Draw loops from enumerated reaction networks
December 2014

Initializations

```
> restart :
> interface(rtablesize = 400) :
> with(ListTools) :
> with(LinearAlgebra) :
> with(VectorCalculus) :
> with(GraphTheory) :
> with(combinat) :
> with(ArrayTools) :
> with(FileTools) :
> _Envsignum0 := 0 :
>
```

Functions for constructing graph from stoichiometric matrix

```
> findZ := proc(A)
    local Z, n, m, i, j :
     n := Dimension(A)[1] :
     m := Dimension(A)[2] :
     Z := Matrix(n, m) :
  for i from 1 to n by 1 do
    for j from 1 to m by 1 do
       if A[i, j] < 0 then Z[i, j] := z[i, j]; end if;      ### what is the z?
    end do;
  end do:
   return(Z) :
   end proc:
```

Find the DSR graph from labels, A and Z

```
> ##Create signed DSR graph: entries are two matrices and the labels of the nodes
   createDSRgraphsigned := proc(mynodes, A, Z)
    local G, n, m, Adj, varsZ, Zsign, varsA, Asign, X :
    n := Dimension(A)[1] : m := Dimension(A)[2] :
    X := Transpose(Z) :
```

```
    varsZ := indets(X) :
    Zsign := subs(seq(varsZ[i] = 1, i = 1 ..numelems(varsZ)), X) :

   Adj := Matrix(n + m, n + m) :
   Adj[[n + 1 ..n + m], [1 ..n]] := Transpose(map(signum, A)) :
   Adj[[1 ..n], [n + 1 ..n + m]] := Transpose(Zsign) :

    G := GraphTheory[Graph](mynodes, Adj, weighted = true) :
    return(G) :
  end proc:
```

Find the DSR graph from labels and A and return the list of edges:

```
> findedgesDSR := proc(mynodes, A)
    local G, Z :
    Z := findZ(A) :
   G := createDSRgraphsigned(mylabels, A, Z) :
   return(Edges(G, weights)) :
  end proc:

>
```

Functions for draw loops from graph

Auxiliary procedures

This procedure **c**omputes the polynomial $p_{A,\,Z}$ in the main text. The input are the matrices A and Z (in the function denoted N and X).

```
> ## compute Mtilde determinant
  computdetS := proc(N, X)
    local M, Mt, F, i, bigdet, n1, s1 :
    n1 := Dimension(N)[1] : s1 := Rank(N) :
    M := N.Transpose(X) :
    Mt := M :
   if s1 < n1 then
     F
       := ReducedRowEchelonForm(Transpose(Matrix([op(NullSpace(Transpose(
      N)))]))) :
    for i from 1 by 1 to Dimension(F)[1] do
       Mt[ArrayTools[SearchArray](F[i])[1]] := F[i] :
    end do:
```

```
    end if:
     bigdet := expand(Determinant(Mt)) :
     return(bigdet) :
    end proc:
```

This function returns the list of monomials that have the wrong sign. The input are the determinant and the wrong sign.

```
> ##Given a determinant and a wrong sign, return the list of wrong monomials
  badterms := proc(deter, mysign)
      local vars, coeflist, monomlist, coeflistsign, wterms, i :
       vars := indets(deter) :
      coeflist := [coeffs(deter, vars, 't')] :
       monomlist := [t] :
     coeflistsign := map(sign, coeflist) :
      wterms := [ ] :
      for i from 1 by 1 to numelems(coeflistsign) do
          if mysign = coeflistsign[i] then wterms := [op(wterms), monomlist[i]] : end if:
      end do:
     return(wterms) :

    end proc:
```

Given a monomial on the entries of a matrix Amatrix, this function finds a matrix from Amatrix such that the variables in the monomial become 1 and the rest are zero.

```
> ##Find submatrix of a matrix corresponding to a monomial
  findmatrix := proc(wmonom, Amatrix)
      local vars, wvarscomp, Anew, i, Anew1 :
      vars := indets(wmonom) :
      wvarscomp := indets(Amatrix) minus vars :
      Anew := subs(seq(wvarscomp[i] = 0, i = 1 ..numelems(wvarscomp)), Amatrix) :
      Anew1 := subs(seq(vars[i] = 1, i = 1 ..numelems(vars)), Anew) :
     return(Anew1) :
    end proc:
```

Find the two submatrices of A and Z corresponding to the monomial, and make A symbolic by introducing a new variable x.

```
> ## Find the two matrices A,Z corresponding to a monomial
  twomatrices := proc(wmonom, A, Z)
      local vars, wvarscomp, Zembed, i, row, col, Aembed, Aembedx, nZ, X :
     X := Transpose(Z) :
      vars := indets(wmonom) :
      wvarscomp := indets(X) minus vars :
      Zembed := subs(seq(wvarscomp[i] = 0, i = 1 ..numelems(wvarscomp)), X) :
      row, col := ArrayTools[SearchArray](Zembed) :
```

```
    Zembed := Zembed[convert(row, list), convert(col, list)] :
    nZ := numelems(vars) :
    Aembed := A[convert(col, list), convert(row, list)] :
    Aembed := map(signum, Aembed) :
    row, col := ArrayTools[SearchArray](Aembed) :
    Aembedx := Matrix(nZ, nZ) :
    for i from 1 by 1 to numelems(col) do
    Aembedx[row[i], col[i]] := Aembed[row[i], col[i]]·x_i :
    end do:
    return(Aembedx, Zembed) :
end proc:
```

Extract the indices of the species and the reaction in the given monomial in the variables of Z

```
> ## Extract indices of species and reaction in the monomial from Z
extractsr := proc(wmonom, Z)
    local vars, wvarscomp, Zembed, i, row, col, X :
    X := Transpose(Z) :
    vars := indets(wmonom) :
    wvarscomp := indets(X) minus vars :          ## indeterminates not in the monomial
    Zembed := subs(seq(wvarscomp[i] = 0, i = 1 ..numelems(wvarscomp)), X) :
    ##set the entries of the indeterminates not in the monomial to zero
    row, col := ArrayTools[SearchArray](Zembed) :
    ##find the nonzero entries of the resulting matrix.
    return(row, col) :  ##return the species and reaction indices
end proc:
```

```
> ##Create DSR graph: entries are two matrices and the labels of the nodes
createDSRgraph := proc(mynodes, A, Z)
    local G, n, m, Adj, varsZ, Zsign, varsA, Asign, X:
    n := Dimension(A)[1] : m := Dimension(A)[2] :
    X := Transpose(Z) :
    varsZ := indets(X) :
    Zsign := subs(seq(varsZ[i] = 1, i = 1 ..numelems(varsZ)), X) :
    varsA := indets(A) :
    Asign := subs(seq(varsA[i] = 1, i = 1 ..numelems(varsA)), A) :

    Adj := Matrix(n + m, n + m) :
    Adj[[n + 1 ..n + m], [1 ..n]] := Transpose(map(signum, Asign)) :
    Adj[[1 ..n], [n + 1 ..n + m]] := Transpose(Zsign) :

    G := GraphTheory[Graph](mynodes, Adj, weighted = true) :
    return(G) :
end proc:
>
```

This function selects the subgraphs that give rise to the monomials with the wrong sign.

```
> ## Select the subgraphs that correspond to the wrong terms of A and Z
  graphlist := proc(mydet, A, Z)
   local srlist, row, col, Gsub, s, wsign, wrongterms, k, wcurrent, Aembedx, Zembed, detZ,
       detAx, wsignA, wrongtermsA, wcurrentA, j, Aembedx1, mynodes :
  Gsub := [ ]:
   srlist := [ ]:
   s := Rank(A) :
  wsign := (-1)^(s+1) :                          ## find wrong sign
  wrongterms := badterms(mydet, wsign) :   ## select the monomials with the wrong sign
   for k from 1 by 1 to numelems(wrongterms) do
       ## for each such monomial, find the associated subgraph
      wcurrent := wrongterms[k] :
      row, col := extractsr(wcurrent, Z) :
       ## find the indices of the species and the reactions in the monomial
     mynodes := [seq(S_col[i], i = 1 ..numelems(col)), seq(R_row[i], i = 1
       ..numelems(row))] :
      Aembedx, Zembed := twomatrices(wcurrent, A, Z) :
       ## the returned Zembed is giving half of the edges of the subgraphs
      detZ := subs(seq(indets(Zembed)[i] = 1, i = 1 ..numelems(indets(Zembed))),
       Determinant(Zembed)) :
     detAx := expand(Determinant(Aembedx)) :
      wsignA := wsign·detZ :
     wrongtermsA := badterms(detAx, wsignA) :
       ## select the monomials with the wrong sign of the subsystem
      for j from 1 by 1 to numelems(wrongtermsA) do
          wcurrentA := wrongtermsA[j] :
          Aembedx1 := findmatrix(wcurrentA, Aembedx) :
       ## find the other half of the edges of the subgraphs
          Gsub := [op(Gsub), createDSRgraph(mynodes, Aembedx1, Zembed)] :
     end do:
   end do:
   return(Gsub) :  ##return the list of graphs
  end proc:
```

Given a list of edges that form a loop, the function returns the edges ordered such that connected they form the loop.

```
> ## Order the edges to have a loop
  orderedge := proc(myedges)
     local ordereedges, endpoint, total, control, k :
    ordereedges := [myedges[1]] :
    endpoint := myedges[1][1][2] :
    total := numelems(myedges) :
   while numelems(ordereedges) < total do
     control := 0 : k := 2 :
     while control = 0 do
       if endpoint = myedges[k][1][1] then
```

```
            orderededges := [op(orderededges), myedges[k]]:
            control := 1:
            endpoint := myedges[k][1][2]:

        end if:
        k := k + 1:
    end do:
end do:
 return(orderededges):
end proc:
```

Find the sequence of signs of the loop

```
> ##Extract the sequence of signs of a loop
extractsign := proc(orderededges)
    local graphsign, i:
    graphsign := []:
    for i from 1 by 1 to numelems(orderededges) do
     graphsign := [op(graphsign), orderededges[i][2]]:
    end do:
 return(graphsign):
end proc:
```

Given a list of graphs, we find the positive feedback loops that they contain and return the sign pattern of each positive feedback loop as well (as those given in Table 1 in the main text).

```
> ##Find the positive feedback loops in the list of graphs
positivefeed :=proc(Gsub)
    local selected, j, mygraph, Gsubcomp, k, mycomp, newgraph, wedges, myprod, i,
     signcycle:
    selected := []:
   signcycle := []:
   Gsubcomp := []:
   for j from 1 by 1 to numelems(Gsub) do
     mygraph := Gsub[j]:
    Gsubcomp := ConnectedComponents(mygraph):
    for k from 1 by 1 to numelems(Gsubcomp) do
      mycomp := Gsubcomp[k]:
     newgraph := InducedSubgraph(mygraph, mycomp):
     wedges := Edges(newgraph, weights):
     myprod := mul(wedges[i][2], i = 1..numelems(wedges)):
     if myprod = 1 then                ##if the loop is positive, select it
        selected := [op(selected), [op(wedges)]]:
    end if:
   end do:
  end do:
```

```
    selected := ListTools[MakeUnique](selected) :
    for k from 1 by 1 to numelems(selected) do
      selected[k] := orderedge(selected[k]) :
      signcycle := [op(signcycle), extractsign(selected[k])] :
    end do:
    return(selected, signcycle) :
  end proc:
```

The main procedure to find the positive loop is the following:

```
> ##main program: find the positive loops
  findloops := proc(A, Z)
    local Gsub, selected, signcycle, mydet :
    mydet := computdetS(A, Z) :  ##find the polynomial p_{A, Z}
    Gsub := graphlist(mydet, A, Z) :
      ##find the list of subgraphs corresponding to the wrong signs
    selected, signcycle := positivefeed(Gsub) :
      ##find the positive feedback loops and their sign pattern
    return(selected) :
  end proc:
```

The second main procedure of the method is the function that draws the selected positive feedback loops. It requires a list with the names of the nodes (see the examples below)

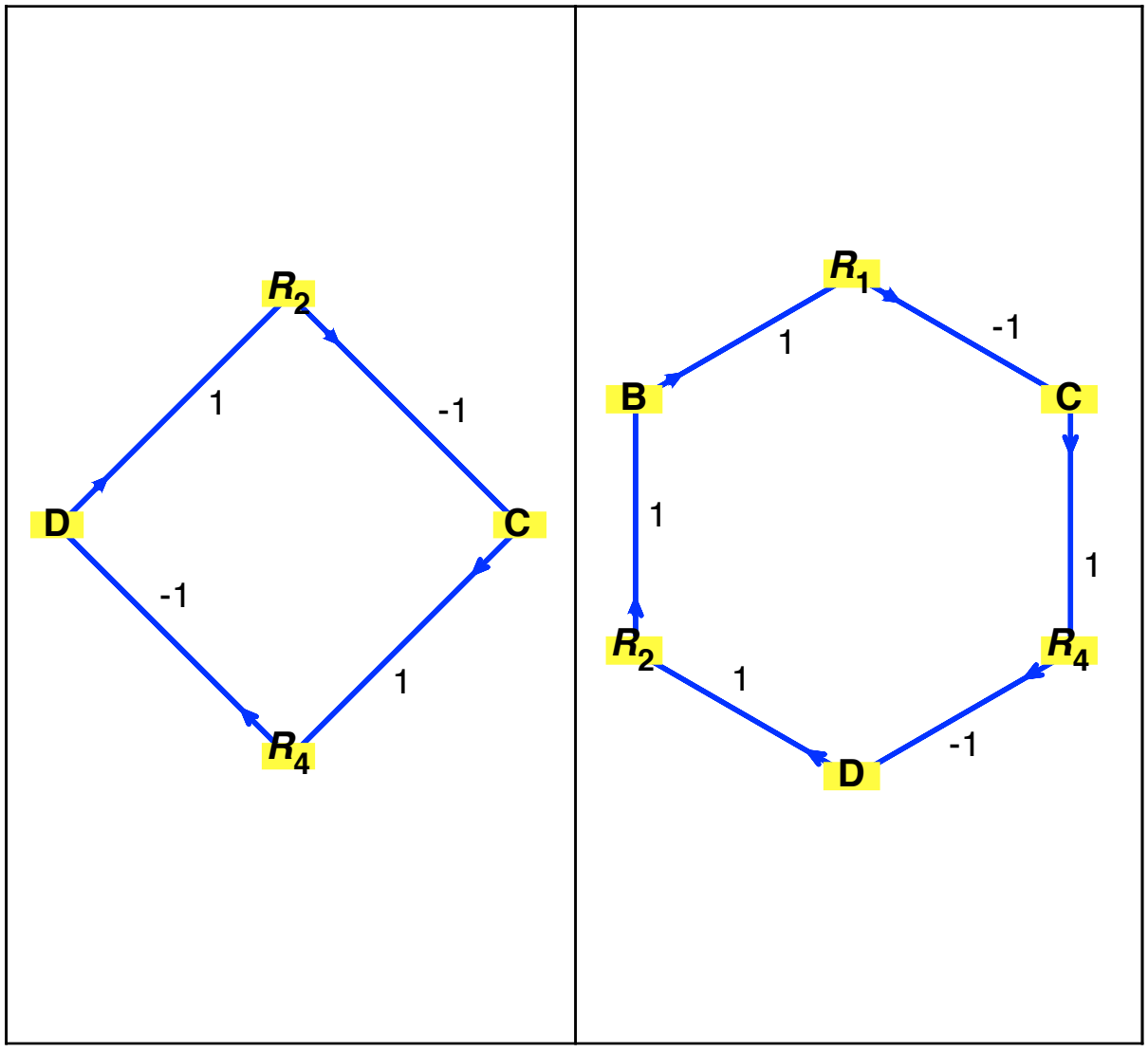
```
> ## draw the positive feedback loops
  drawloops := proc(selected, speciesord)
    local loops, i, vertices, speciesdic, selected2 :
    loops := [ ]:
    speciesdic := {seq(S_i = speciesord[i], i = 1 ..numelems(speciesord))} :
    selected2 := subs(speciesdic, selected) :
    for i from 1 by 1 to numelems(selected2) do
      vertices := ListTools[MakeUnique]([seq(op(selected2[i][j][1]), j = 1
      ..numelems(selected2[i]))]) :
      loops := [op(loops), Digraph(vertices, {op(selected2[i])})] :
    end do:
    DrawGraph(loops, style = circle);

  end proc:

>
```

```
> speciesord := ["A", "B", "C", "D", "E"]:
```

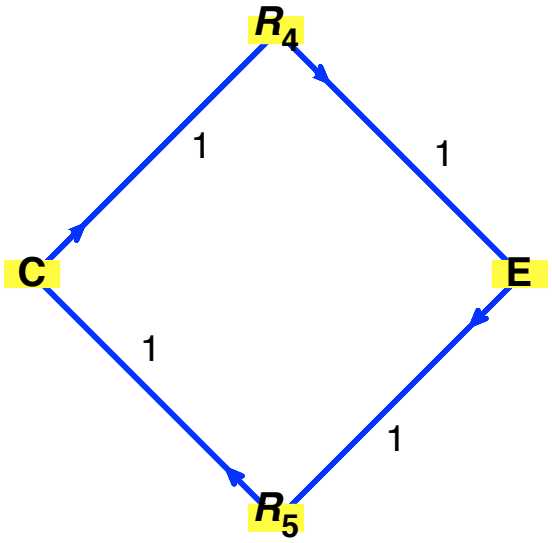
Non-multistationary: with competition loop, with intersecting loops

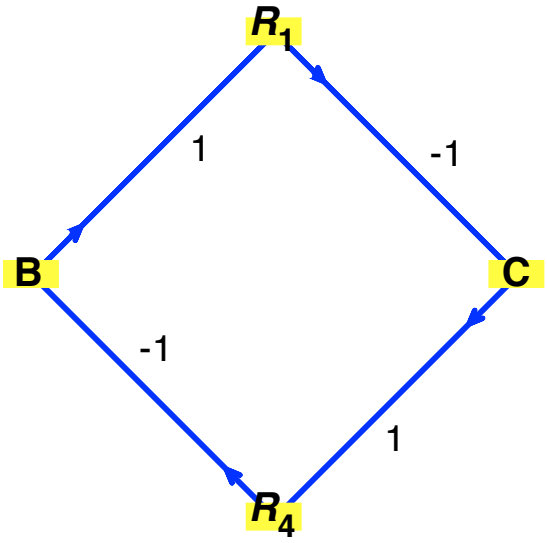
```
> uniquefolder := "5species/nonmultistationary/unique_competitionloop_intersectingloops":
> uniquefiles := ListDirectory(uniquefolder):
> m := nops(uniquefiles):
> for i from 1 to m by 1 do uni := op(i, uniquefiles): A := ImportMatrix(cat(uniquefolder,
      "/", uni)): Z := findZ(A): selected := findloops(A, Z): printf("No. %d: %s \n", i,
      uni); print(drawloops(selected, speciesord)); end do:
```

