

# 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
createDSRgraphsinged := 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 := createDSRgraphsinged(mylabels, A, Z) :
  return(Edges(G, weights)) :
end proc:

>

```

## ▼ Functions for draw loops from graph

### ▼ Auxiliary procedures

This procedure computes 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:
end proc:

```

```

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, AnewI :
  vars := indets(wmonom) :
  wvarscomp := indets(Amatrix) minus vars :
  Anew := subs(seq(wvarscomp[i] = 0, i = 1 .. numelems(wvarscomp)), Amatrix) :
  AnewI := subs(seq(vars[i] = 1, i = 1 .. numelems(vars)), Anew) :
  return (AnewI) :
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]]·xi :
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( $S_{col[i]}$ ,  $i = 1 \dots \text{numelems}(col)$ ), seq( $R_{row[i]}$ ,  $i = 1 \dots \text{numelems}(row)$ )] :
    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,  $i = 1 \dots \text{numelems}(\text{indets}(Zembed))$ ),
      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
orderededge := proc(myedges)
  local orderededges, endpoint, total, control, k :
  orderededges := [myedges[1]] :
  endpoint := myedges[1][1][2] :
  total := numelems(myedges) :
  while numelems(orderededges) < 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:
end proc:

```

```

selected := ListTools[MakeUnique](selected) :
for k from 1 by 1 to numelems(selected) do
  selected[k] := orderededge(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(Si = 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:

```

```

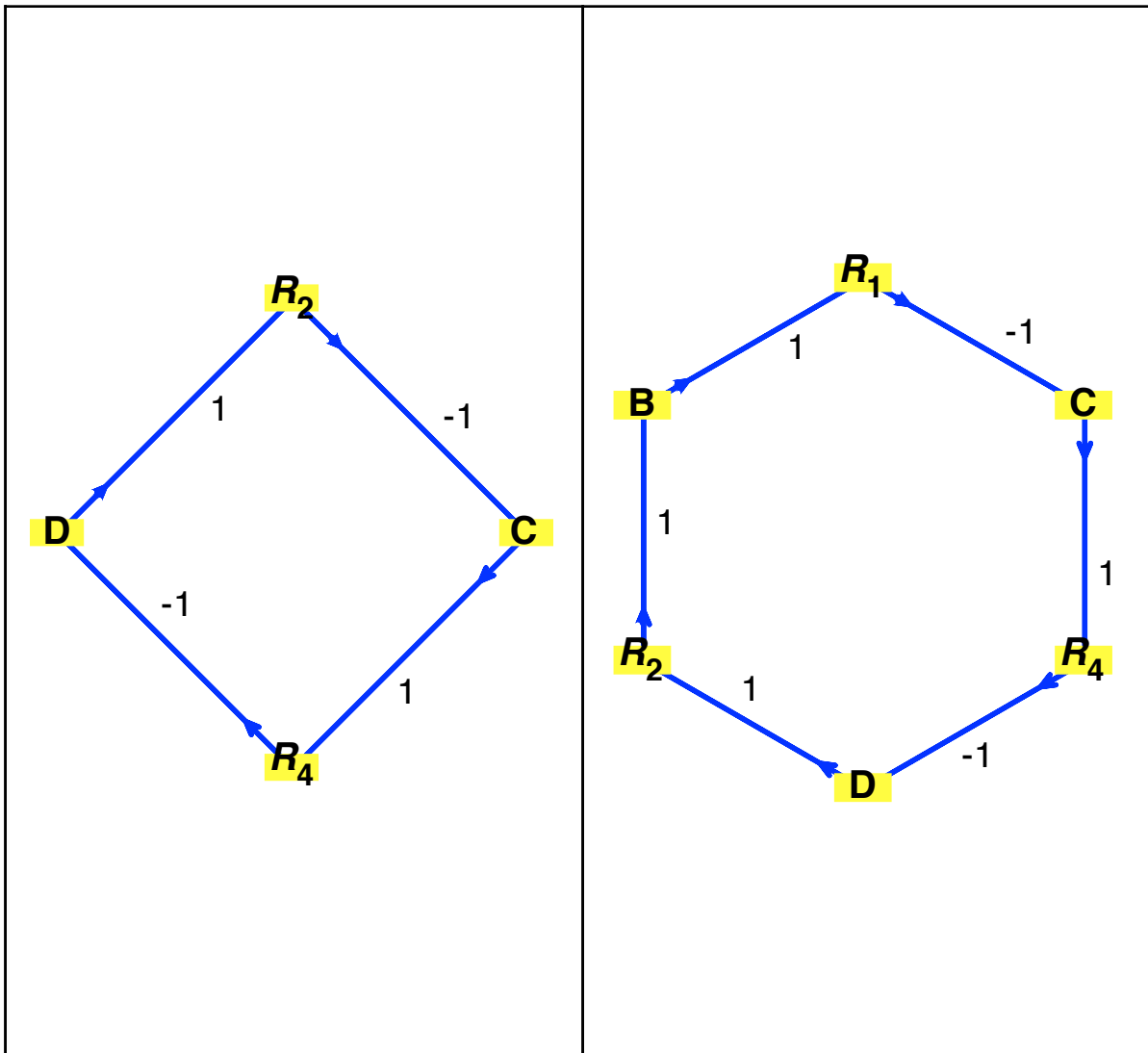
>

```

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

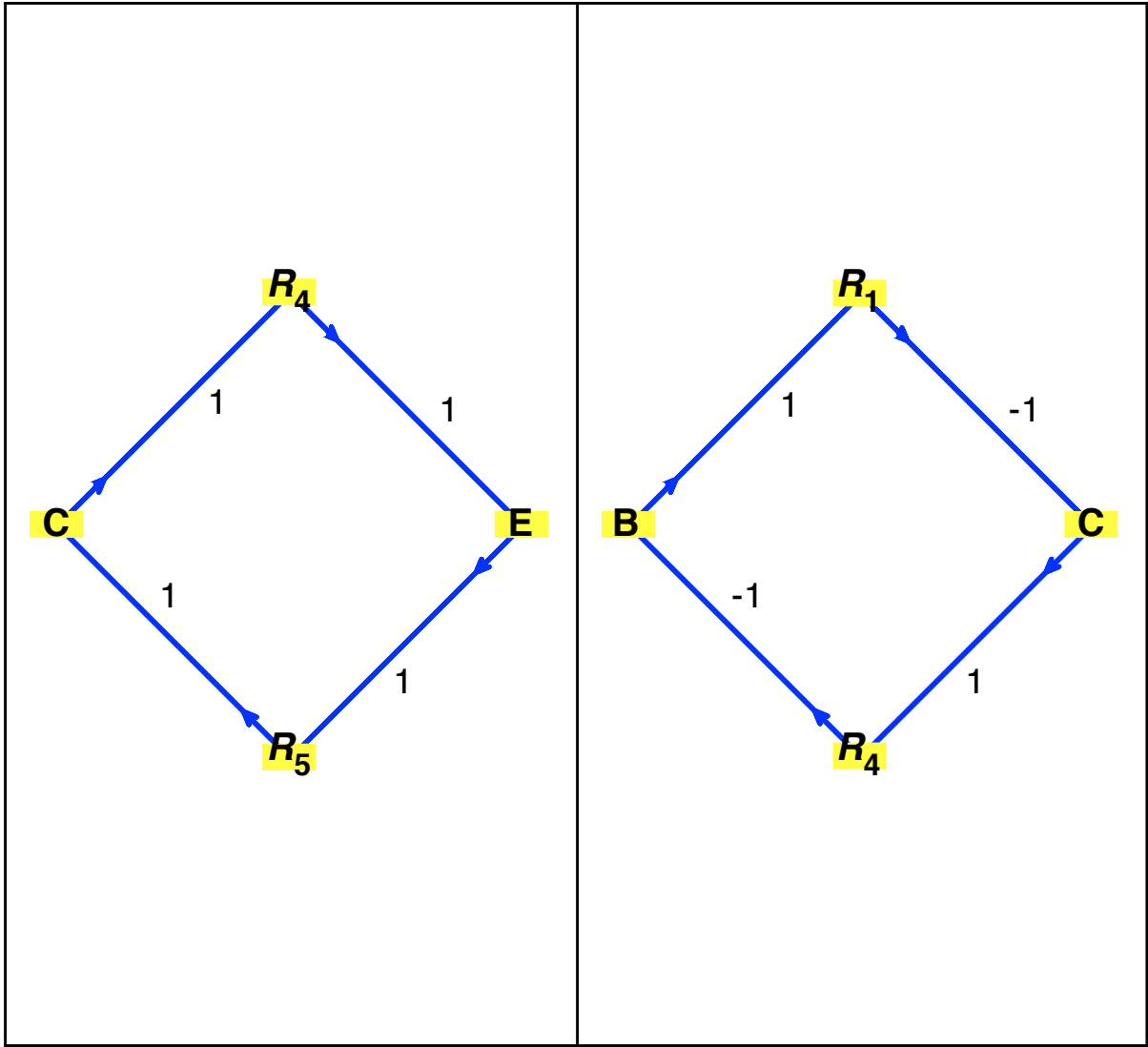
## Non-multistationary: with competition loop, with intersecting loops

```
[> 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: injectivel_2962.csv
```

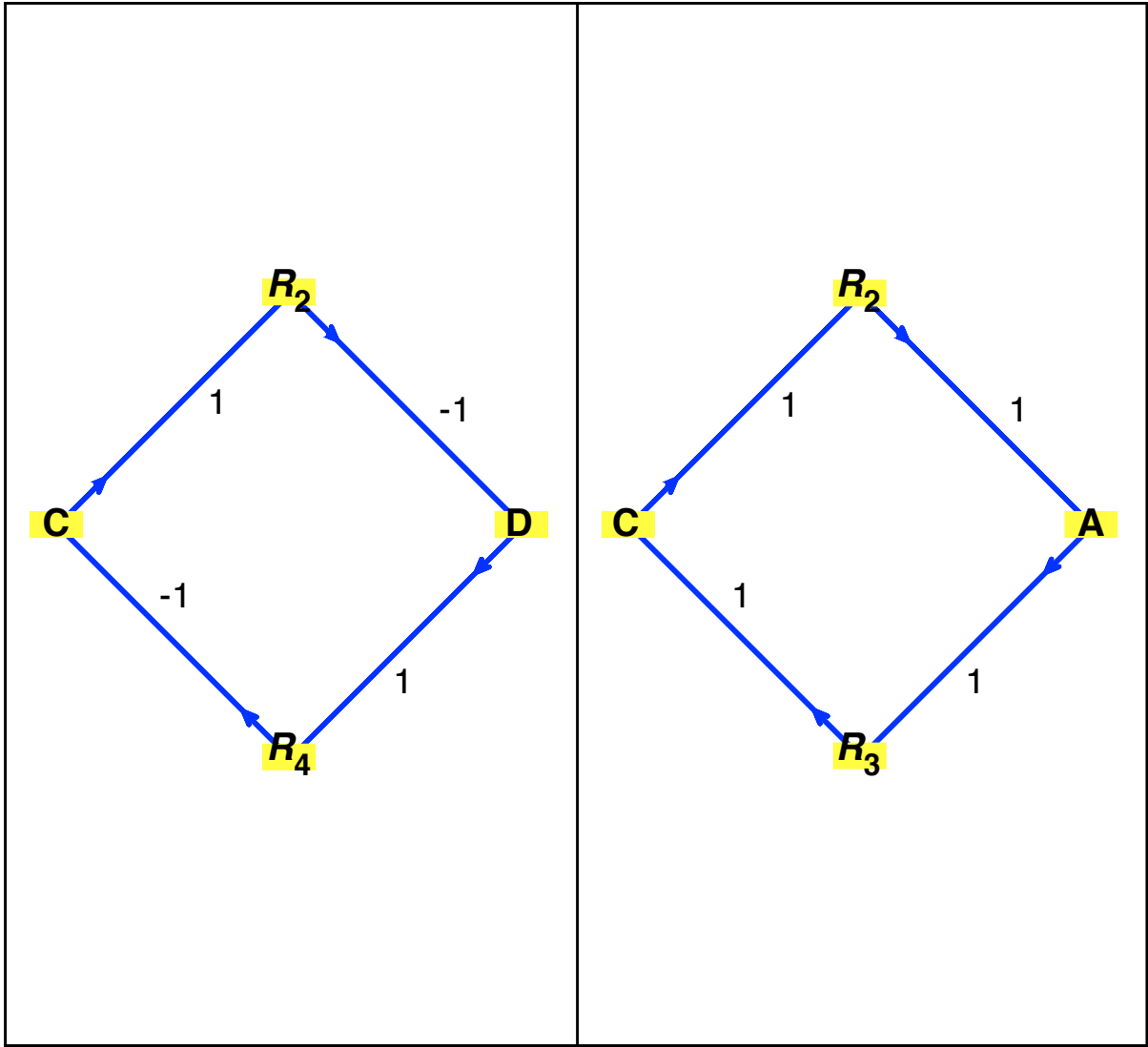


No. 2: injectivel\_3111.csv

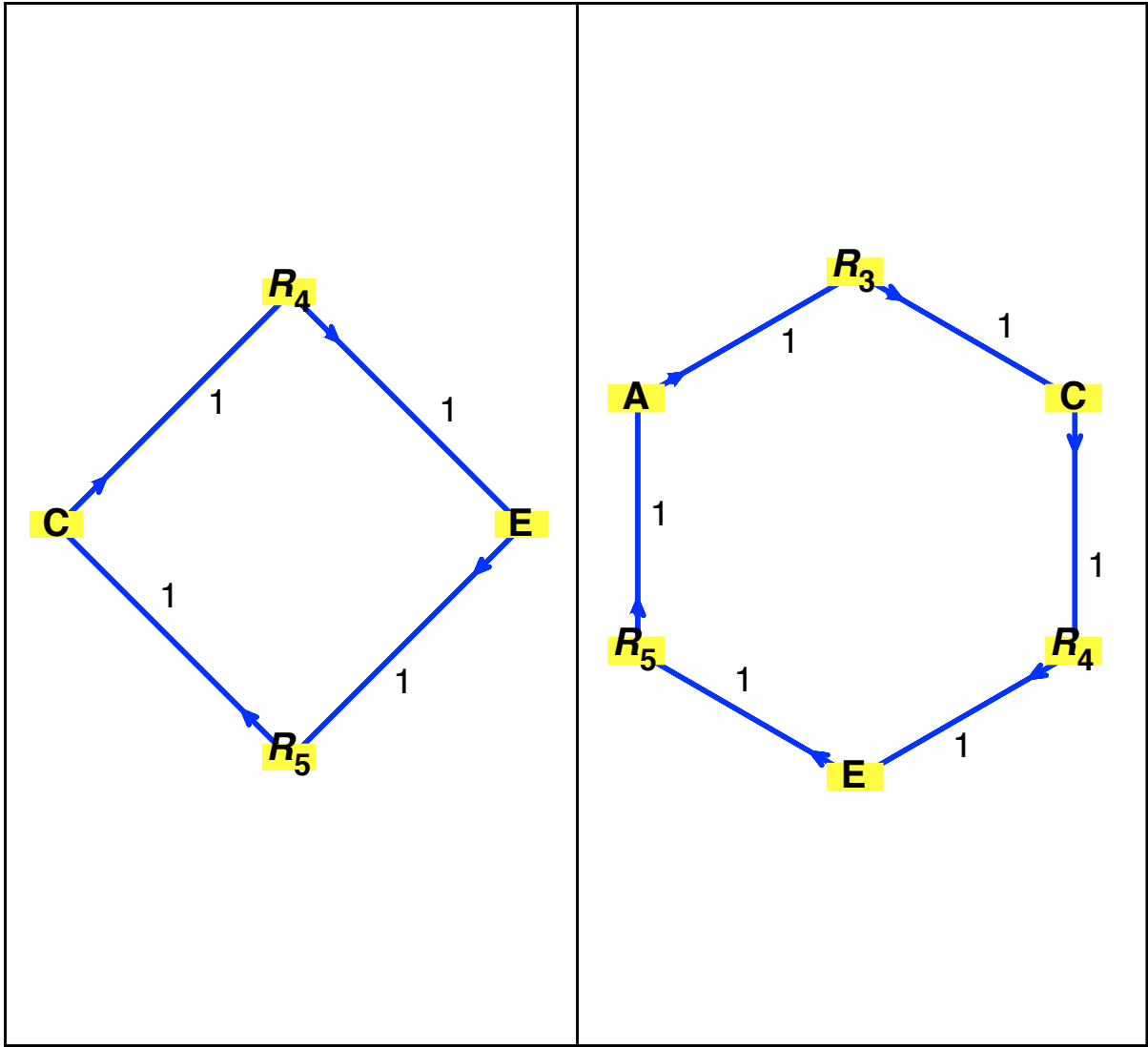




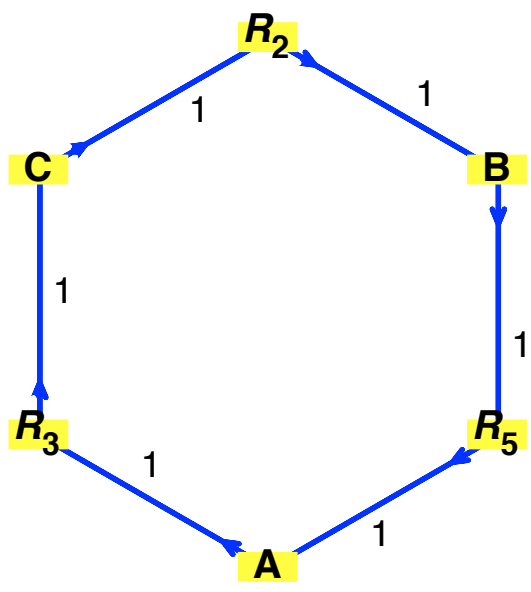
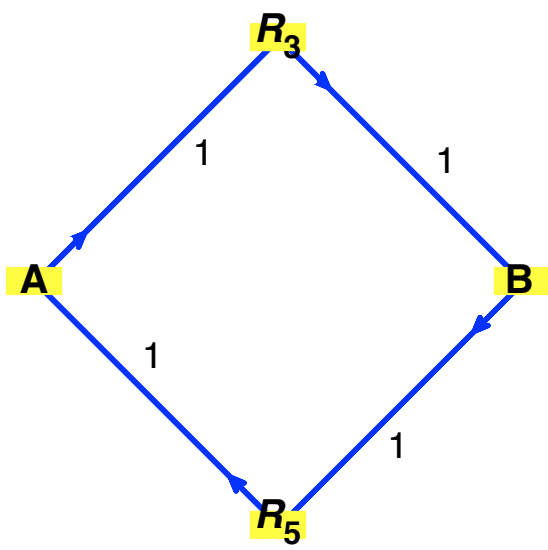
No. 3: injectivel\_3119.csv

