```
1  ---------- MODULE StateSpace ----------
   The graph representation of n-ary ordered state space and 2D state space used in CJupiter and
   XJupiter, respectively.
6  EXTENDS JupiterCtx, GraphsUtil
7  ----------
8  IsSS(G) ≜  A state space is a digraph with labeled edges.
9        ∧ IsGraph(G)  It is a digraph (represented by a record).
10       ∧ G.node ⊆ (SUBSET Oid)  Each node represents a document state, i.e., a set of Oid.
11       ∧ G.edge ⊆ [from : G.node, to : G.node, cop : Cop]  Each edge is labeled with a Cop operation.

13 EmptySS ≜ EmptyGraph
14 ----------
15 Locate(cop, ss) ≜  Locate the (unique) node in state space ss that matches the context of cop.
16    CHOOSE n ∈ ss.node : n = cop.ctx

18  RECURSIVE ExtractCopSeq(_, _, _) \* Extract a Cop sequence starting with u in ss at replica r.
19  ExtractCopSeq(NextEdge(_, _, _), r, u, ss) ≜
20    IF u = ds[r] THEN ⟨⟩
21    ELSE LET e ≜ NextEdge(r, u, ss)
22       IN ⟨e.cop⟩ ∘ ExtractCopSeq(NextEdge, r, e.to, ss)

24 xForm(NextEdge(_, _, _), r, cop, ss) ≜  Transform cop with an operation sequence
25    LET u ≜ Locate(cop, ss)              in state space ss at replica r.
26        v ≜ u ∪ {cop.oid}
27        RECURSIVE xFormHelper(_, _, _, _)
28        xFormHelper(uh, vh, coph, xss) ≜  xss: eXtra ss created during transformation
29           IF uh = ds[r] THEN [xcop ↦ coph,
30                               xss ↦ xss,
31                               lss ↦ [node ↦ {vh},
32                                      edge ↦ {[from ↦ uh, to ↦ vh, cop ↦ coph]}]]
33           ELSE LET e ≜ NextEdge(r, uh, ss)
34                    copprime ≜ e.cop
35                    uprime ≜ e.to
36                    vprime ≜ vh ∪ {copprime.oid}
37                    coph2copprime ≜ COT(coph, copprime)
38                    copprime2coph ≜ COT(copprime, coph)
39                IN  xFormHelper(uprime, vprime, coph2copprime,
40                      xss ⊕ [node ↦ {vprime},
41                             edge ↦ {[from ↦ vh, to ↦ vprime, cop ↦ copprime2coph],
42                                     [from ↦ uprime, to ↦ vprime, cop ↦ coph2copprime]}])
43    IN  xFormHelper(u, v, cop, [node ↦ {v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop]}])

45 xFormCopCopsSS(cop, cops) ≜ \* Transform cop against cops (a sequence of Cop) on state space.
46   LET RECURSIVE xFormCopCopsSSHelper(_, _, _)
47     xFormCopCopsSSHelper(coph, copsh, xss) ≜
48       LET u ≜ coph.ctx
```

```
49              v ≜ u ∪ {coph.oid}
50          uvSS ≜ [node ↦ {u, v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ coph]}]
51          IN  IF copsh = ⟨⟩ THEN [xcop ↦ coph, xss ↦ xss ⊕ uvSS, lss ↦ uvSS]
52            ELSE LET copprimeh ≜ Head(copsh)
53                     uprime ≜ u ∪ {copprimeh.oid}
54                     vprime ≜ u ∪ {coph.oid, copprimeh.oid}
55                 coph2copprimeh ≜ COT(coph, copprimeh)
56                 copprimeh2coph ≜ COT(copprimeh, coph)
57               IN  xFormCopCopsSSHelper(coph2copprimeh, Tail(copsh),
58                     xss ⊕ [node ↦ {u, v},
59                            edge ↦ {[from ↦ u, to ↦ v, cop ↦ coph],
60                                    [from ↦ u, to ↦ uprime, cop ↦ copprimeh],
61                                    [from ↦ v, to ↦ vprime, cop ↦ copprimeh2coph]}])
62    IN  xFormCopCopsSSHelper(cop, cops, EmptySS)
63
64  xFormSS(cop, copprime) ≜ \* Transform cop against copprime on state space.
65    LET u ≜ cop.ctx      \* Return the extra state space.
66      v ≜ u ∪ {cop.oid}
67      uprime ≜ u ∪ {copprime.oid}
68      vprime ≜ u ∪ {cop.oid, copprime.oid}
69      cop2copprime ≜ COT(cop, copprime)
70      copprime2cop ≜ COT(copprime, cop)
71    IN  [node ↦ {u, v, uprime, vprime},
72         edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop],
73                 [from ↦ u, to ↦ uprime, cop ↦ copprime],
74                 [from ↦ v, to ↦ vprime, cop ↦ copprime2cop],
75                 [from ↦ uprime, to ↦ vprime, cop ↦ cop2copprime]}]
76
```

* Modification History
* Last modified *Wed Jan* 09 15:31:43 *CST* 2019 by *hengxin*
* Created *Wed Dec* 19 18:15:25 *CST* 2018 by *hengxin*