No. 1: injective1_2962.csv

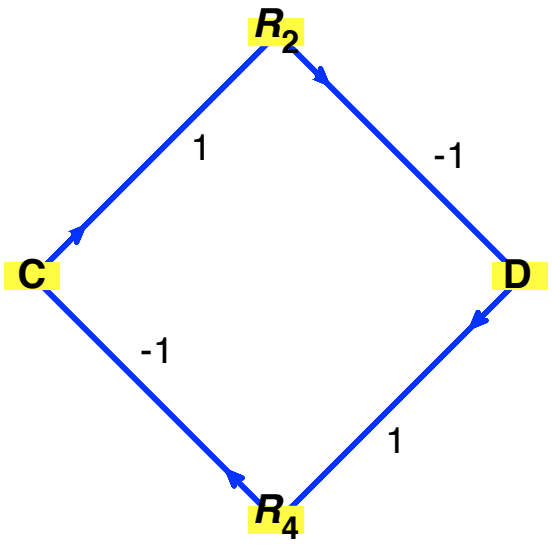


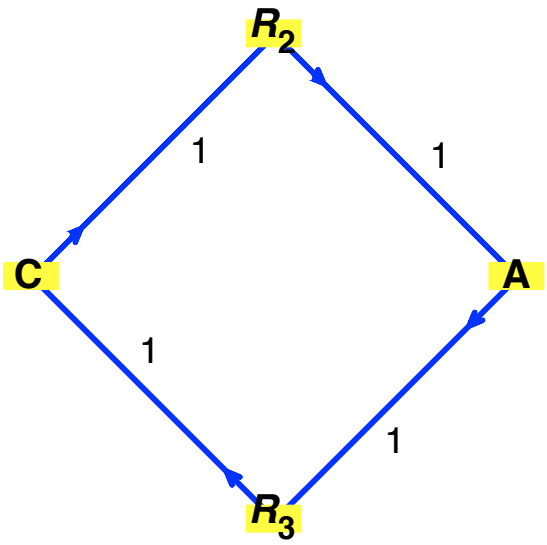
No. 2: injective1_3111.csv



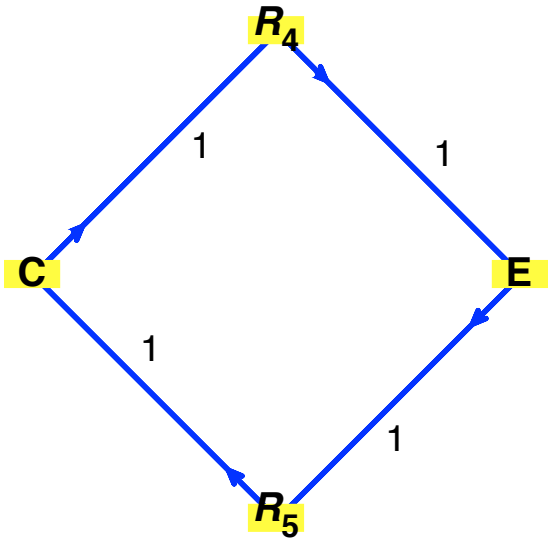


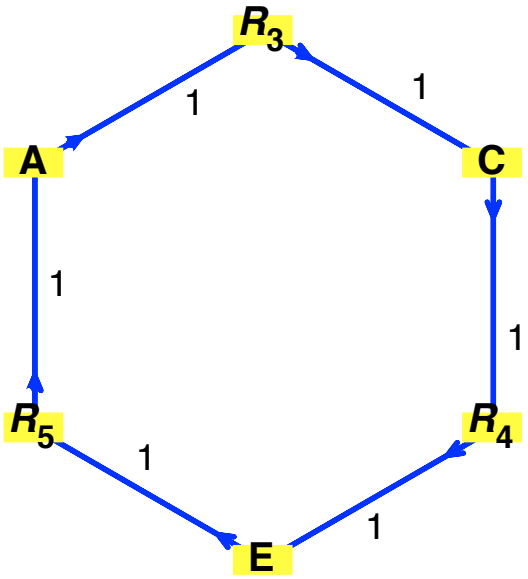
No. 3: injective1_3119.csv



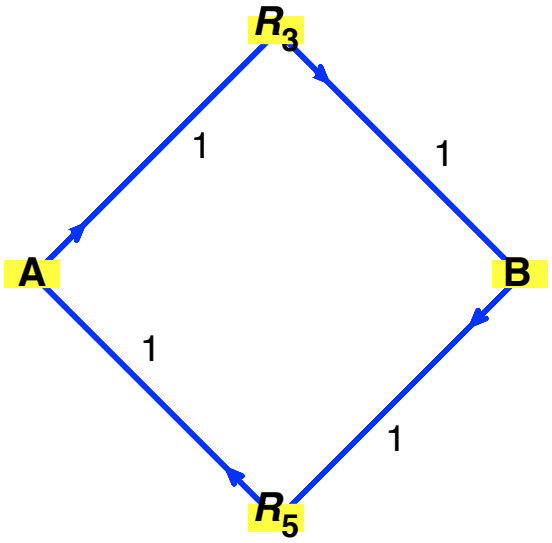


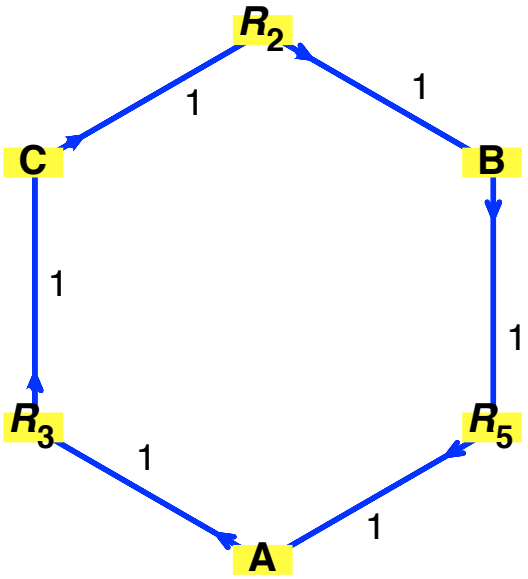
No. 4: injectiveEx1_1267.csv

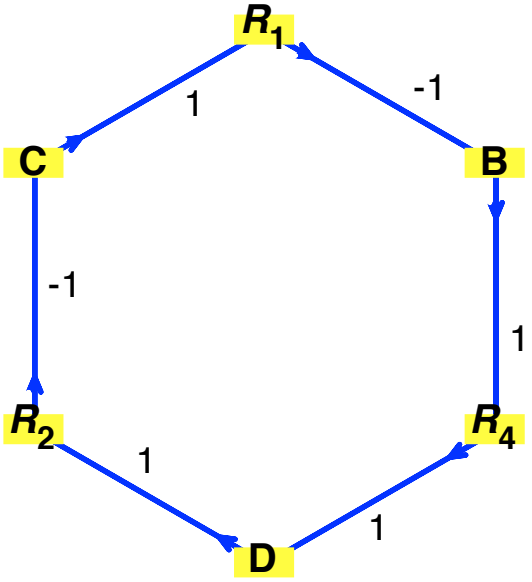


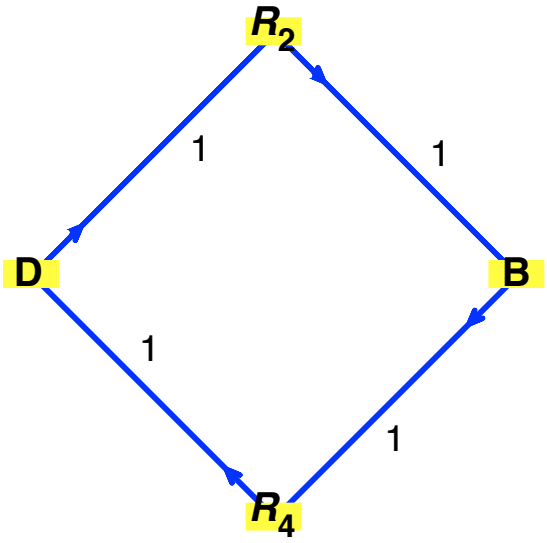


No. 5: injectiveEx1_2956.csv

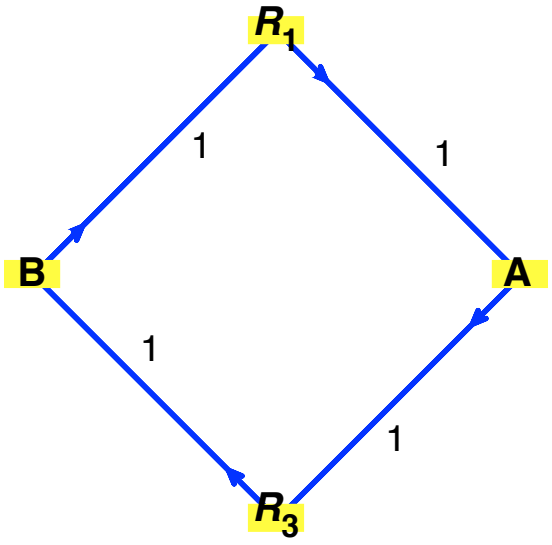

R_3
1
1
A
B
1
1
R_5

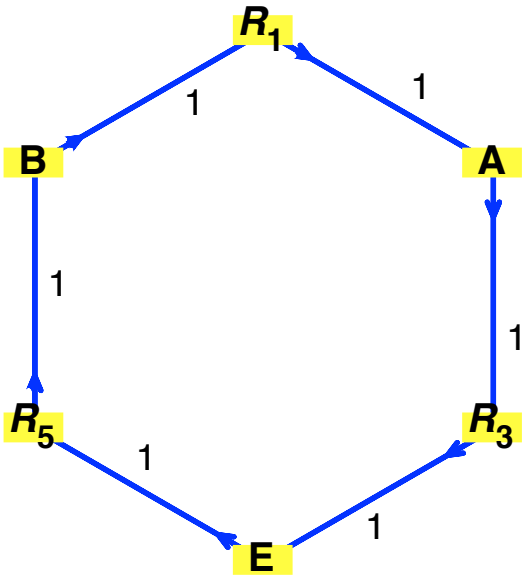

R_2
1
1
C
B
1
1
R_3
R_5
1
1
A



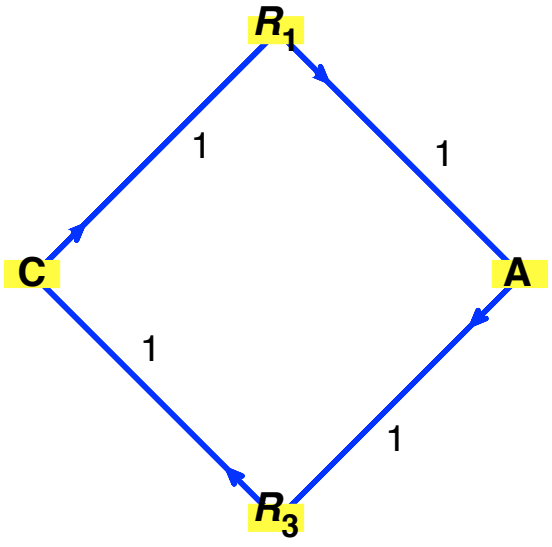


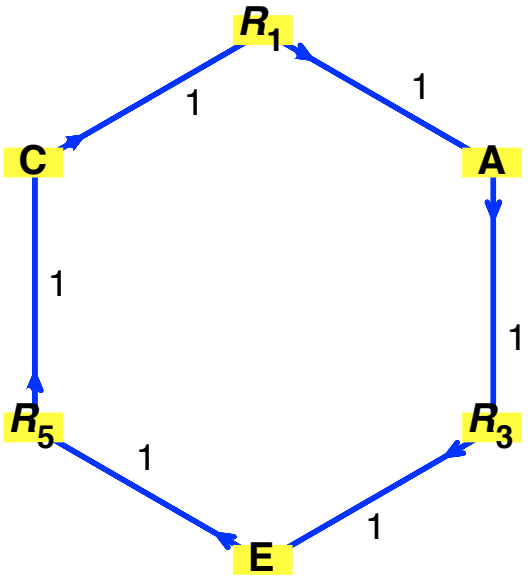
No. 6: injectiveEx1_2979.csv



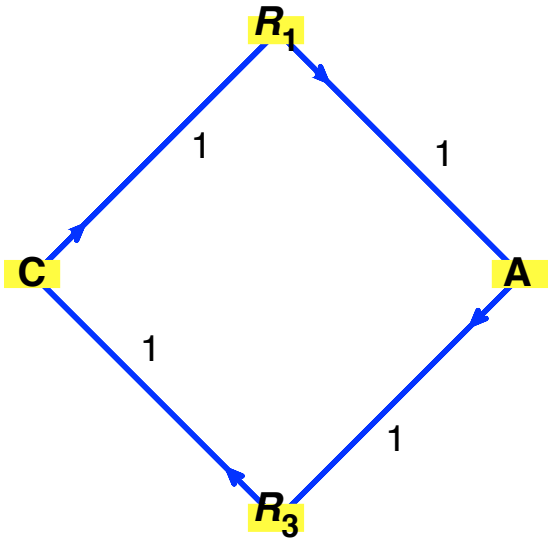


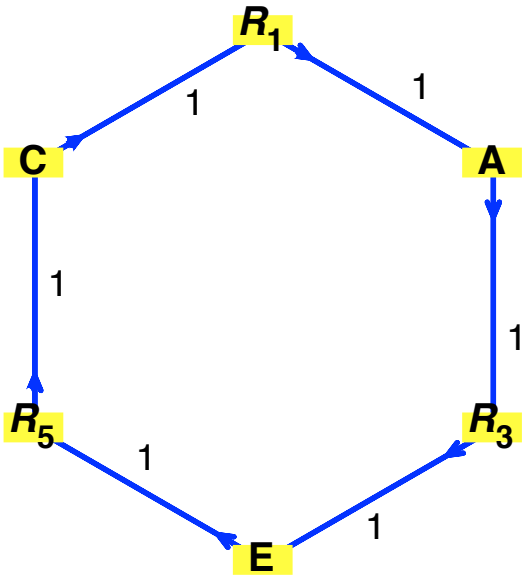
No. 7: injectiveEx1_2987.csv

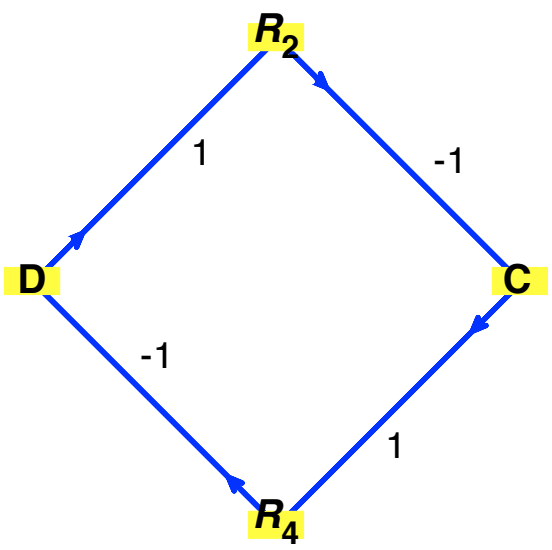


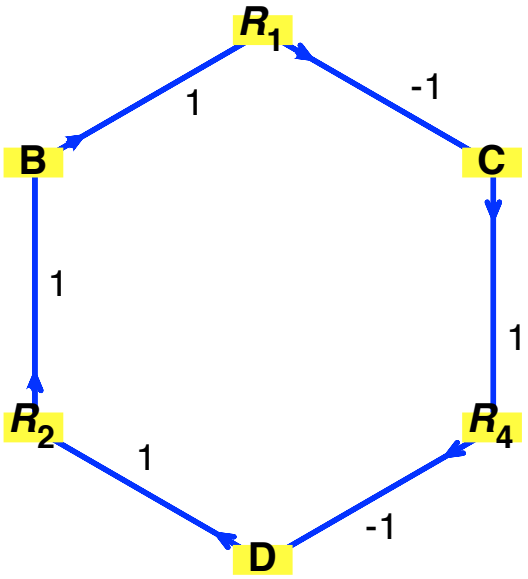


No. 8: injectiveEx1_2988.csv

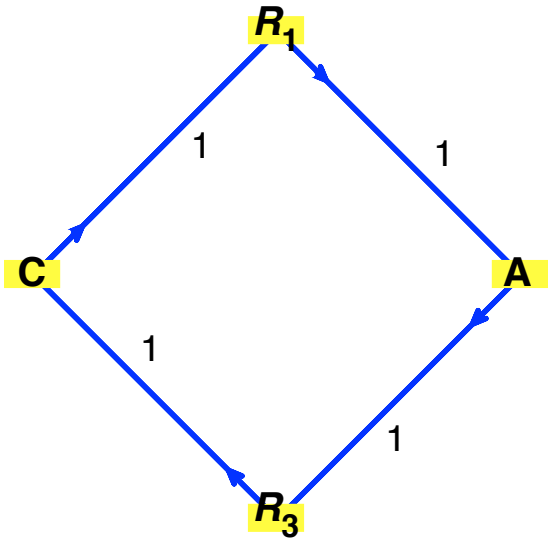

R_1
1
1
C
A
1
1
R_3

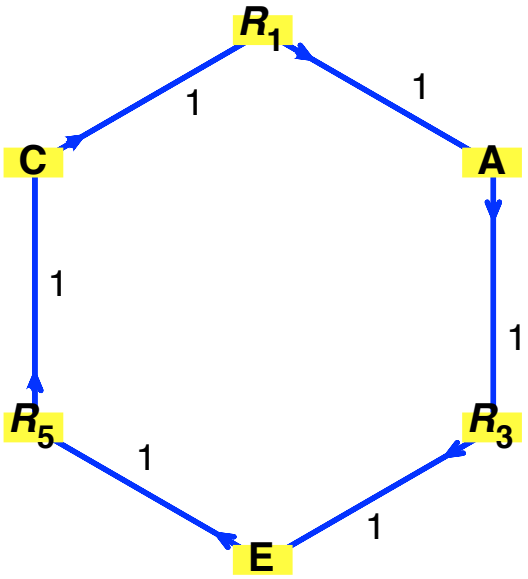

R_1
1
1
C
A
1
1
R_5
R_3
1
1
E

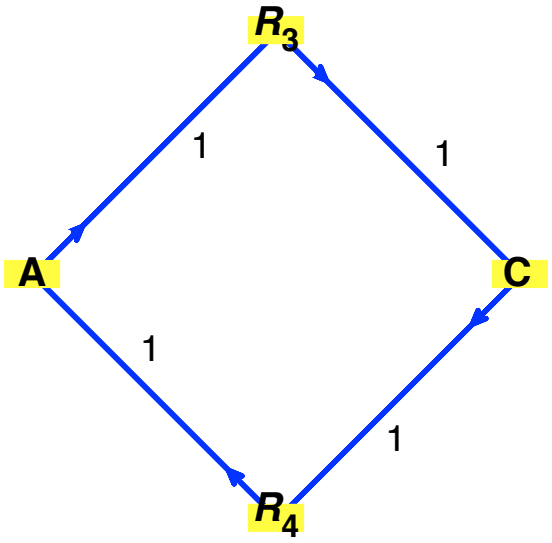


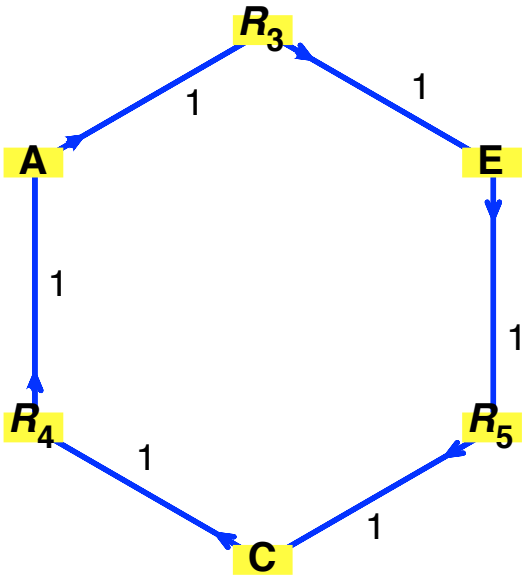


No. 9: injectiveEx1_2989.csv

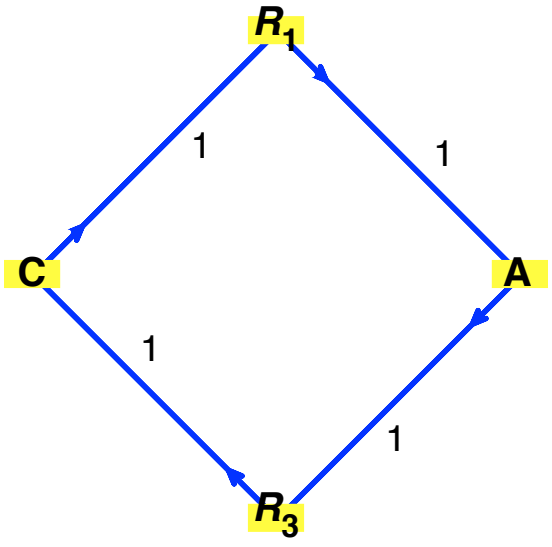

R_1
1
1
C
A
1
1
R_3

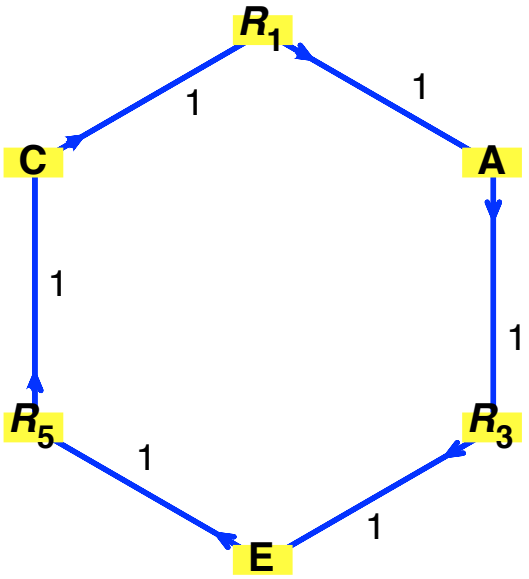

R_1
1
1
C
A
1
1
R_5
R_3
1
1
E



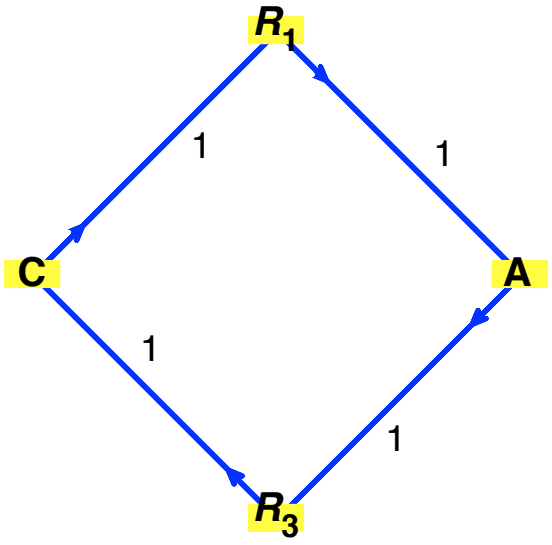


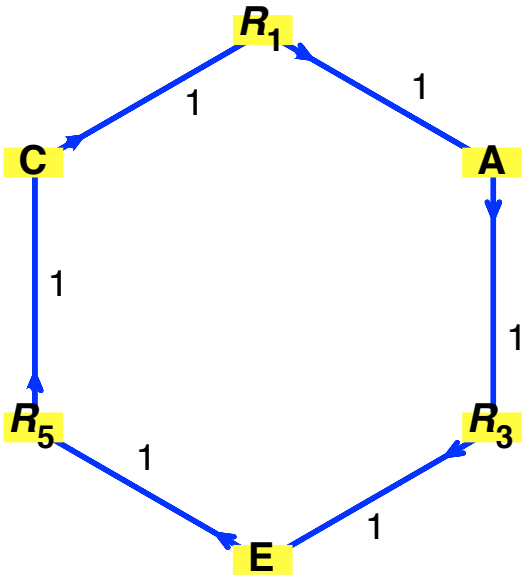
No. 10: injectiveEx1_2991.csv



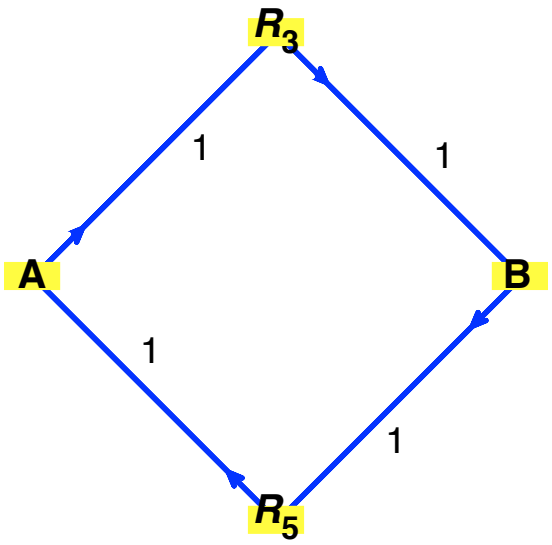


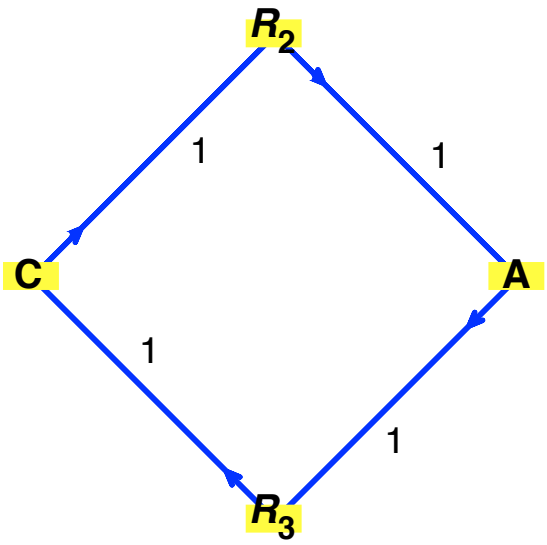
No. 11: injectiveEx1_2993.csv

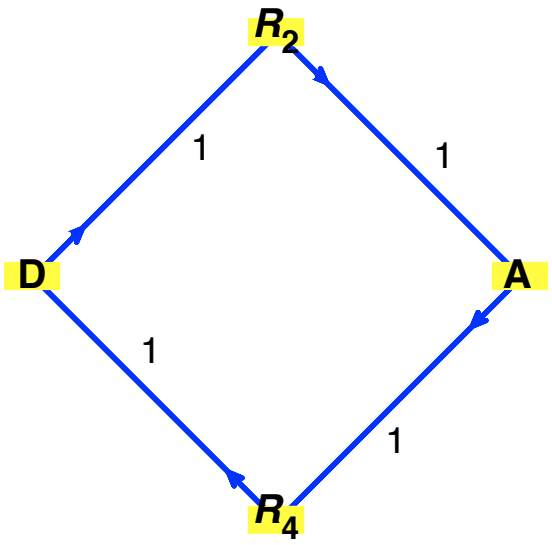


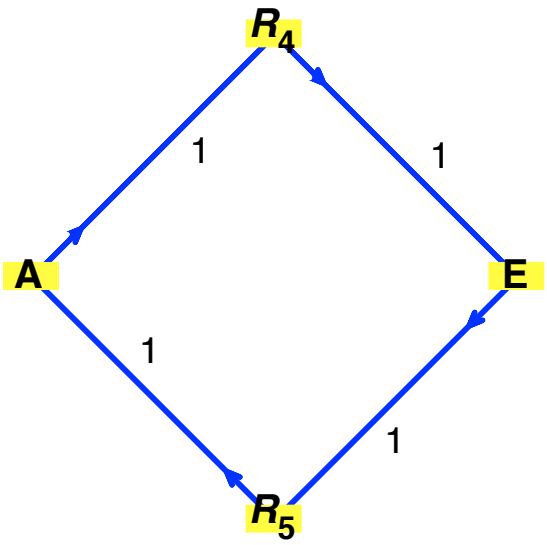


No. 12: injectiveEx1_3103.csv

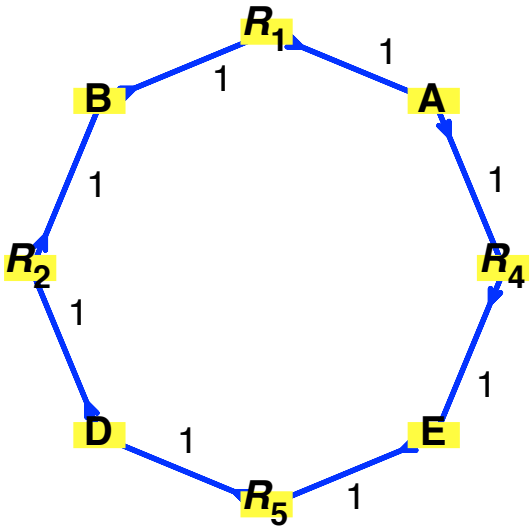

R_3
1
1
A
B
1
1
R_5

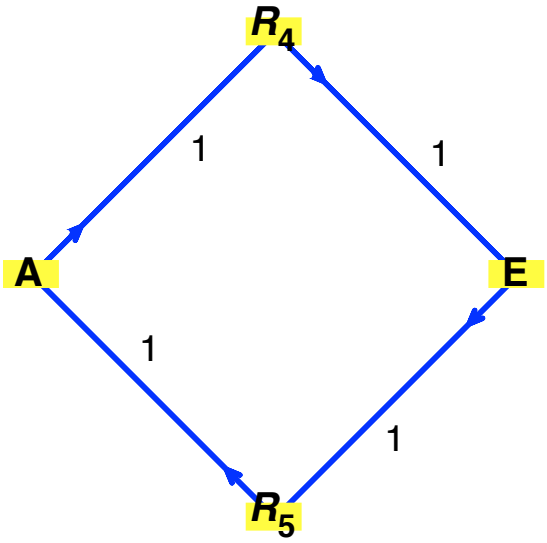

R_2
1
1
C
A
1
1
R_3



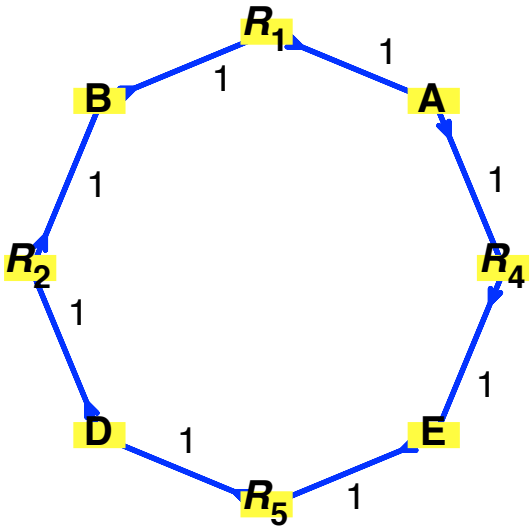


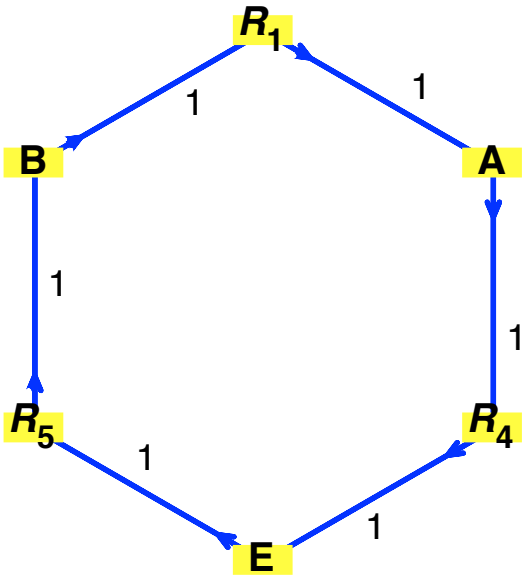
No. 13: injectiveEx3_1250.csv



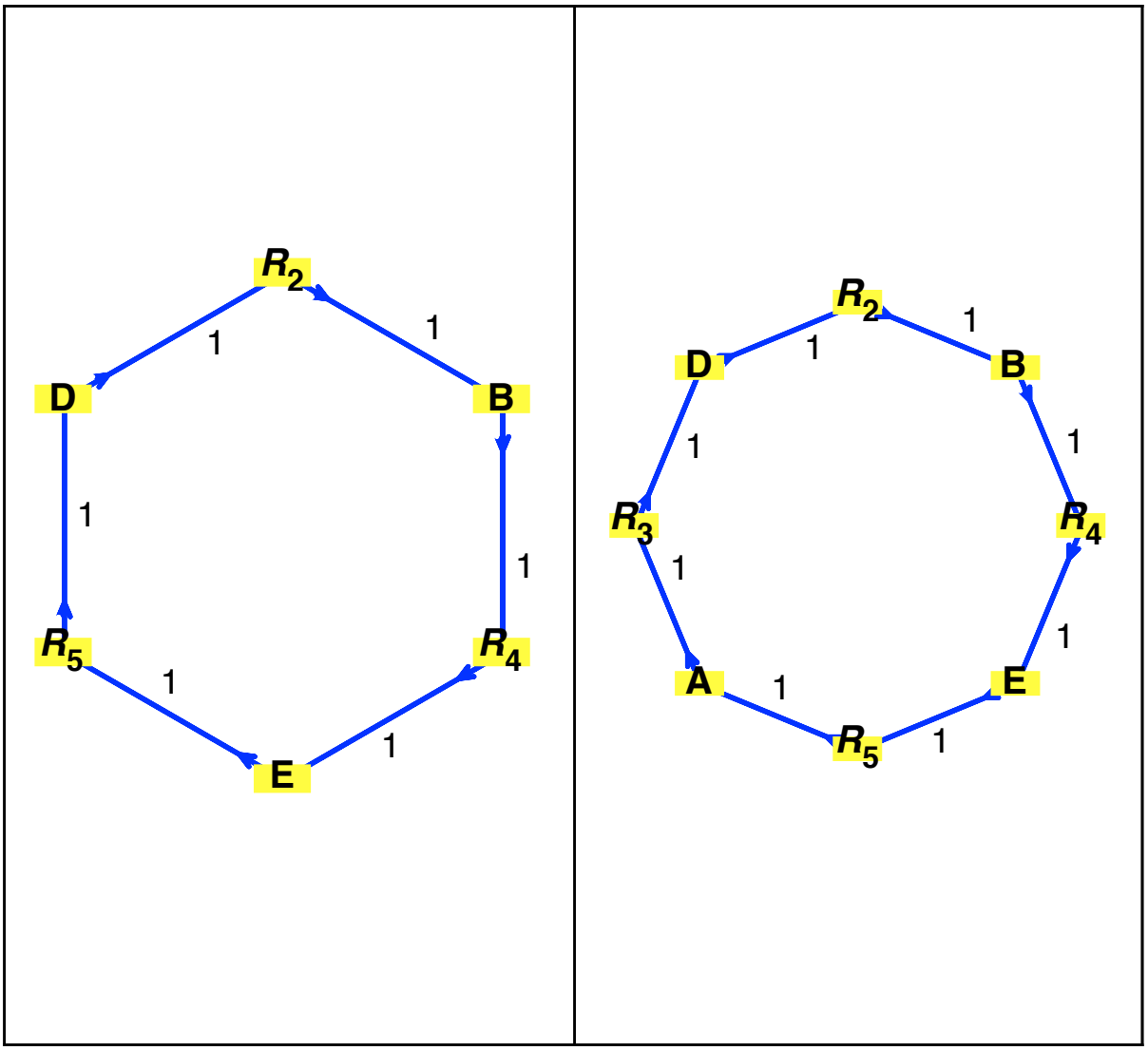


No. 14: injectiveEx3_1252.csv

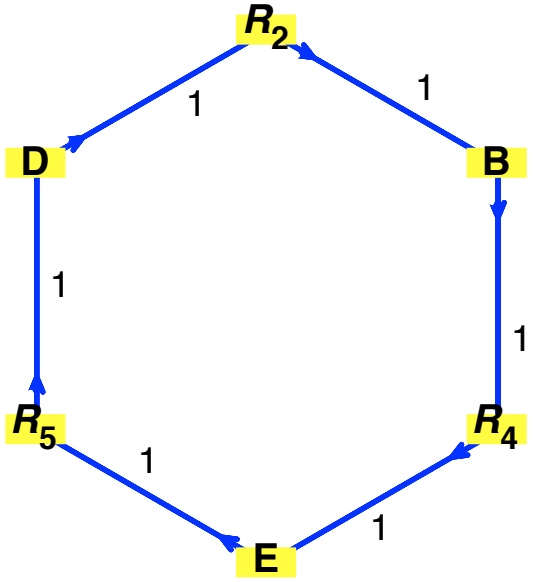


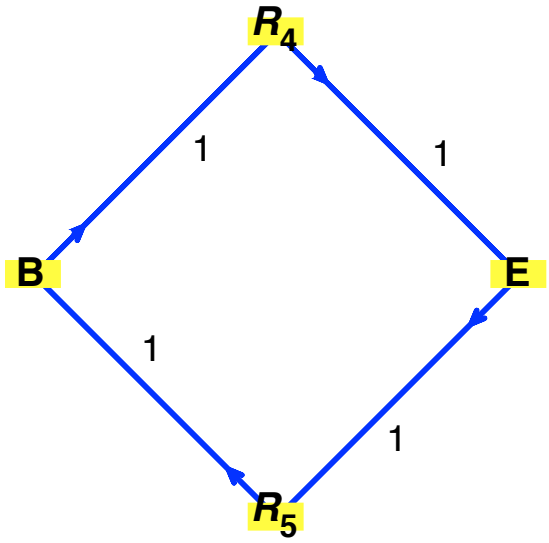


No. 15: injectiveEx3_1268.csv

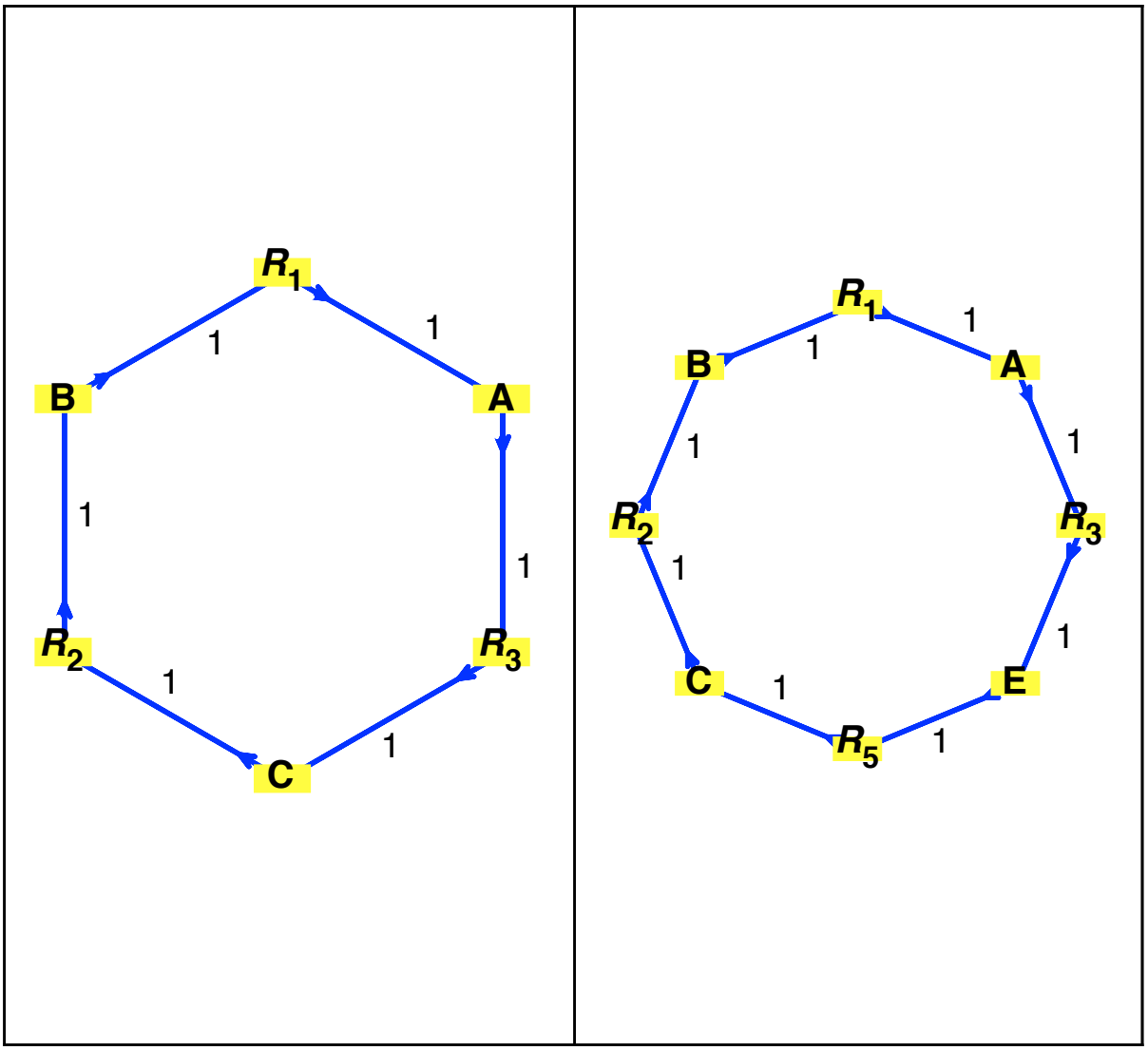


No. 16: injectiveEx3_1270.csv

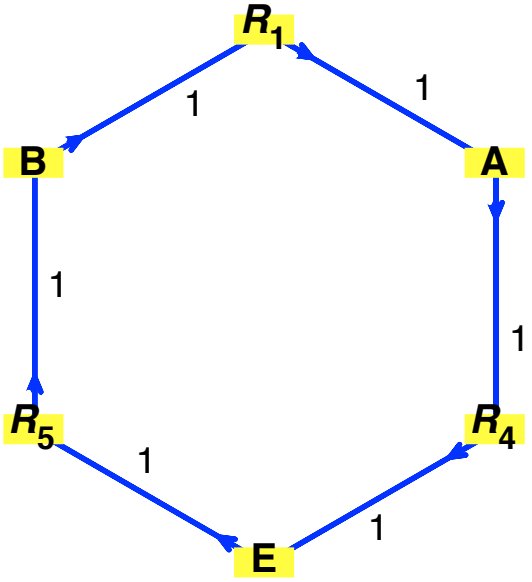


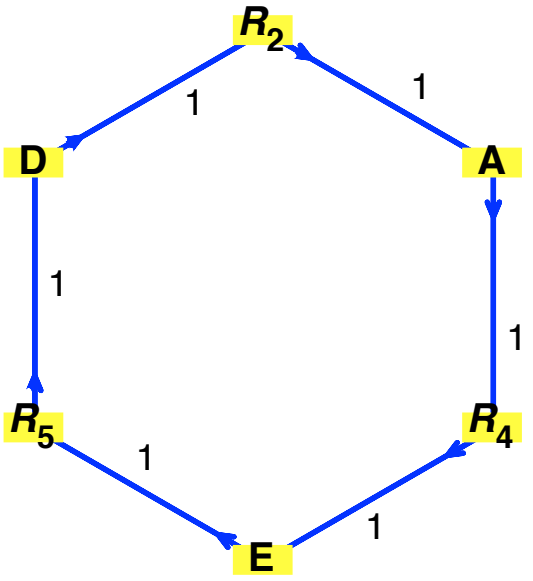


No. 17: injectiveEx3_1302.csv

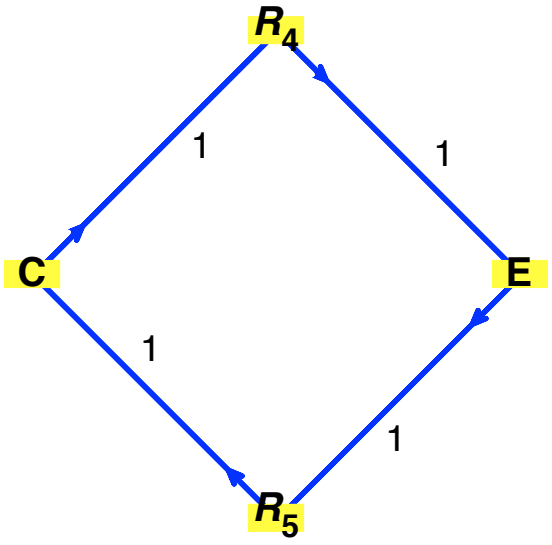


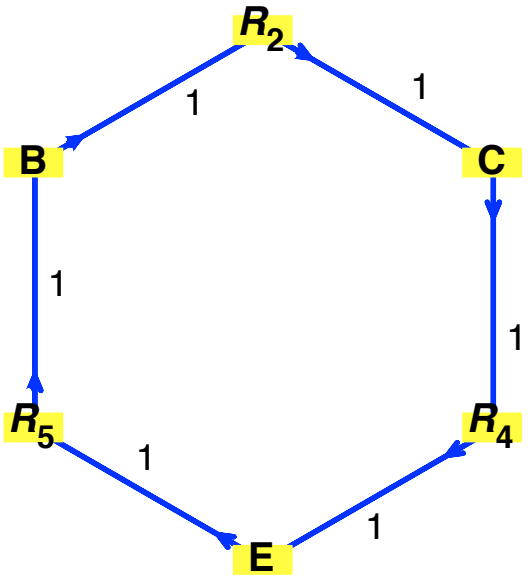
No. 18: injectiveEx3_1369.csv

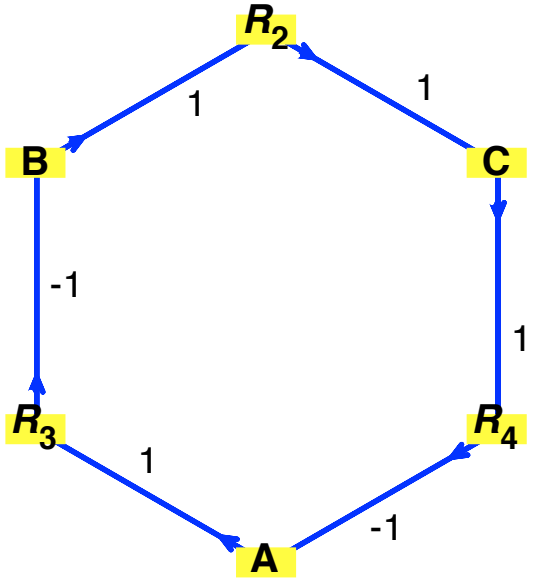


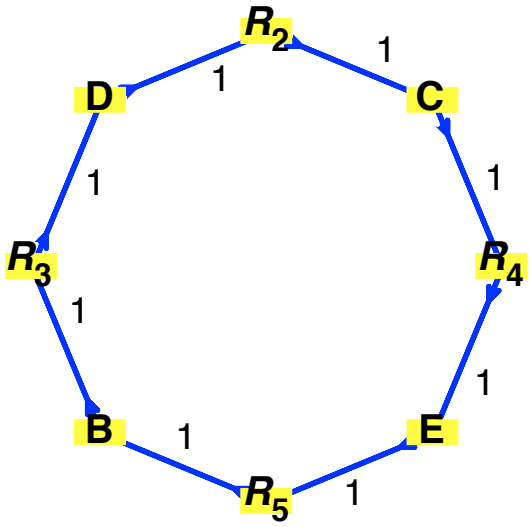


No. 19: injectiveEx3_1473.csv

