```
┌──────────────────────────── MODULE AbsJupiter ────────────────────────────┐
Abstract Jupiter, inspired by the COT algorithm proposed by Sun and Sun; see TPDS′2009.
EXTENDS JupiterSerial
├──────────────────────────────────────────────────────────────────────────┤
VARIABLES
    copss    copss[r]: the state space (i.e., a set) of Cop maintained at replia r ∈ Replica

vars ≜ ⟨intVars, ctxVars, serialVars, copss⟩
├──────────────────────────────────────────────────────────────────────────┤
TypeOK ≜
    ∧  TypeOKInt
    ∧  TypeOKCtx
    ∧  TypeOKSerial
    ∧  copss ∈ [Replica → SUBSET Cop]
├──────────────────────────────────────────────────────────────────────────┤
Init ≜
    ∧ InitInt
    ∧ InitCtx
    ∧ InitSerial
    ∧ copss = [r ∈ Replica ↦ {}]
├──────────────────────────────────────────────────────────────────────────┤
RECURSIVE xForm(_, _)        Transform cop at replica r ∈ Replica.
xForm(r, cop) ≜              Return the transformed cop and the state space copss[r] after transformation.
    LET ctxDiff ≜ ds[r] \ cop.ctx    THEOREM : cop.ctx ⊆ ds[r]
        RECURSIVE xFormHelper(_, _, _)
         xFormHelper(coph, ctxDiffh, copssr) ≜
            IF ctxDiffh = {} THEN [xcop ↦ coph, xcopss ↦ copssr]
             ELSE LET foph ≜ CHOOSE op ∈ ctxDiffh :   the first op in serial
                                   ∀ opprime ∈ ctxDiffh \ {op} : tb(op, opprime, serial[r])
                          fcophDict ≜ {op ∈ copssr : op.oid = foph ∧ op.ctx = coph.ctx}
                          fcoph ≜ CHOOSE op ∈ fcophDict : TRUE   THEOREM : Cardinality(fophDict) = 1
                          xcoph ≜ COT(coph, fcoph)
                         xfcoph ≜ COT(fcoph, coph)
                      IN   xFormHelper(xcoph, ctxDiffh \ {foph}, copssr ∪ {xcoph, xfcoph})
    IN   xFormHelper(cop, ctxDiff, copss[r] ∪ {cop})

Perform(r, cop) ≜
    LET xform ≜ xForm(r, cop)    [xcop, xcopss]
    IN   ∧ copss′ = [copss EXCEPT ![r] = xform.xcopss]
         ∧ SetNewAop(r, xform.xcop.op)

ClientPerform(c, cop) ≜ Perform(c, cop)

ServerPerform(cop) ≜
    ∧ Perform(Server, cop)
    ∧ Comm!SSendSame(ClientOf(cop), cop)
```

```
--------------------------------------------------------------------------------
DoOp(c, op) ≜
      LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq[c]], ctx ↦ ds[c]]
      IN  ∧ ClientPerform(c, cop)
          ∧ Comm!CSend(cop)

Do(c) ≜
      ∧ DoInt(DoOp, c)
      ∧ DoCtx(c)
      ∧ DoSerial(c)

Rev(c) ≜
      ∧ RevInt(ClientPerform, c)
      ∧ RevCtx(c)
      ∧ RevSerial(c)

SRev ≜
     ∧ SRevInt(ServerPerform)
     ∧ SRevCtx
     ∧ SRevSerial
--------------------------------------------------------------------------------
Next ≜
     ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
     ∨ SRev

Fairness ≜
     WF_vars(SRev ∨ ∃ c ∈ Client : Rev(c))

Spec ≜ Init ∧ □[Next]_vars  ∧ Fairness
--------------------------------------------------------------------------------
Compactness ≜
     Comm!EmptyChannel ⇒ Cardinality(Range(copss)) = 1

THEOREM Spec ⇒ Compactness
================================================================================
```

* Modification History

* Last modified Sat *Jan* 05 17:28:25 *CST* 2019 by *hengxin*

* Created *Wed Dec* 05 19:55:52 *CST* 2018 by *hengxin*