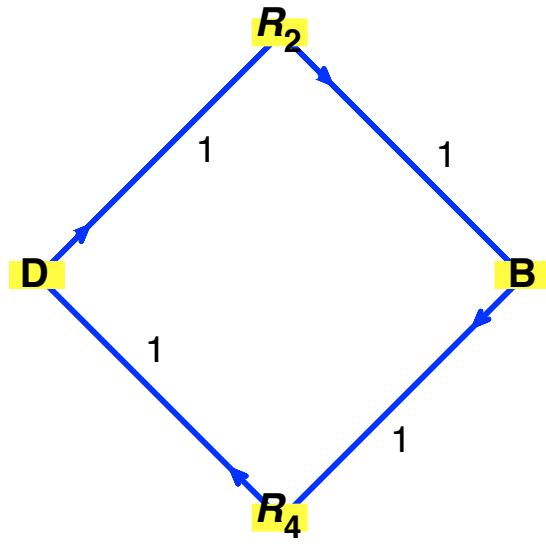
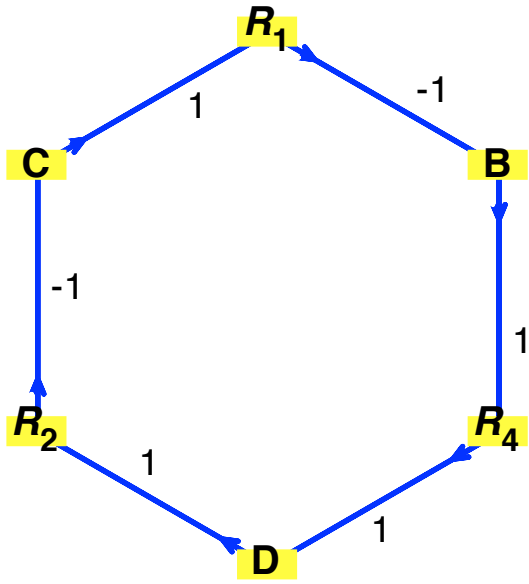


No. 4: injectiveEx1\_1267.csv

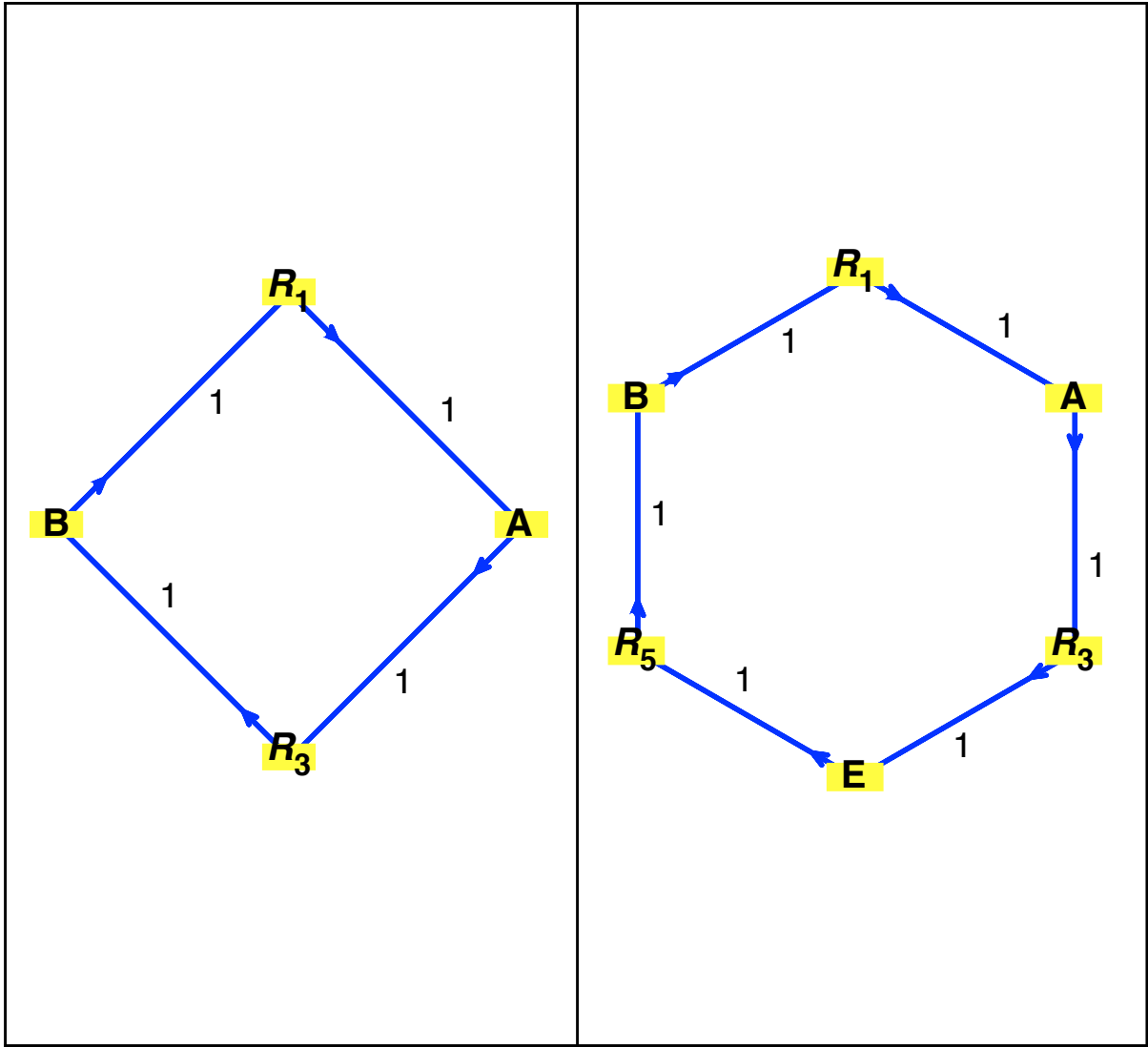


No. 5: injectiveEx1\_2956.csv

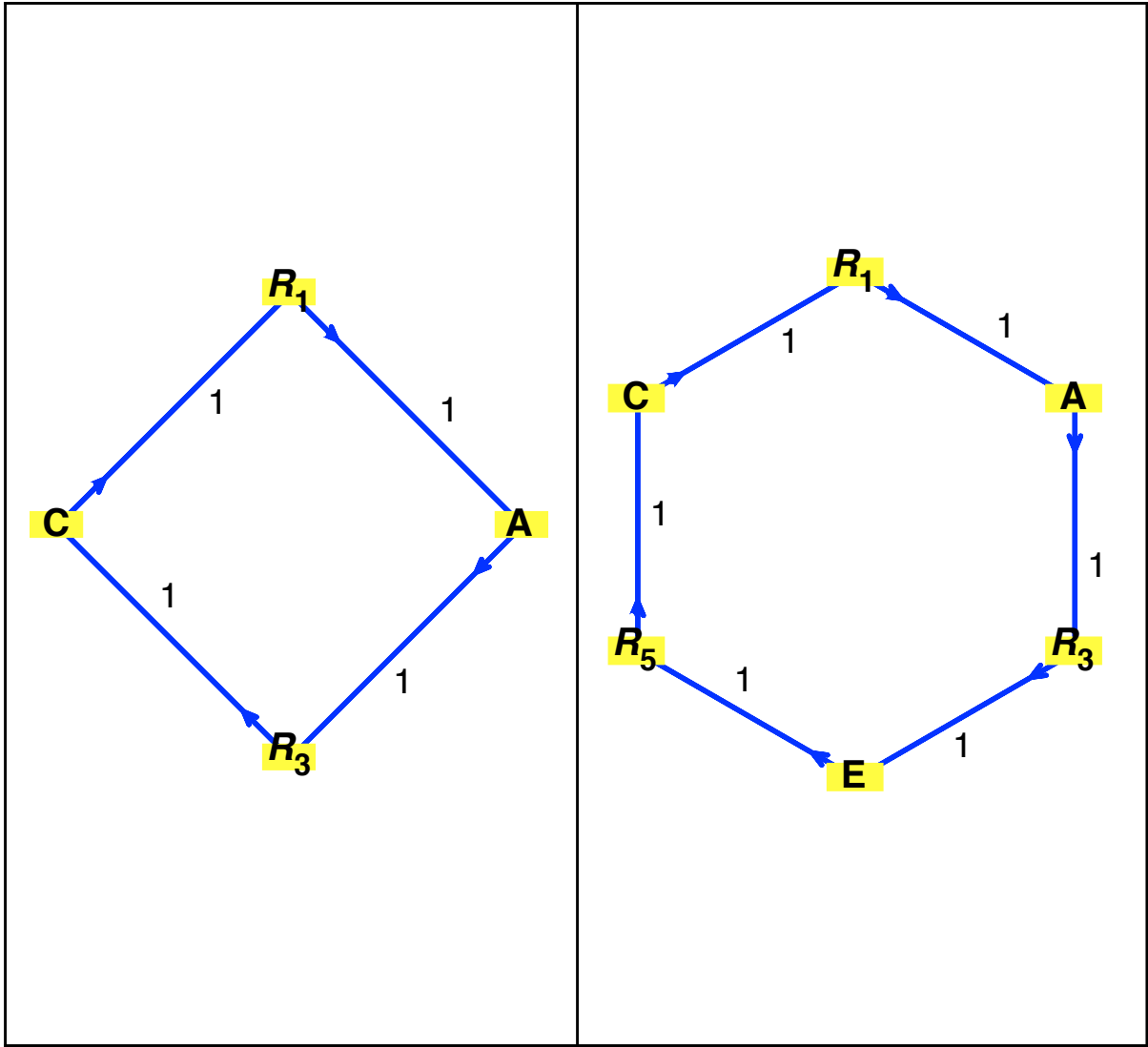




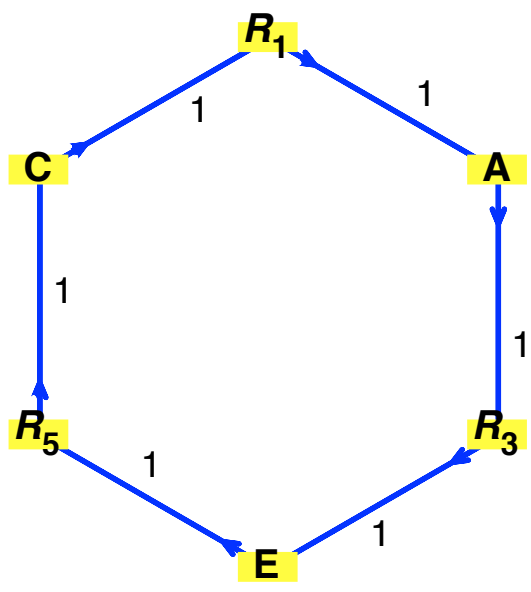
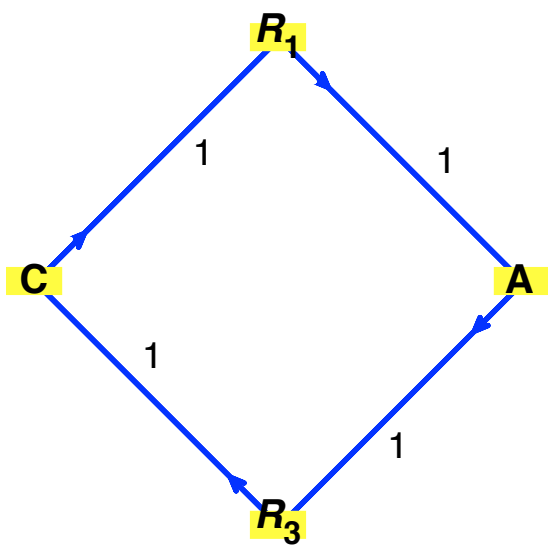
No. 6: injectiveEx1\_2979.csv



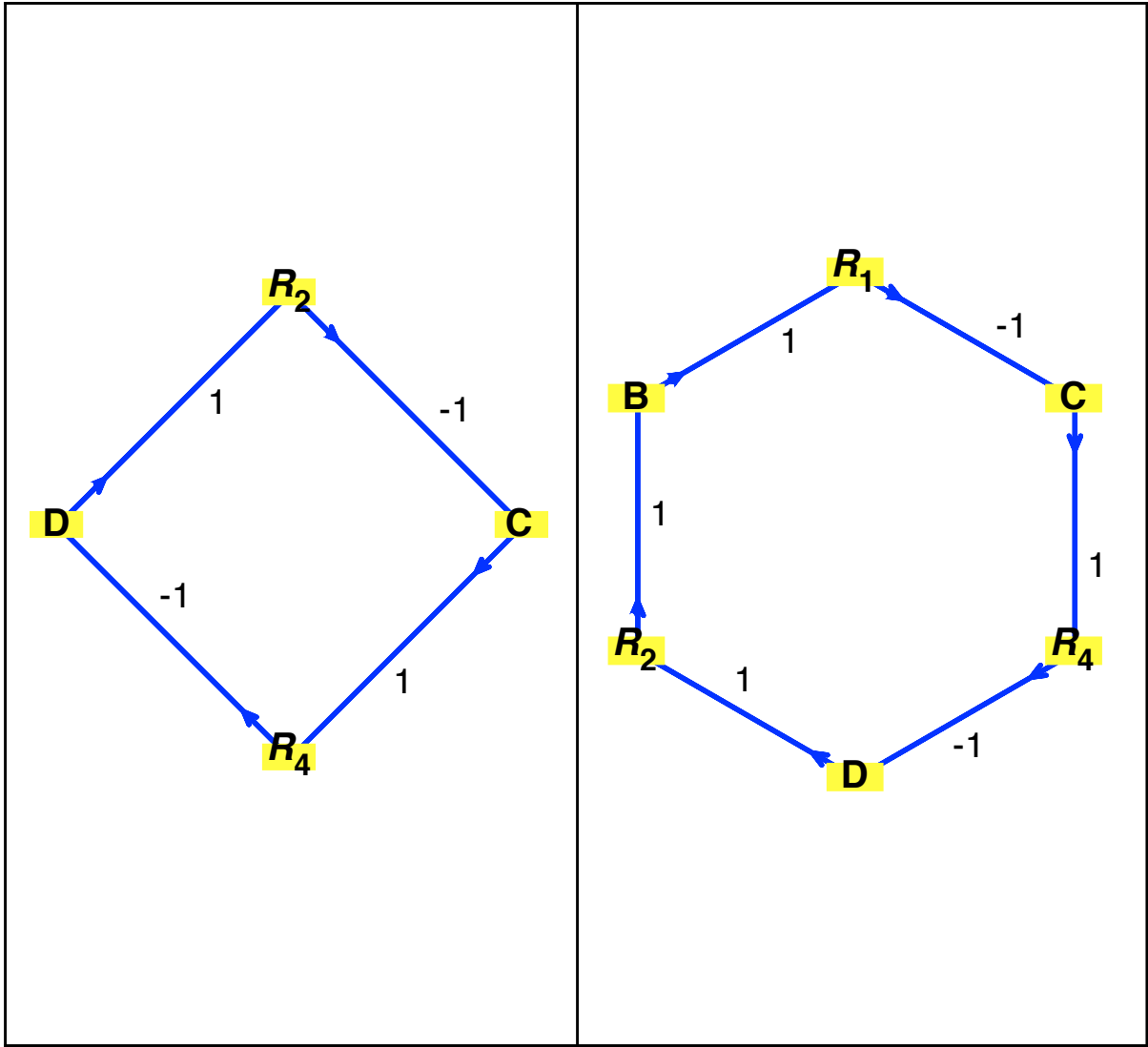
No. 7: injectiveEx1\_2987.csv



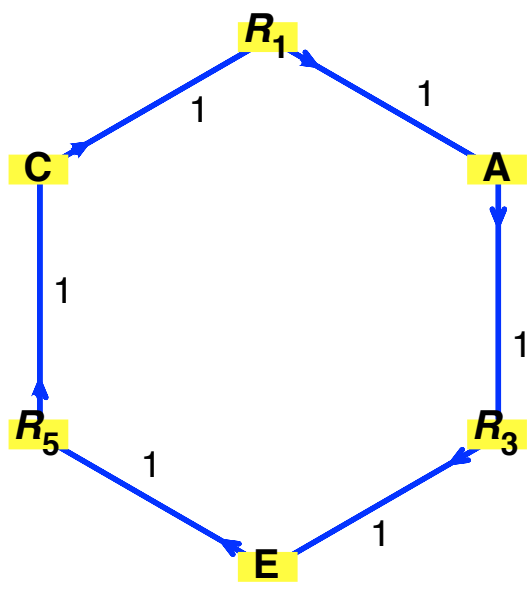
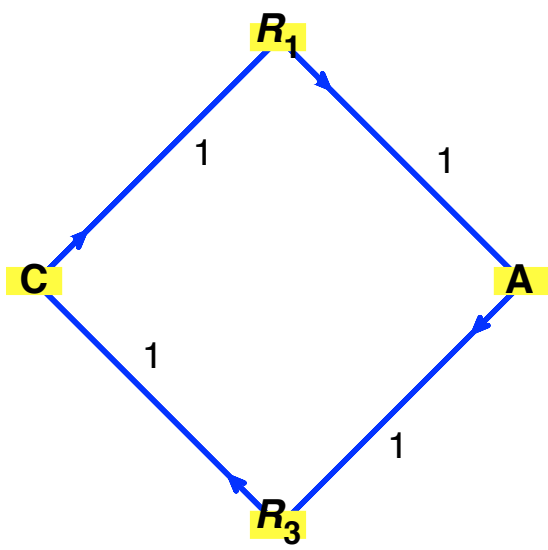
No. 8: injectiveEx1\_2988.csv