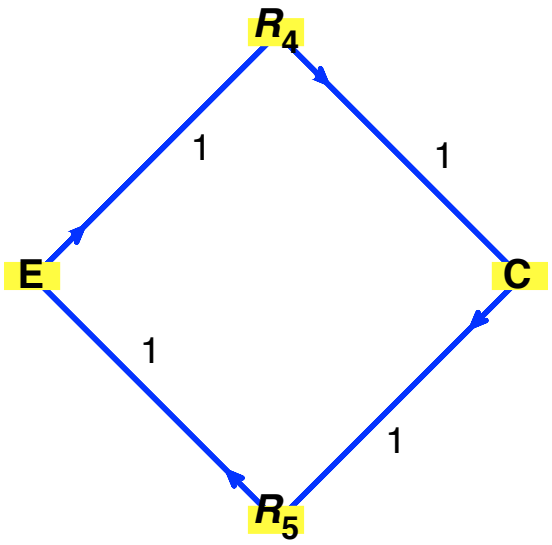

R4
1
1
C
E
1
1
R5

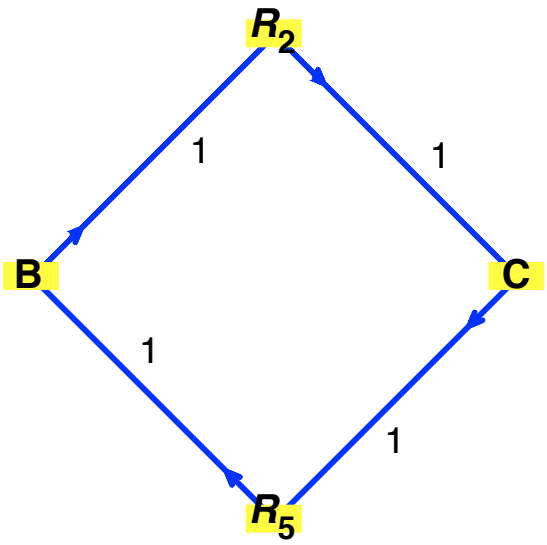

R2
1
1
B
C
1
1
R5
R4
1
1
E

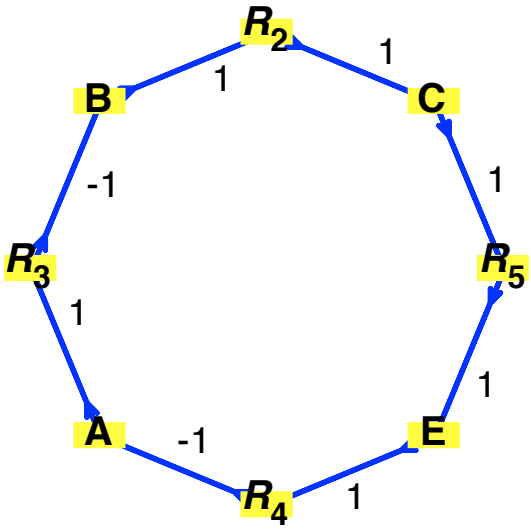


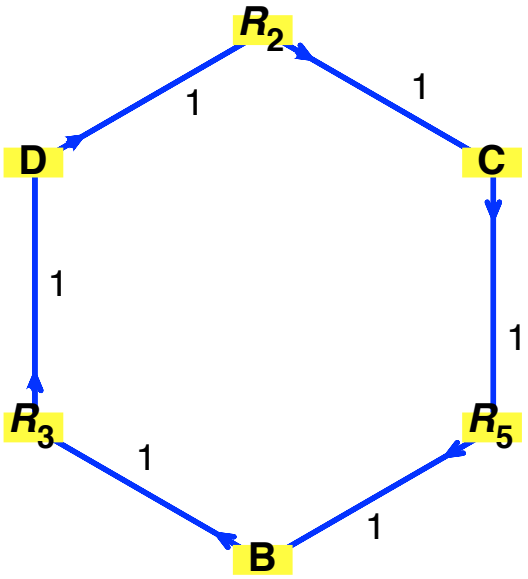


No. 20: injectiveEx3_1474.csv

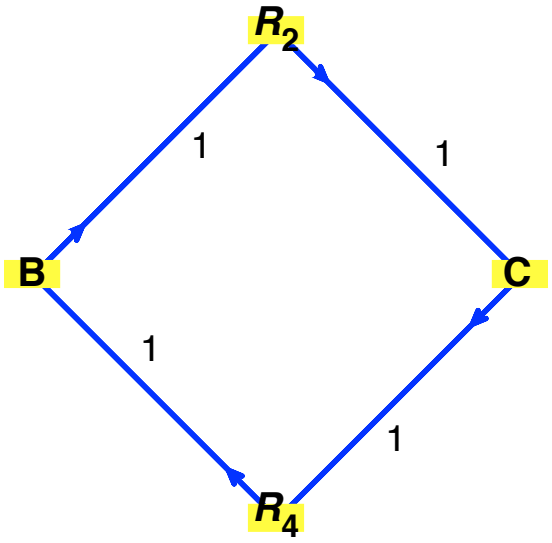

R_4
1
1
E
C
1
1
R_5

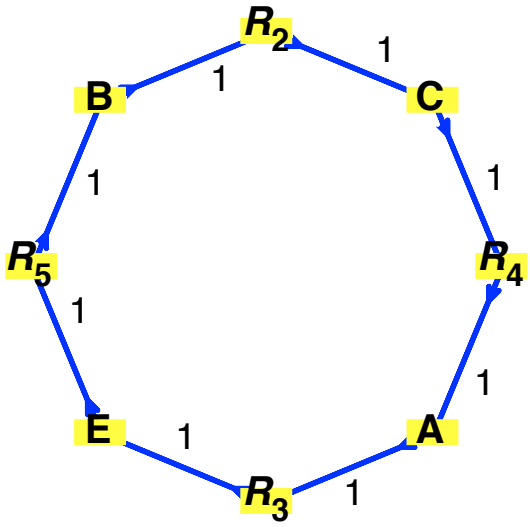

R_2
1
1
B
C
1
1
R_5



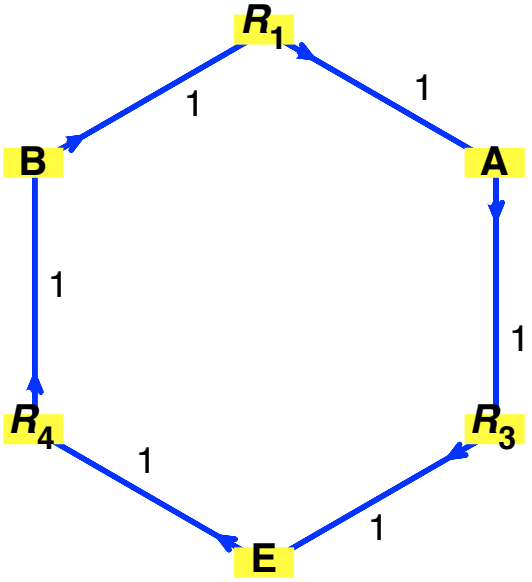


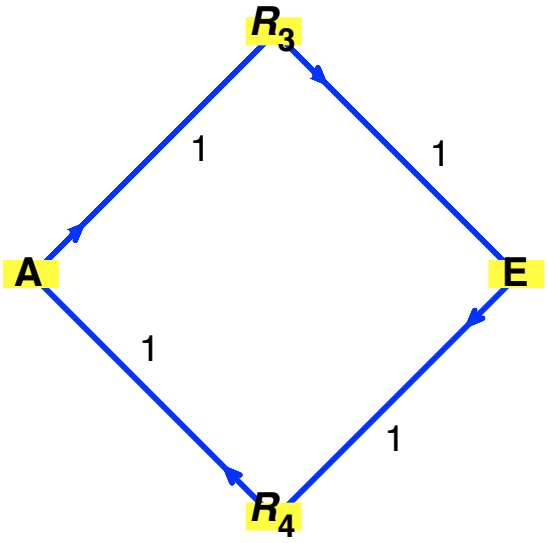
No. 21: injectiveEx3_1516.csv

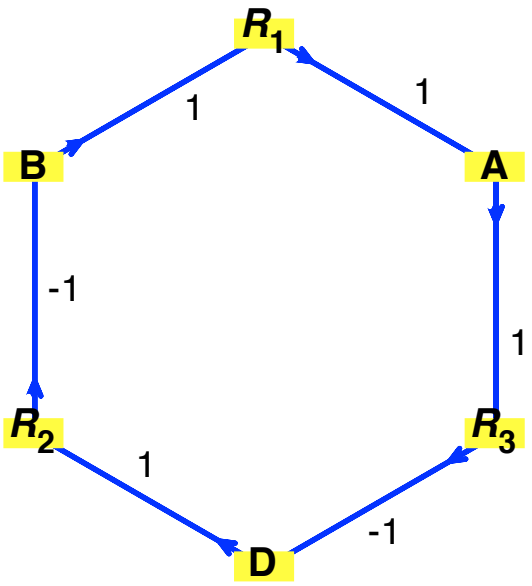




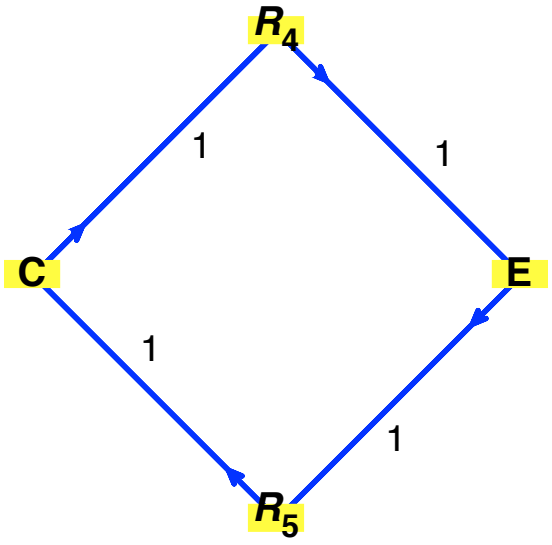
No. 22: injectiveEx3_1518.csv

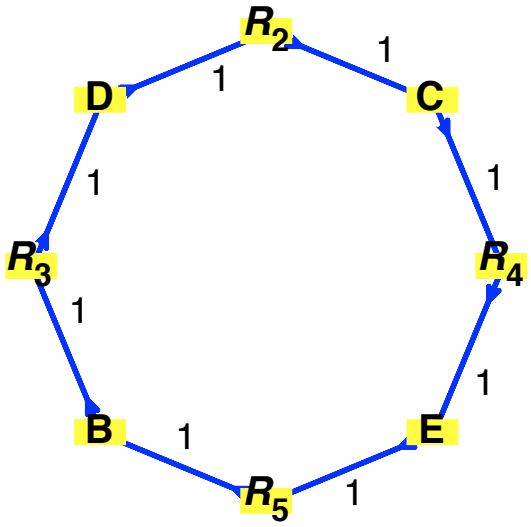

R1
1
1
B
A
1
1
R4
R3
1
1
E


R3
1
1
A
E
1
1
R4

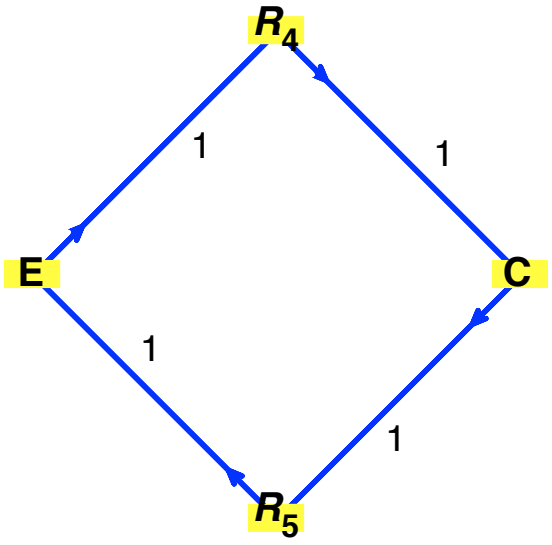


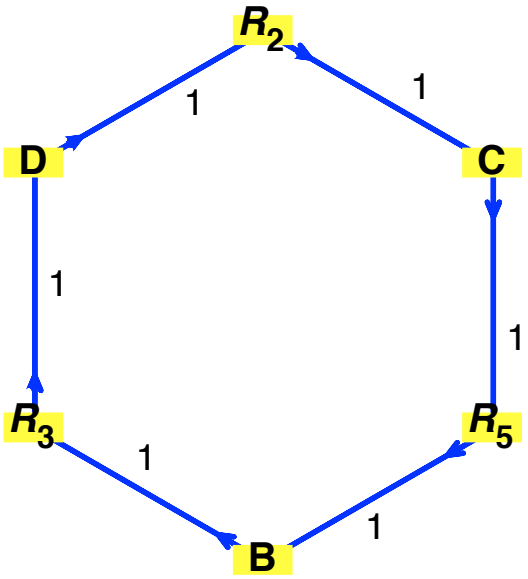
No. 23: injectiveEx3_1721.csv



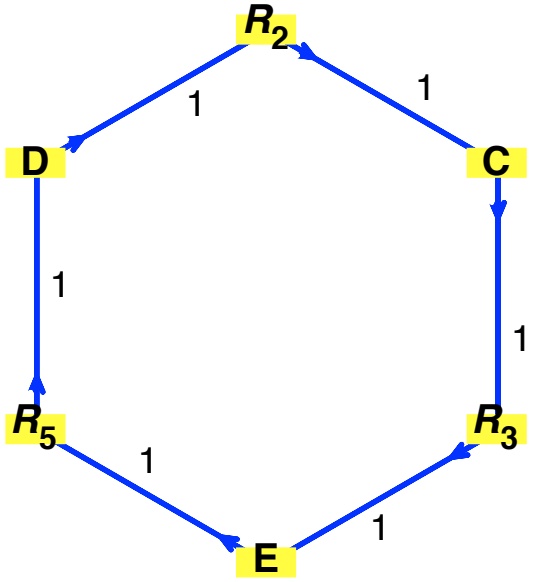


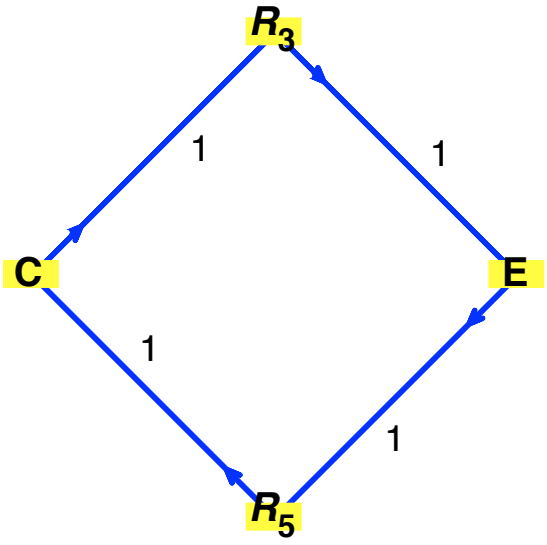
No. 24: injectiveEx3_1722.csv



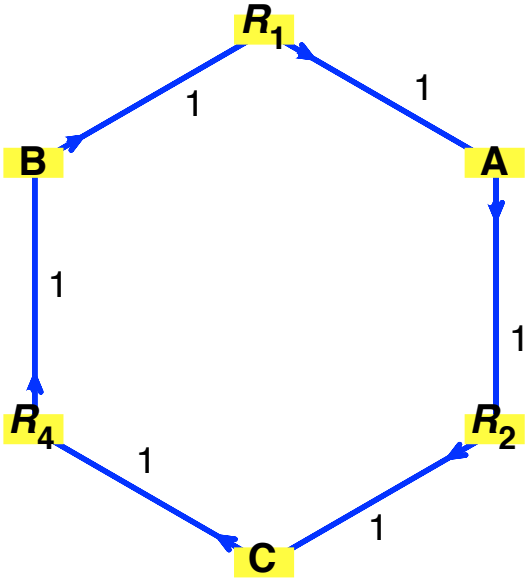


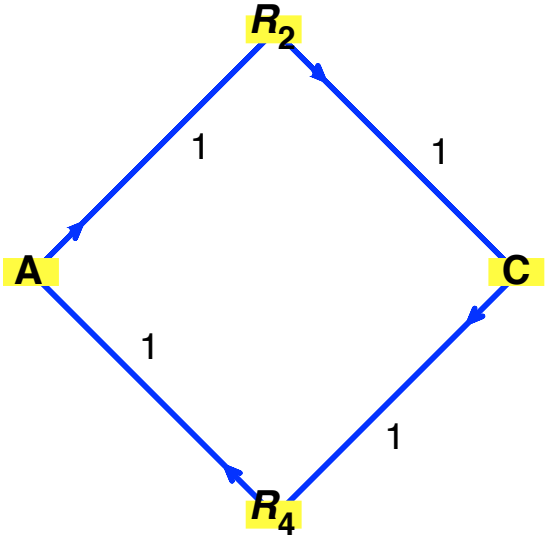
No. 25: injectiveEx3_1743.csv

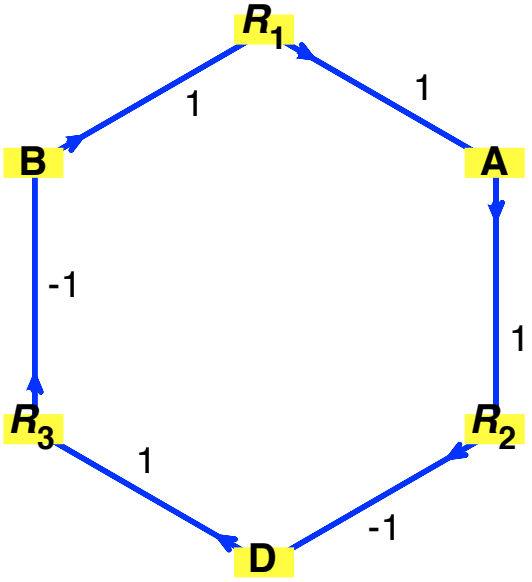


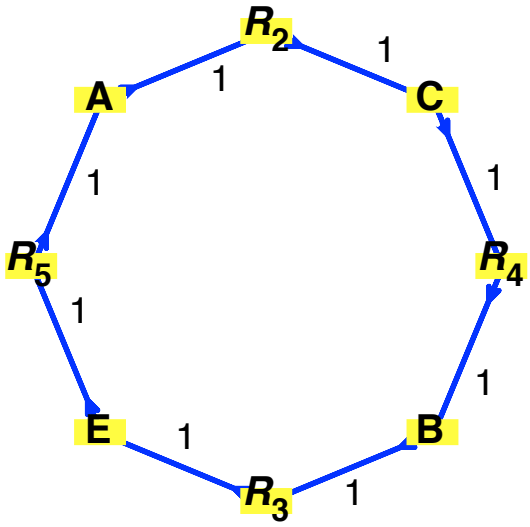


No. 26: injectiveEx3_1796.csv

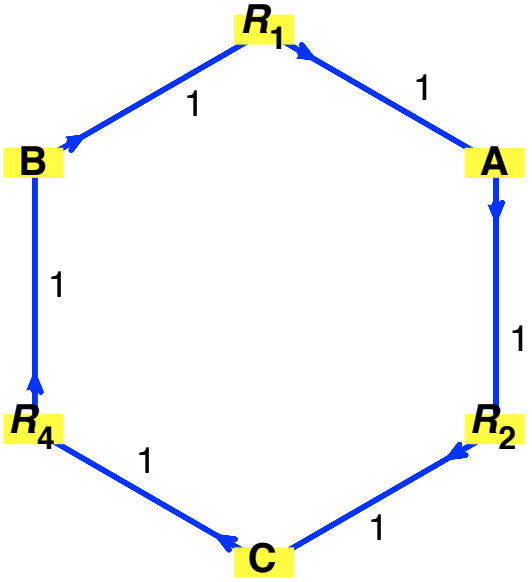

R1
1
1
B
A
1
1
R4
R2
1
1
C

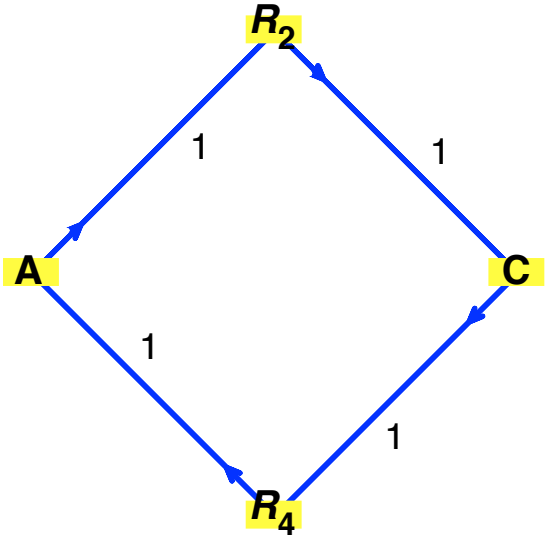

R2
1
1
A
C
1
1
R4

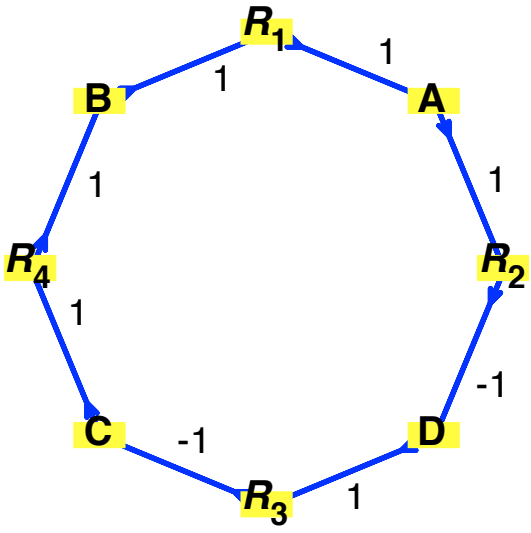


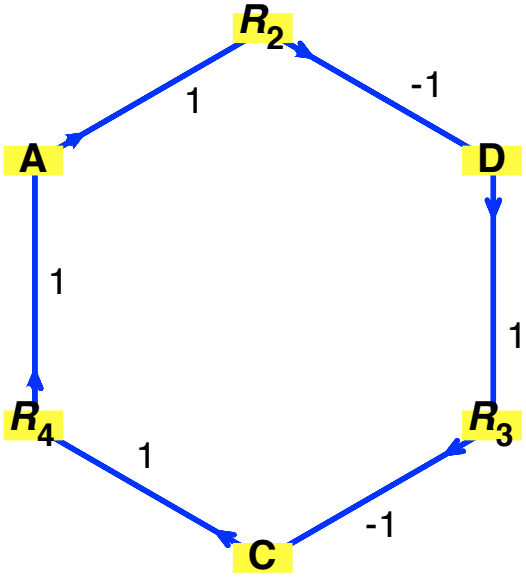


No. 27: injectiveEx3_1814.csv

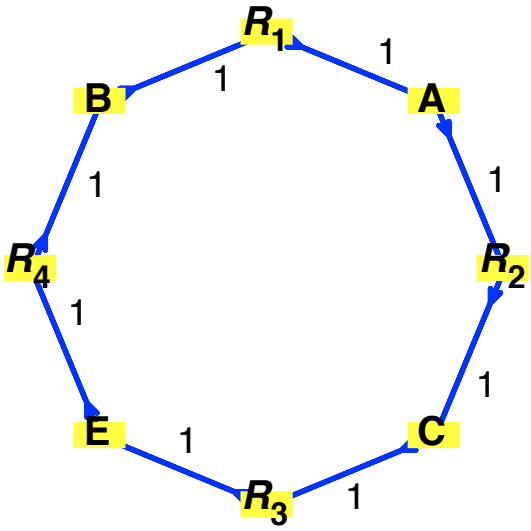

R1
1
1
B
A
1
1
R4
R2
1
1
C

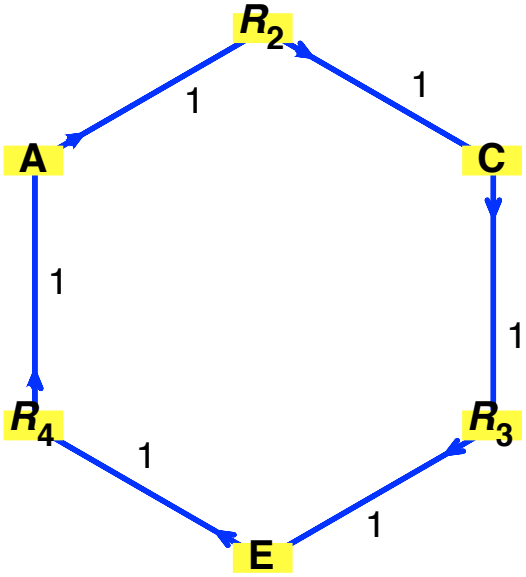

R2
1
1
A
C
1
1
R4



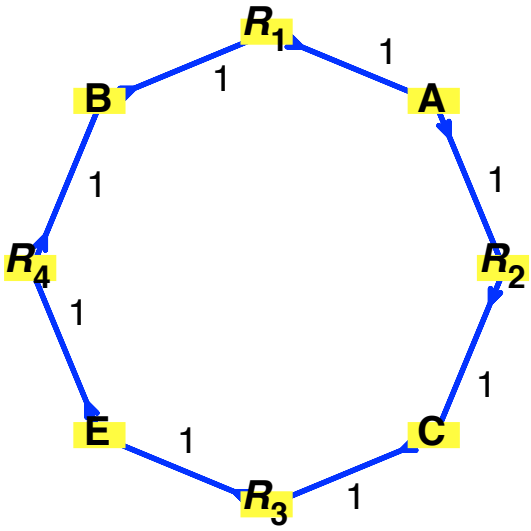


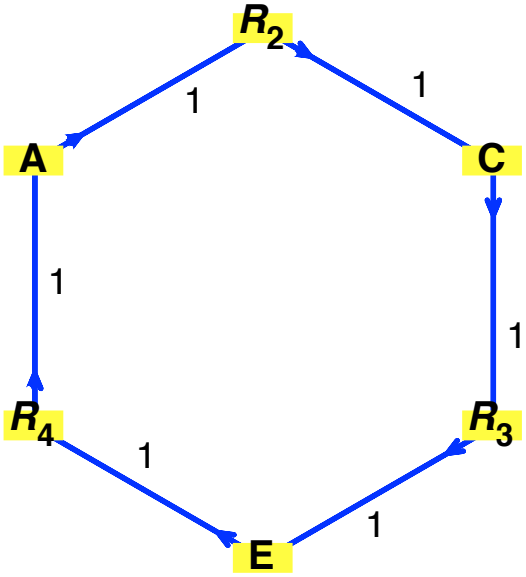
No. 28: injectiveEx3_1816.csv



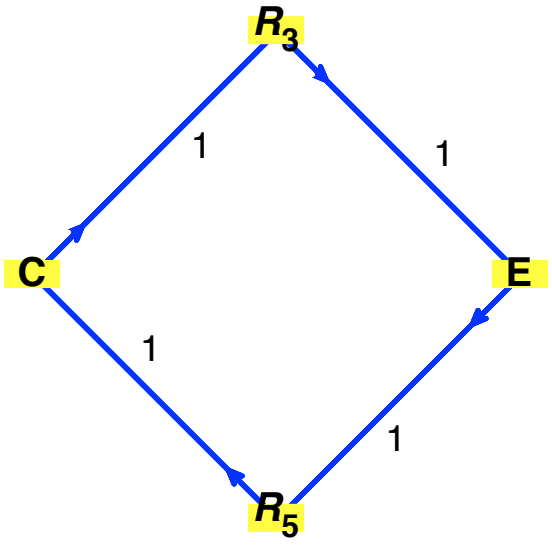


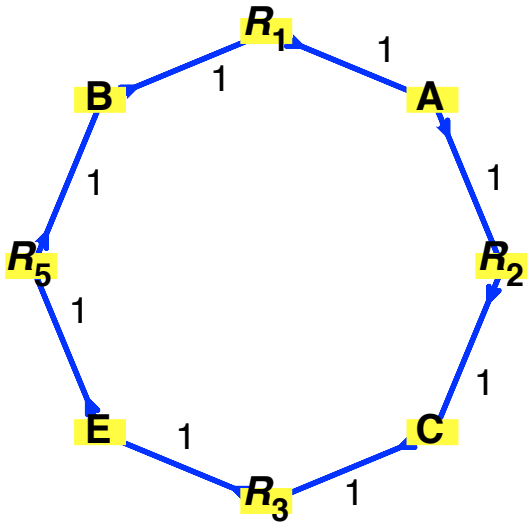
No. 29: injectiveEx3_1817.csv



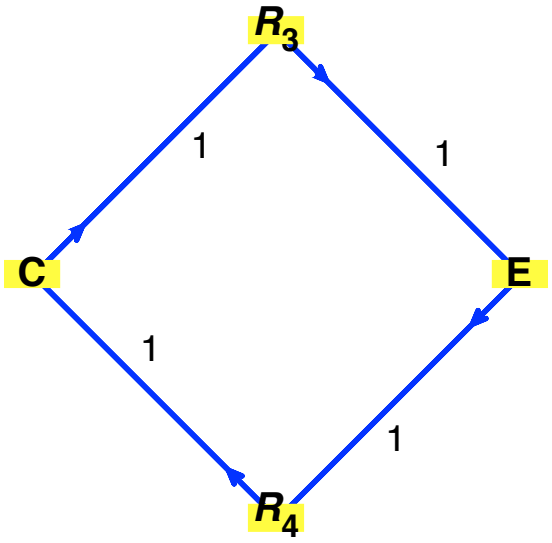


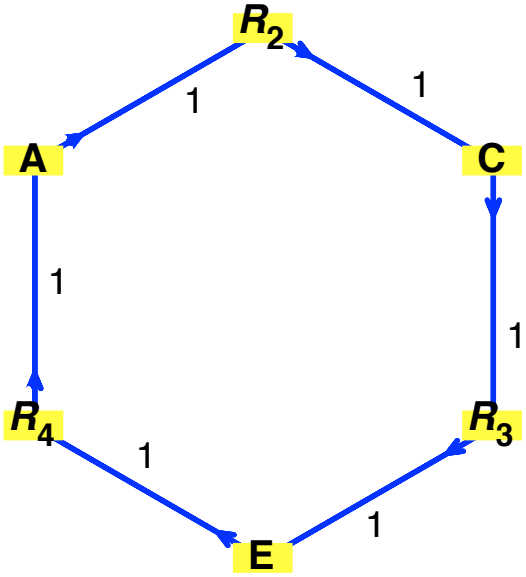
No. 30: injectiveEx3_1819.csv



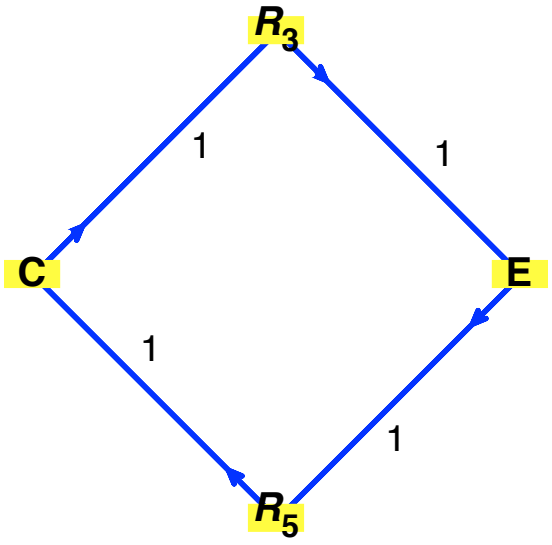


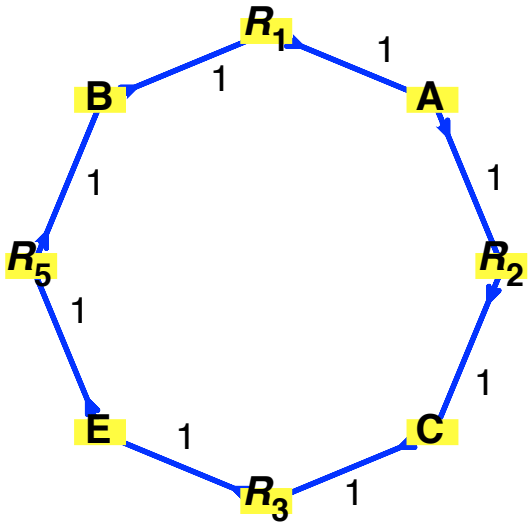
No. 31: injectiveEx3_1820.csv



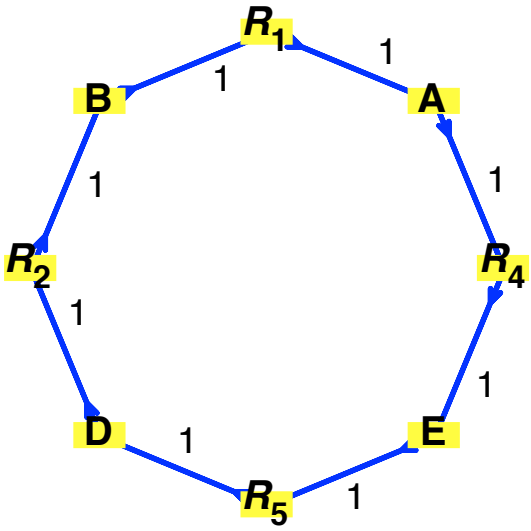


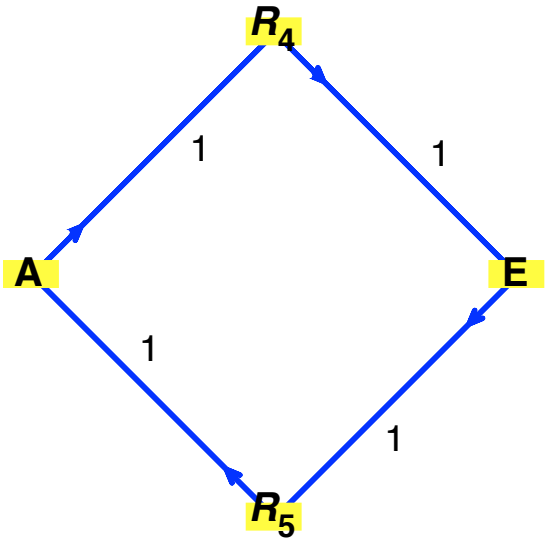
No. 32: injectiveEx3_1821.csv



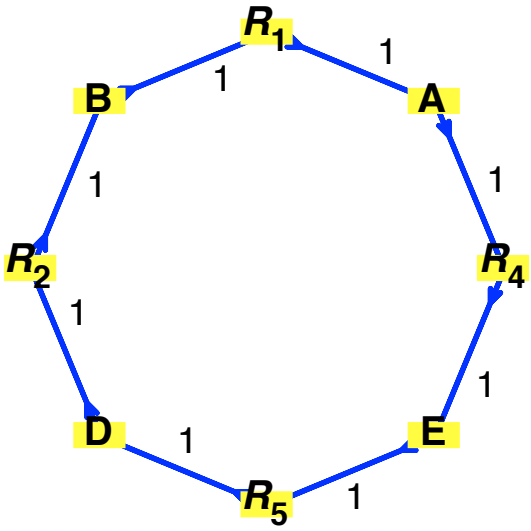


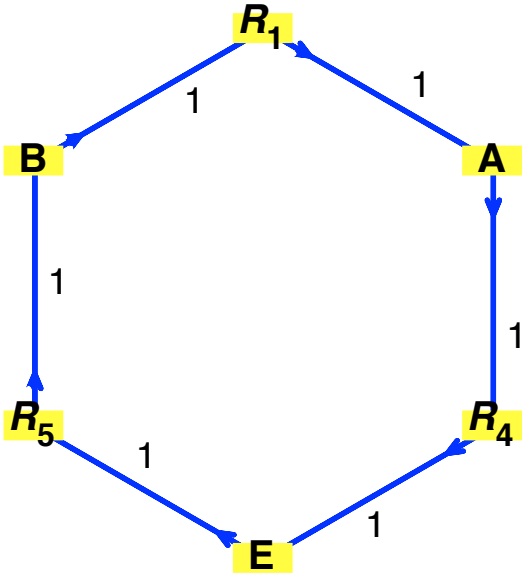
No. 33: injectiveEx3_2945.csv



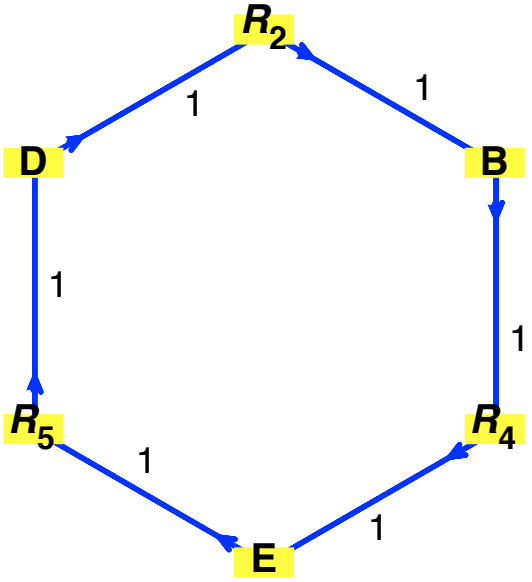


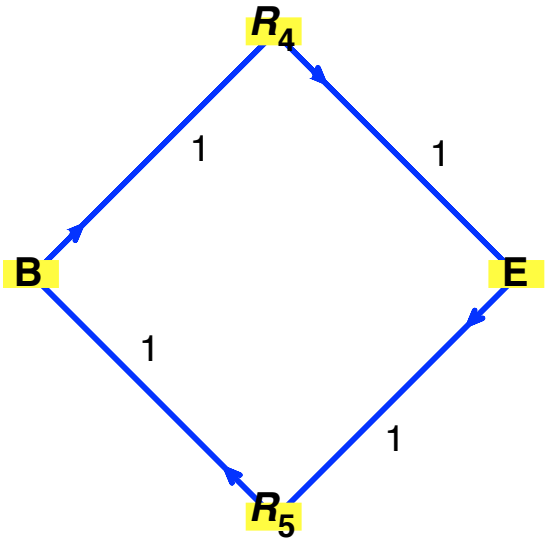
No. 34: injectiveEx3_2946.csv

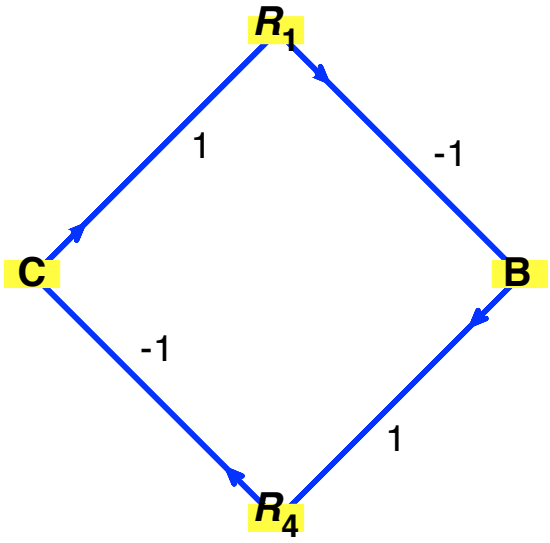


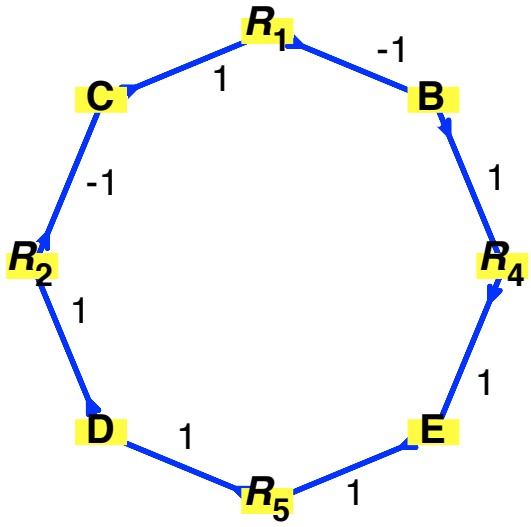


No. 35: injectiveEx3_2954.csv

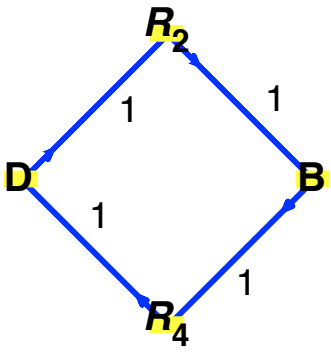
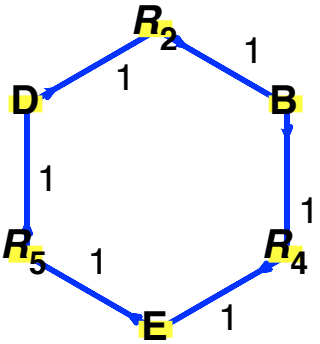
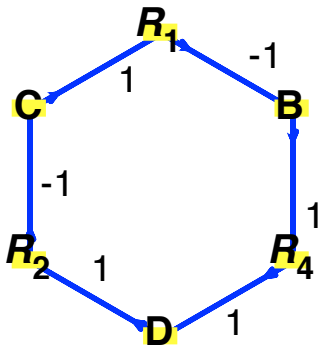

R2
1
1
D
B
1
1
R5
R4
1
1
E

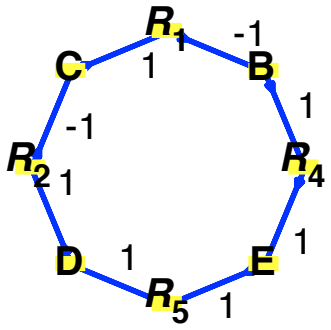

R4
1
1
B
E
1
1
R5

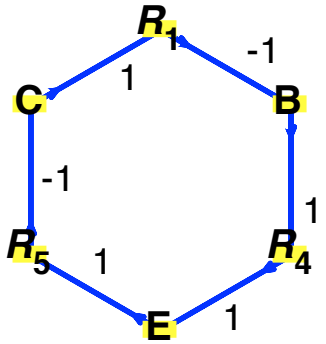




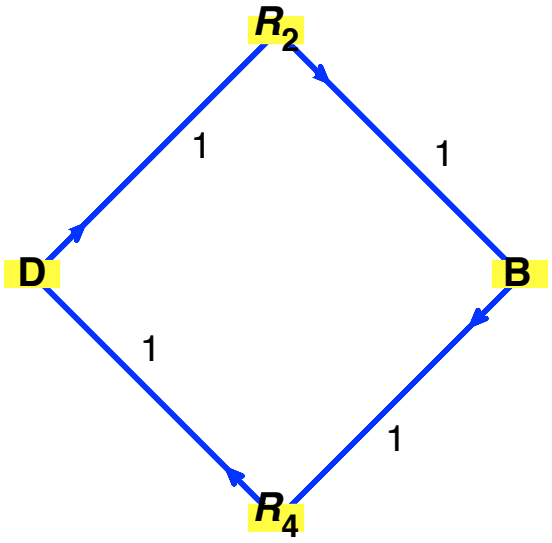
No. 36: injectiveEx3_2955.csv

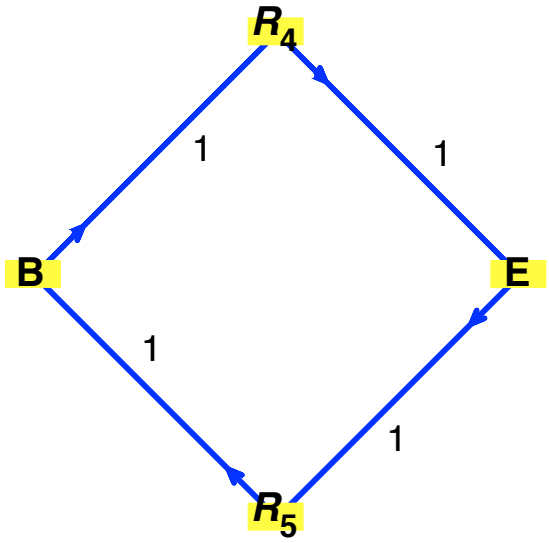
 R2 1 1 D B 1 1 R4	 R2 1 1 D B 1 1 R5 1 R4 E 1	 R1 -1 1 C B -1 1 R2 1 R4 D 1