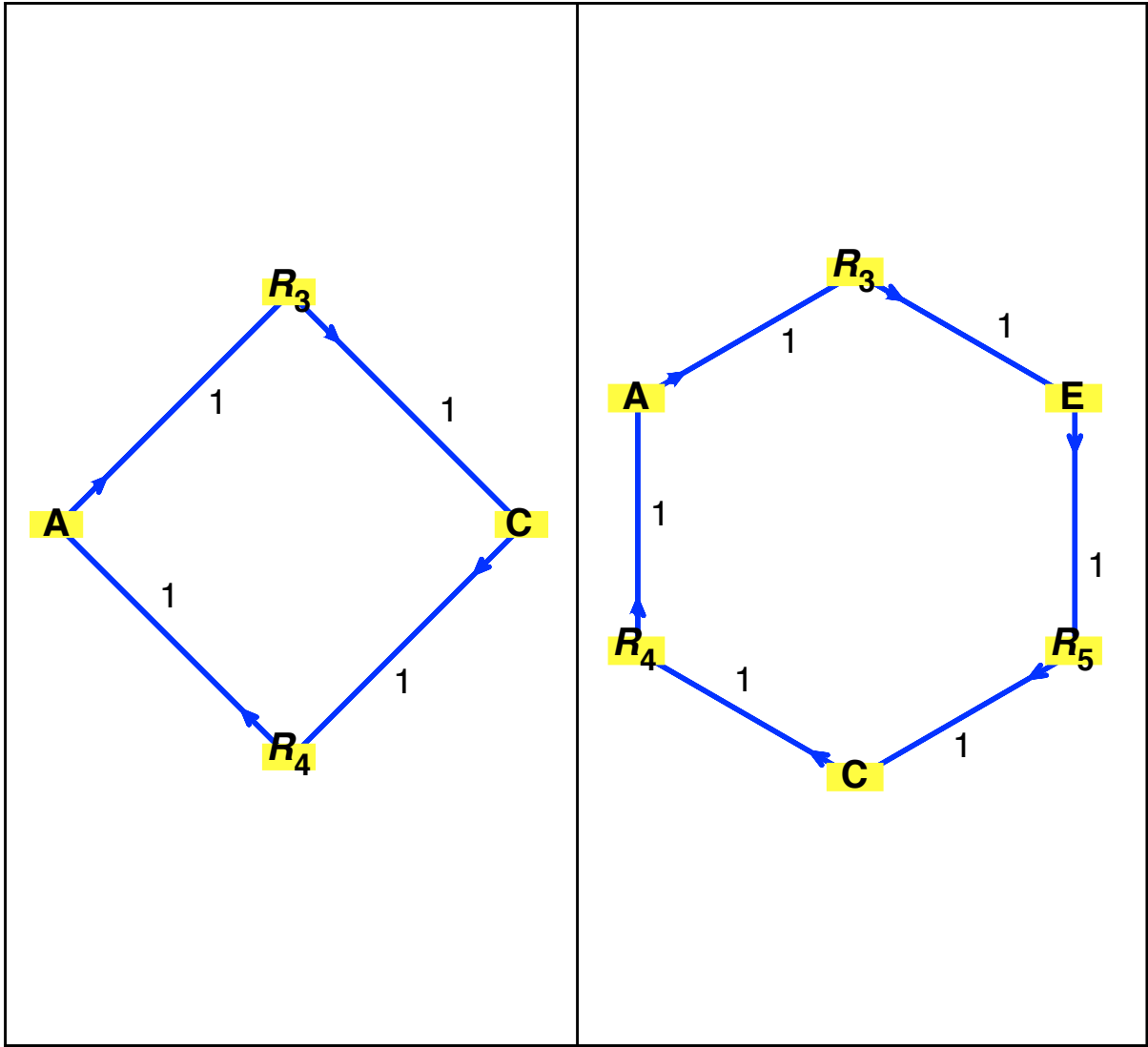




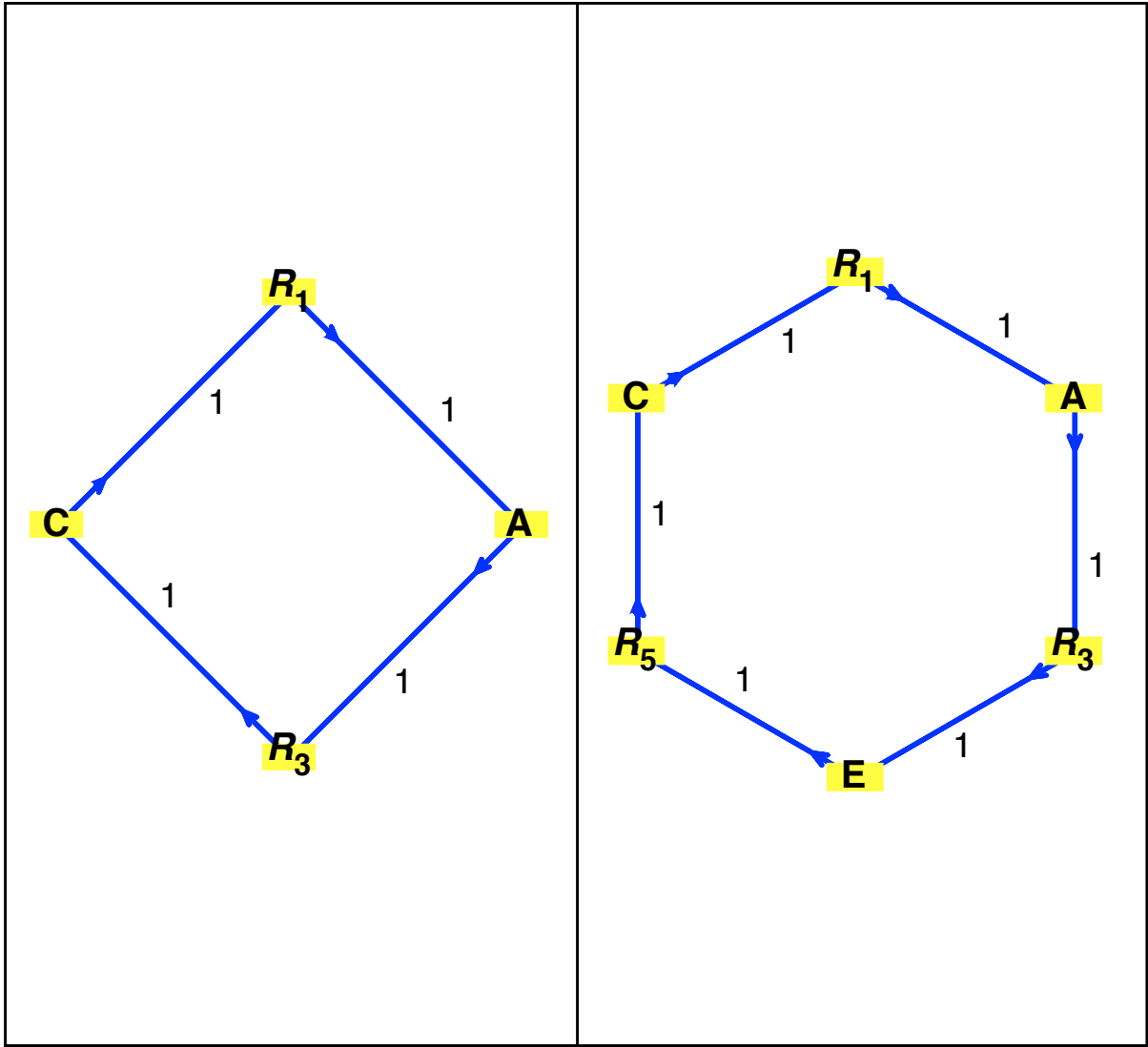


No. 9: injectiveEx1\_2989.csv

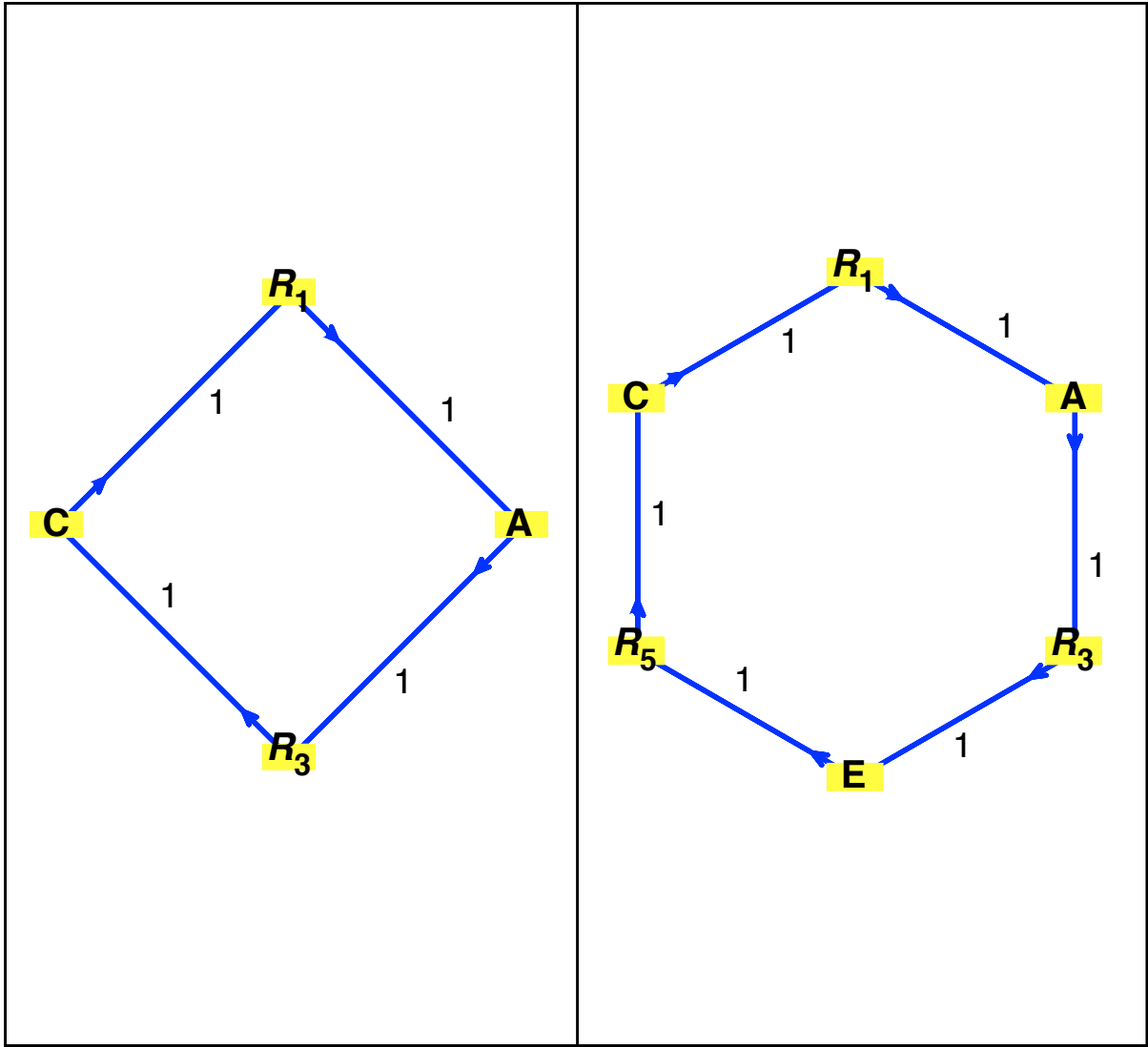




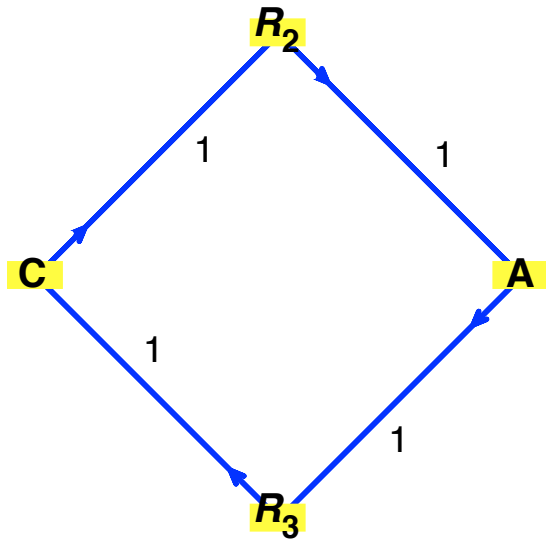
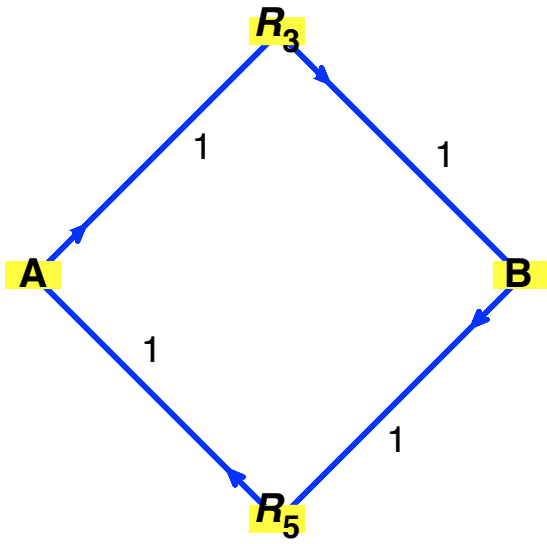
No. 10: injectiveEx1\_2991.csv

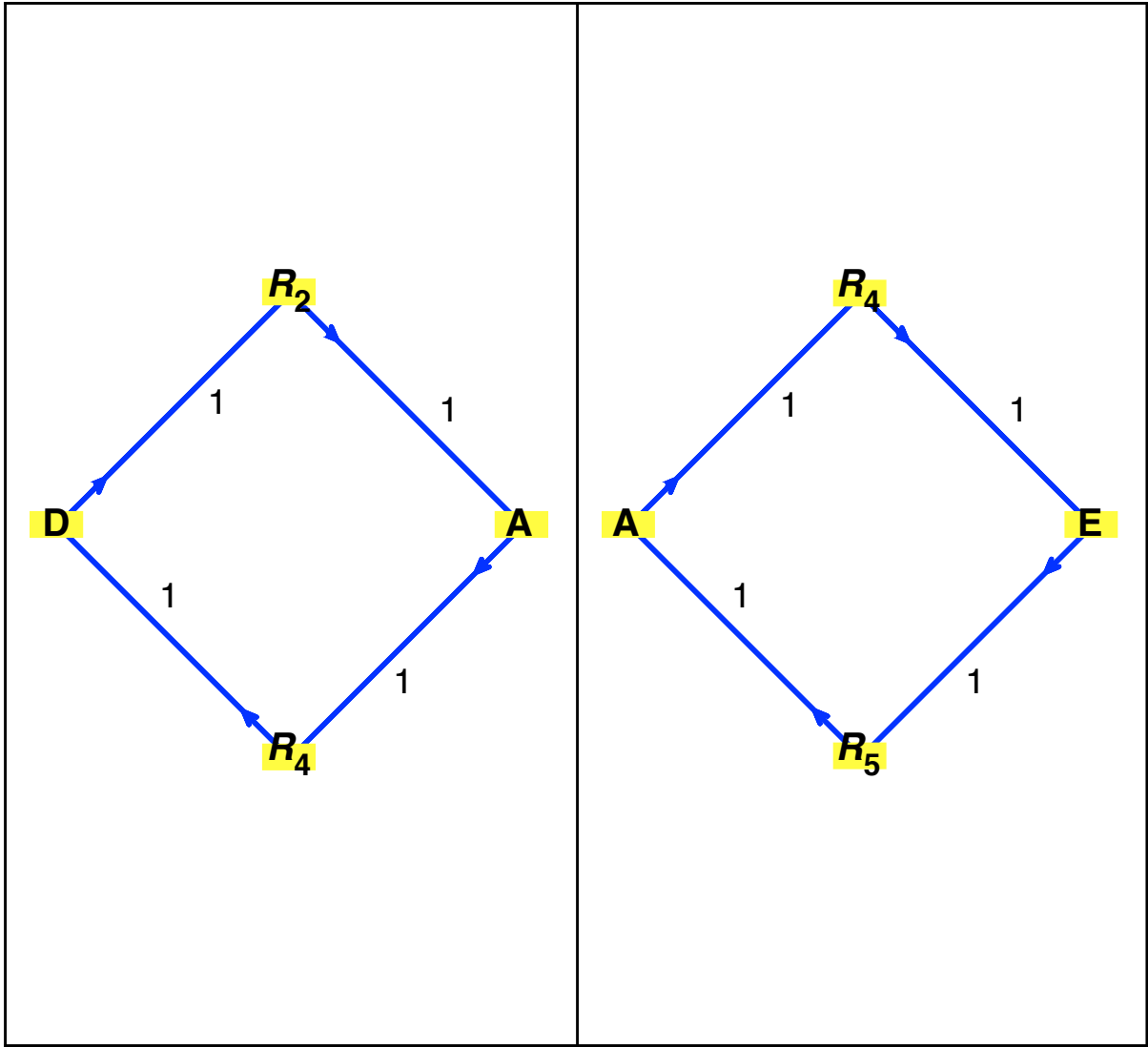


No. 11: injectiveEx1\_2993.csv

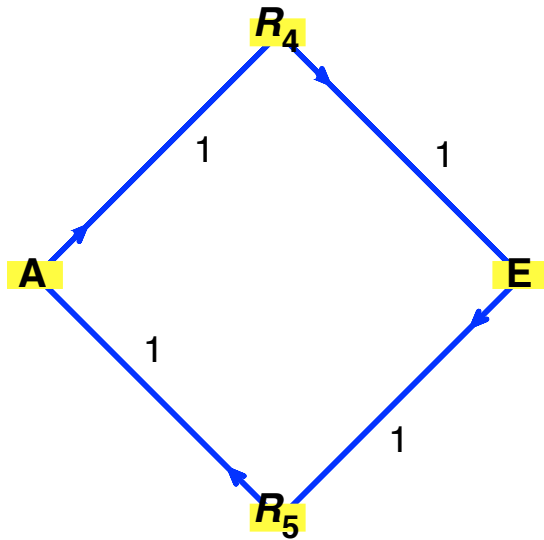
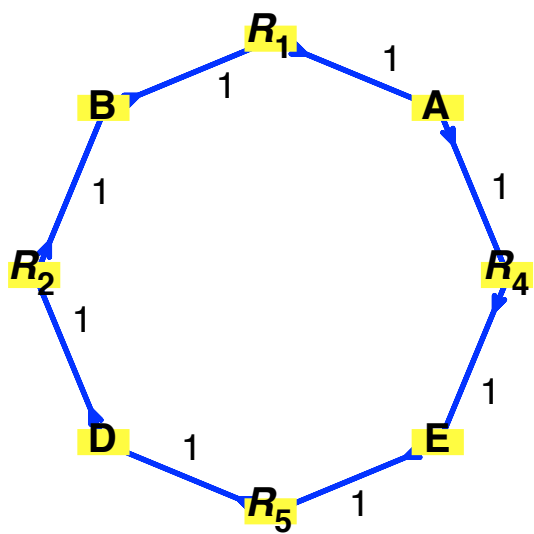