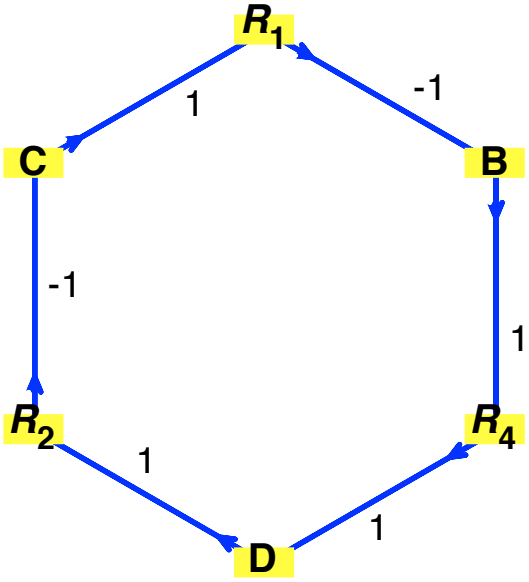


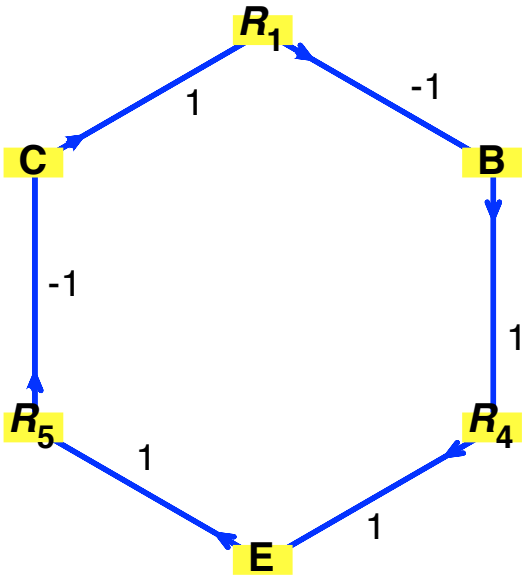


No. 37: injectiveEx3_2957.csv

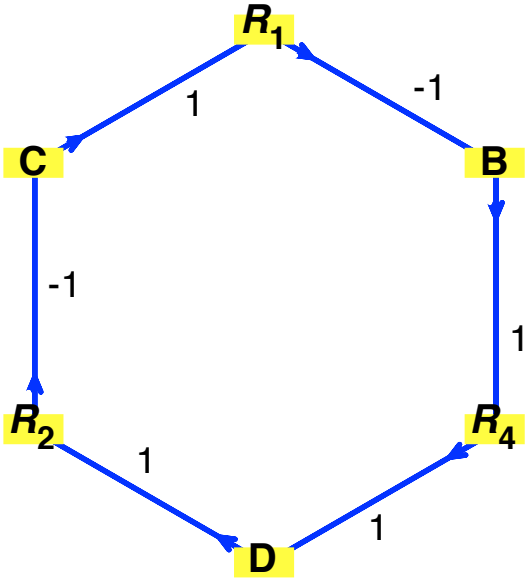

R2
1
1
D
B
1
1
R4

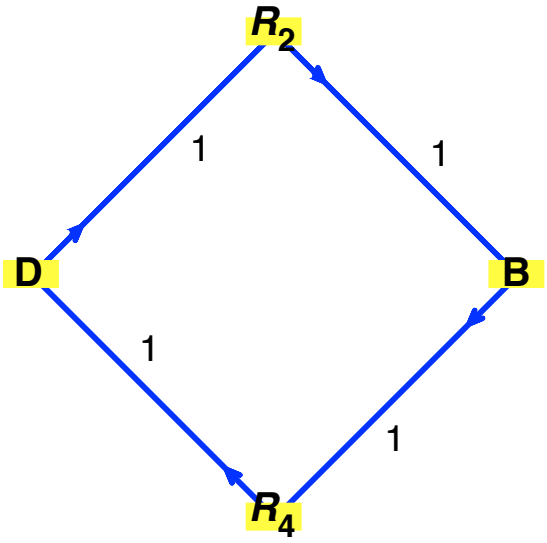

R4
1
1
B
E
1
1
R5



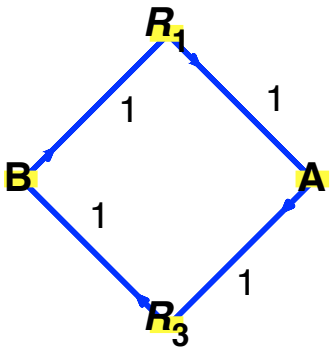


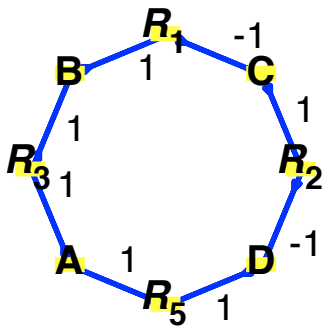
No. 38: injectiveEx3_2958.csv

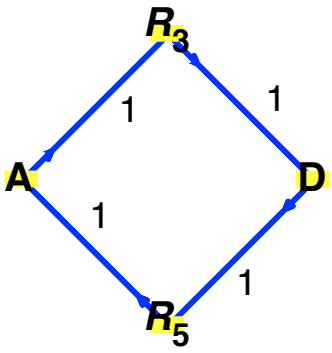


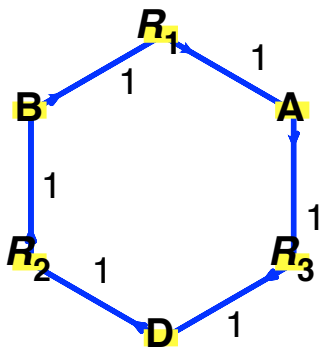


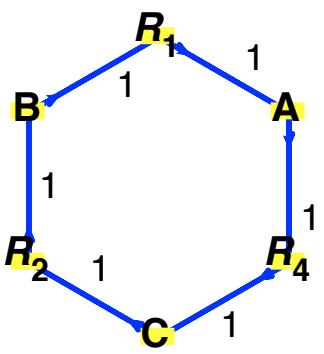
No. 39: injectiveEx3_2965.csv

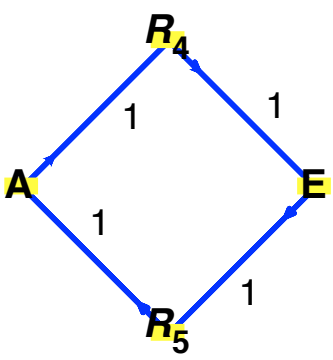

R_1
1
1
B
A
1
1
R_3

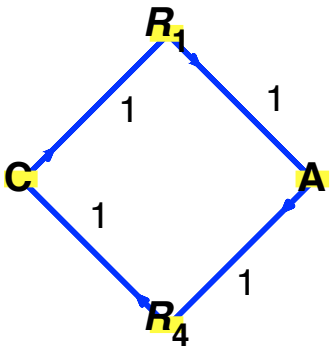

R_1
-1
B
C
1
1
R_3
1
R_2
A
1
D
-1
R_5
1

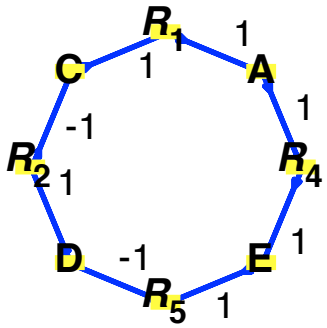

R_3
1
1
A
D
1
1
R_5


R1
1
1
B
A
1
1
R2
1
R3
D
1

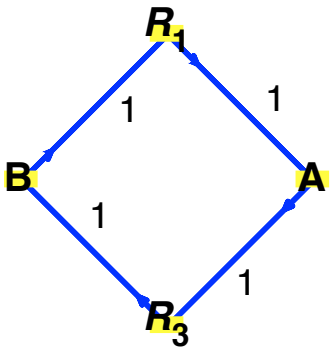

R1
1
1
B
A
1
1
R2
1
R4
C
1

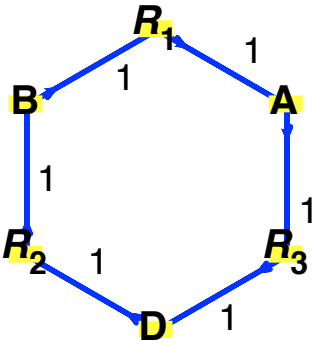

R4
1
1
A
E
1
1
R5

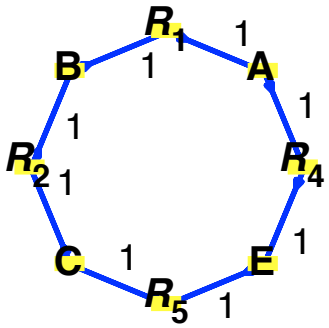


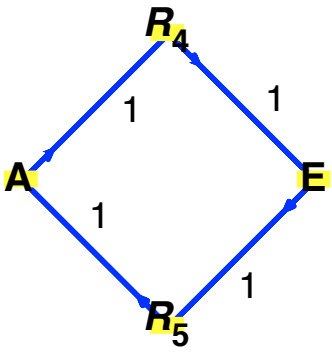


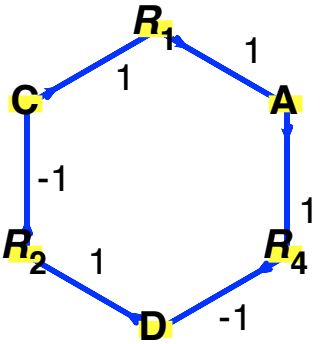
No. 40: injectiveEx3_2969.csv

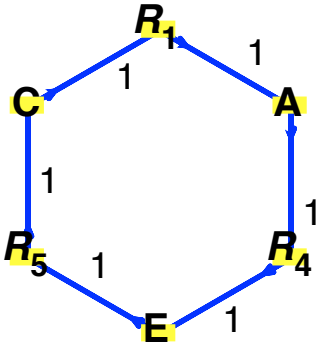

R1
1
1
B
A
1
1
R3


R1
1
1
B
A
1
1
R2
1
R3
D
1

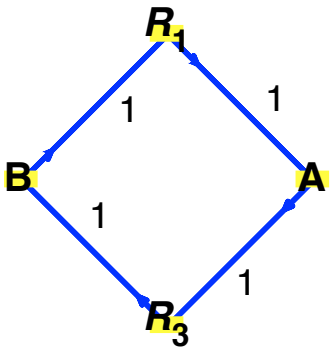

R1
1
1
B
A
1
1
R2
1
R4
1
C
1
E
R5
1

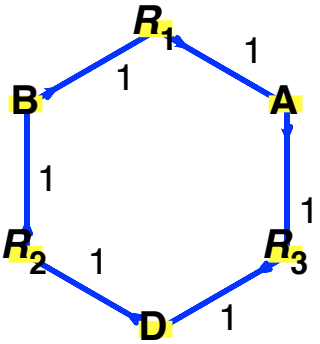


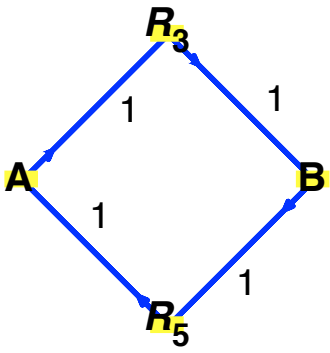


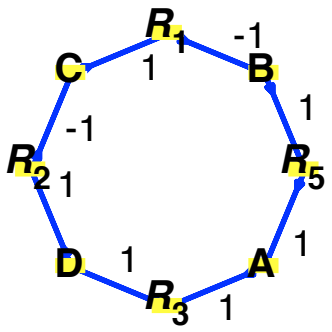


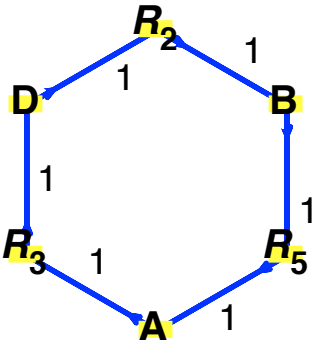
No. 41: injectiveEx3_2970.csv


R1
1
1
B
A
1
1
R3

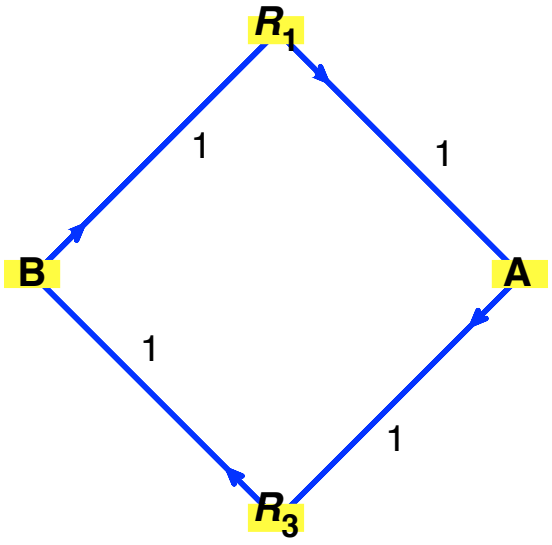

R1
1
1
B
A
1
1
R2
1
R3
D
1

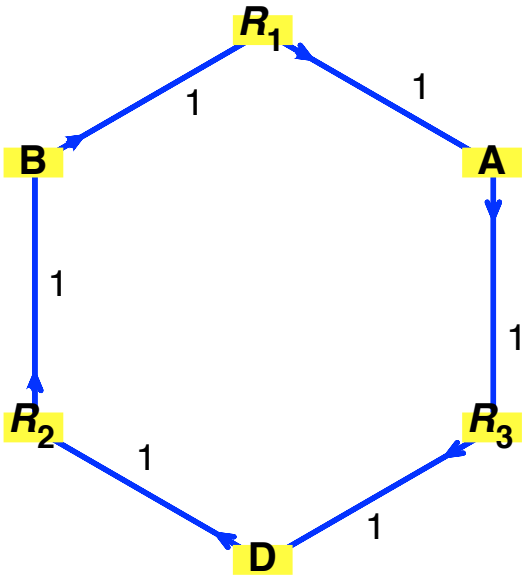

R3
1
1
A
B
1
1
R5



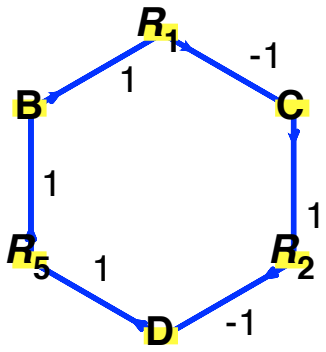


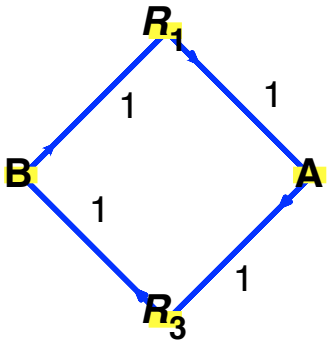
No. 42: injectiveEx3_2971.csv

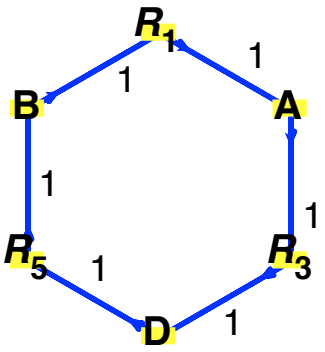


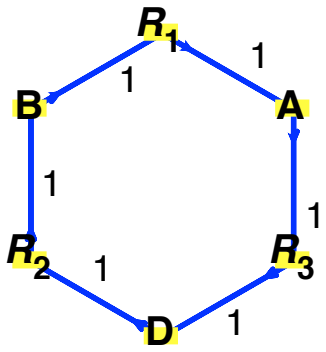


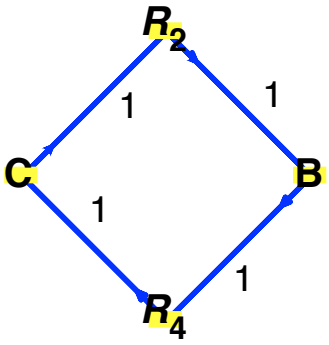
No. 43: injectiveEx3_2972.csv

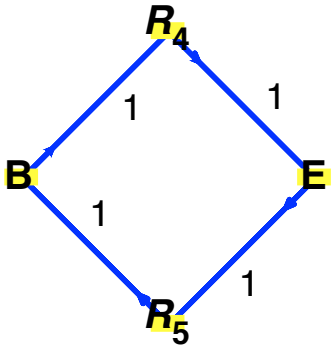

R1
-1
1
B
C
1
1
R5
R2
1
D
-1


R1
1
1
B
A
1
1
R3

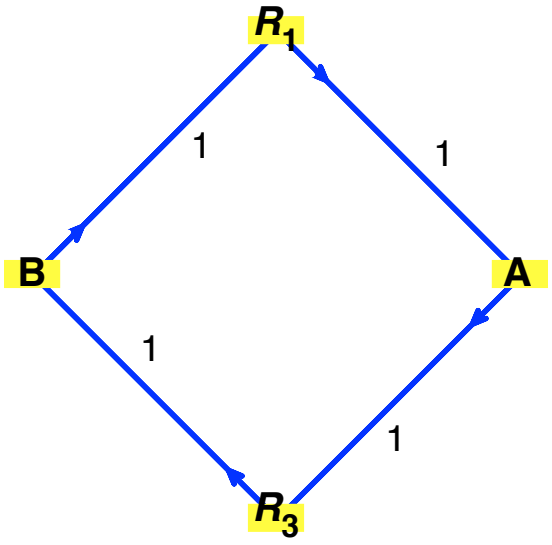

R1
1
1
B
A
1
1
R5
R3
1
D
1

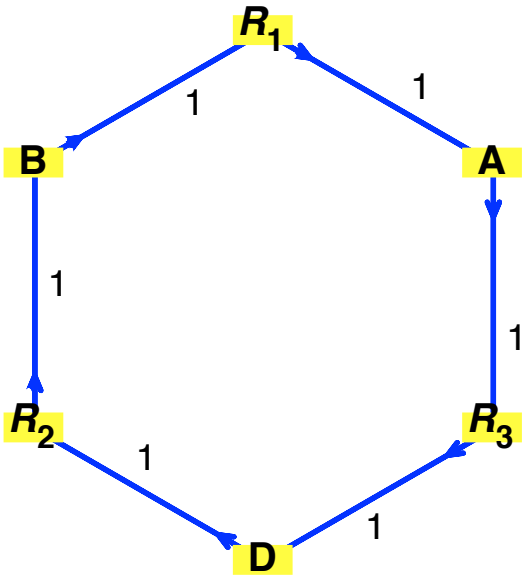


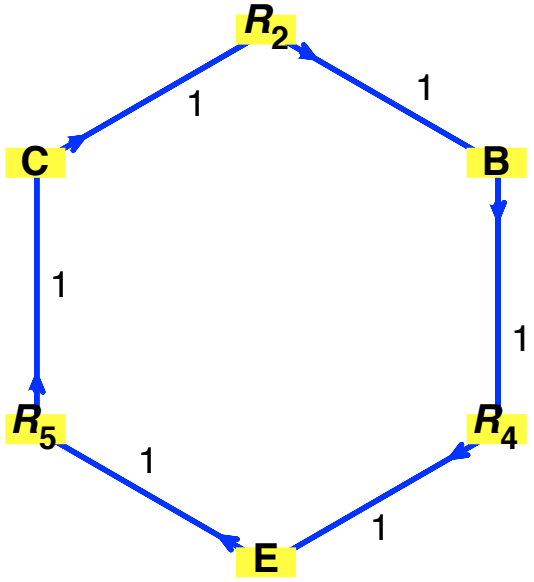


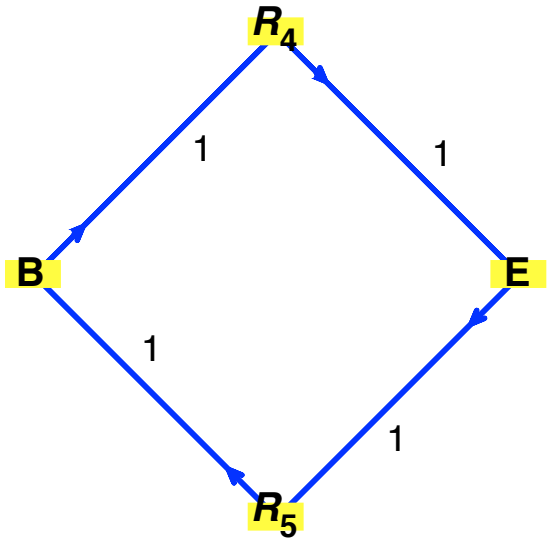


No. 44: injectiveEx3_2974.csv

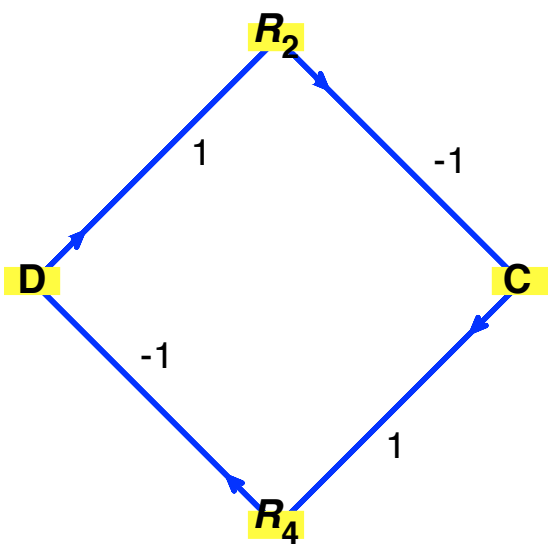

R1
1
1
B
A
1
1
R3

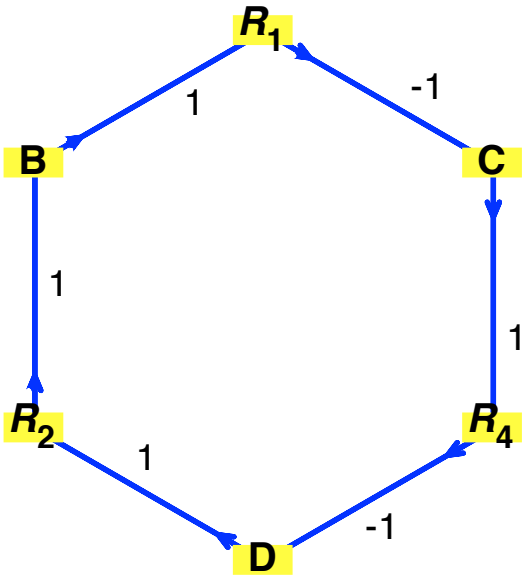

R1
1
1
B
A
1
1
R2
R3
1
1
D

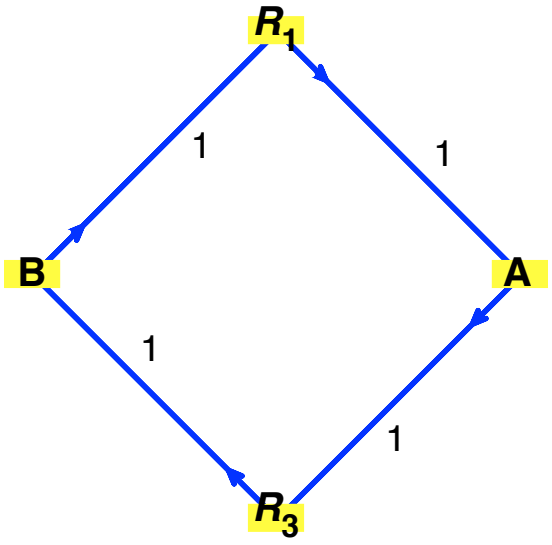


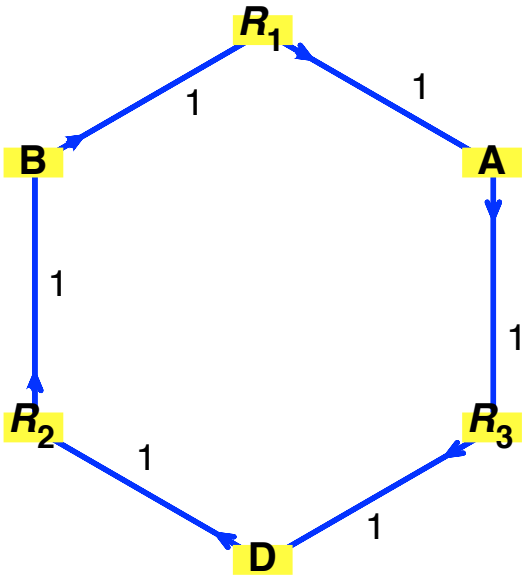


No. 45: injectiveEx3_2975.csv

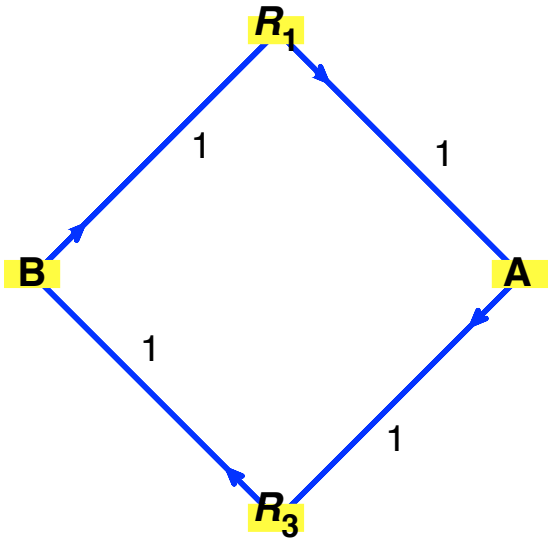

R2
1
-1
D
C
-1
1
R4

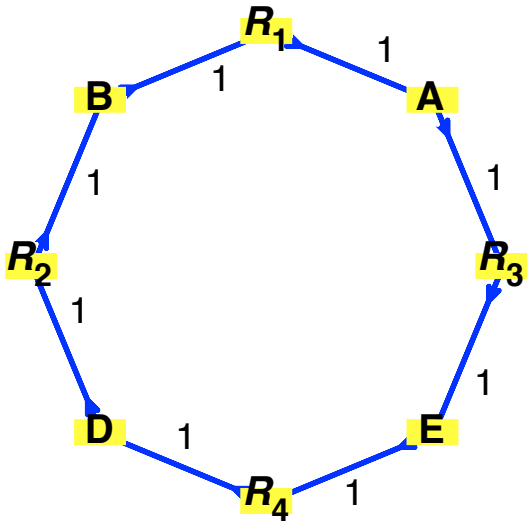

R1
1
-1
B
C
1
1
R2
R4
1
-1
D



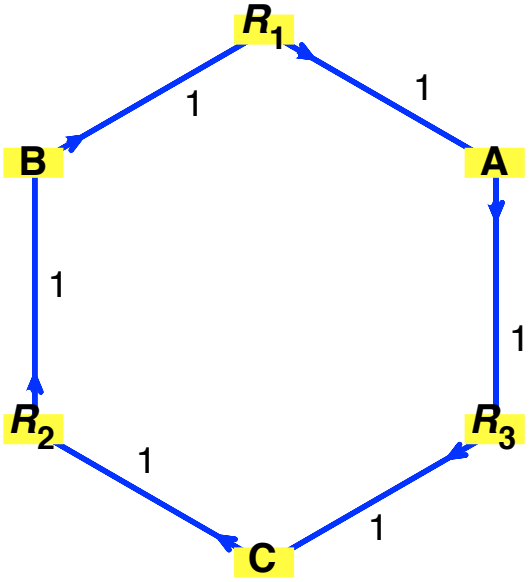


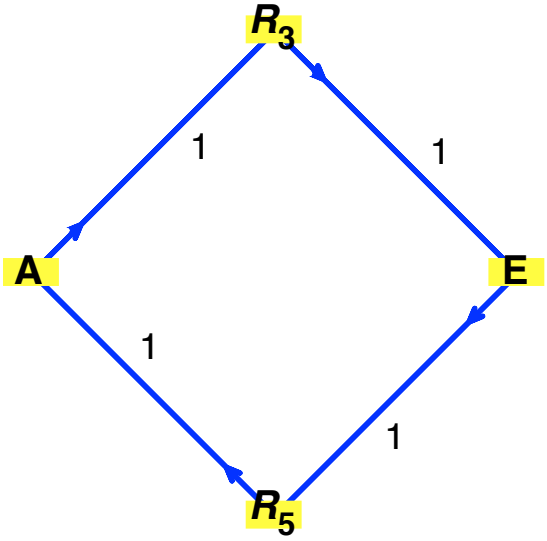
No. 46: injectiveEx3_2980.csv

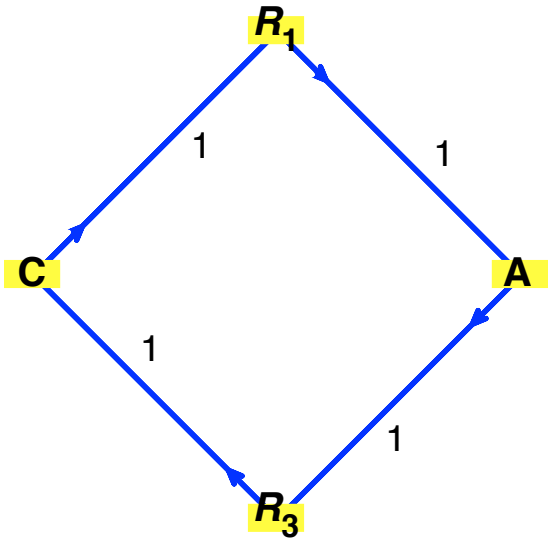


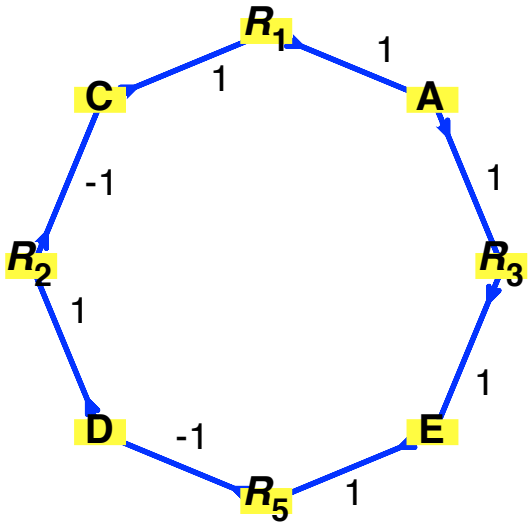


No. 47: injectiveEx3_2983.csv

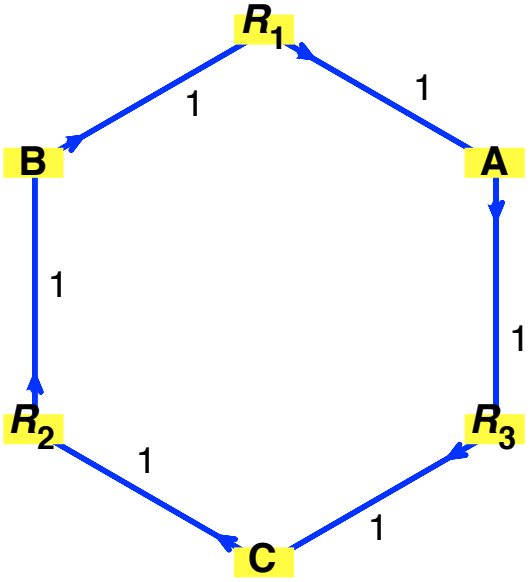

R1
1
1
B
A
1
1
R2
R3
1
1
C

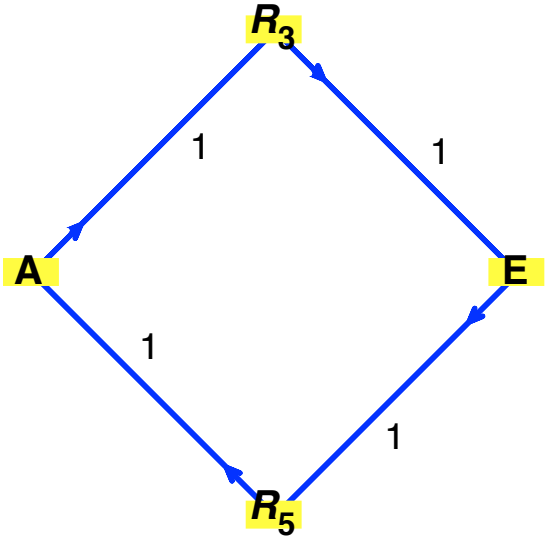

R3
1
1
A
E
1
1
R5

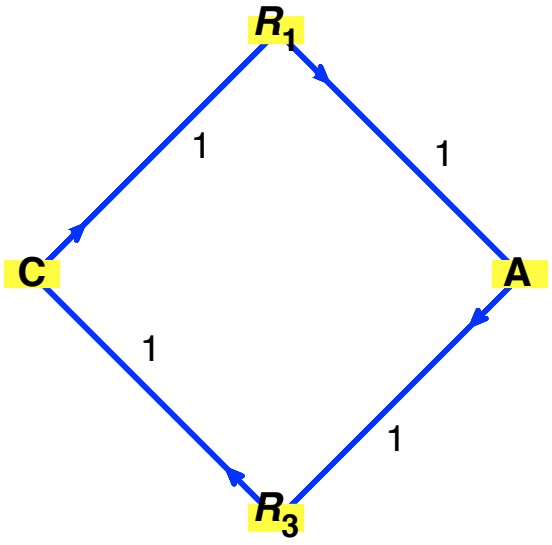


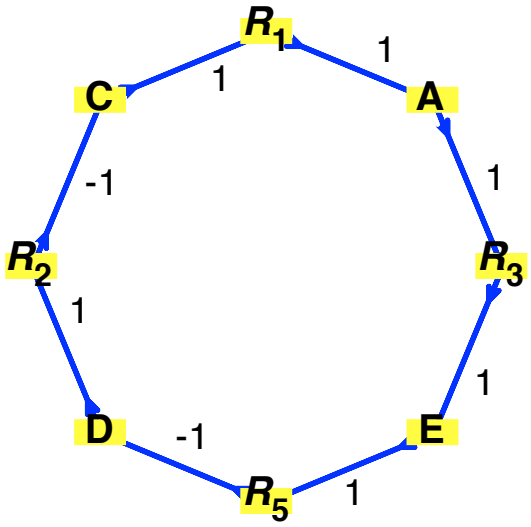


No. 48: injectiveEx3_2986.csv

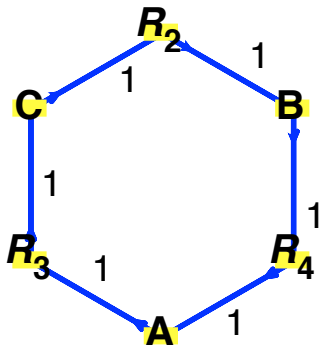

R1
1
1
B
A
1
1
R2
R3
1
1
C

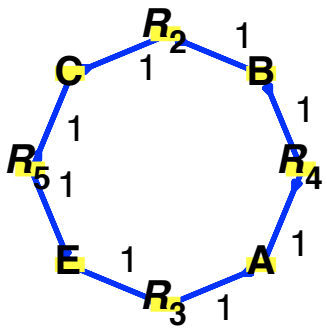

R3
1
1
A
E
1
1
R5

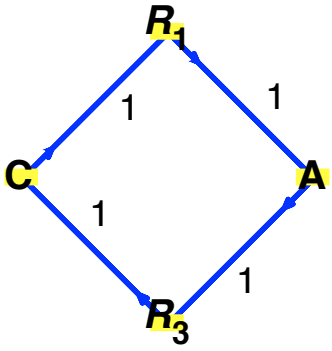


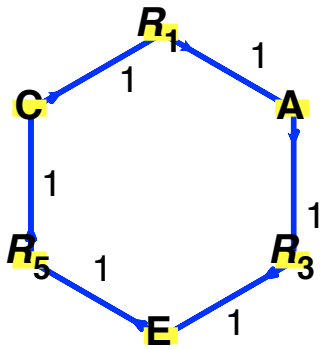


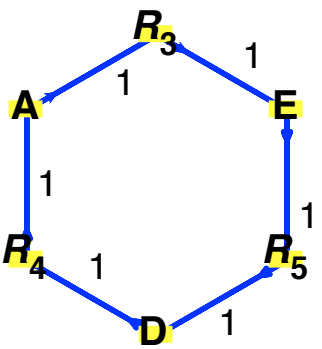
No. 49: injectiveEx3_2990.csv


R2
C
B
R3
R4
A
1
1
1
1
1
1

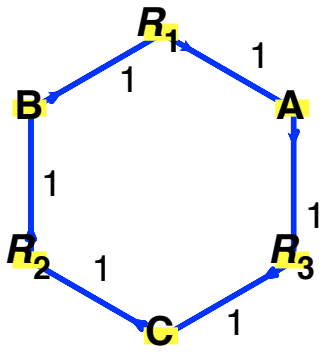

R2
C
B
R5
R4
E
A
R3
1
1
1
1
1
1
1
1

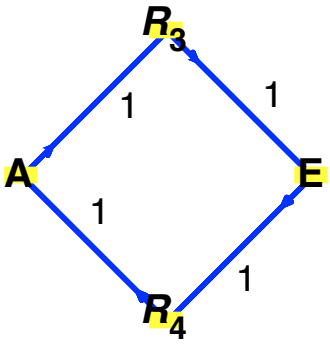

R1
C
A
R3
1
1
1
1

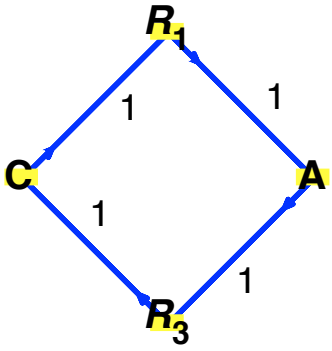


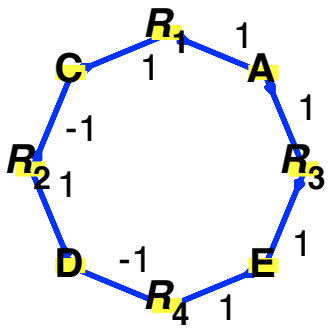


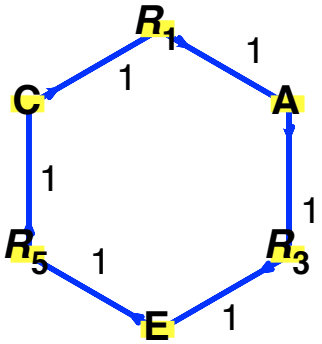
No. 50: injectiveEx3_2992.csv

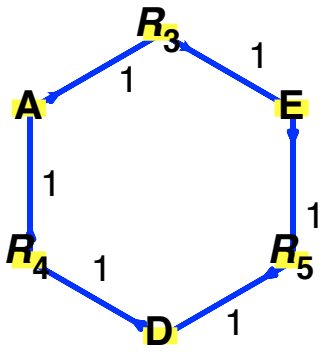

R_1
1
1
B
A
1
1
R_2
1
R_3
C
1


R_3
1
1
A
E
1
1
R_4

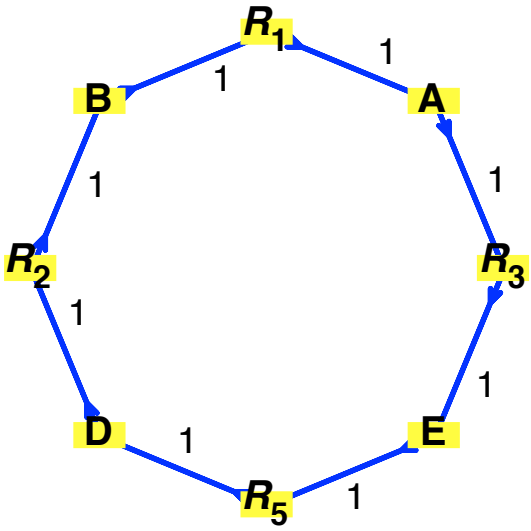

R_1
1
1
C
A
1
1
R_3

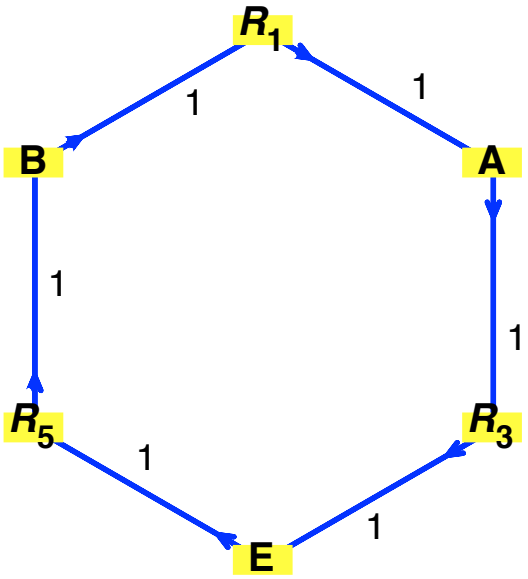




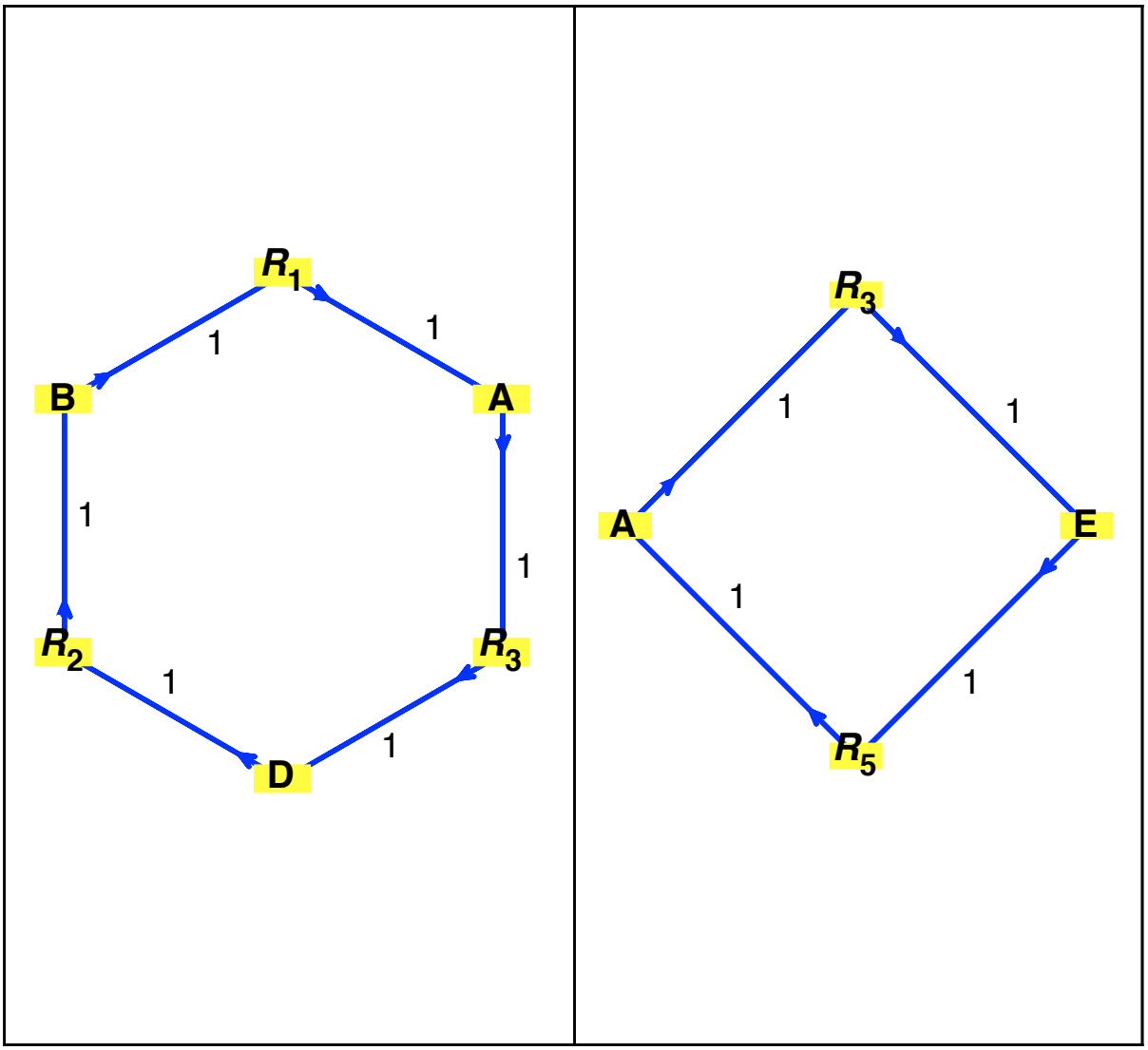


No. 51: injectiveEx3_2996.csv

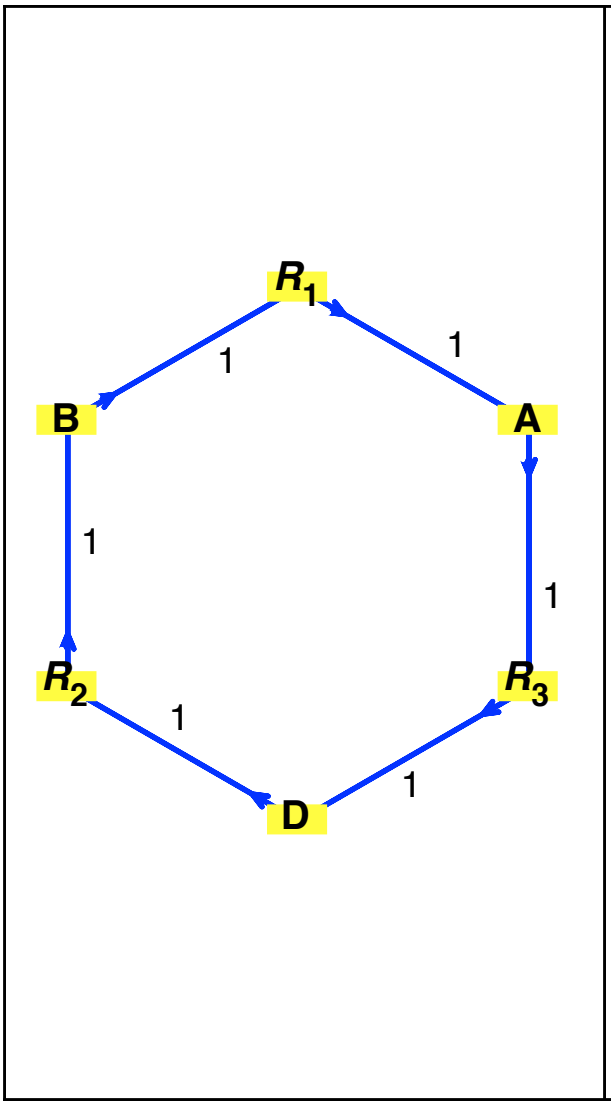


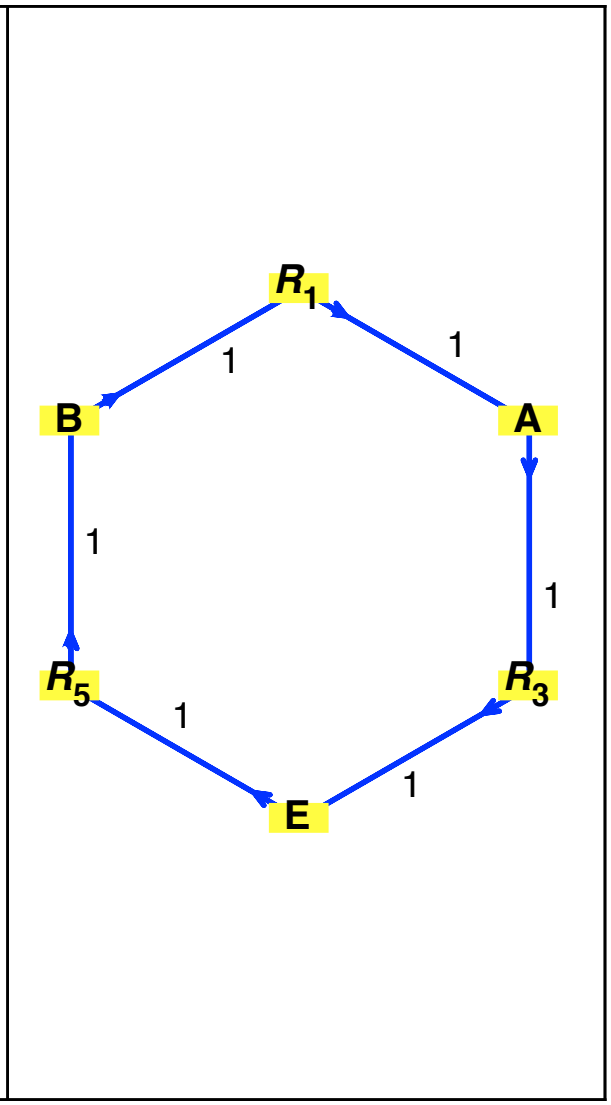


No. 52: injectiveEx3_3000.csv

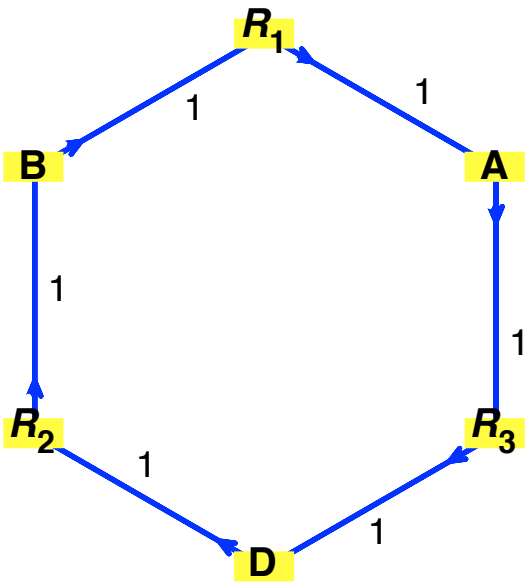


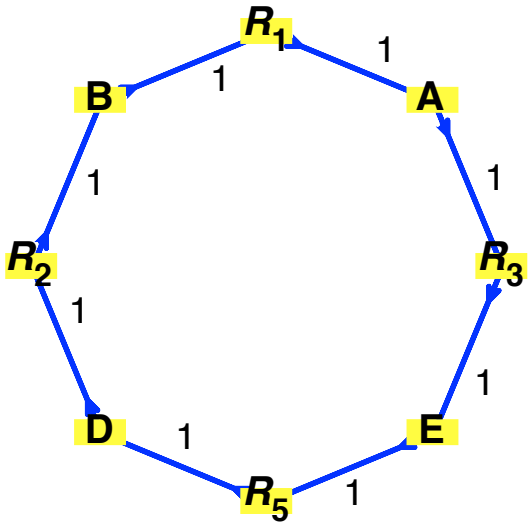
No. 53: injectiveEx3_3002.csv

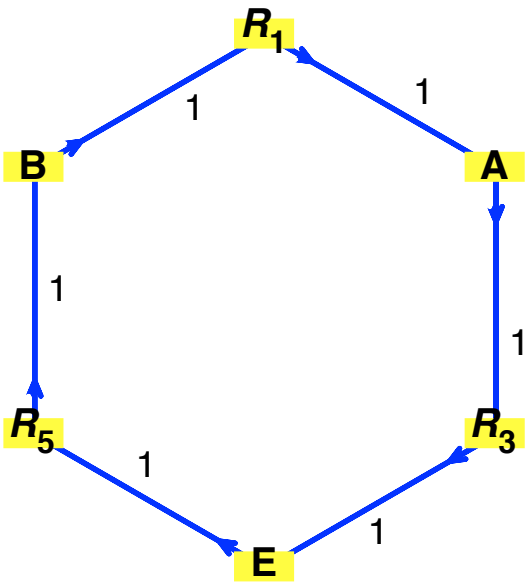




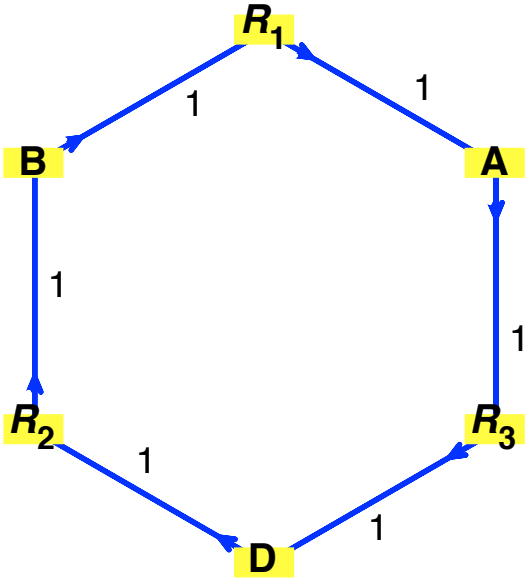
No. 54: injectiveEx3_3003.csv

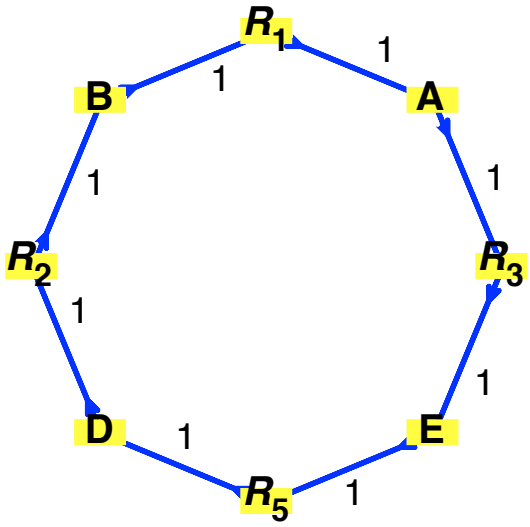

R1
1
1
B
A
1
1
R2
R3
1
1
D


R1
1
1
B
A
1
1
R2
R3
1
1
D
E
1
R5
1

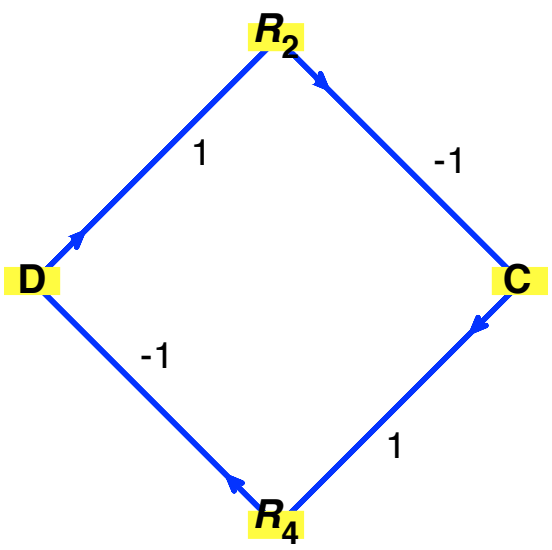


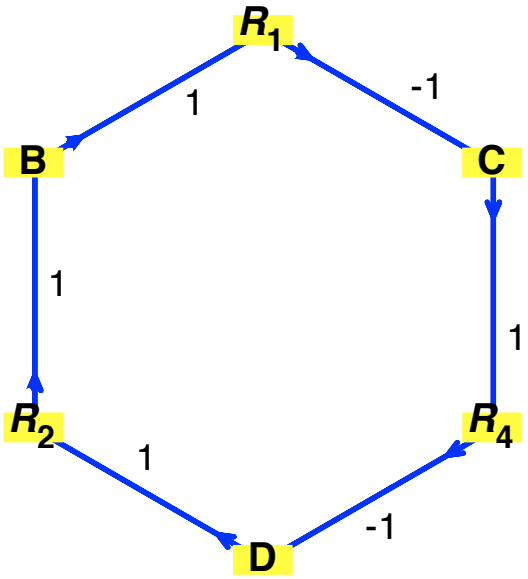
No. 55: injectiveEx3_3004.csv

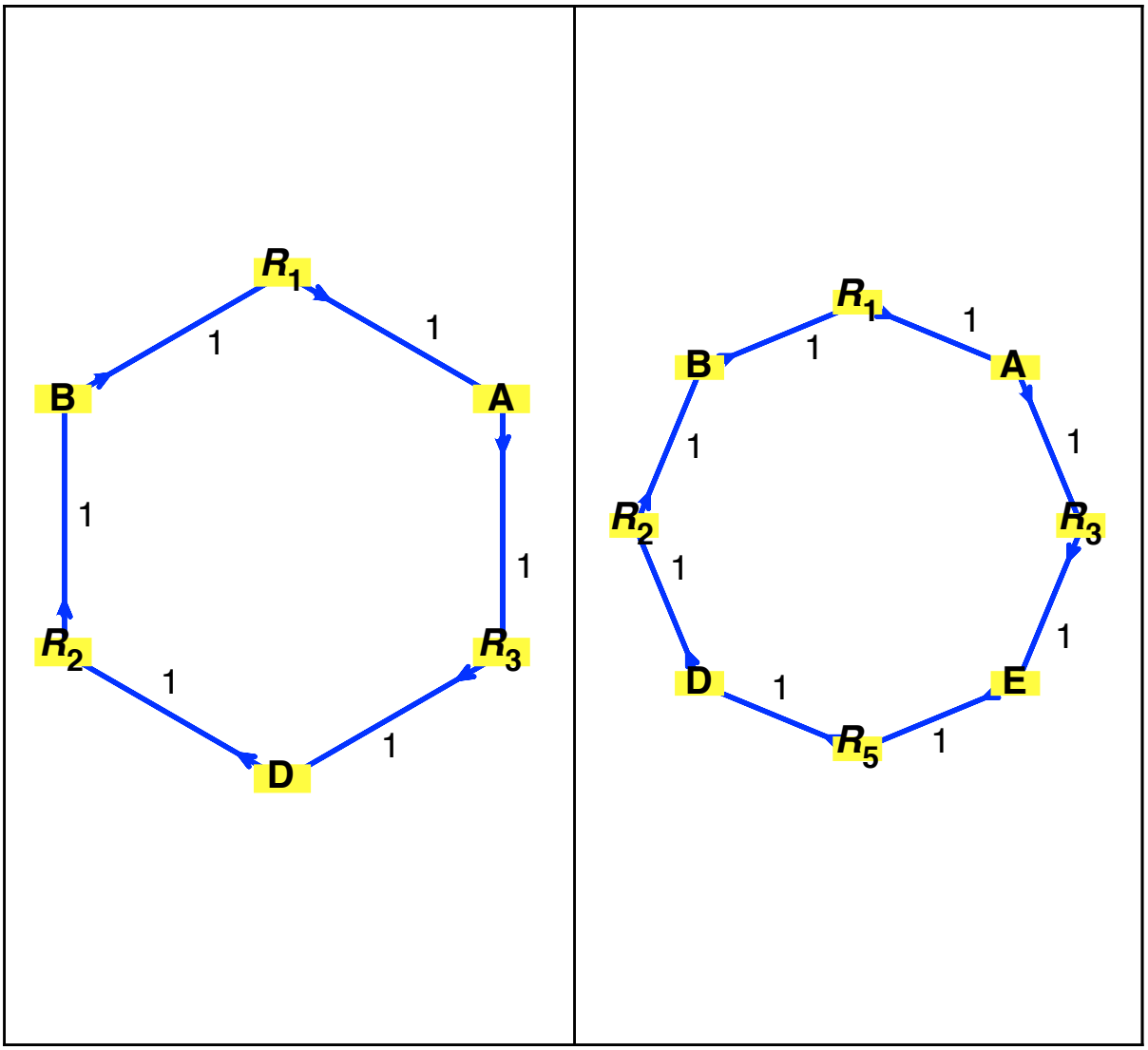




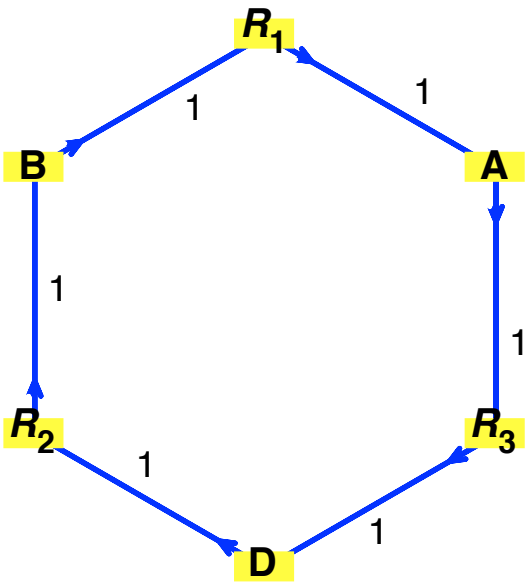
No. 56: injectiveEx3_3007.csv

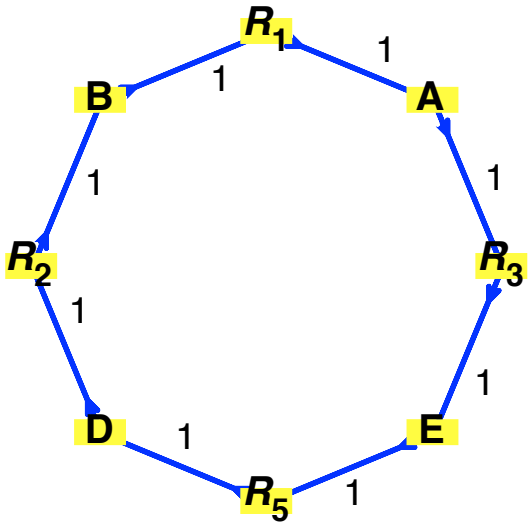

R2
1
-1
D
C
-1
1
R4

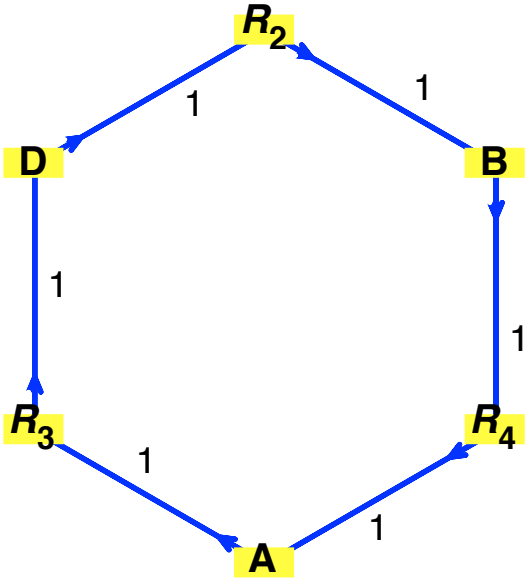

R1
1
-1
B
C
1
1
R2
R4
1
-1
D

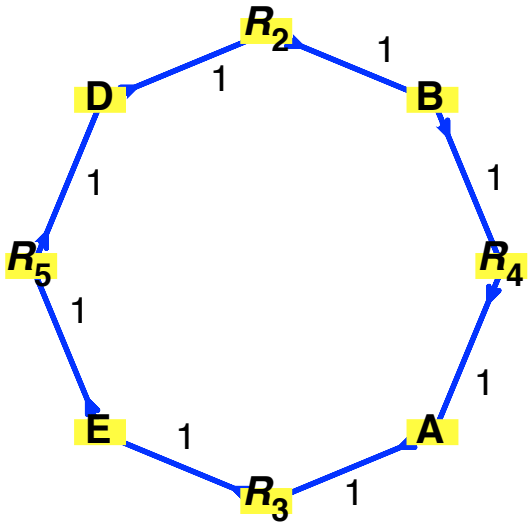


No. 57: injectiveEx3_3009.csv

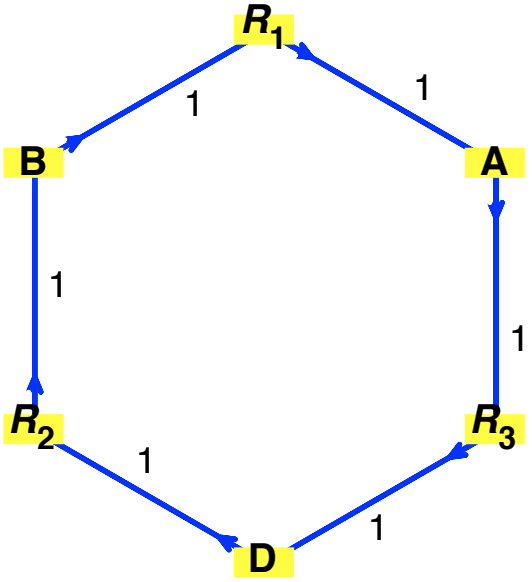

R1
1
1
B
A
1
1
R2
R3
1
1
D

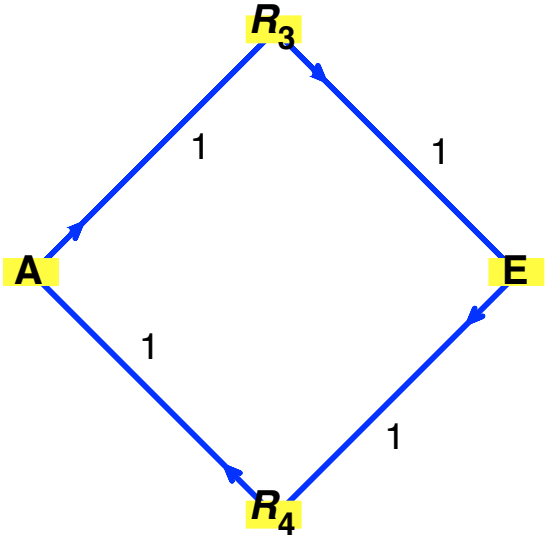

R1
1
1
B
A
1
1
R2
R3
1
1
D
E
1
R5
1

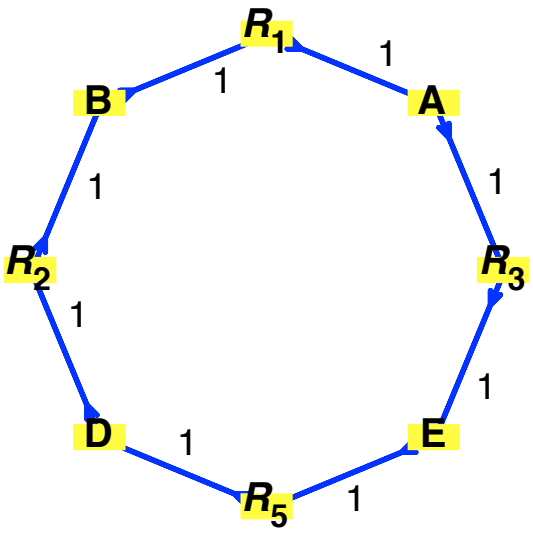




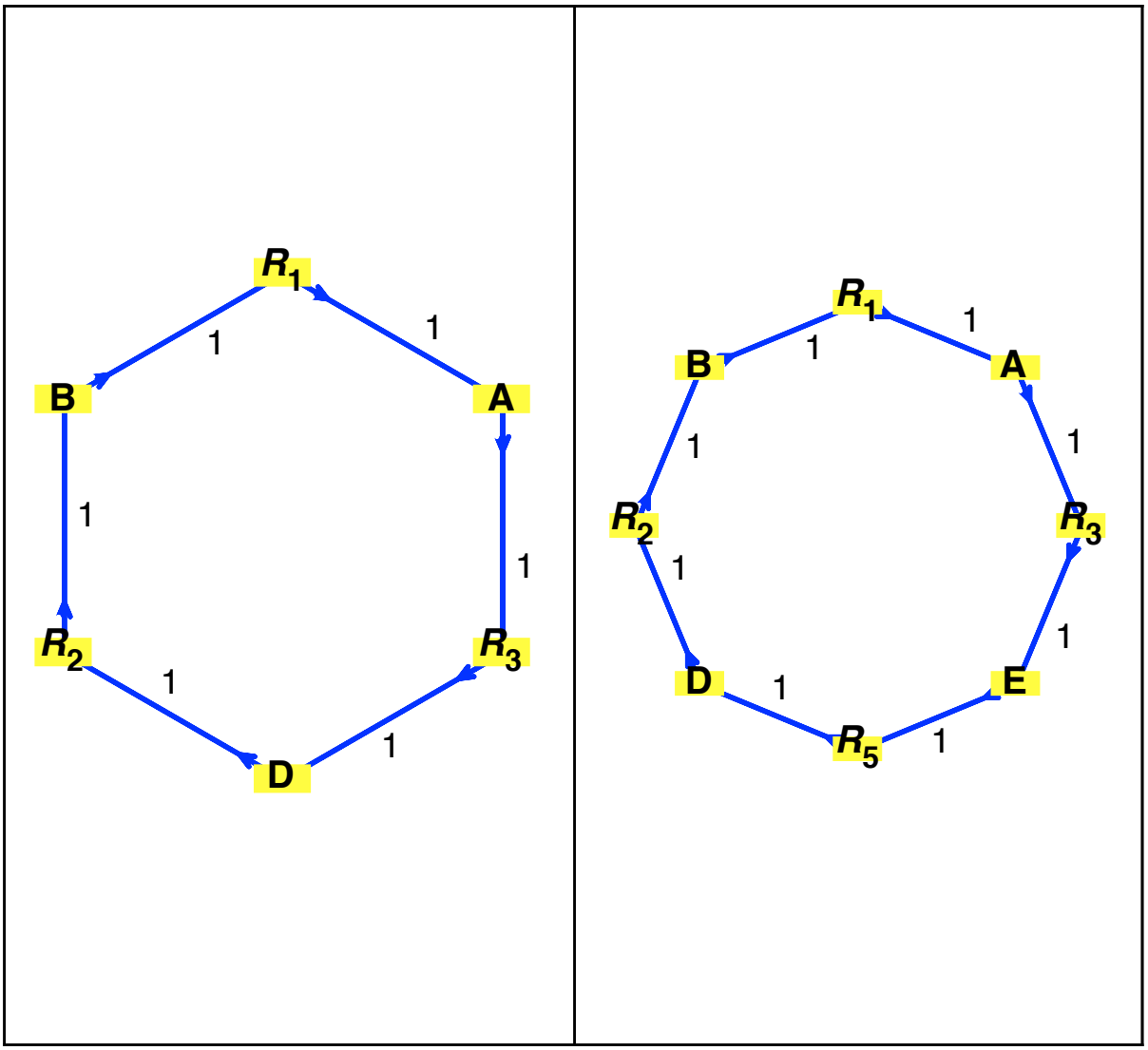
No. 58: injectiveEx3_3011.csv


R1
1
1
B
A
1
1
R2
R3
1
1
D

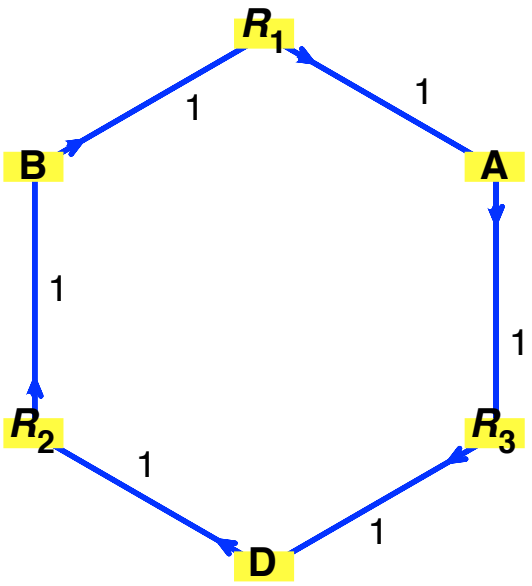

R3
1
1
A
E
1
1
R4

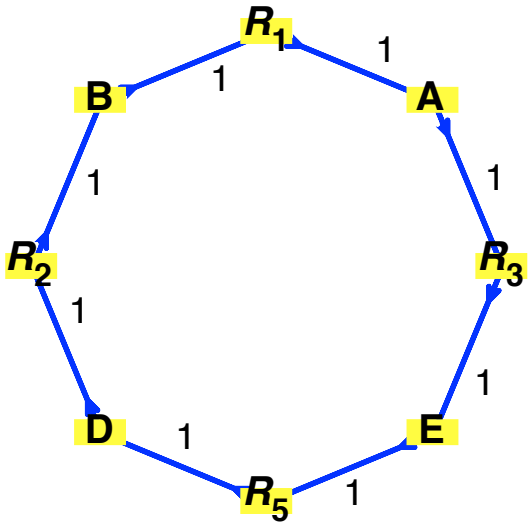


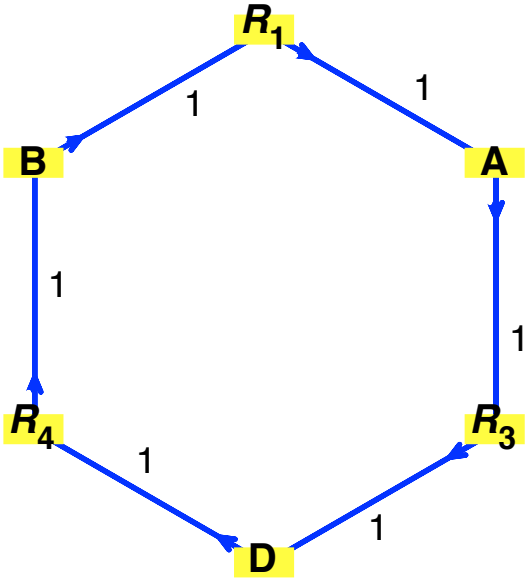
No. 59: injectiveEx3_3012.csv

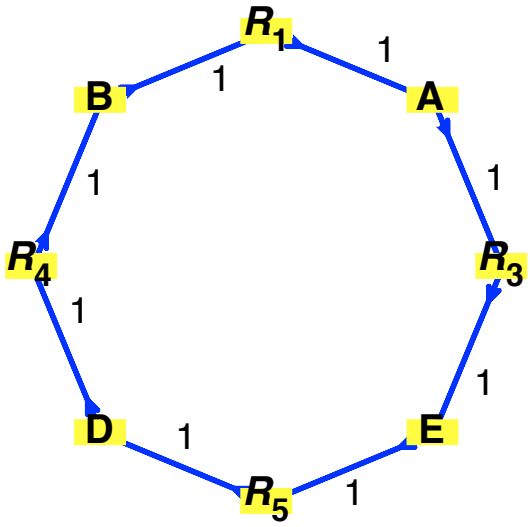


No. 60: injectiveEx3_3014.csv

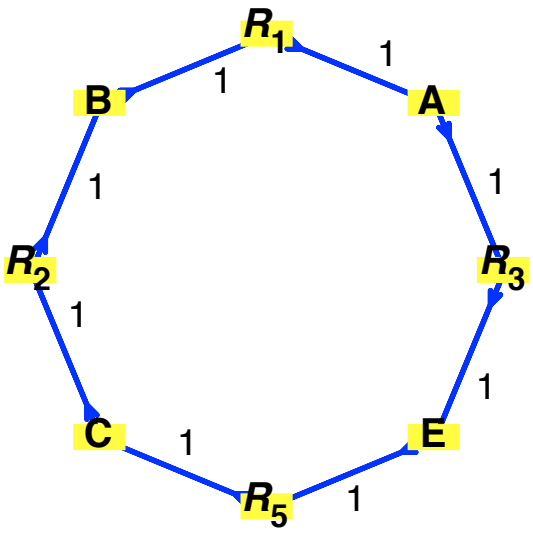

R1
1
1
B
A
1
1
R2
R3
1
1
D

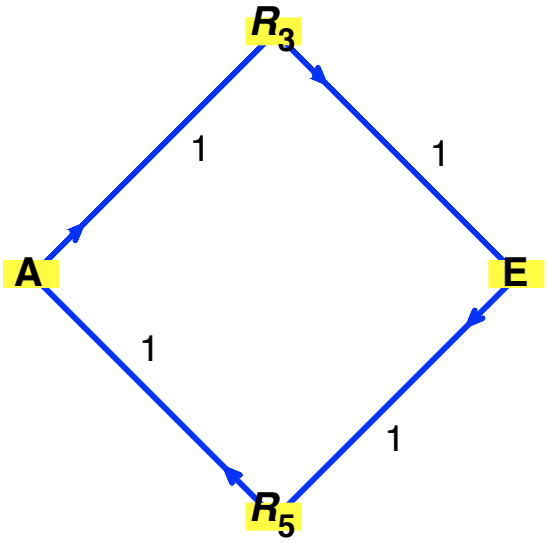

R1
1
1
B
A
1
1
R2
R3
1
1
D
E
1
R5
1

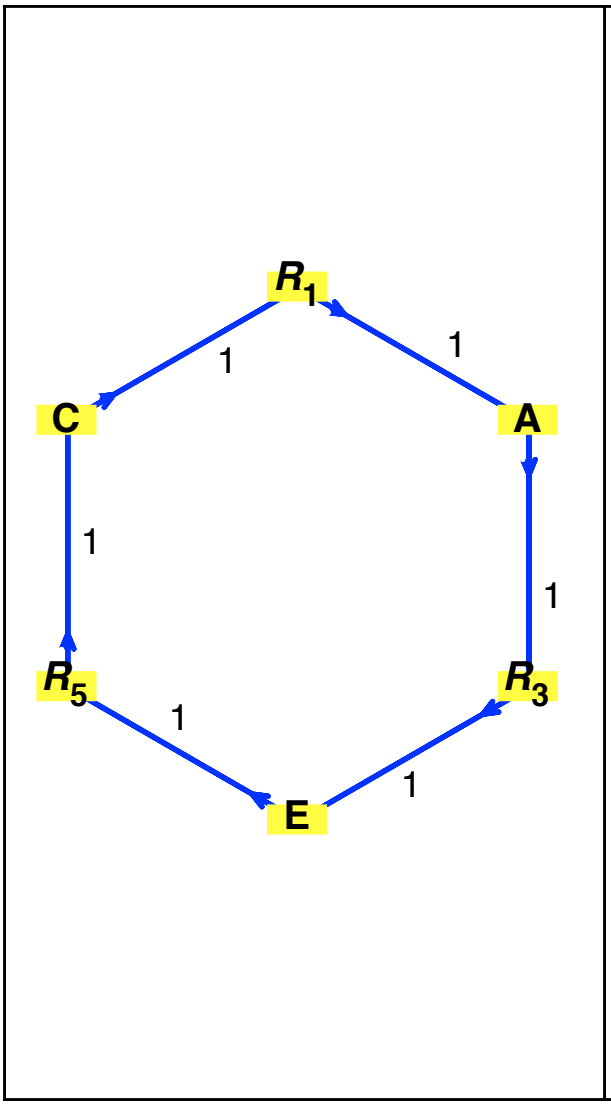


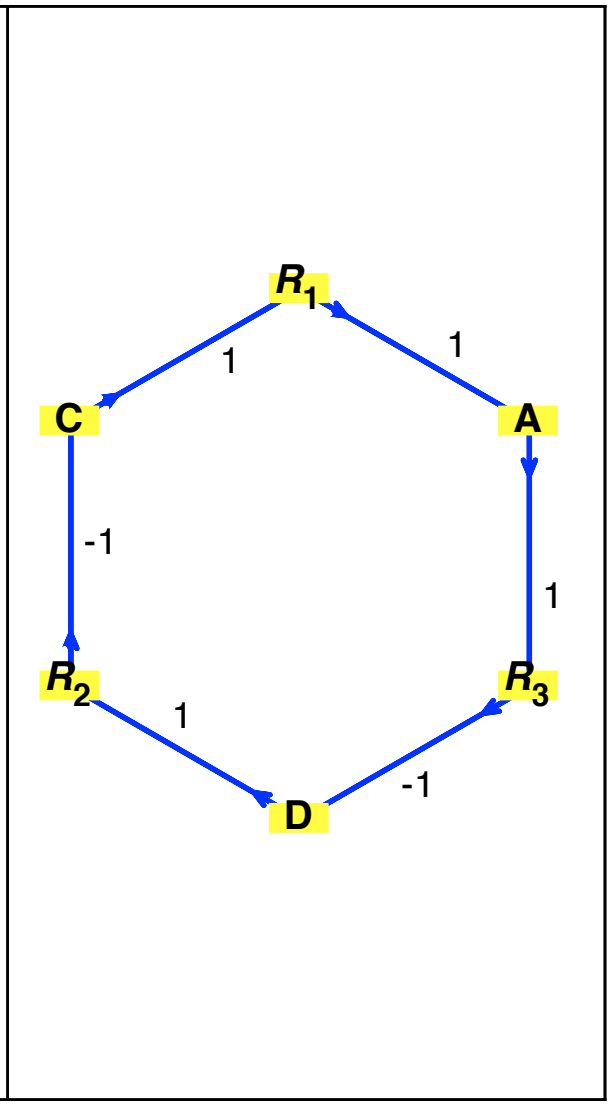


No. 61: injectiveEx3_3015.csv

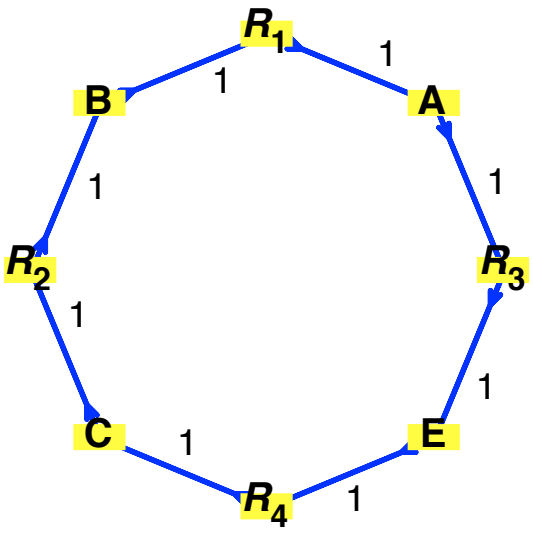

R_1
1
1
B
A
1
1
R_2
R_3
1
1
C
E
1
1
R_5

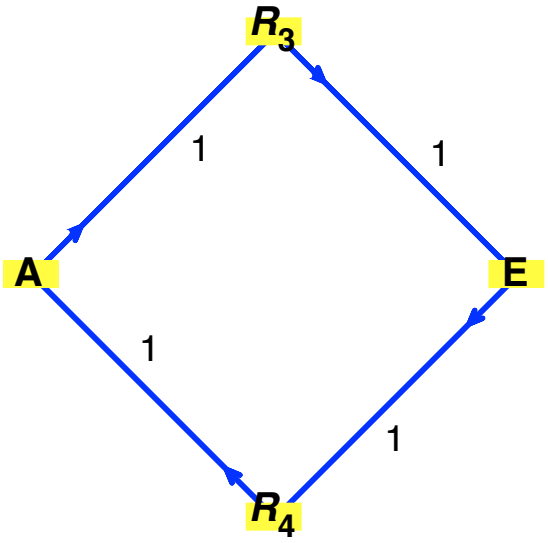

R_3
1
1
A
E
1
1
R_5

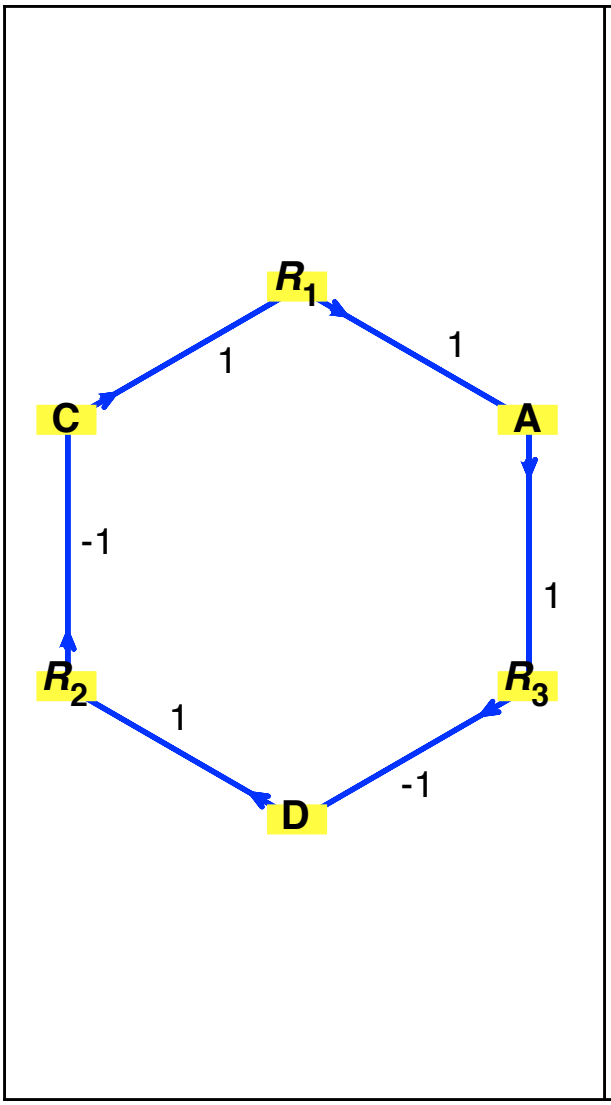


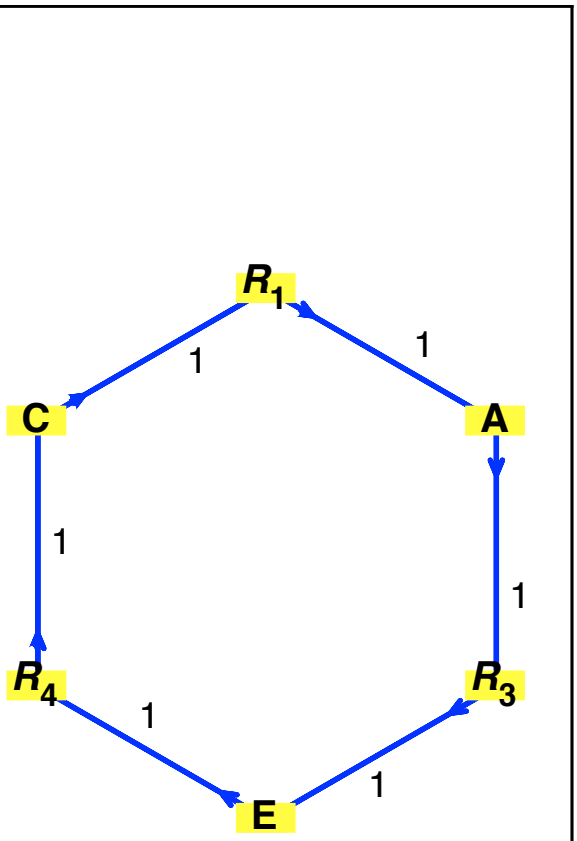


No. 62: injectiveEx3_3017.csv

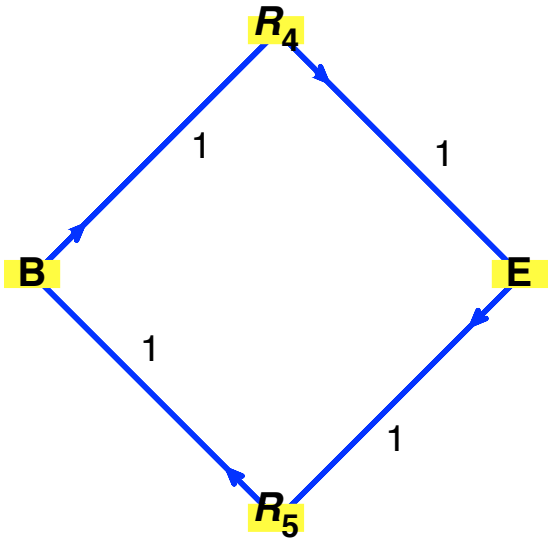

R1
1
1
B
A
1
1
R2
R3
1
1
C
1
E
R4
1

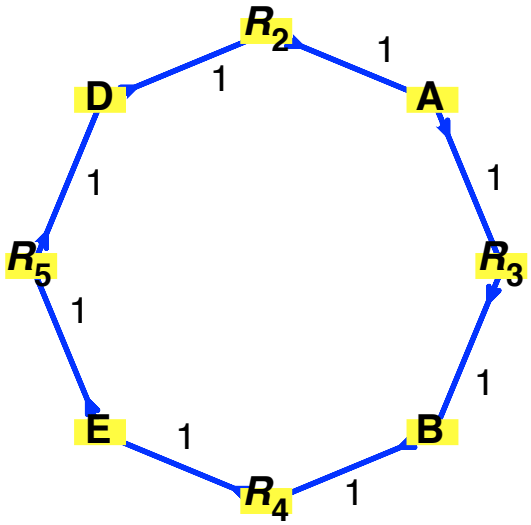

R3
1
1
A
E
1
1
R4

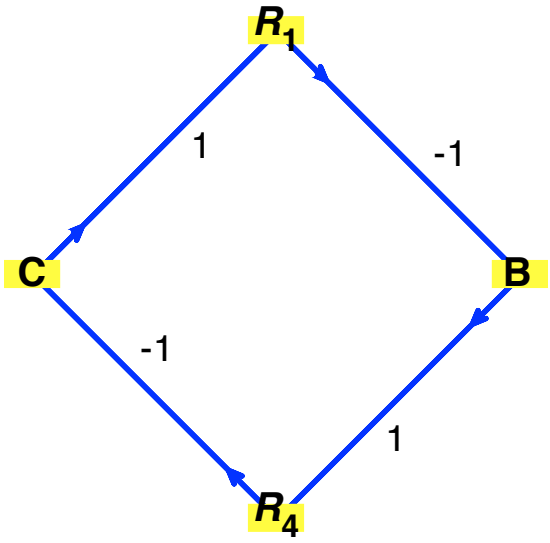


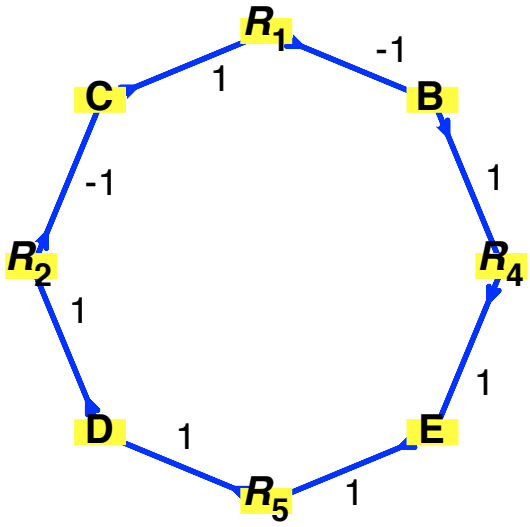


No. 63: injectiveEx3_3110.csv

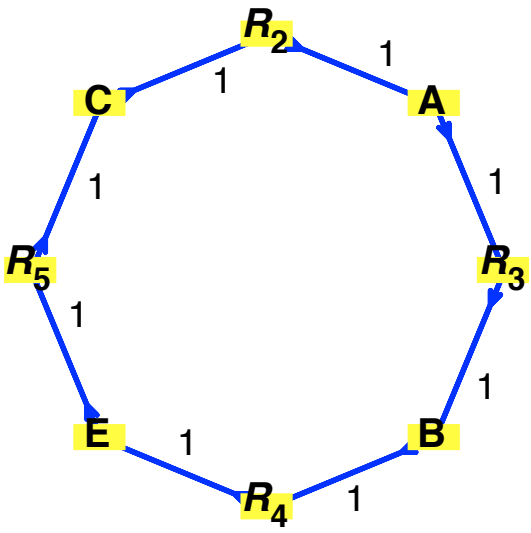

R_4
1
1
B
E
1
1
R_5

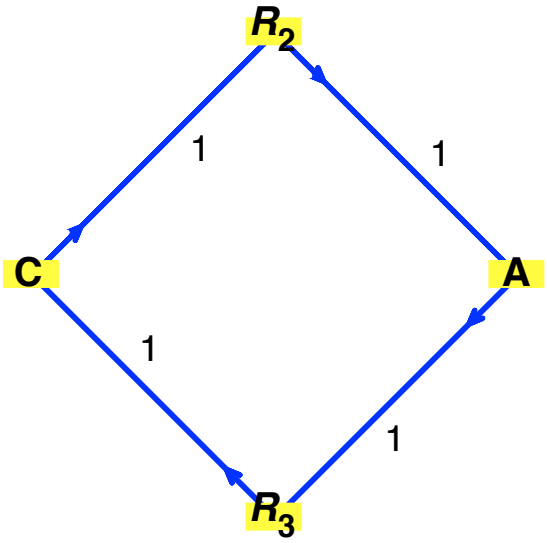

R_2
1
1
D
A
1
1
R_5
R_3
1
1
E
1
B
R_4
1



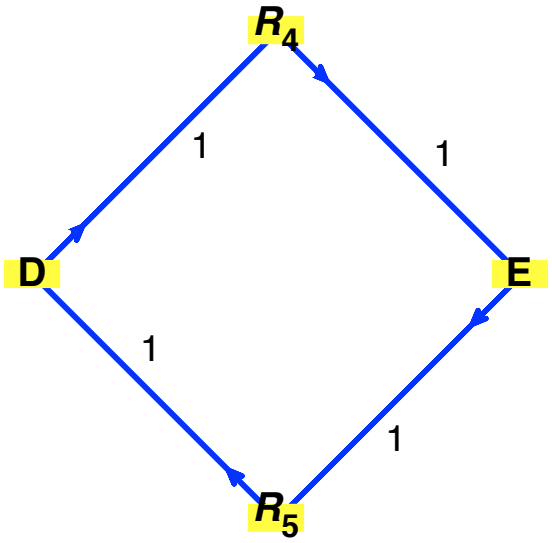


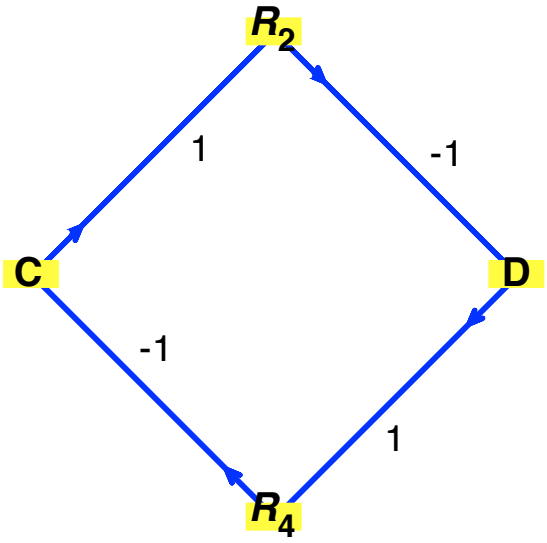
No. 64: injectiveEx3_3114.csv

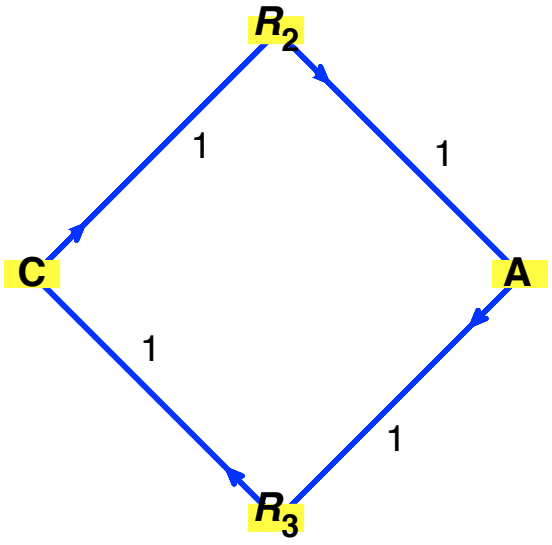


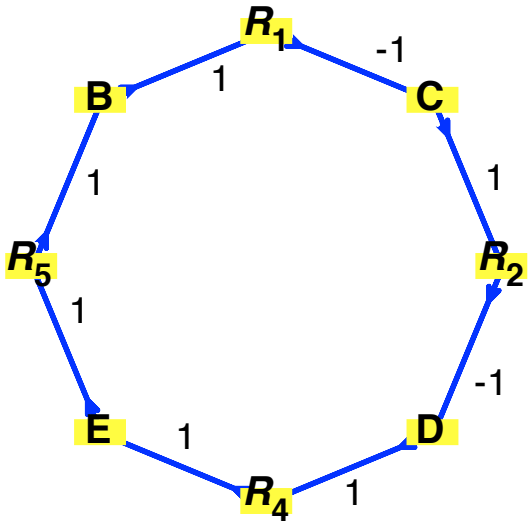


No. 65: injectiveEx3_3118.csv

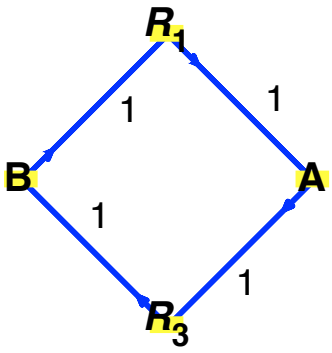

R_4
1
1
D
E
1
1
R_5

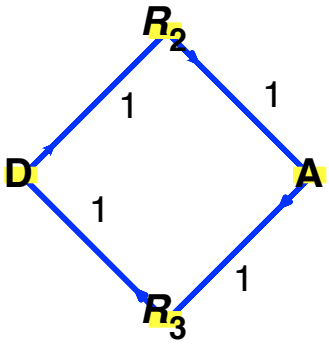

R_2
1
-1
C
D
-1
1
R_4

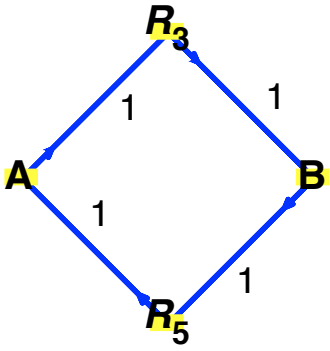


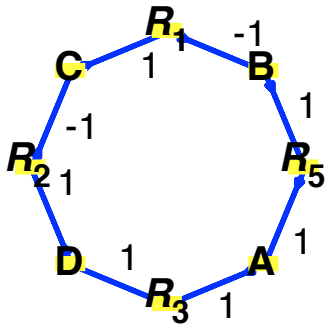


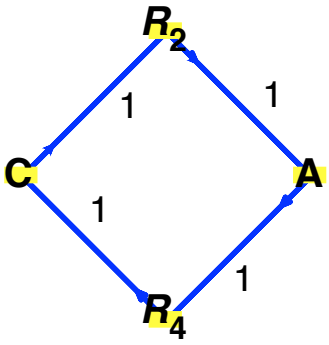
No. 66: injectiveEx3_3123.csv

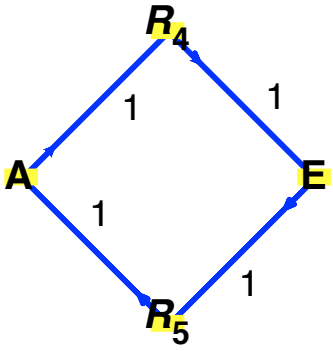

R1
1
1
B
A
1
1
R3

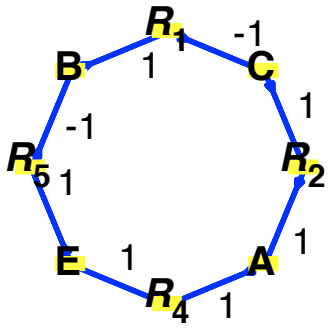

R2
1
1
D
A
1
1
R3

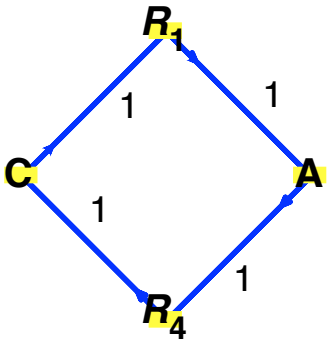

R3
1
1
A
B
1
1
R5


R1
-1
C
1
B
1
-1
R2
1
R5
D
1
1
A
R3
1

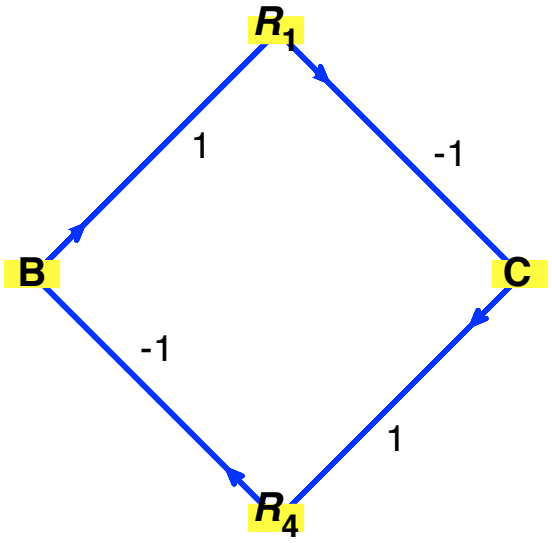

R2
1
1
C
A
1
1
R4