No. 12: injectiveEx1\_3103.csv



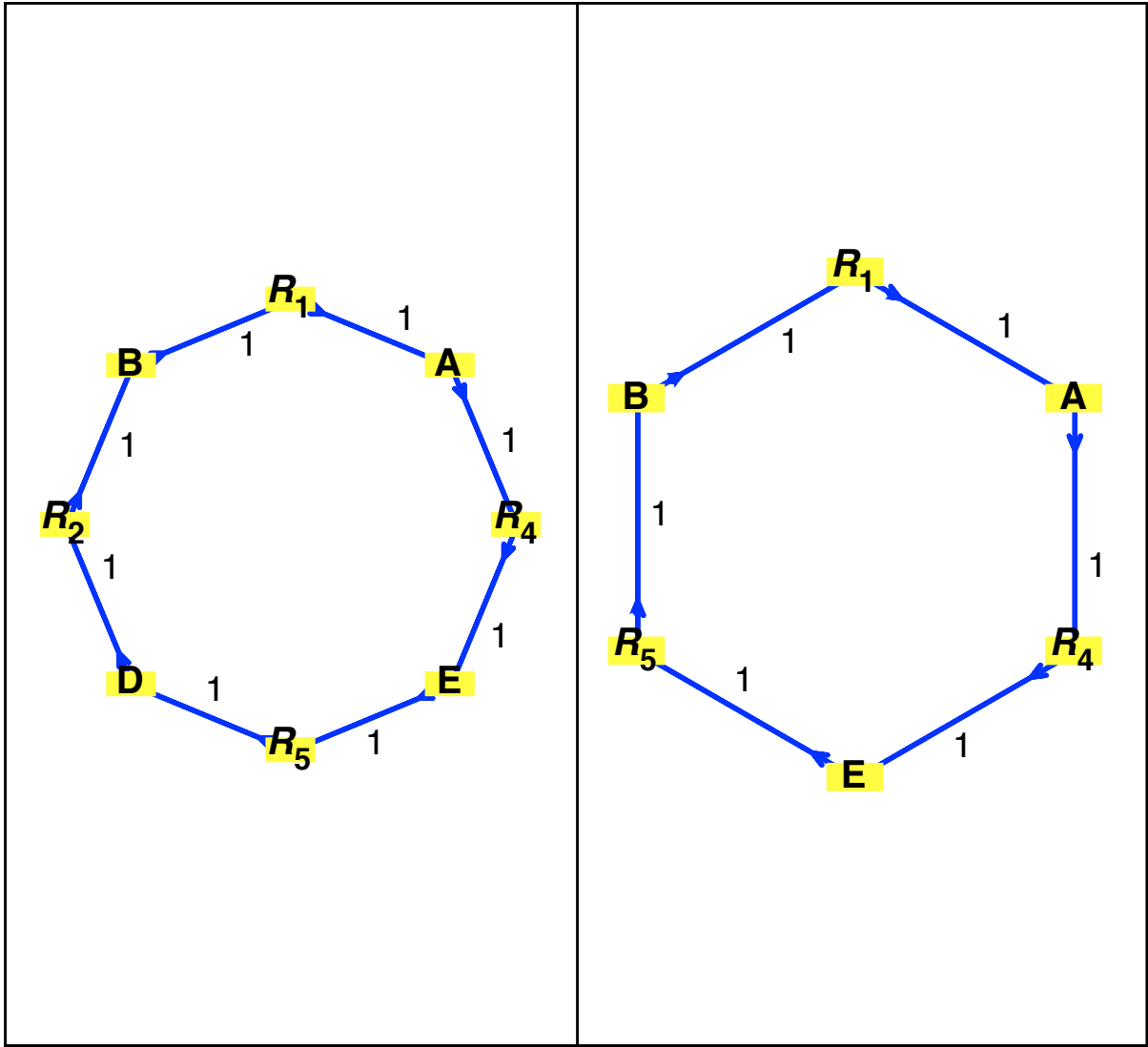


No. 13: injectiveEx3\_1250.csv

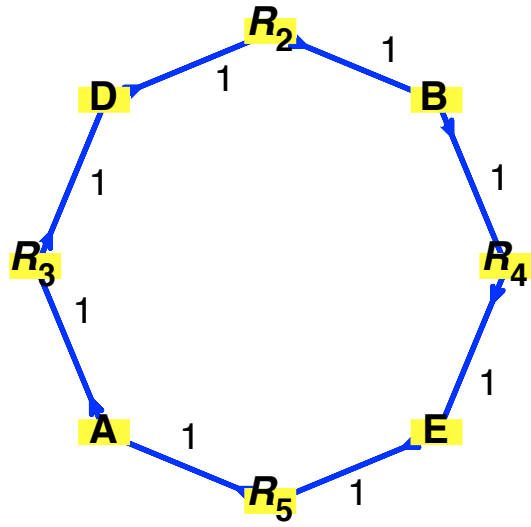
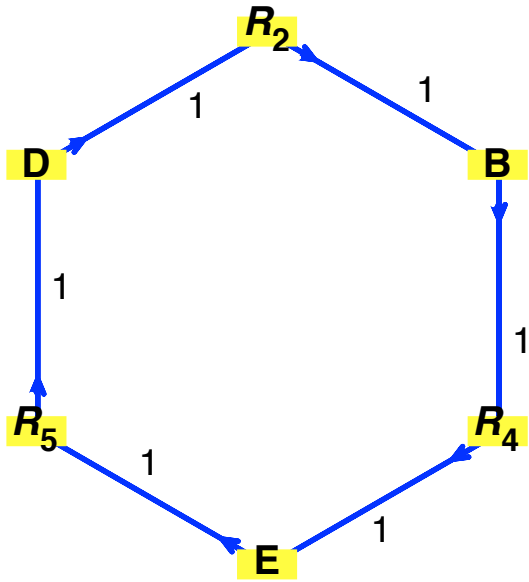


No. 14: injectiveEx3\_1252.csv

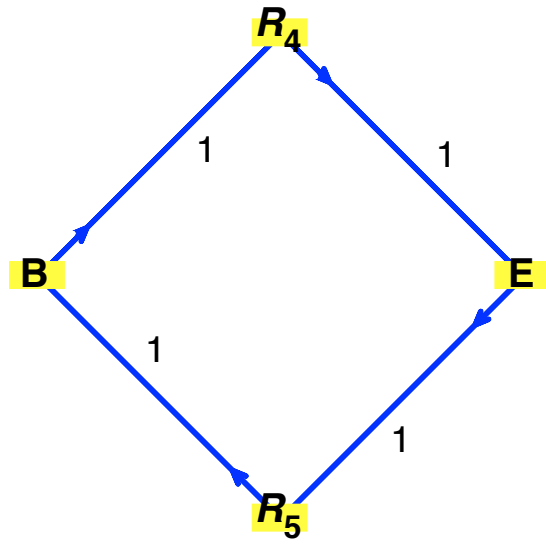
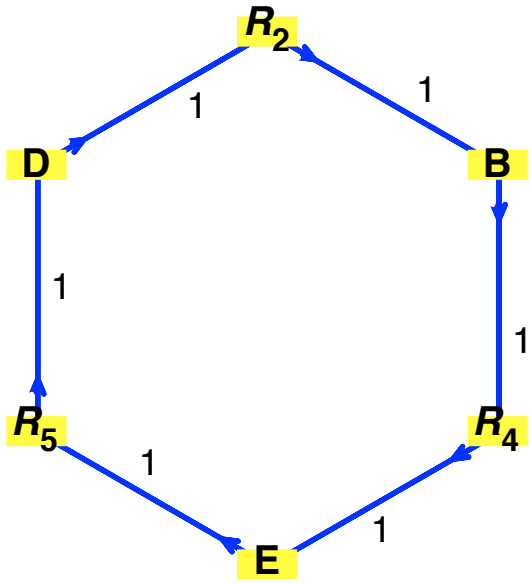




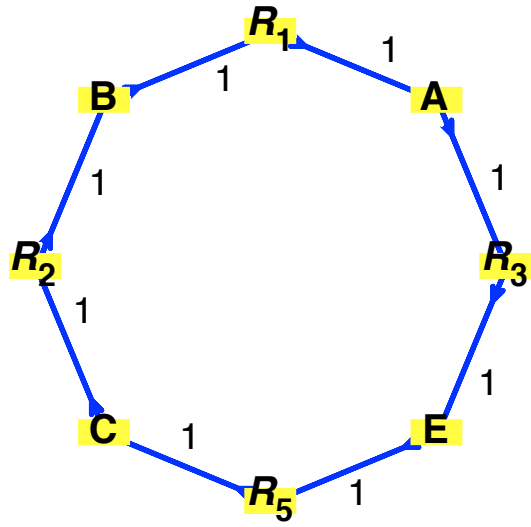
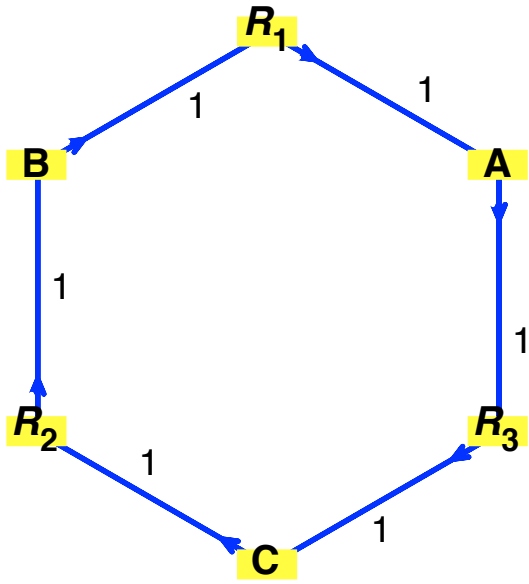
No. 15: injectiveEx3\_1268.csv



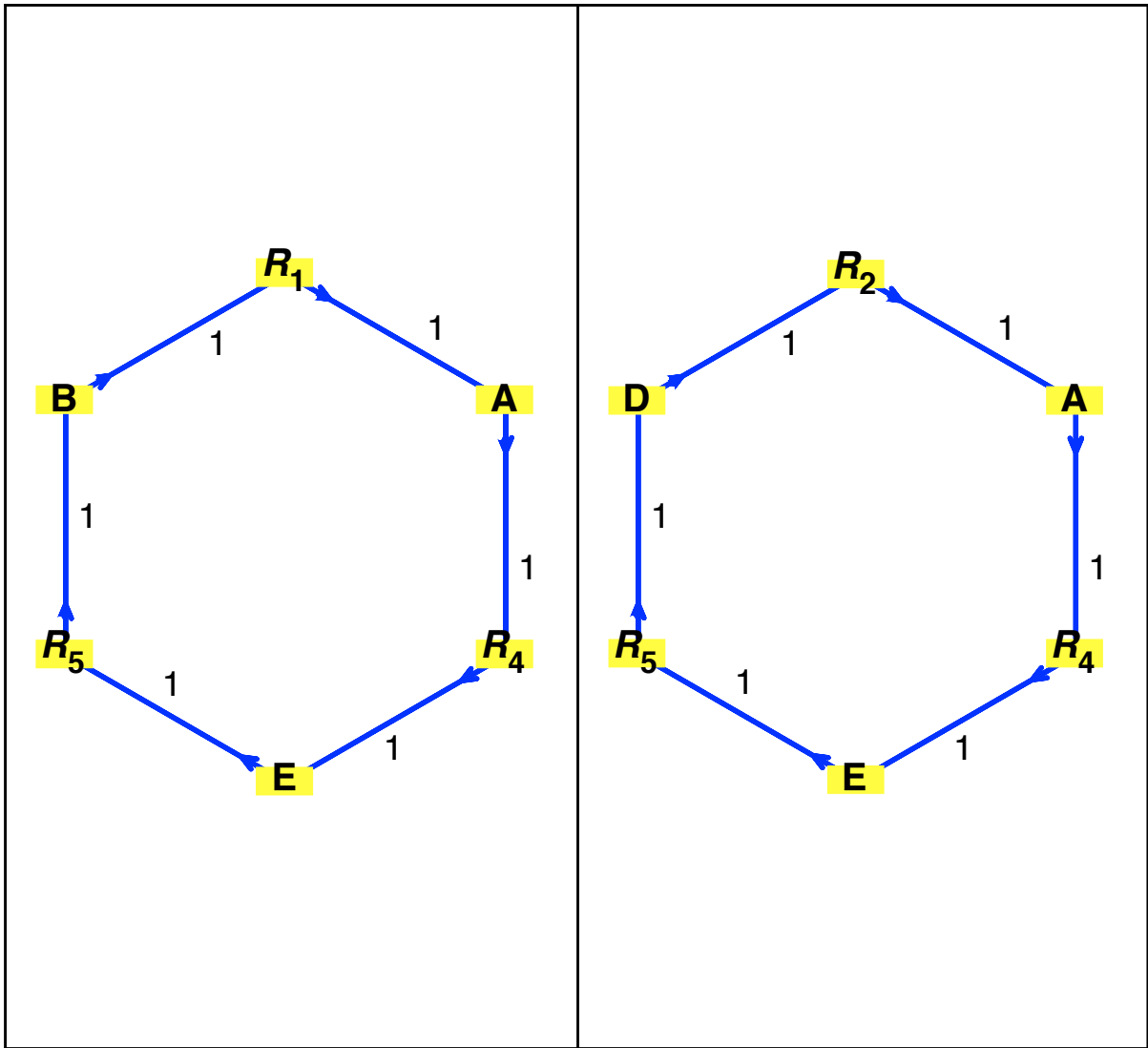
No. 16: injectiveEx3\_1270.csv



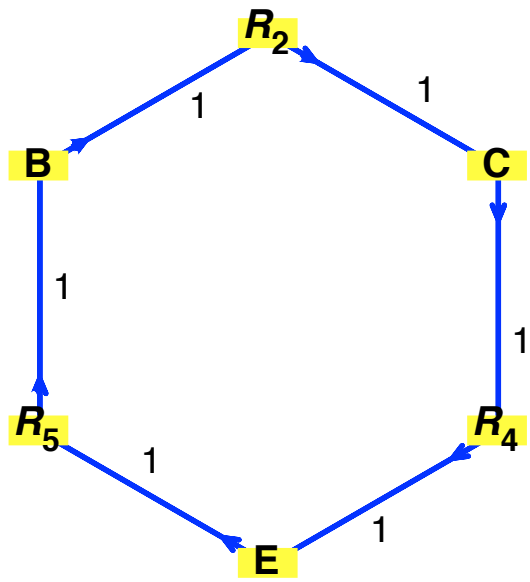
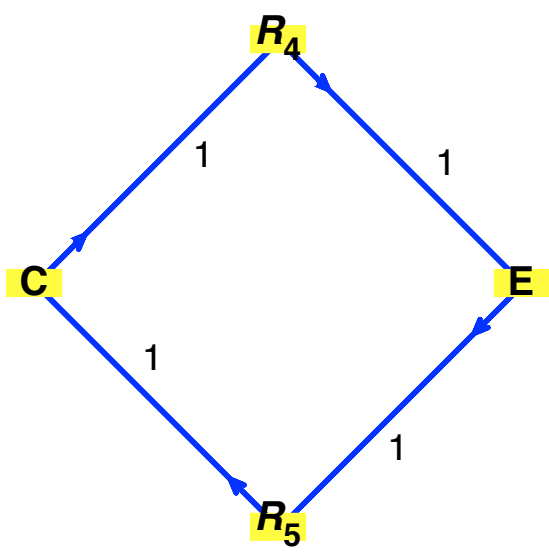
No. 17: injectiveEx3\_1302.csv