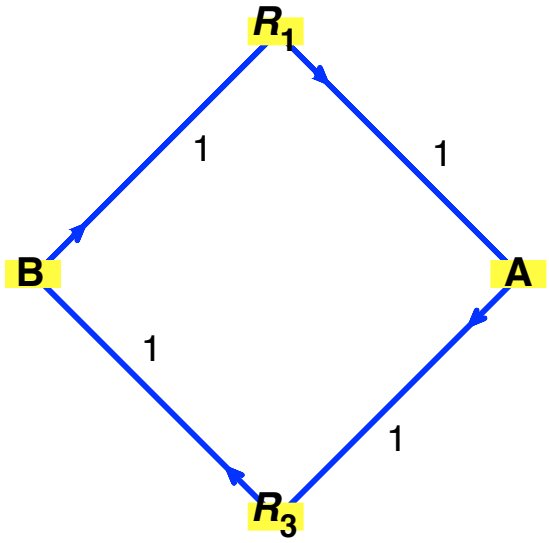

R4
1
1
A
E
1
1
R5

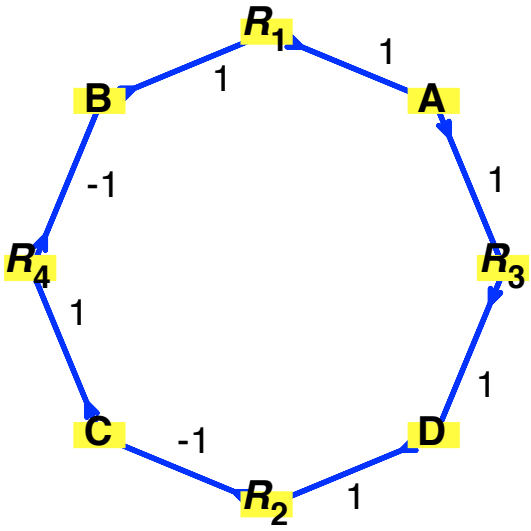


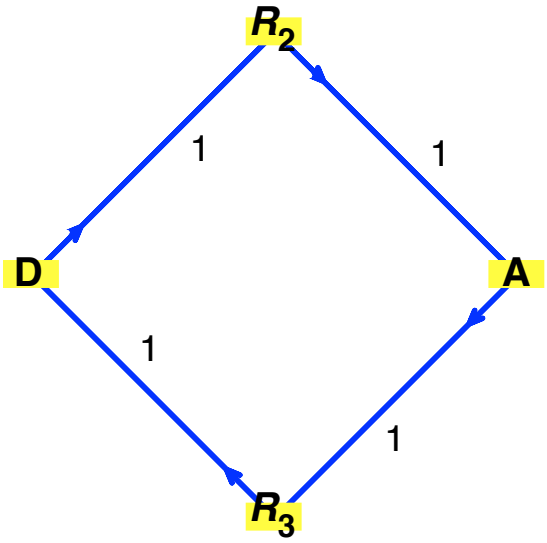


No. 67: injectiveEx3_3128.csv

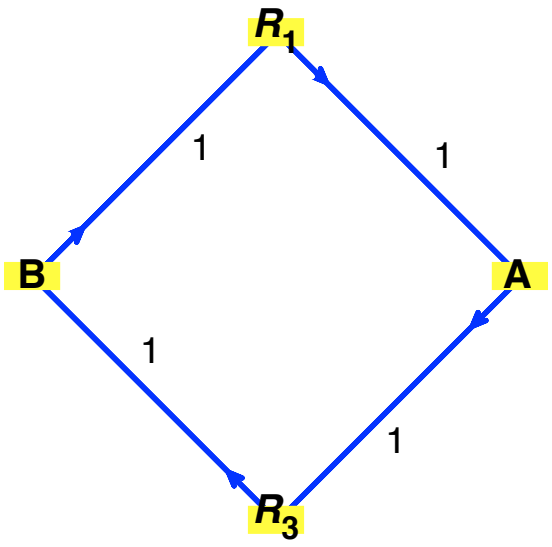

R_1
1
-1
B
C
-1
1
R_4

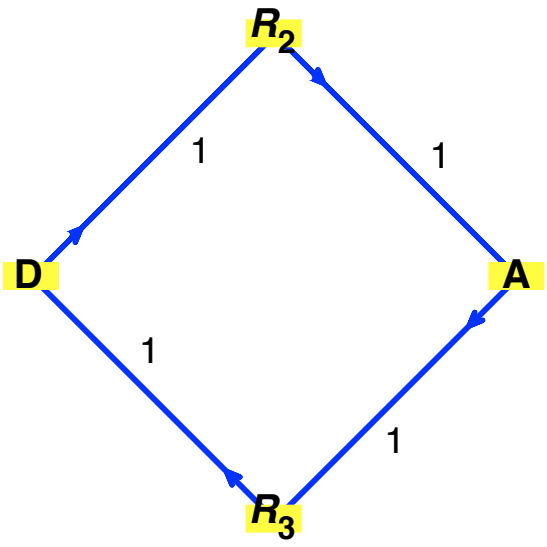

R_1
1
1
B
A
1
1
R_3

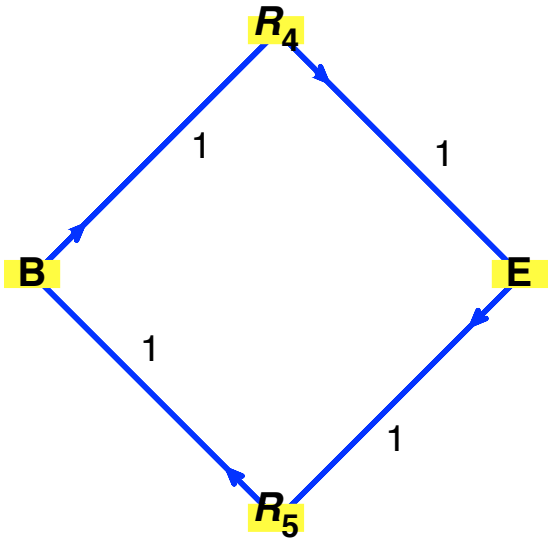


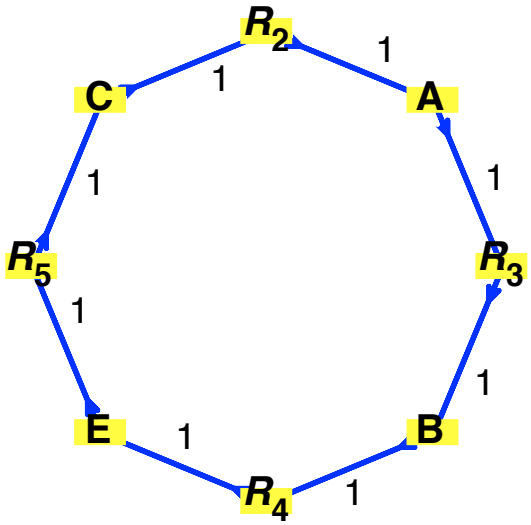


No. 68: injectiveEx3_3130.csv

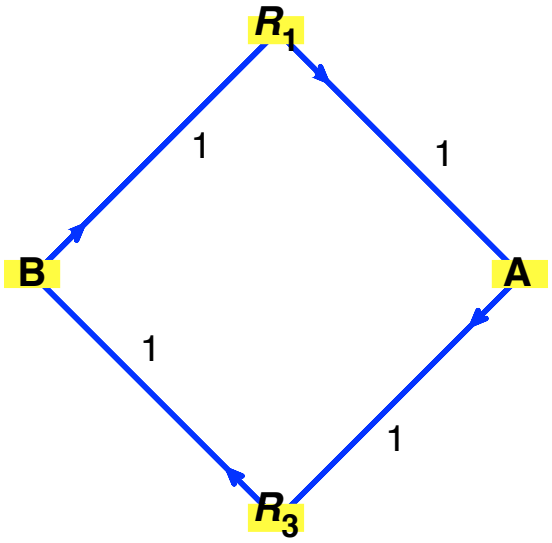

R_1
1
1
B
A
1
1
R_3

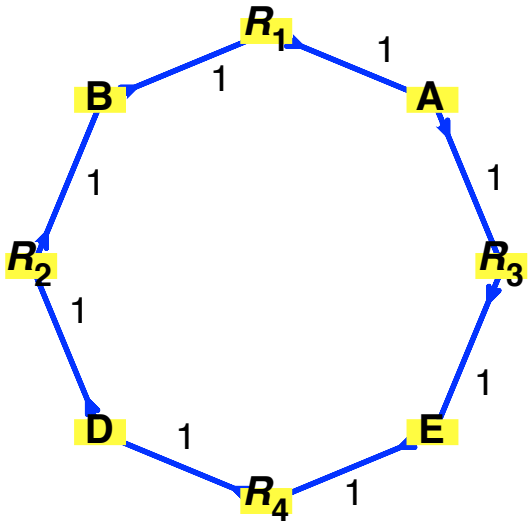

R_2
1
1
D
A
1
1
R_3

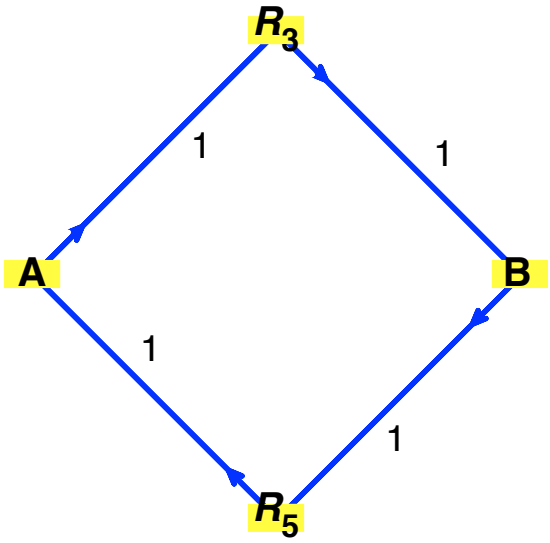


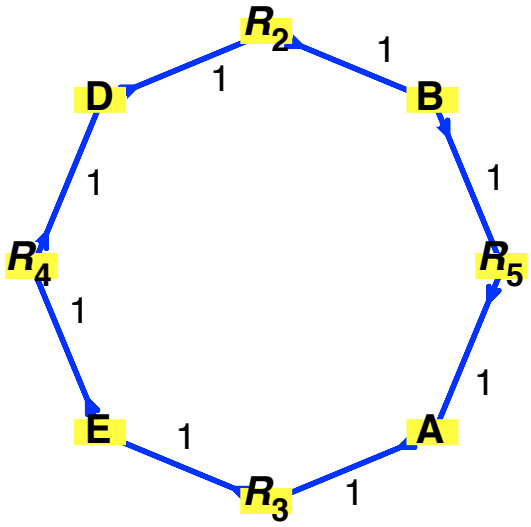


No. 69: injectiveEx3_3553.csv

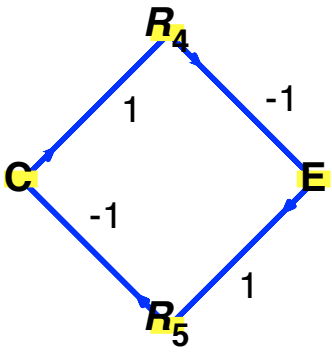

R1
1
1
B
A
1
1
R3

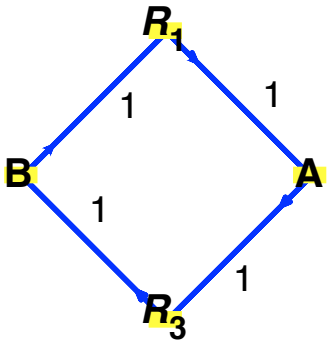

R1
1
1
B
A
1
1
R2
R3
1
1
D
1
E
R4
1

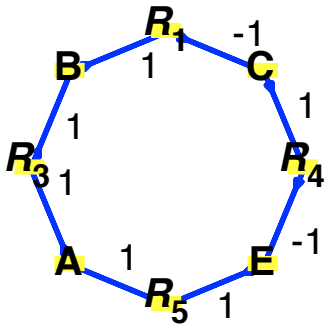


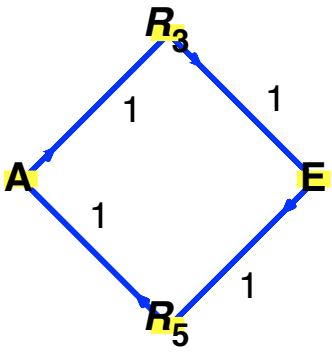


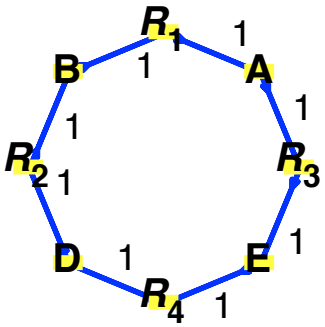
No. 70: injectiveEx3_3555.csv

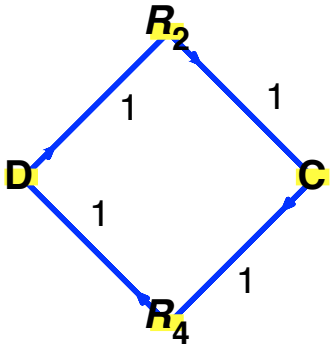

R4
1
-1
C
E
-1
1
R5


R1
1
1
B
A
1
1
R3

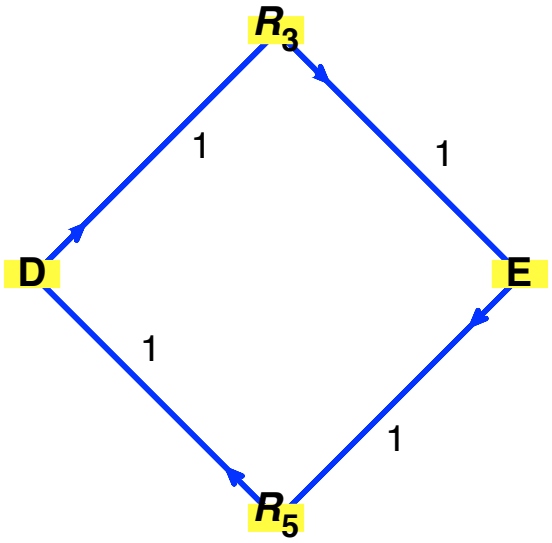

R1
-1
B
1
C
1
1
R3
1
R4
-1
A
1
E
R5
1

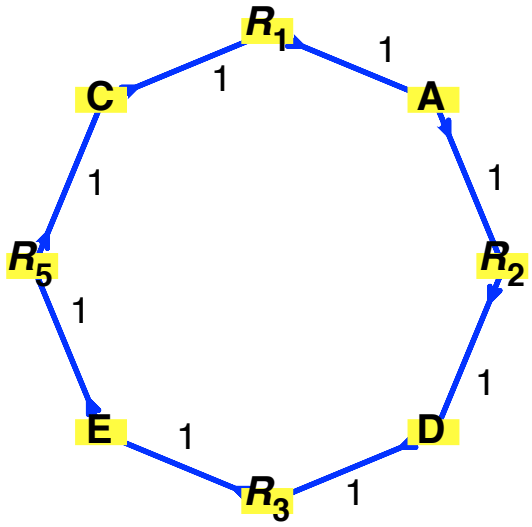


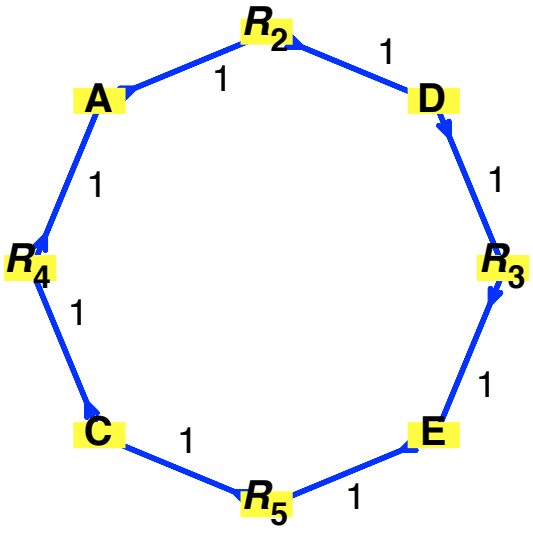




No. 71: injectiveEx3_4324.csv


R_3
1
1
D
E
1
1
R_5


R_1
1
1
C
A
1
1
R_5
R_2
1
1
E
D
1
R_3
1



>

>

Multistationary: with competition loop, with intersecting loops

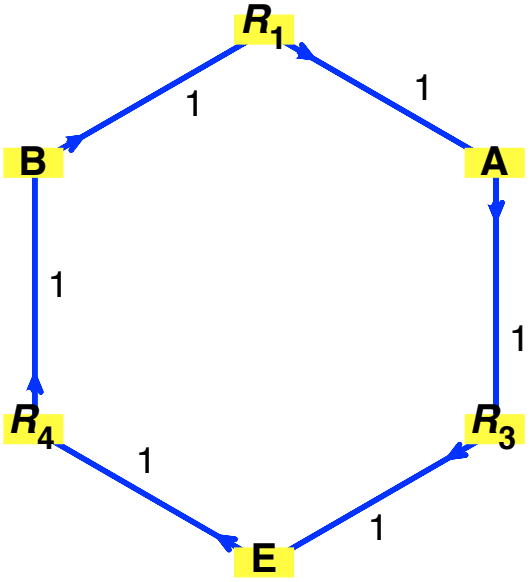
> *uniquefolder* := "5species/multistationary/unique_competitionloop_intersectingloops" :

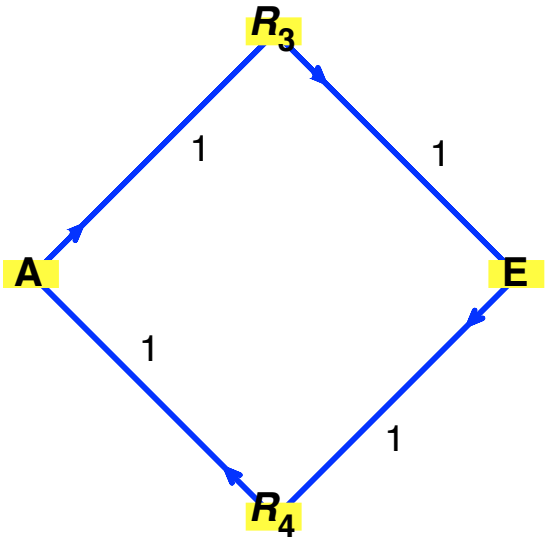
> *uniquefiles* := *ListDirectory*(*uniquefolder*) :

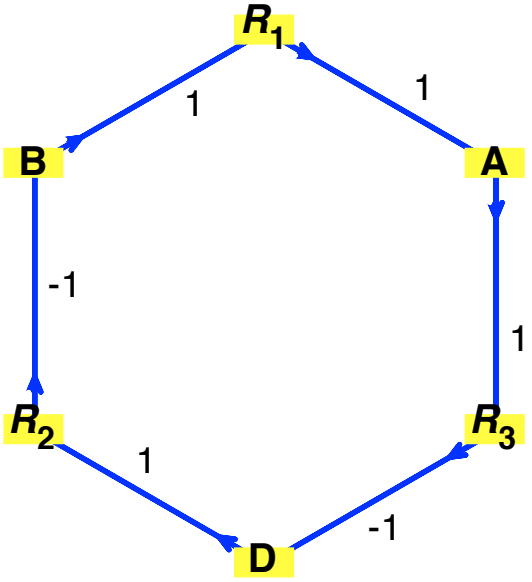
> *m* := *nops*(*uniquefiles*) :

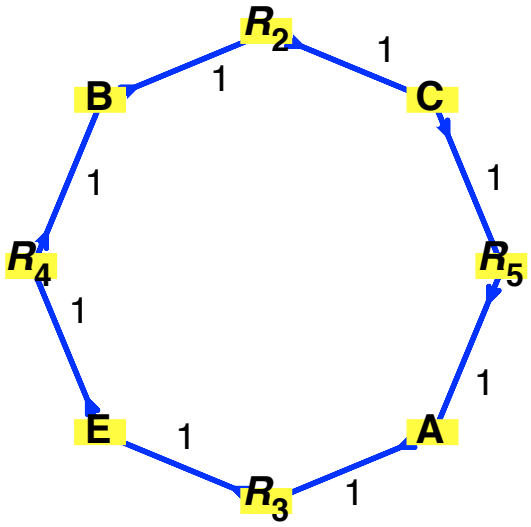
> **for** *i* **from** 1 **to** *m* **by** 1 **do** *uni* := *op*(*i*, *uniquefiles*) : *A* := *ImportMatrix*(*cat*(*uniquefolder*, "/", *uni*)) : *Z* := *findZ*(*A*) : *selected* := *findloops*(*A*, *Z*) : *printf*("No. %d: %s \n", *i*, *uni*); *print*(*drawloops*(*selected*, *speciesord*)); **end do**:

```
No. 1: injectiveEx0_1517.csv
```

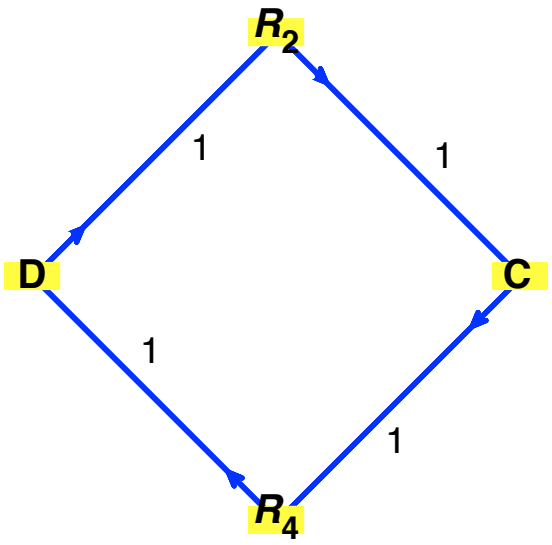

R1
1
1
B
A
1
1
R4
R3
1
1
E

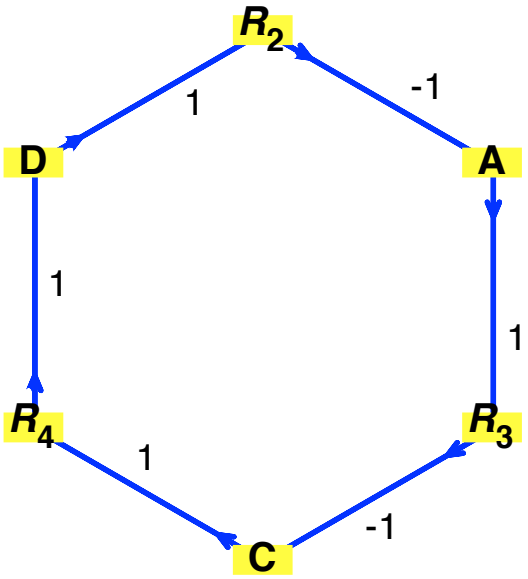

R3
1
1
A
E
1
1
R4



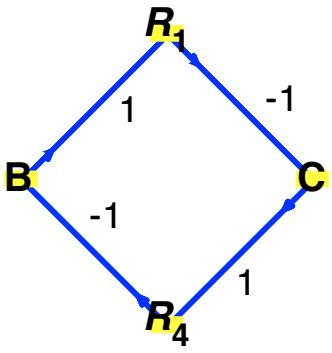


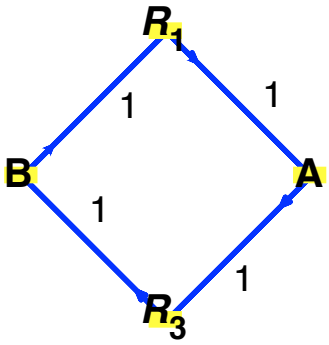
No. 2: injectiveEx0_1742.csv

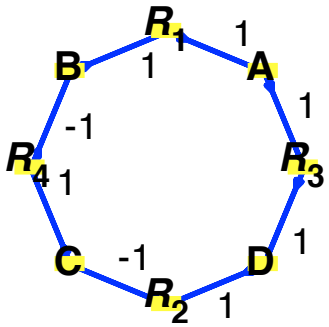


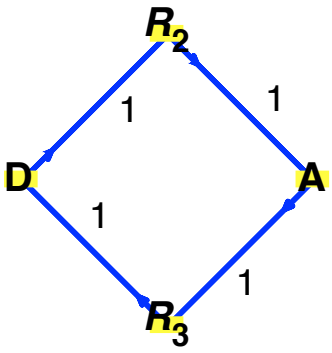


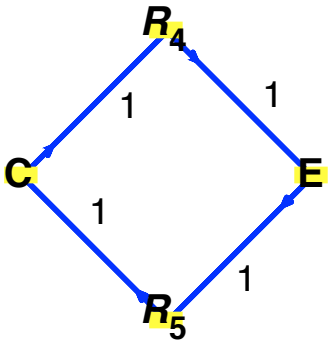
No. 3: injectiveEx0_3129.csv

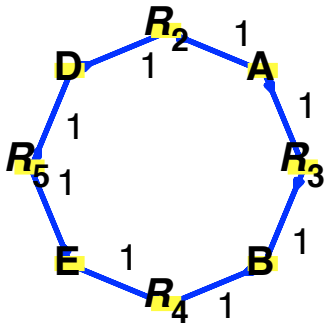

R1
1
-1
B
C
-1
1
R4


R1
1
1
B
A
1
R3

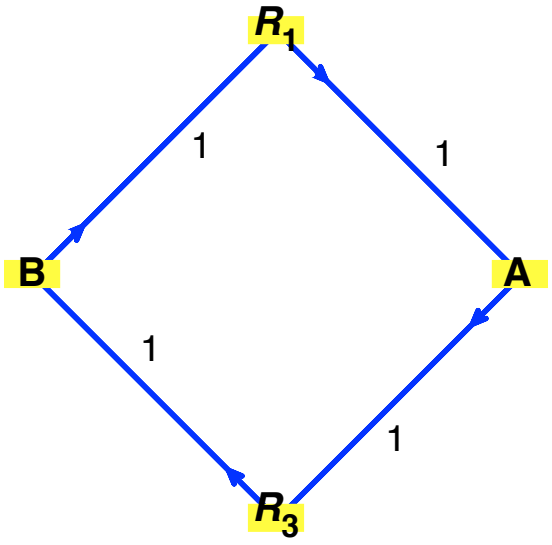

R1
1
1
B
A
1
-1
R4
1
R3
1
C
-1
D
R2
1

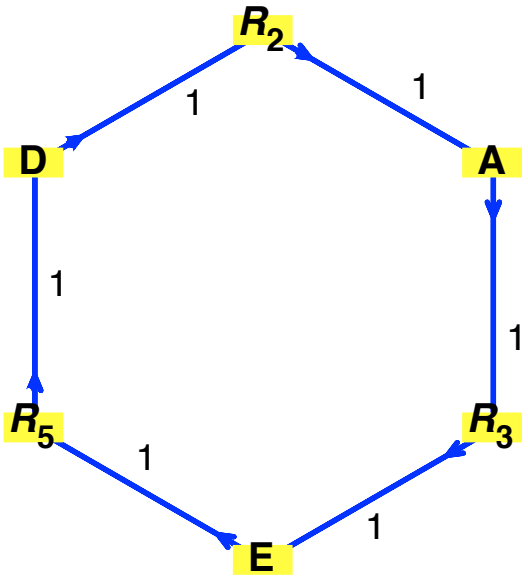


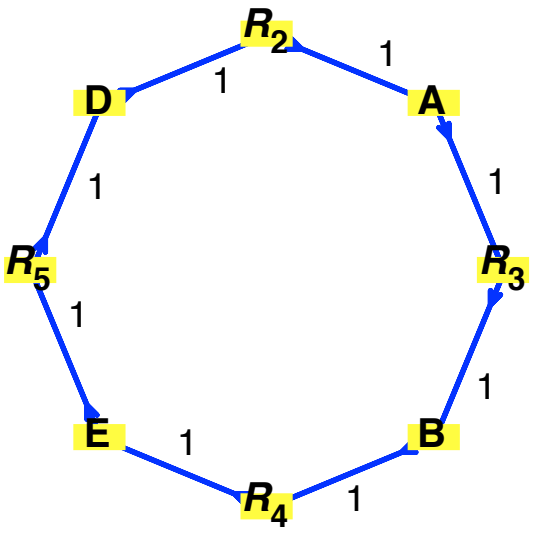




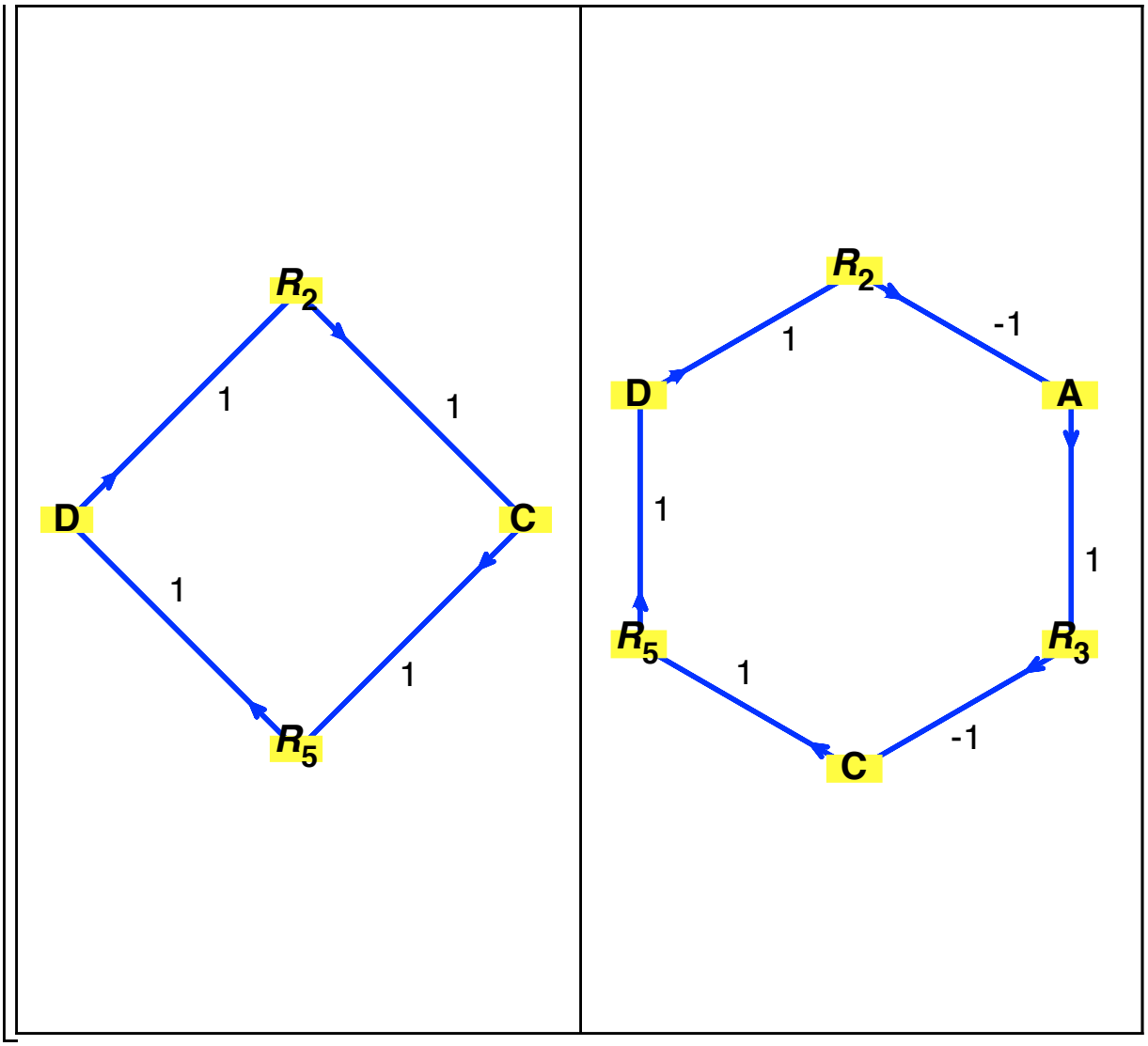
No. 4: injectiveEx0_3141.csv


R1
1
1
B
A
1
1
R3


R2
1
1
D
A
1
1
R5
R3
1
1
E

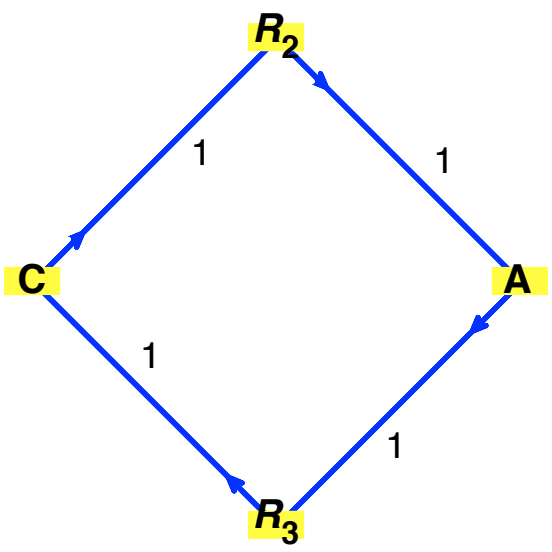


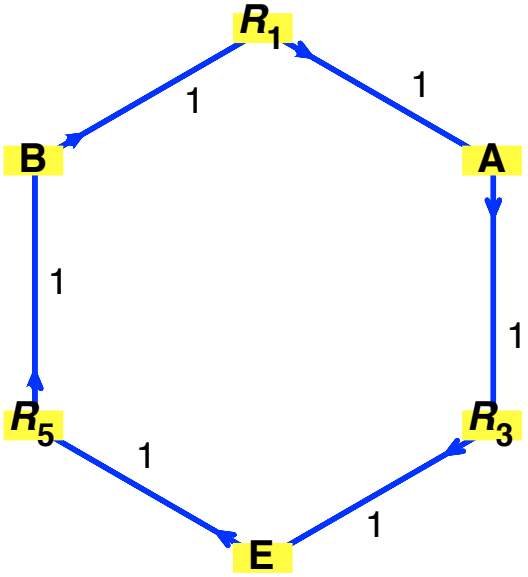
No. 5: injectiveEx2_1741.csv

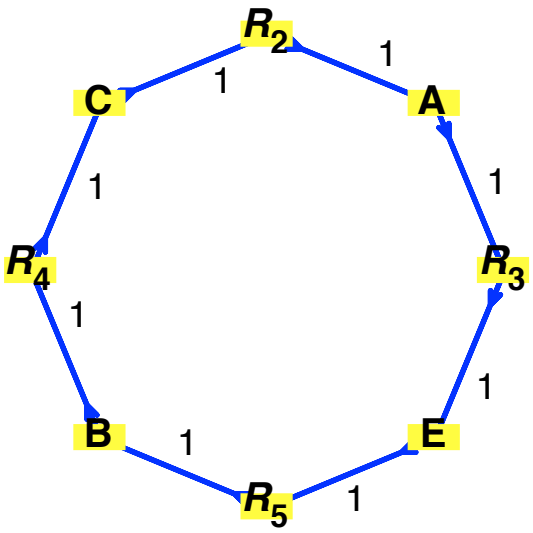


Multistationary: no competition loop, with intersecting loops

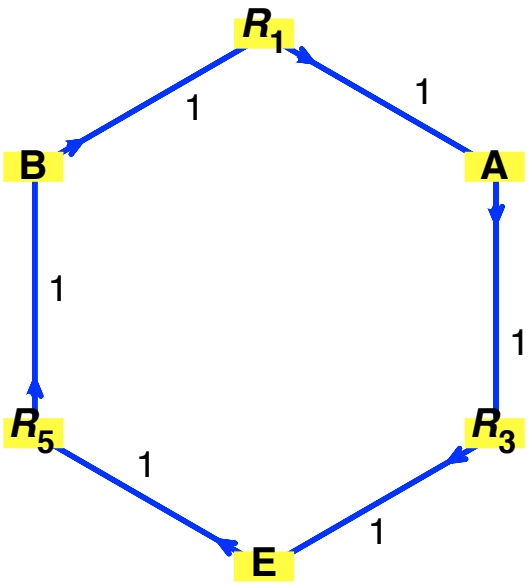
```
> uniquefolder := "5species/multistationary/unique_nocompetitionloop_intersectingloops" :
> uniquefiles := ListDirectory(uniquefolder) :
> m := nops(uniquefiles) :
> for i from 1 to m by 1 do uni := op(i, uniquefiles) : A := ImportMatrix(cat(uniquefolder,
      "/", uni)) : Z := findZ(A) : selected := findloops(A, Z) : printf("No. %d: %s \n", i,
      uni); print(drawloops(selected, speciesord)); end do:
No. 1: injectiveEx0_1387.csv
```

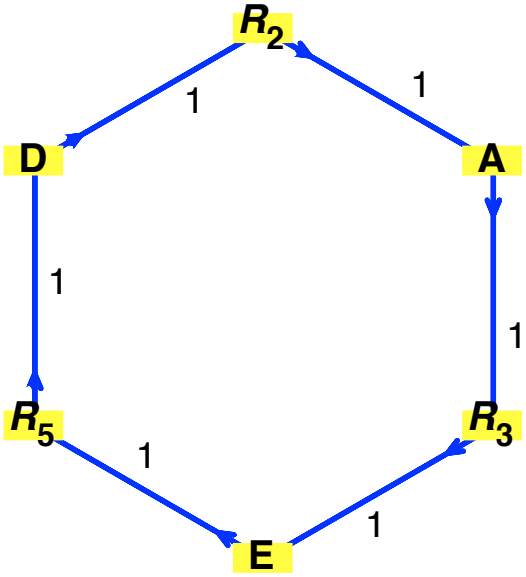

R2
1
1
C
A
1
1
R3

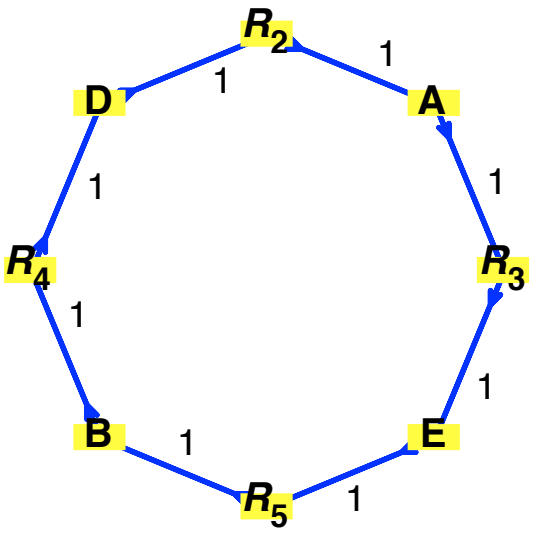

R1
1
1
B
A
1
1
R5
R3
1
1
E



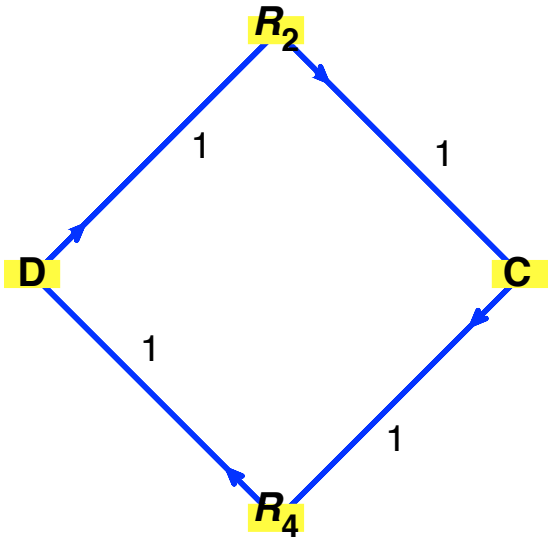
No. 2: injectiveEx0_1392.csv

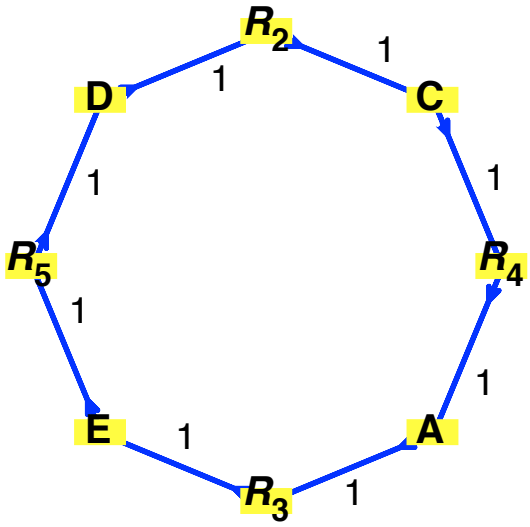

R1
1
1
B
A
1
1
R5
R3
1
1
E


R2
1
1
D
A
1
1
R5
R3
1
1
E

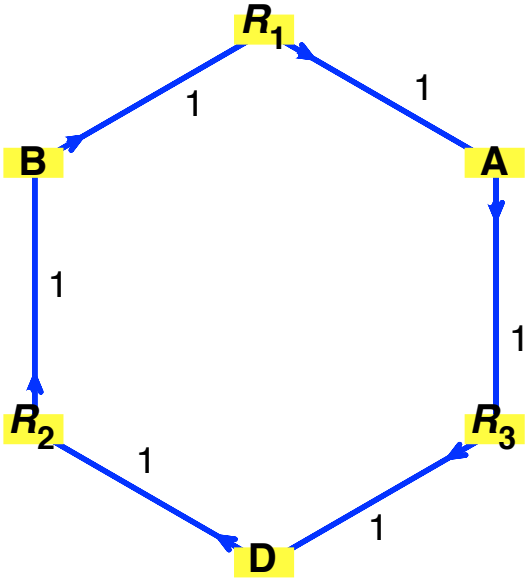


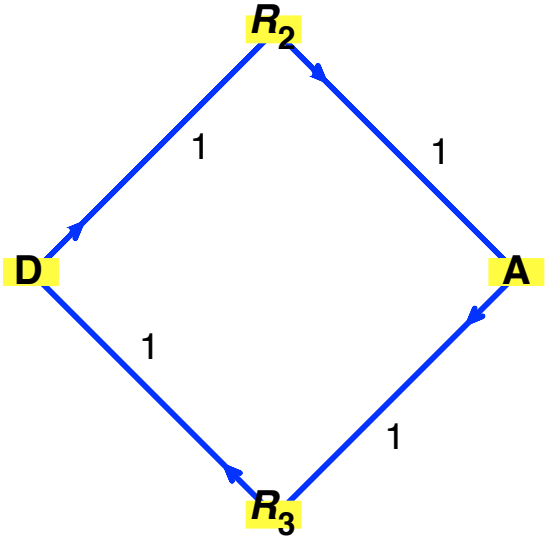
No. 3: injectiveEx0_1487.csv

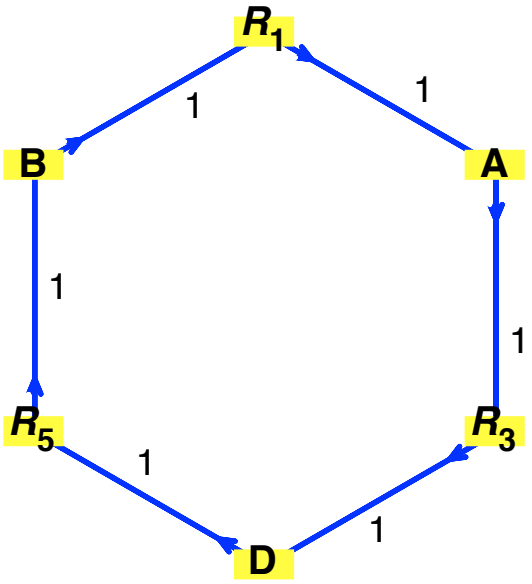


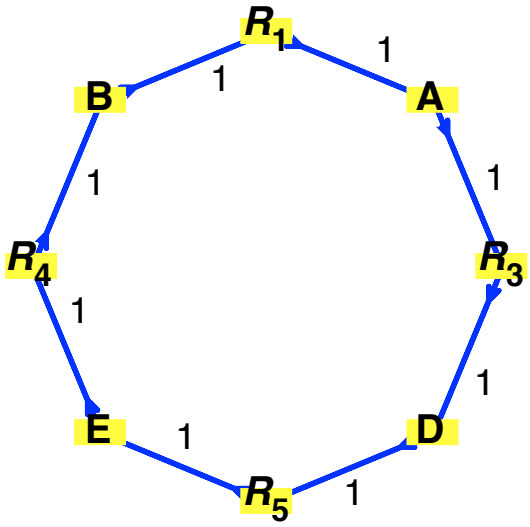


No. 4: injectiveEx0_3733.csv

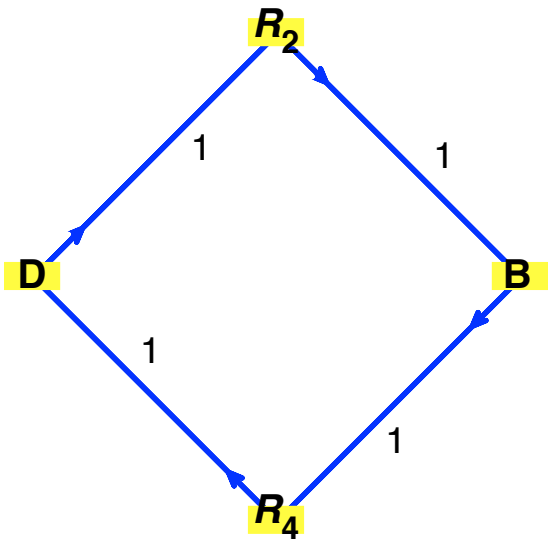

R1
1
1
B
A
1
1
R2
R3
1
1
D

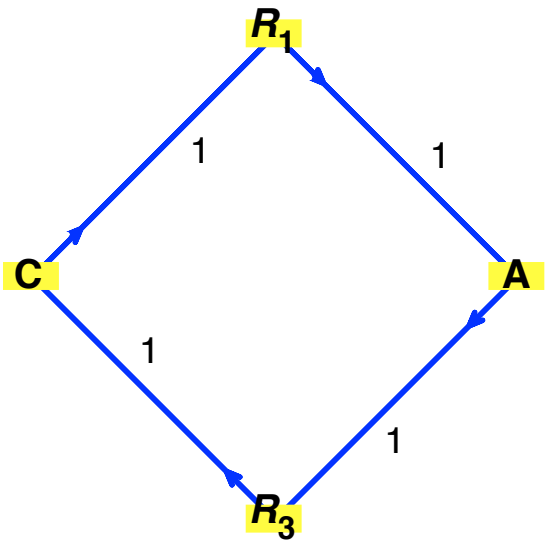

R2
1
1
D
A
1
1
R3

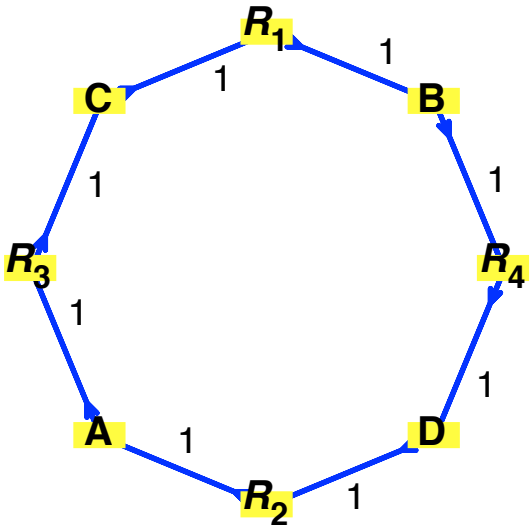


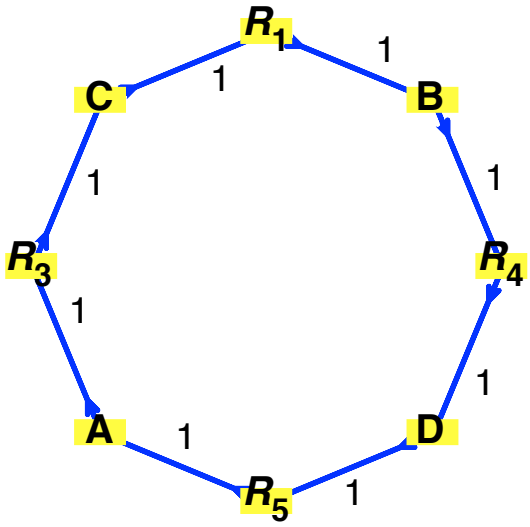


No. 5: injectiveEx0_5631.csv

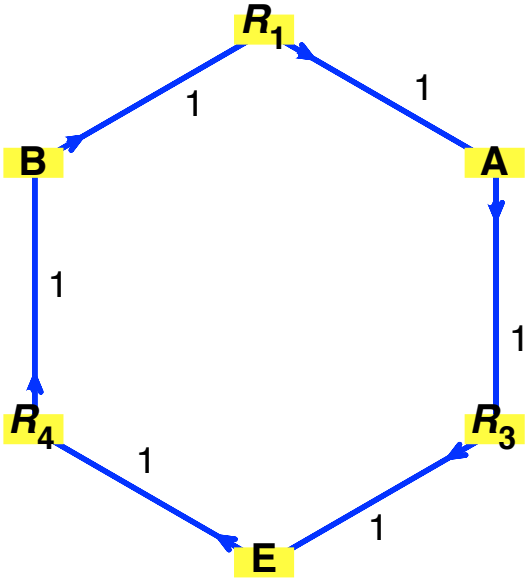

R_2
1
1
D
B
1
1
R_4

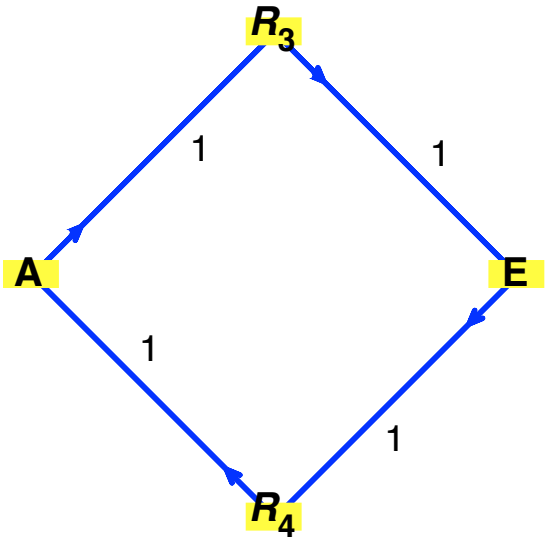

R_1
1
1
C
A
1
1
R_3

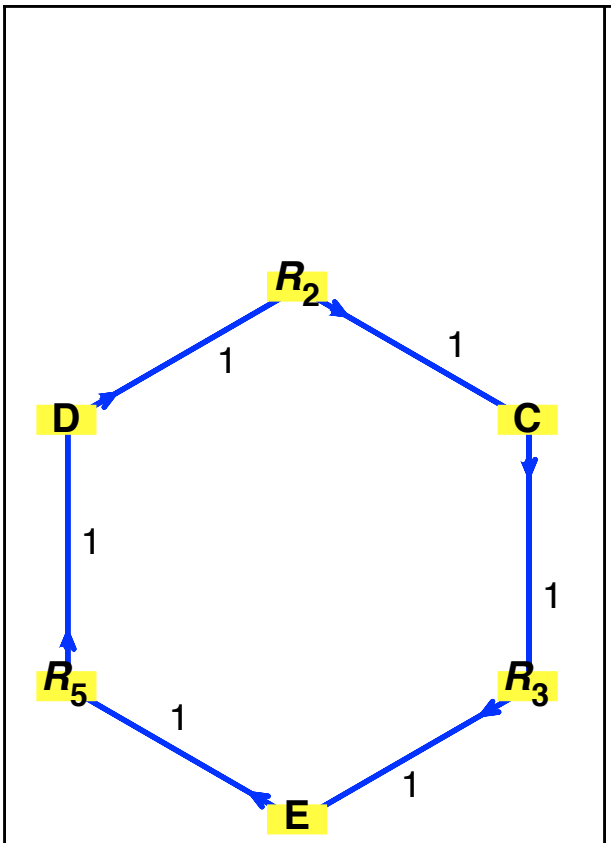


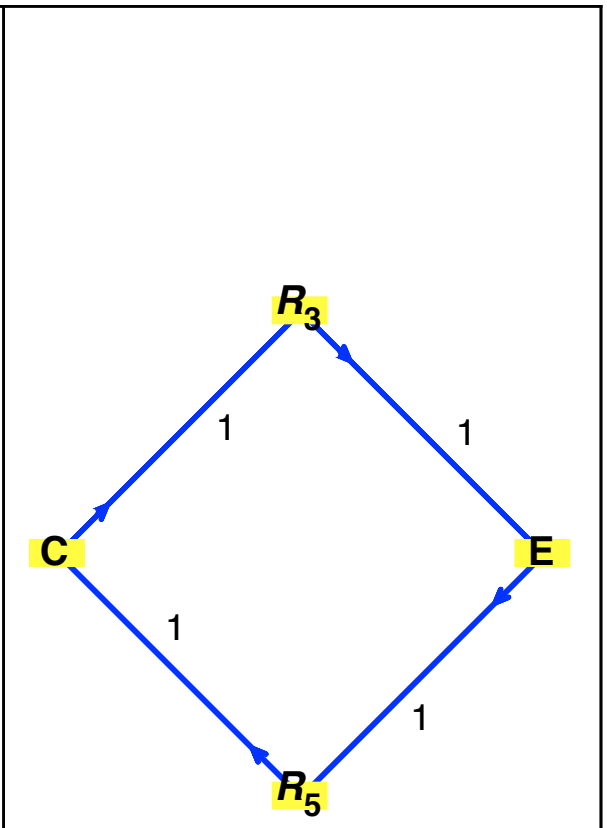


No. 6: injectiveEx2_146.cvs.csv

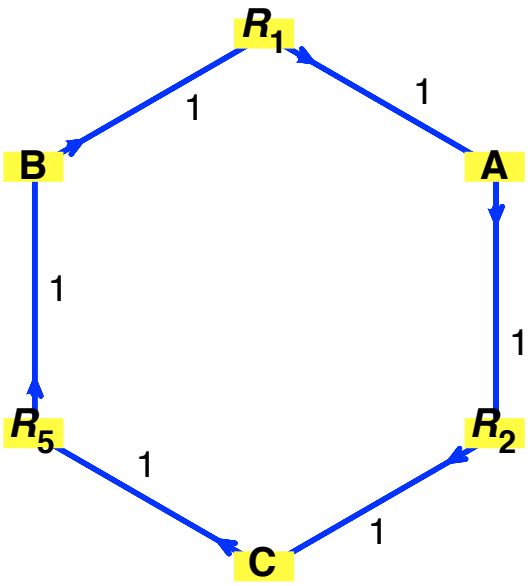

R1
1
1
B
A
1
1
R4
R3
1
1
E

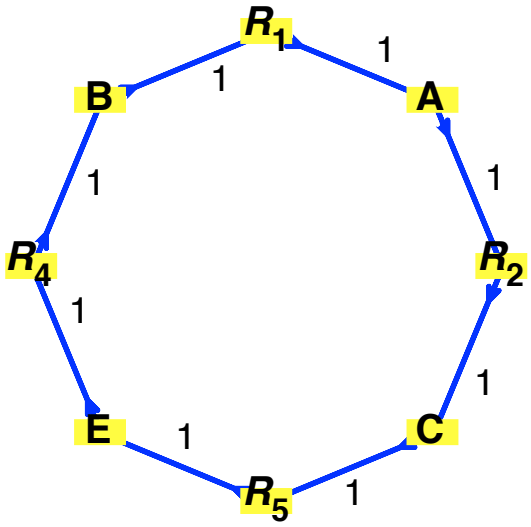

R3
1
1
A
E
1
1
R4

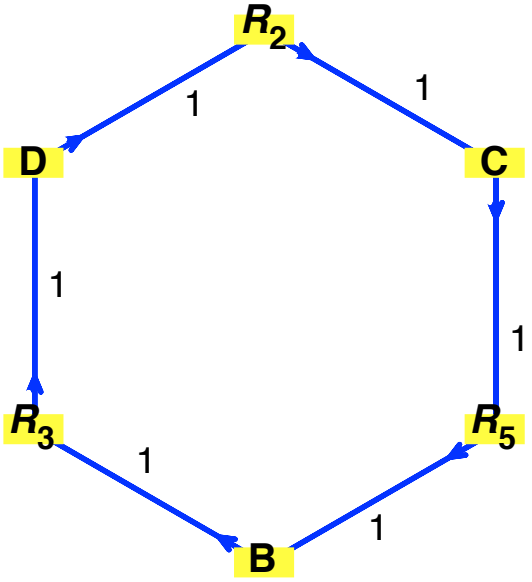


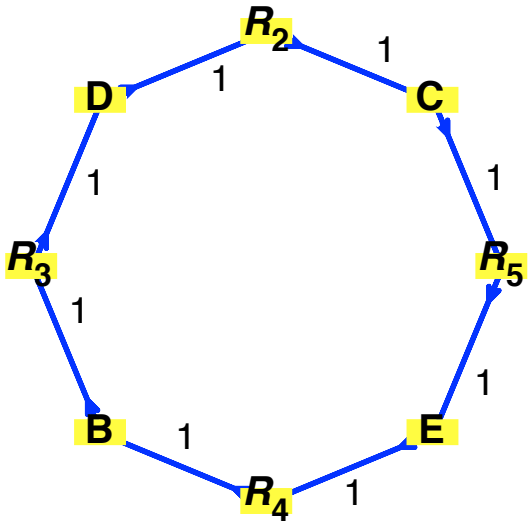


No. 7: injectiveEx2_1573.csv

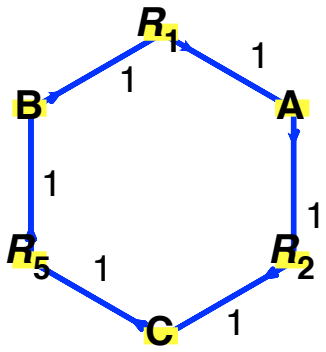

R1
1
1
B
A
1
1
R5
R2
1
1
C

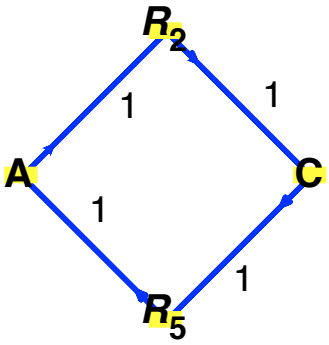

R1
1
1
B
A
1
1
R4
R2
1
1
E
C
1
R5
1

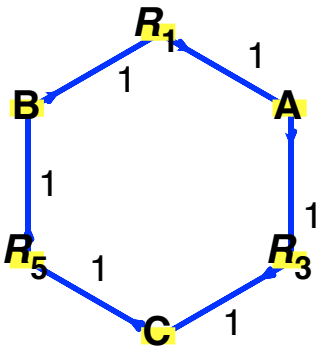


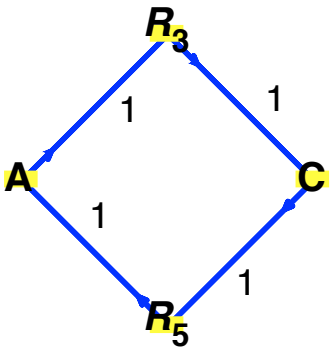


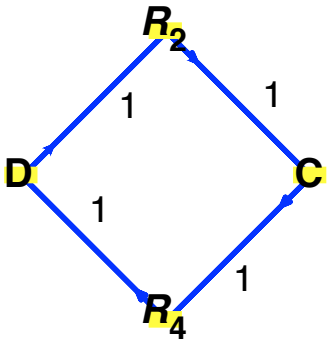
No. 8: injectiveEx2_1749.csv

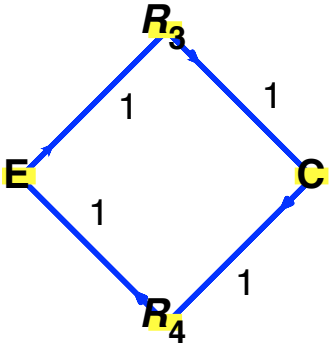

R1
1
1
B
A
1
1
R5
1
R2
C
1


R2
1
1
A
C
1
1
R5

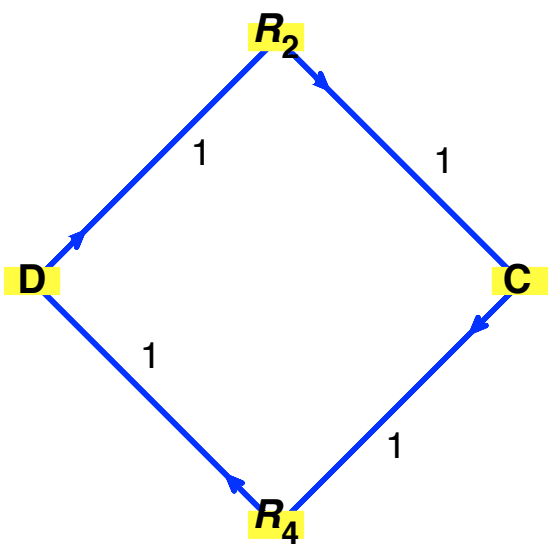

R1
1
1
B
A
1
1
R5
1
R3
C
1

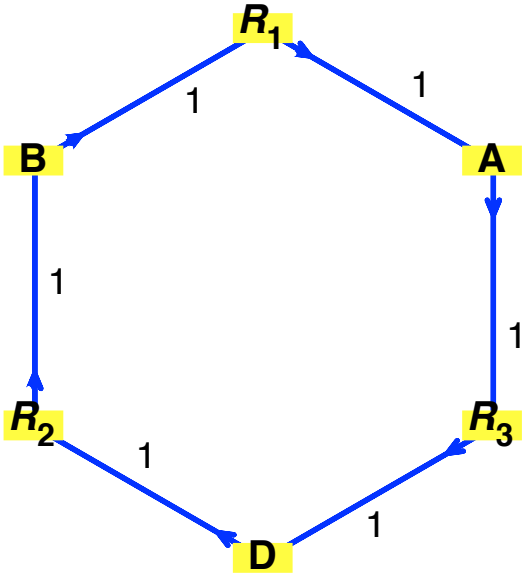


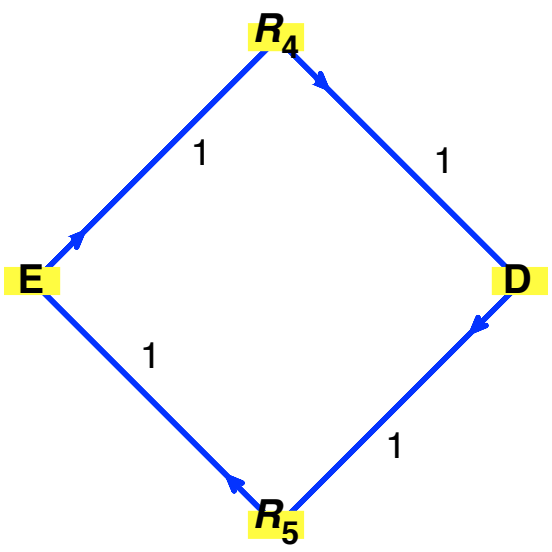


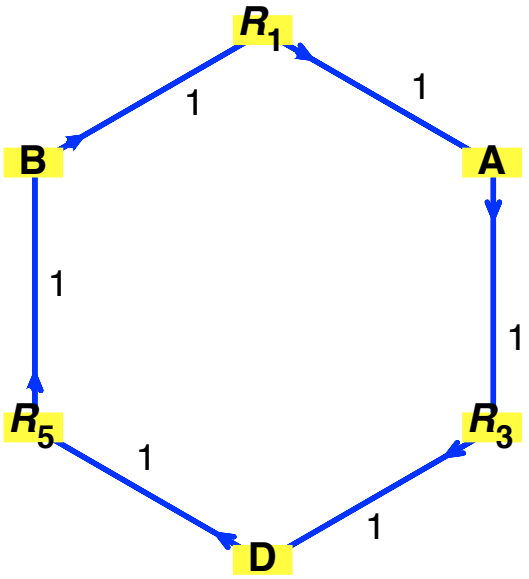


No. 9: injectiveEx2_2052.csv

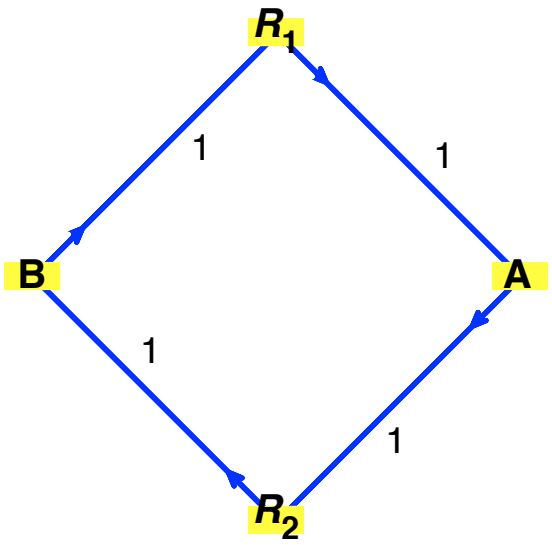

R2
1
1
D
C
1
1
R4

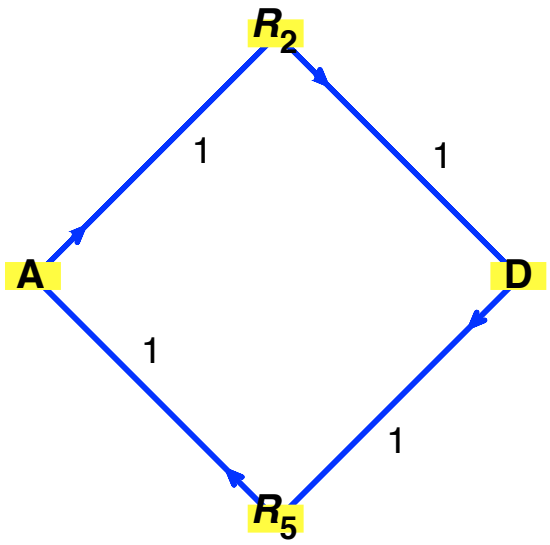

R1
1
1
B
A
1
1
R2
R3
1
1
D

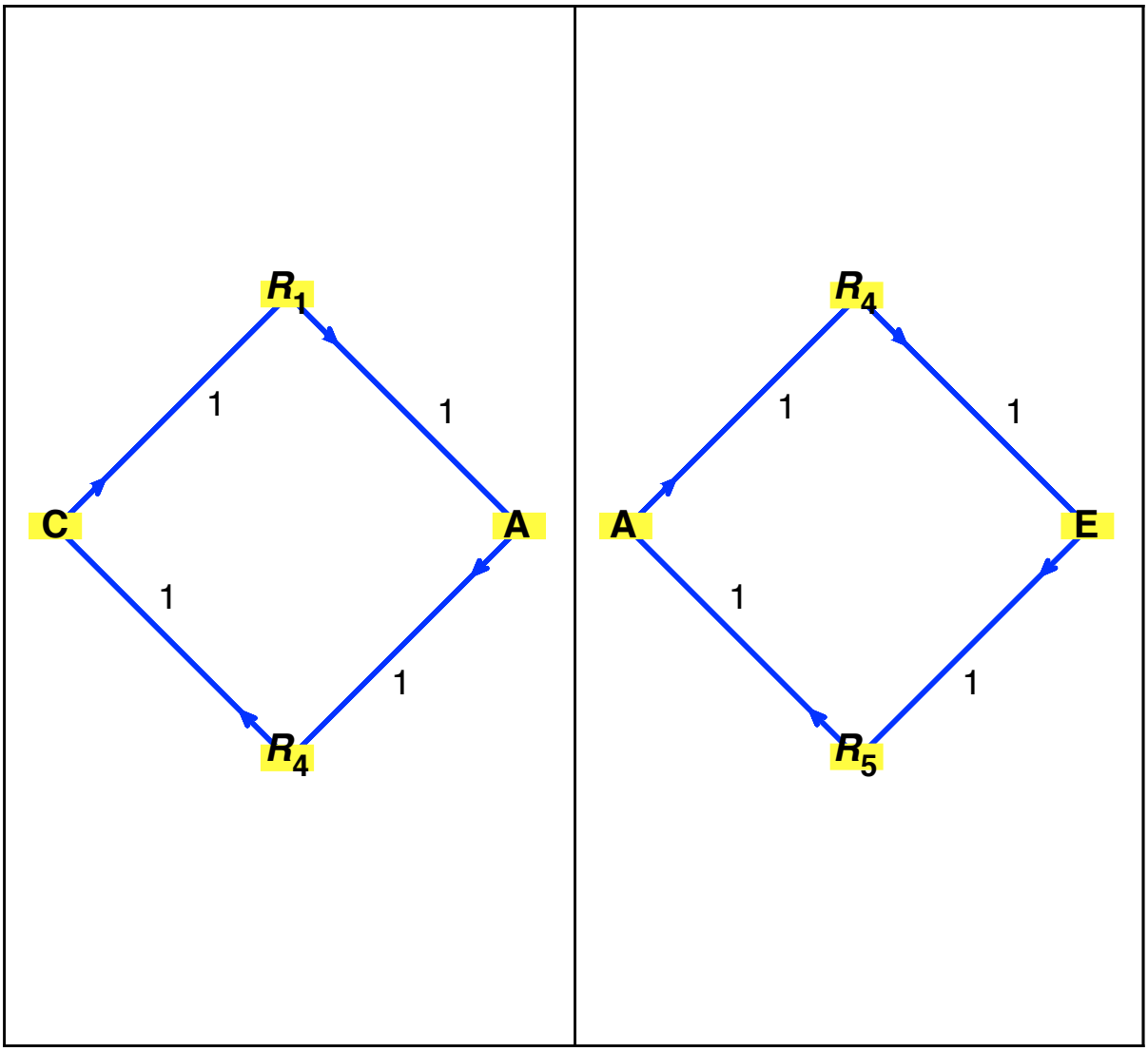




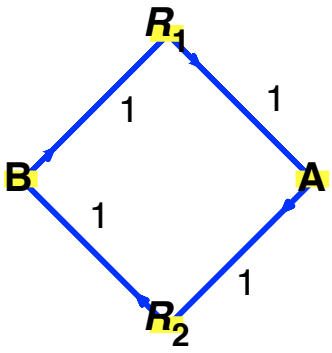
No. 10: injectiveEx2_4180.csv

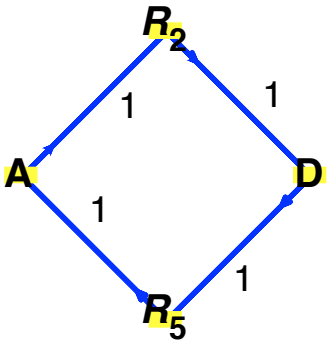

R1
1
1
B
A
1
1
R2

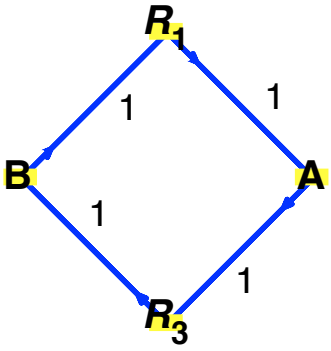

R2
1
1
A
D
1
1
R5

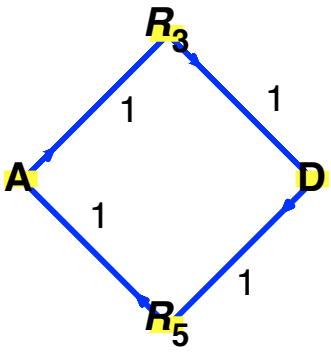


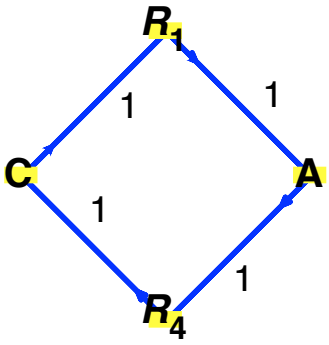
No. 11: injectiveEx2_4203.csv WRONG.csv

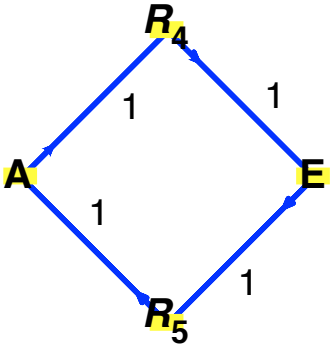

R1
1
1
B
A
1
1
R2


R2
1
1
A
D
1
1
R5

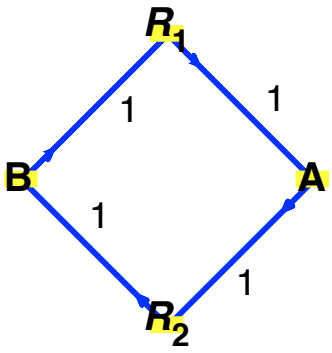

R1
1
1
B
A
1
1
R3

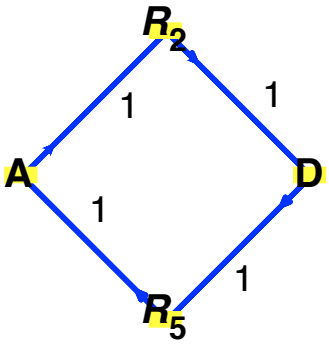


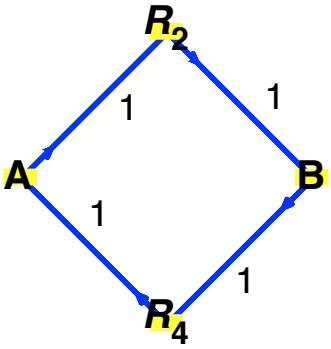


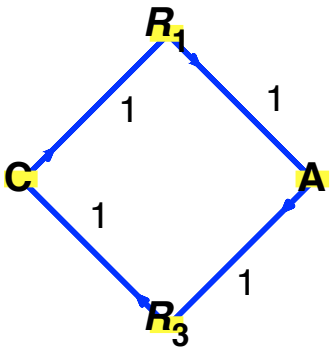


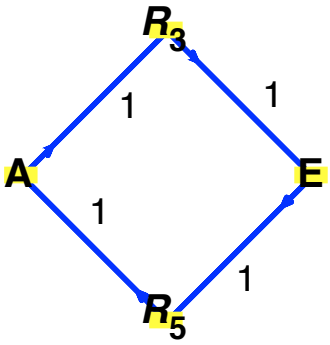
No. 12: injectiveEx2_4242.csv WRONG.csv

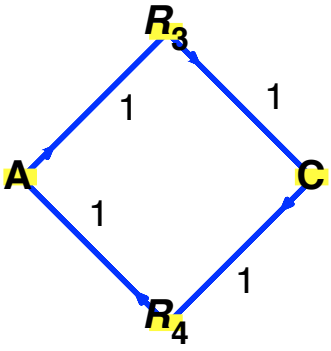

R_1
1
1
B
A
1
1
R_2


R_2
1
1
A
D
1
1
R_5


R_2
1
1
A
B
1
1
R_4


R_1
1
1
C
A
1
1
R_3


R_3
1
1
A
E
1
1
R_5


R_3
1
1
A
C
1
1
R_4

>