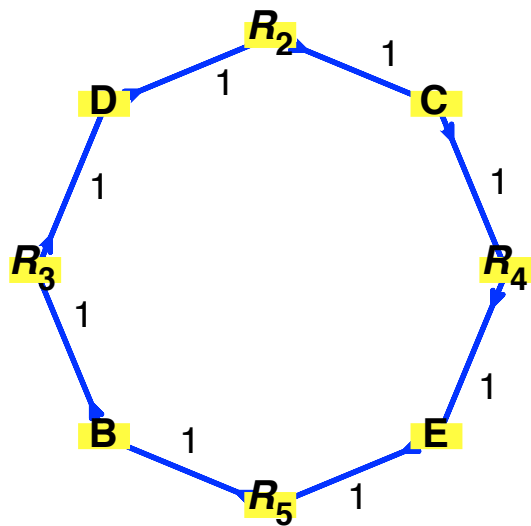
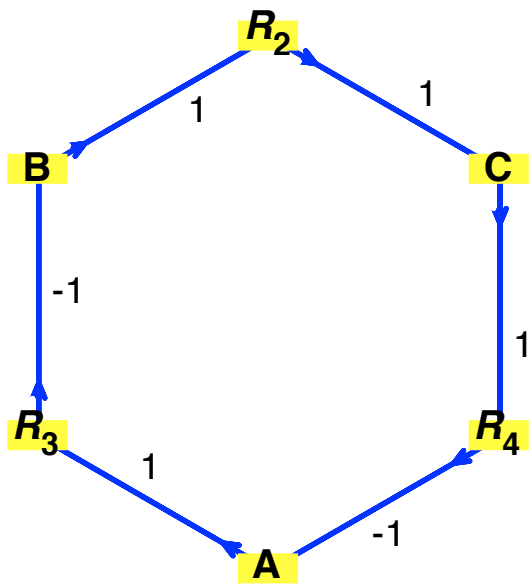


No. 18: injectiveEx3\_1369.csv

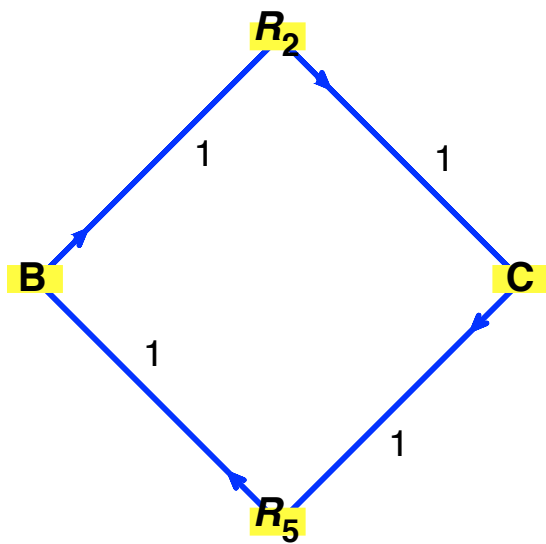
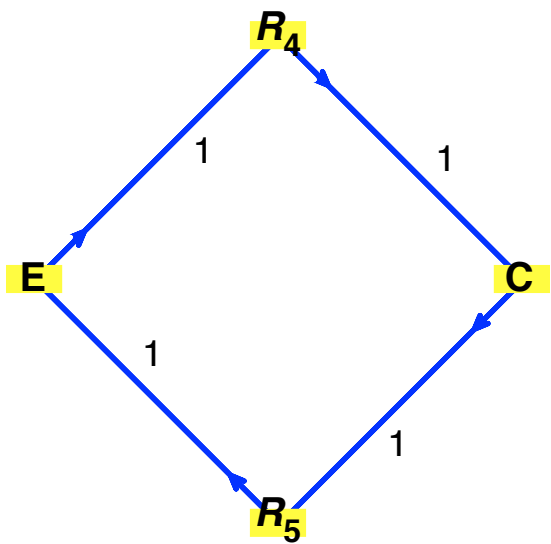


No. 19: injectiveEx3\_1473.csv

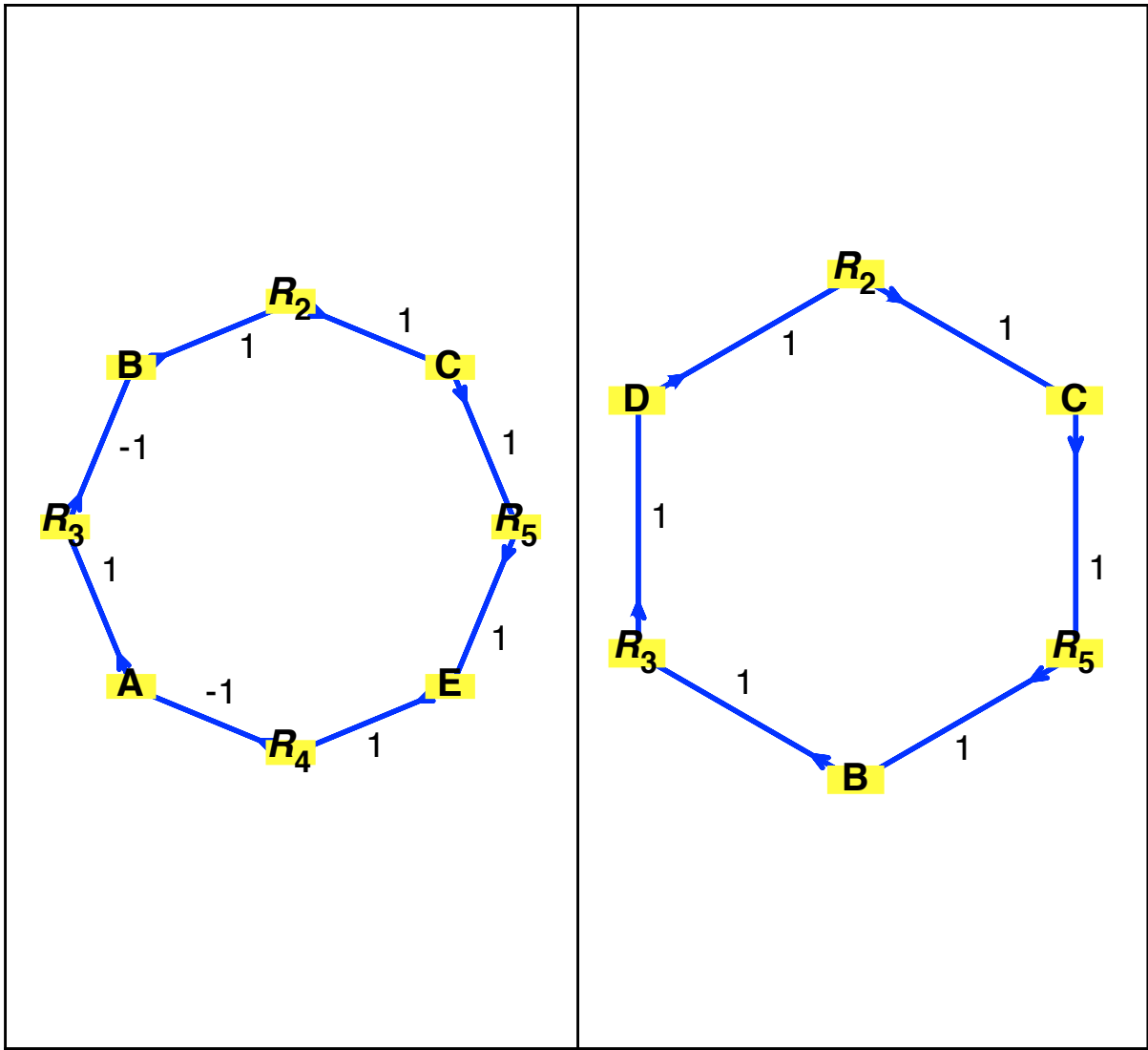




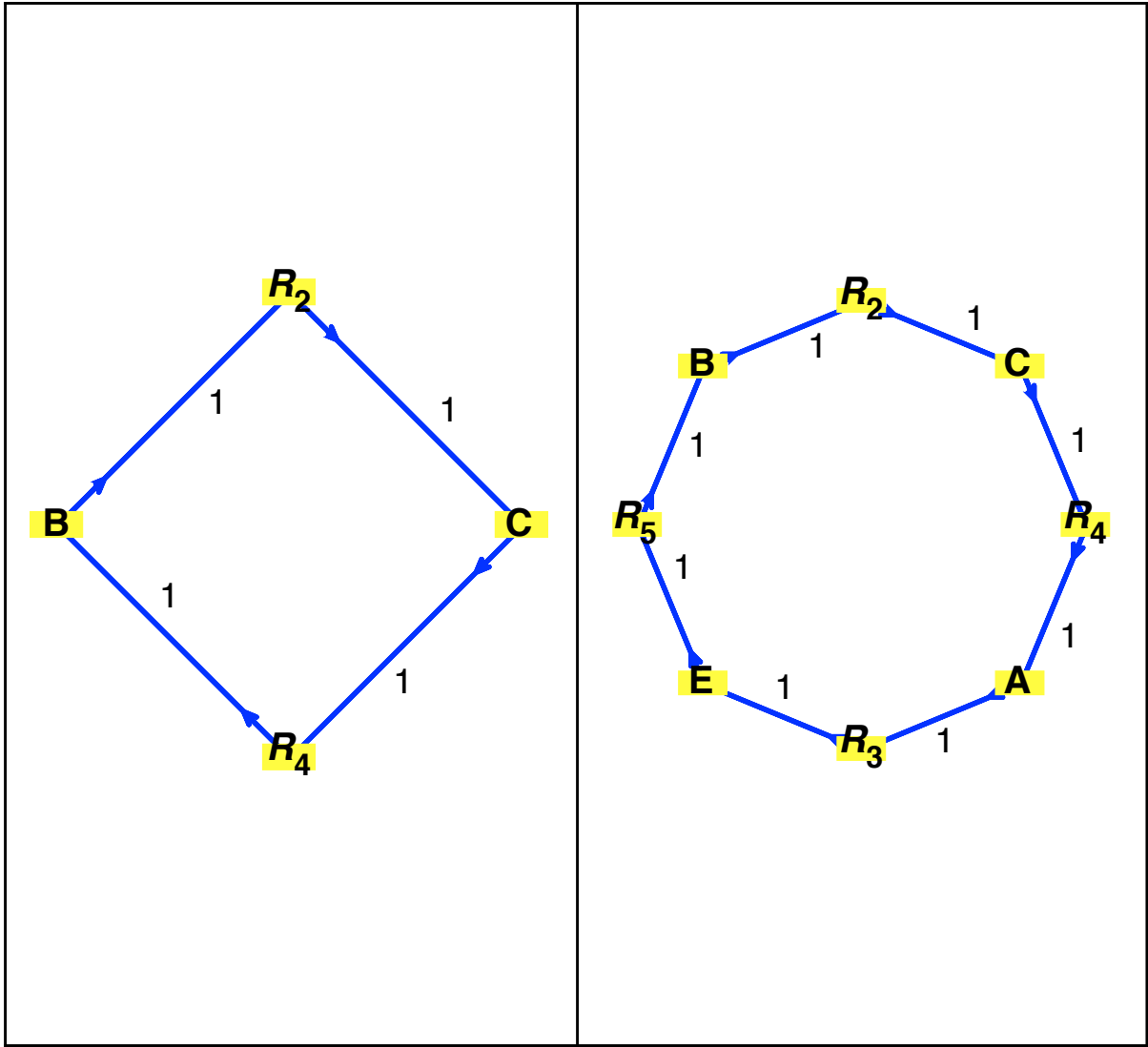
No. 20: injectiveEx3\_1474.csv



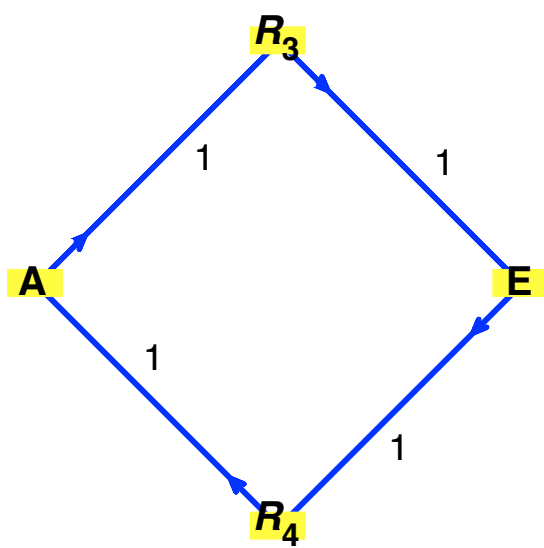
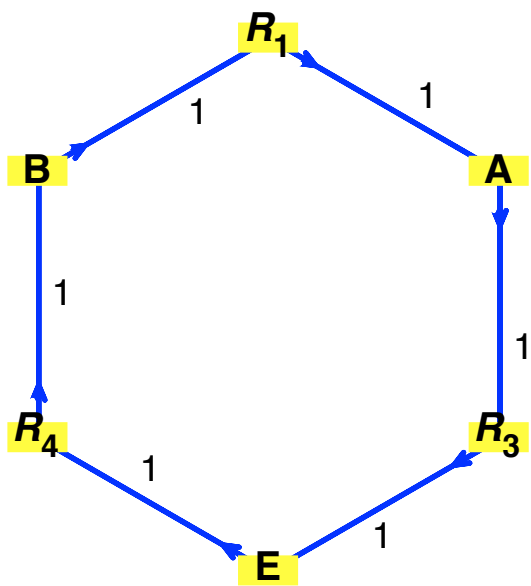


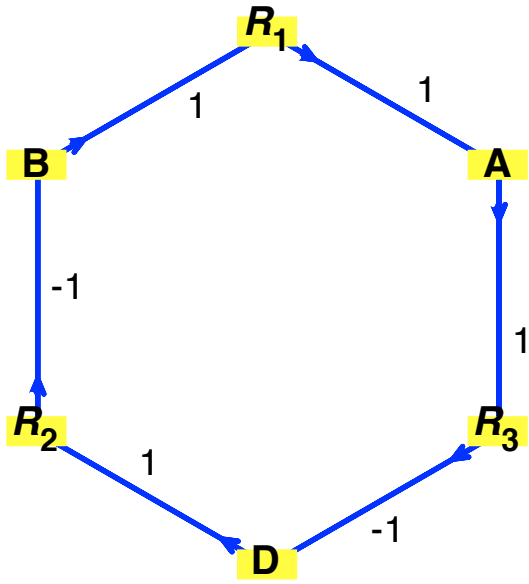


No. 21: injectiveEx3\_1516.csv

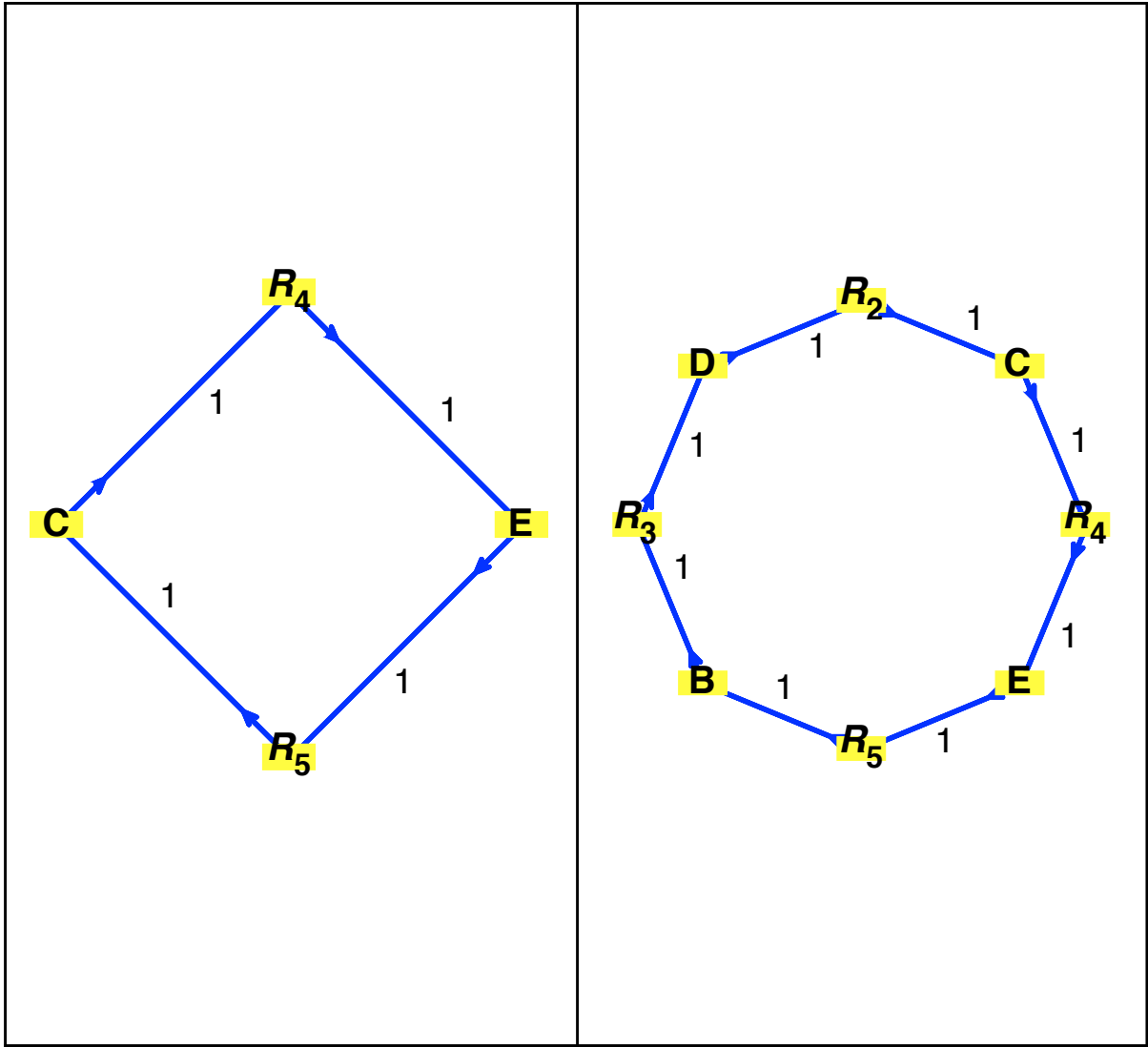


No. 22: injectiveEx3\_1518.csv

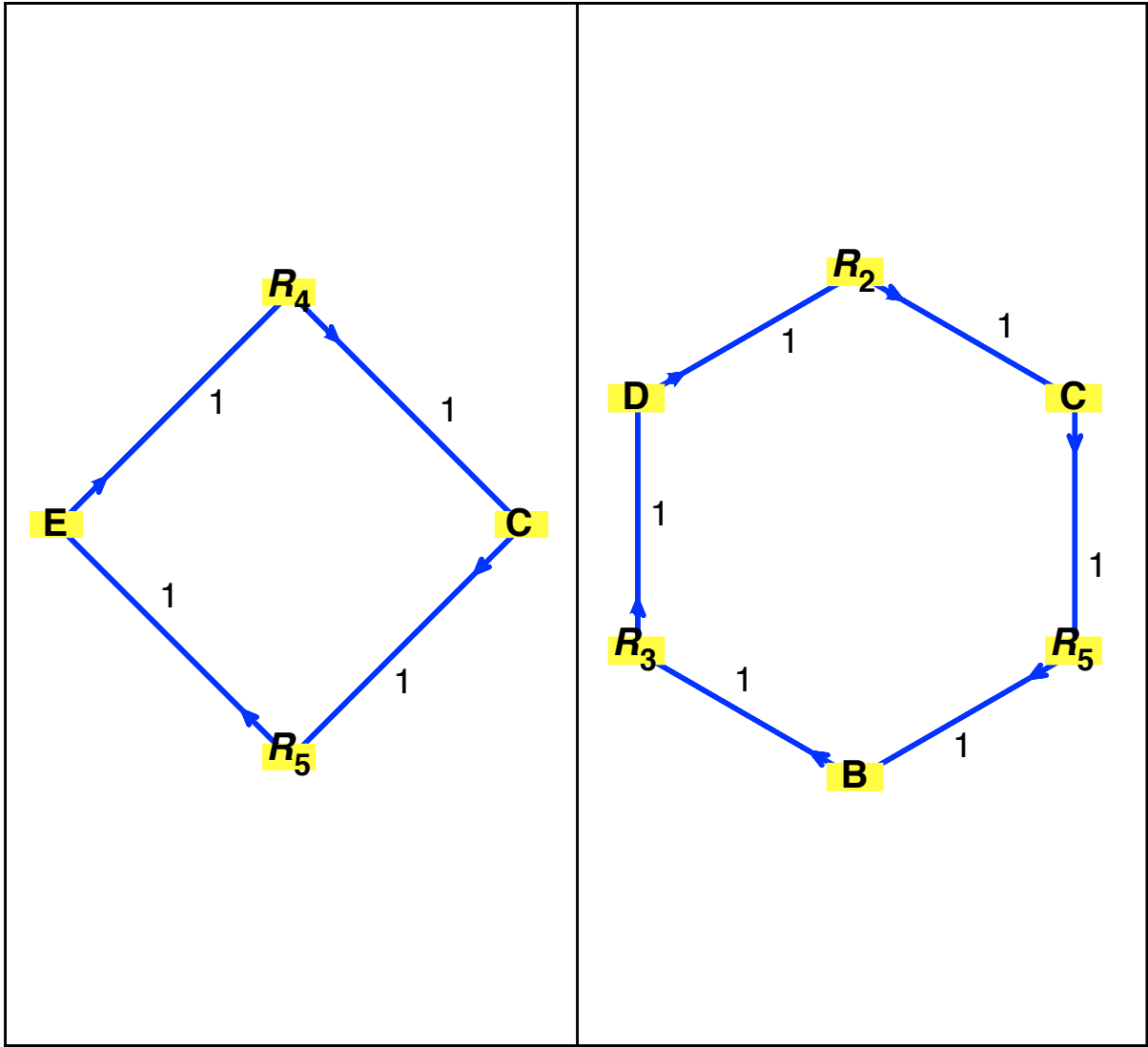




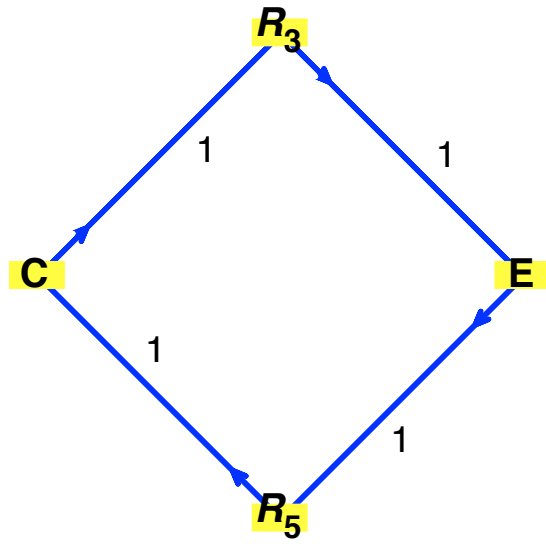
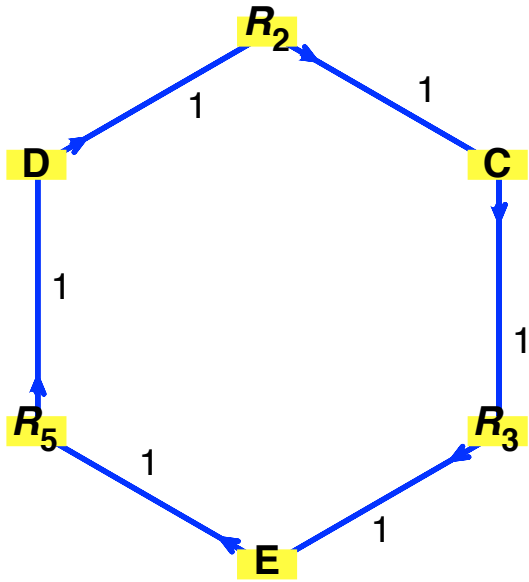
No. 23: injectiveEx3\_1721.csv



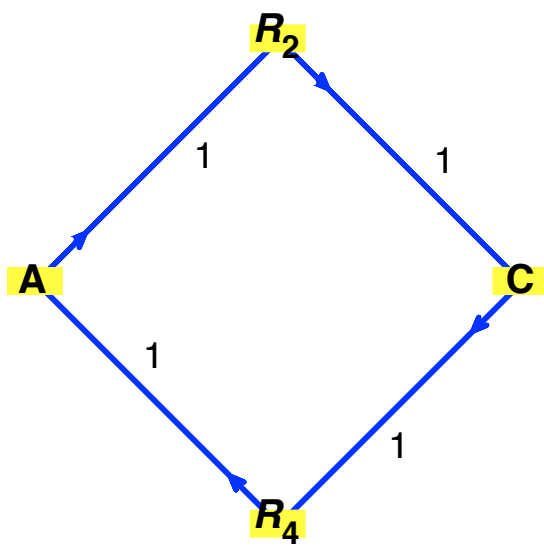
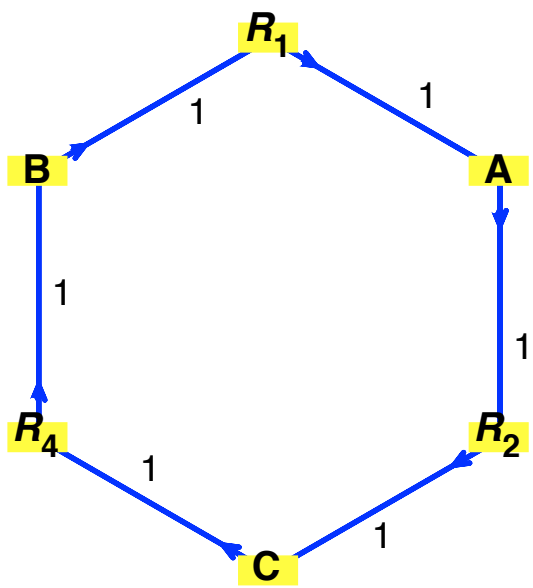
No. 24: injectiveEx3\_1722.csv



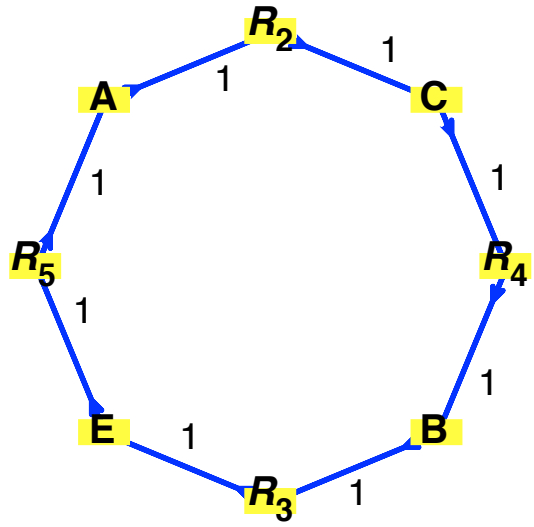
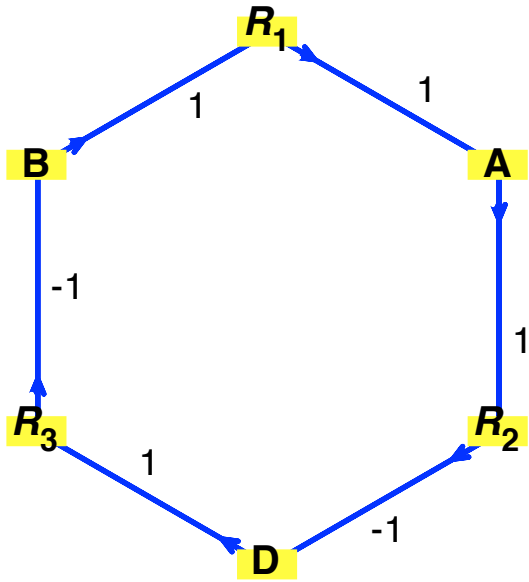
No. 25: injectiveEx3\_1743.csv



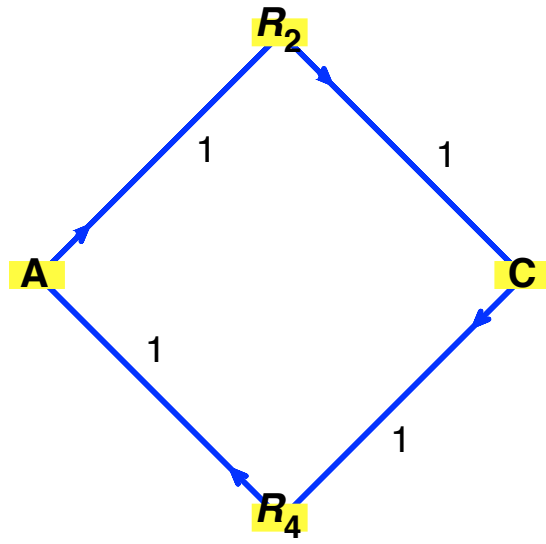
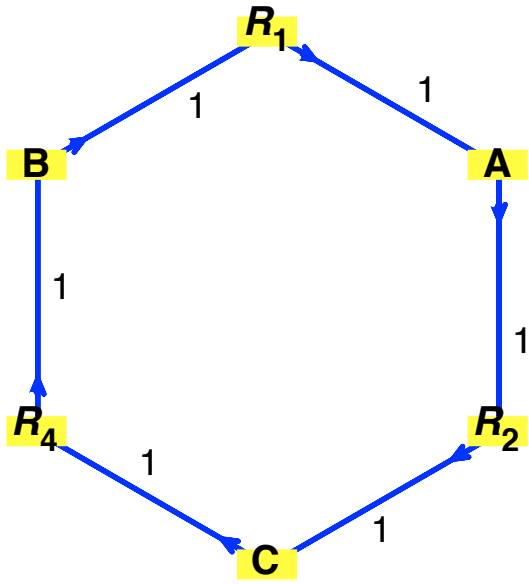
No. 26: injectiveEx3\_1796.csv

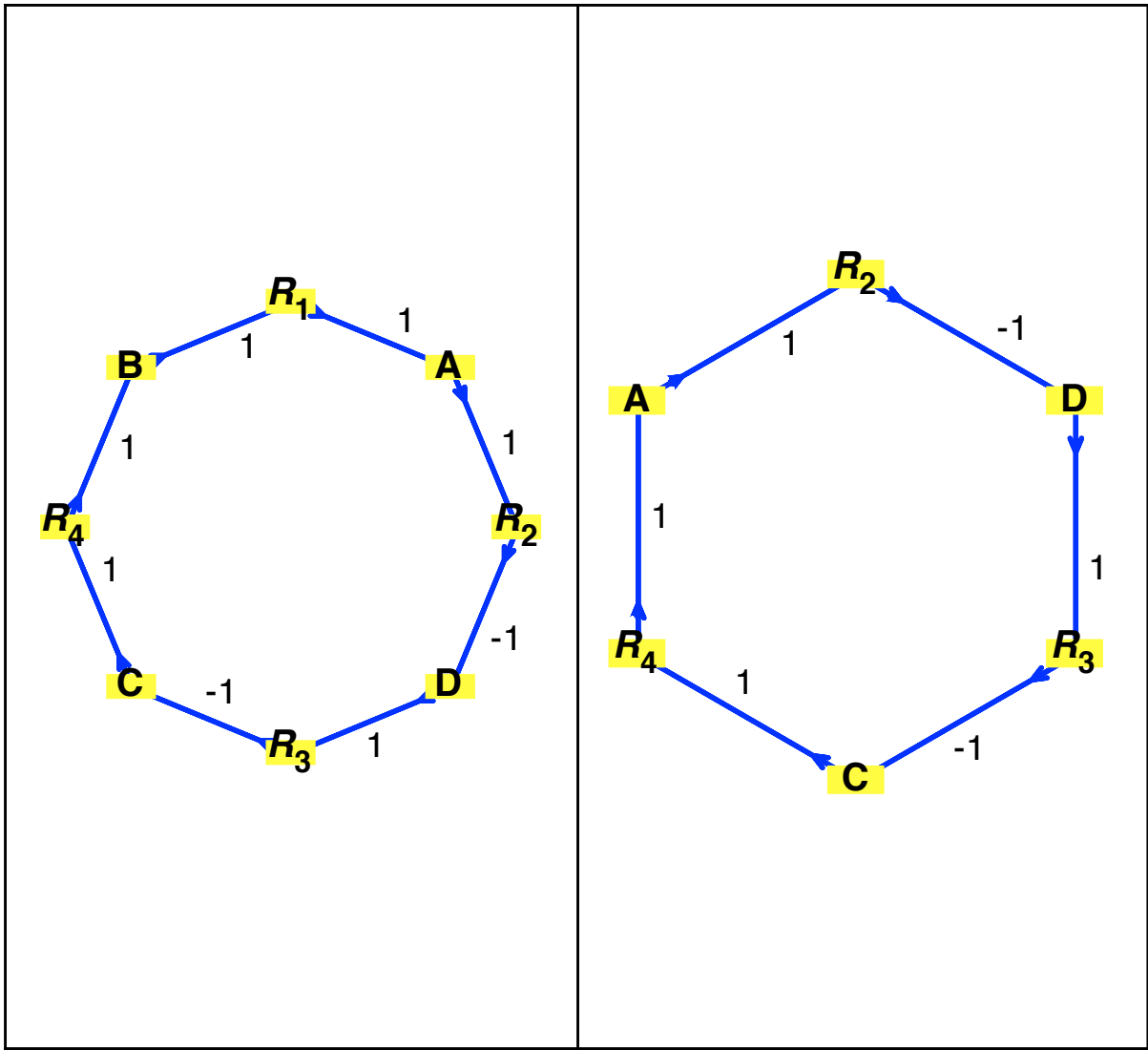




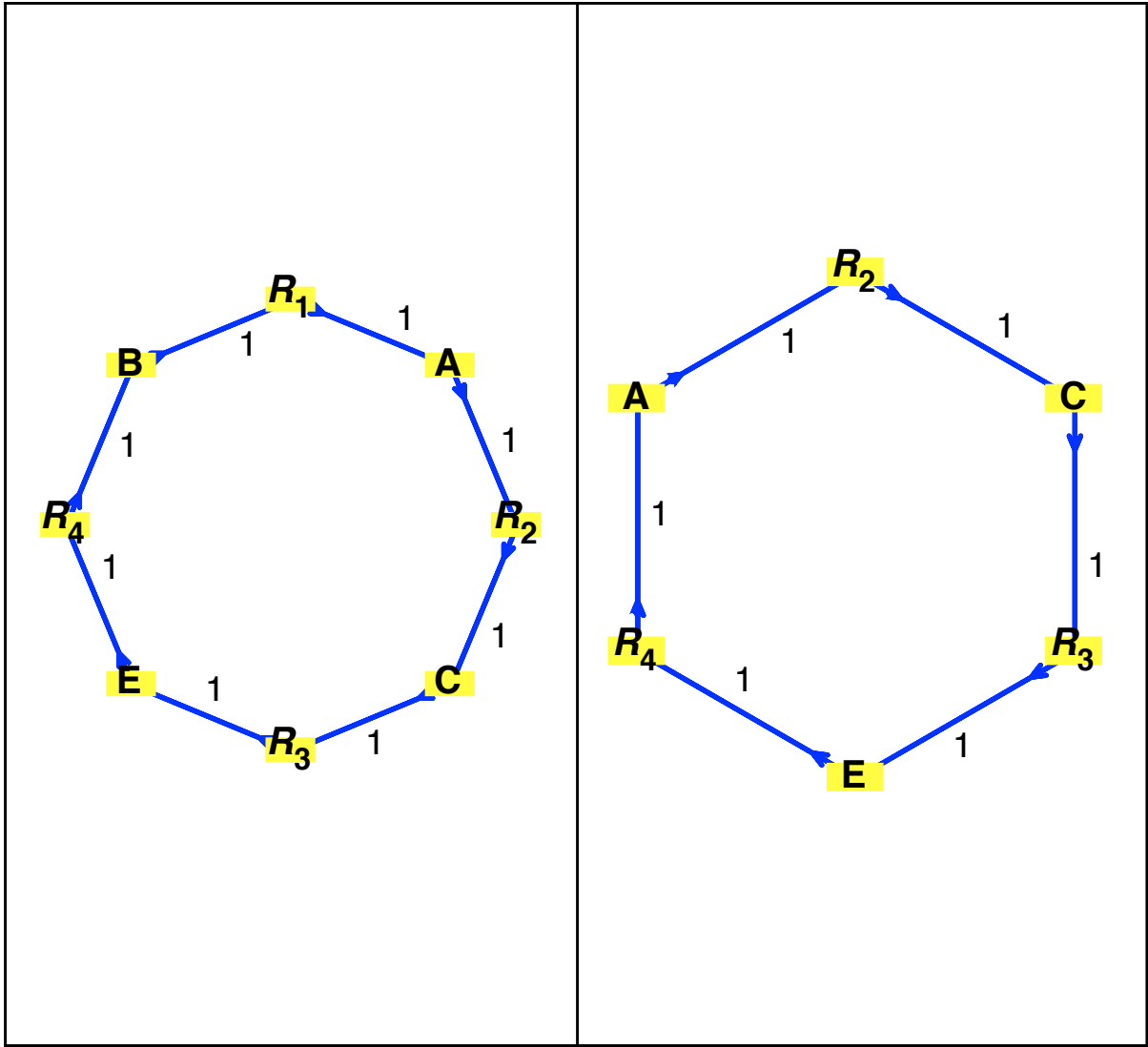


No. 27: injectiveEx3\_1814.csv

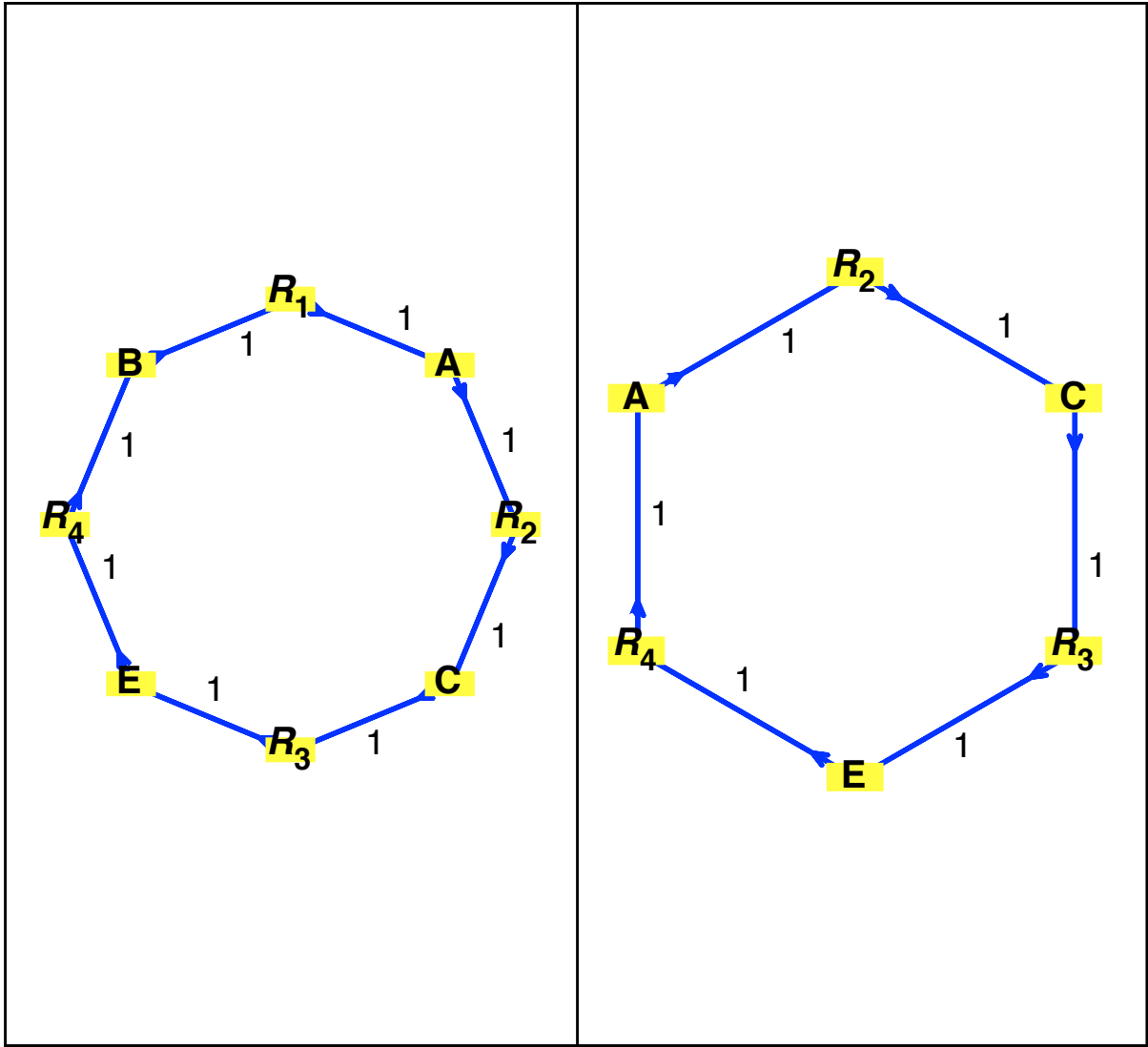




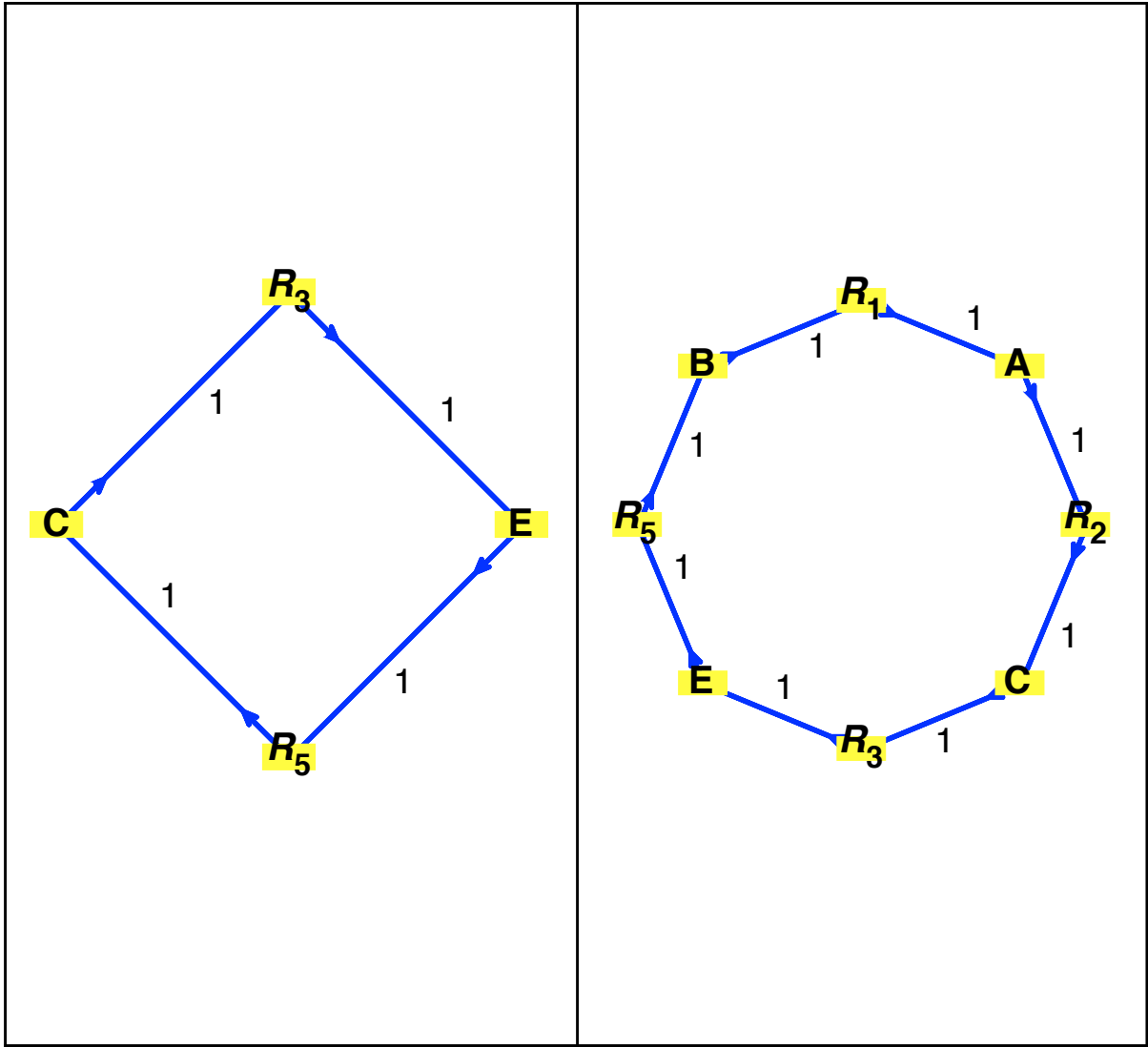
No. 28: injectiveEx3\_1816.csv



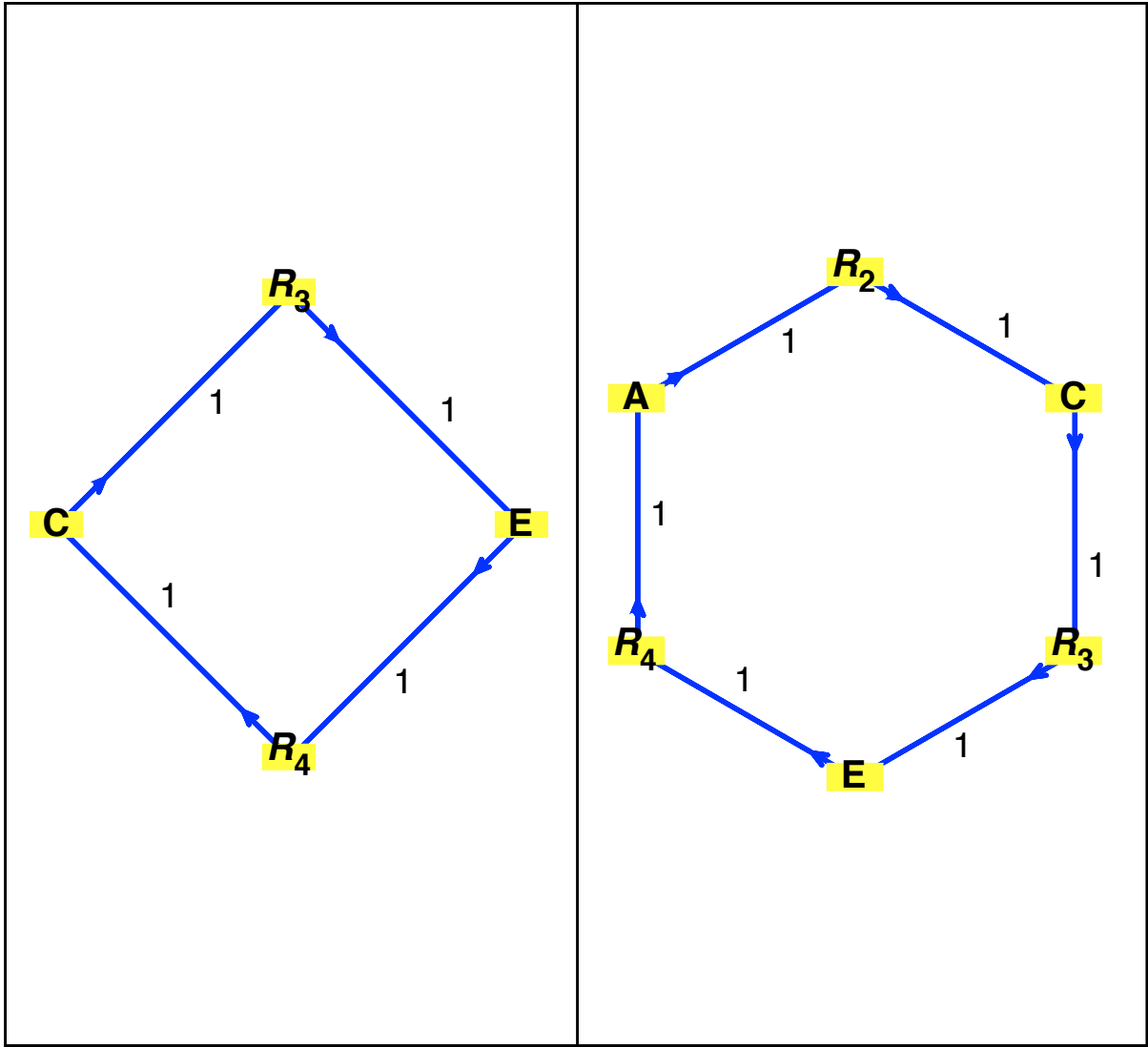
No. 29: injectiveEx3\_1817.csv



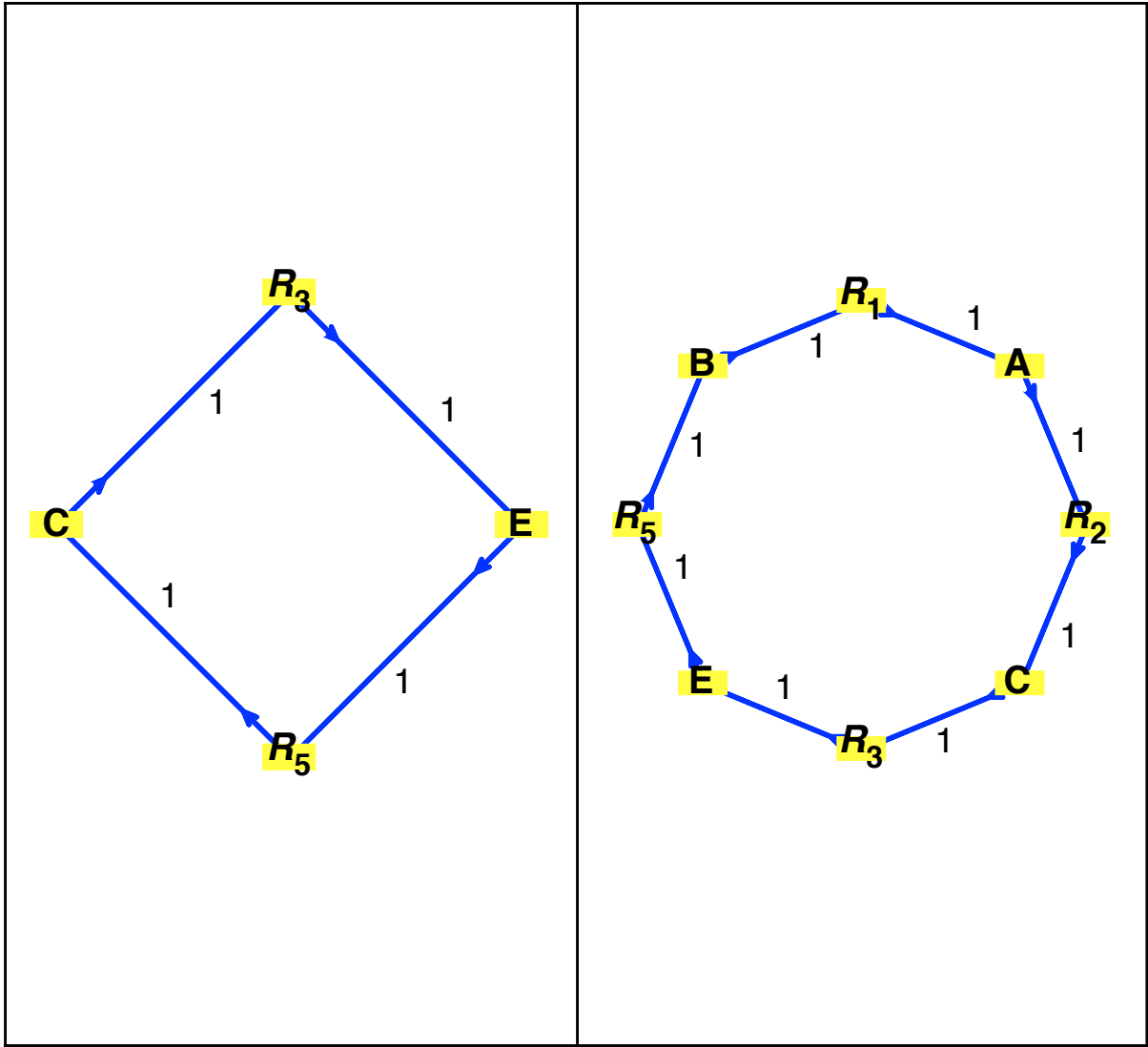
No. 30: injectiveEx3\_1819.csv



No. 31: injectiveEx3\_1820.csv

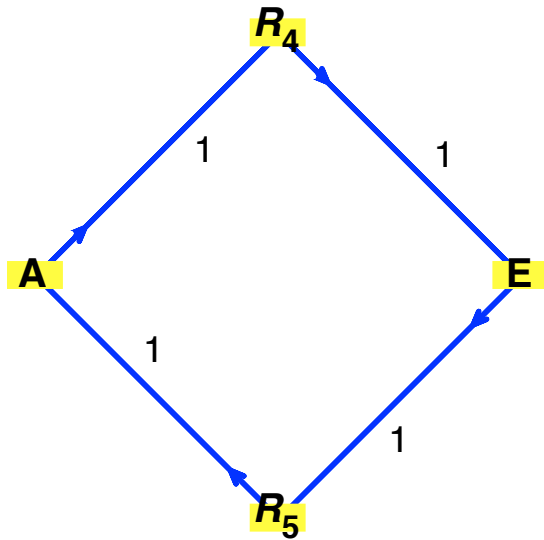
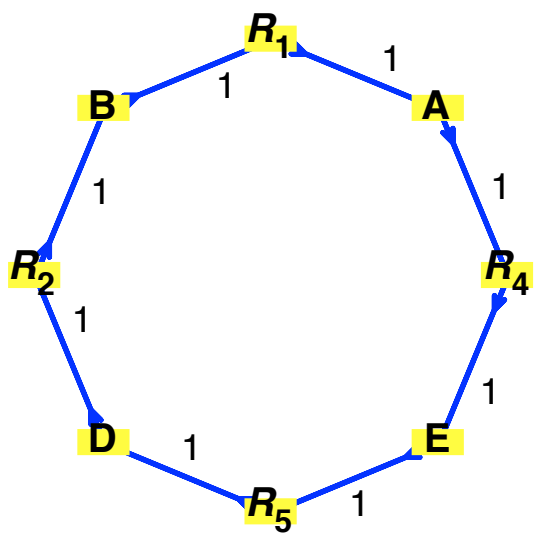


No. 32: injectiveEx3\_1821.csv

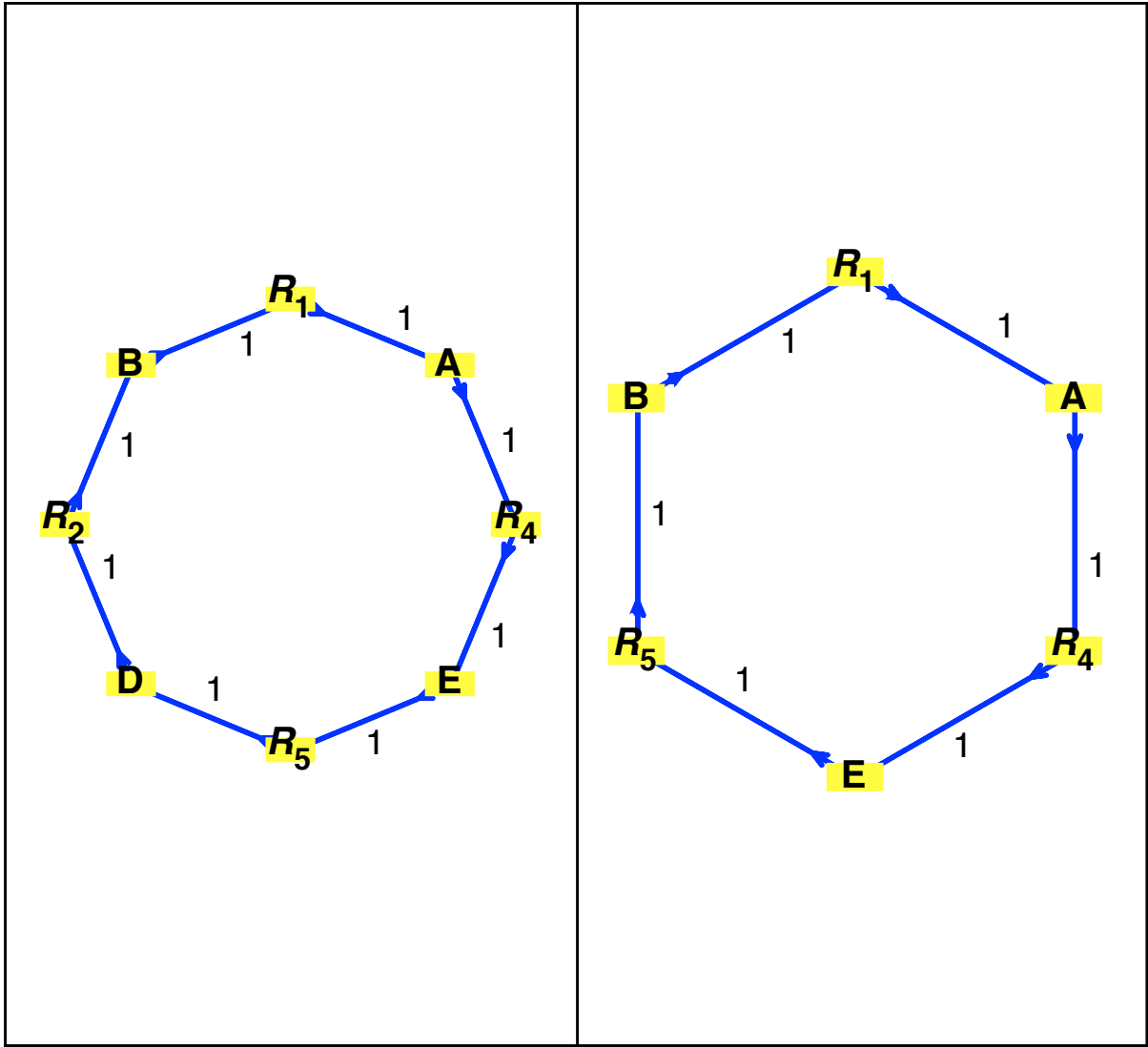


No. 33: injectiveEx3\_2945.csv

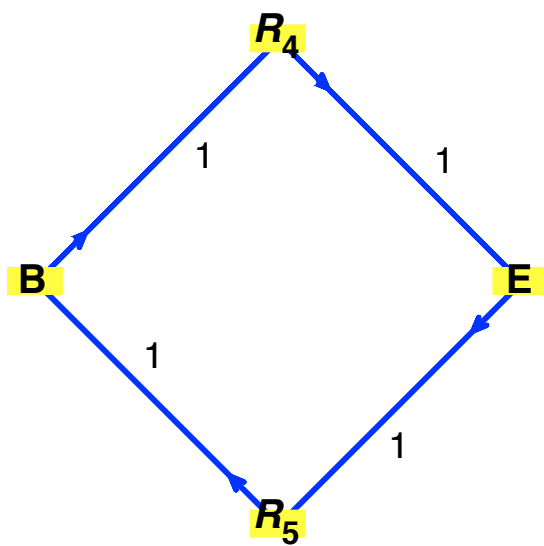
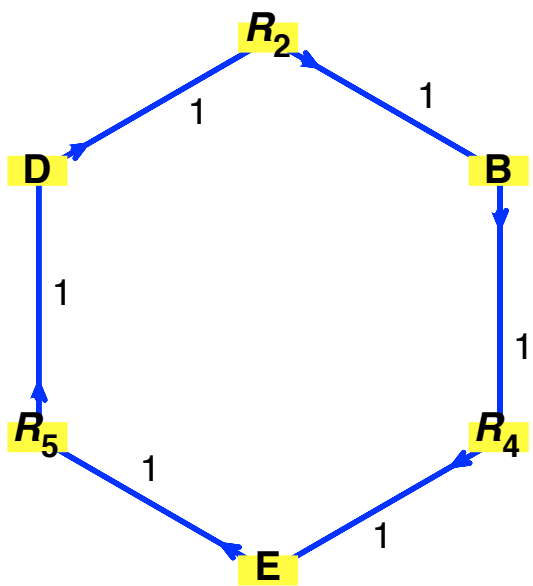


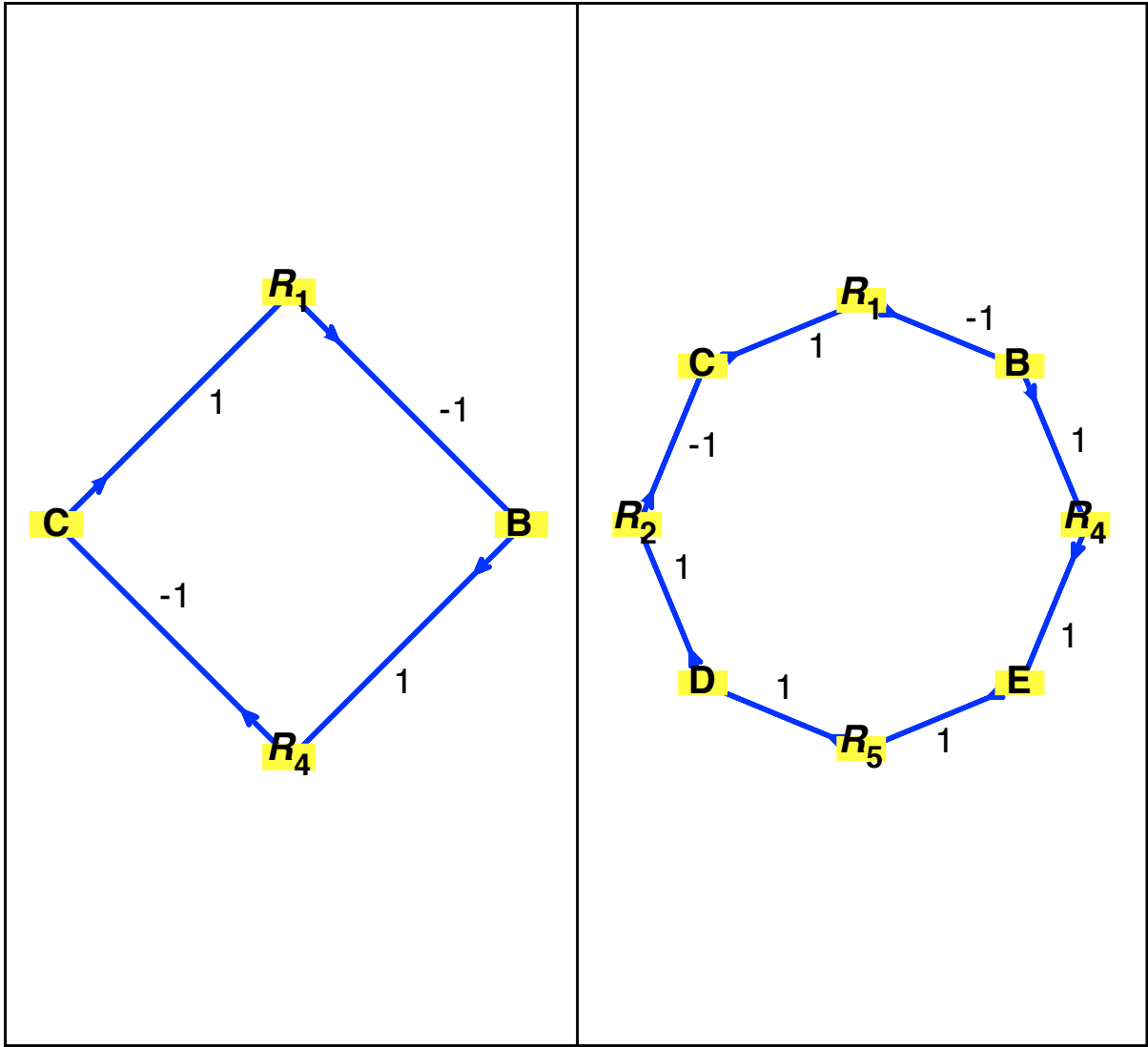


No. 34: injectiveEx3\_2946.csv

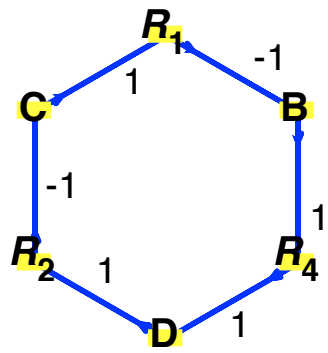
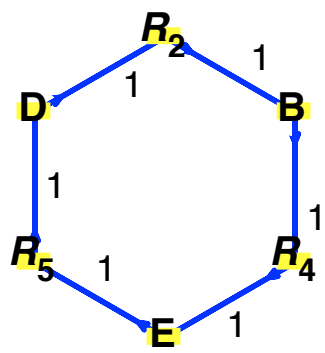
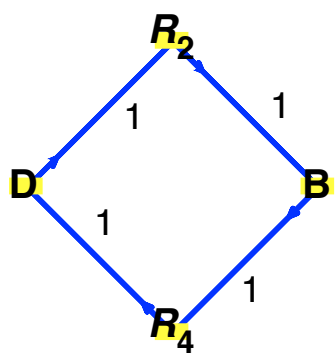


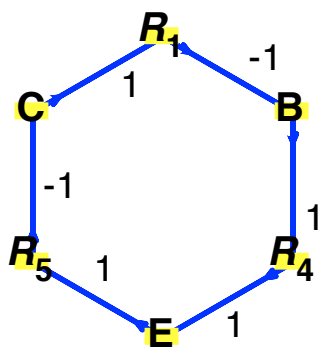
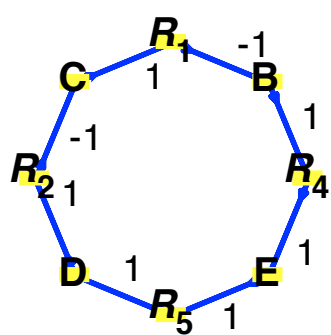
No. 35: injectiveEx3\_2954.csv



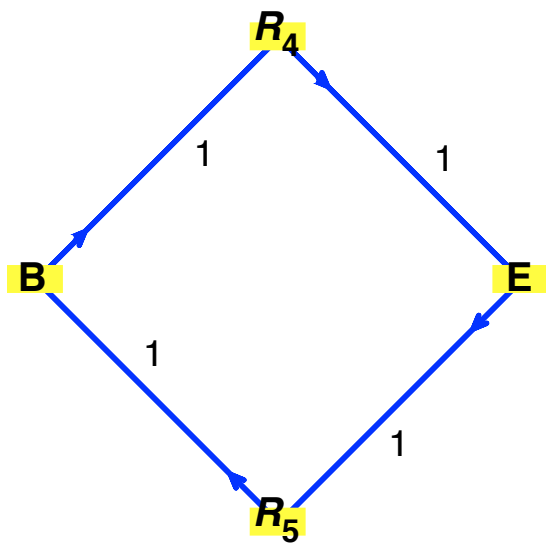
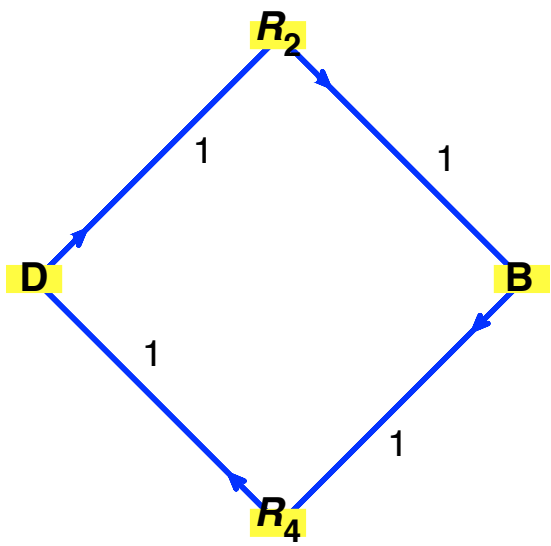


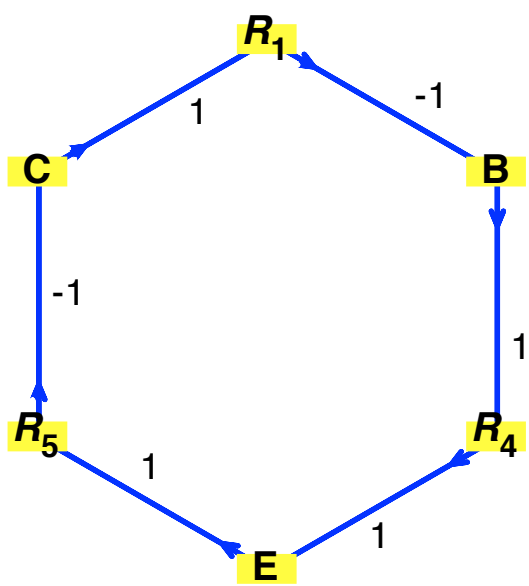
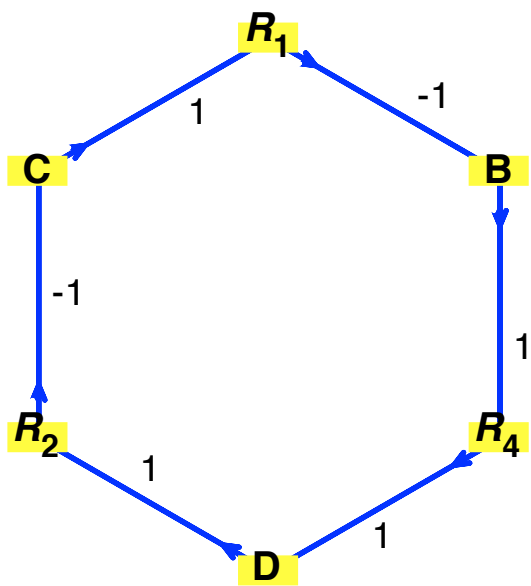
No. 36: injectiveEx3\_2955.csv





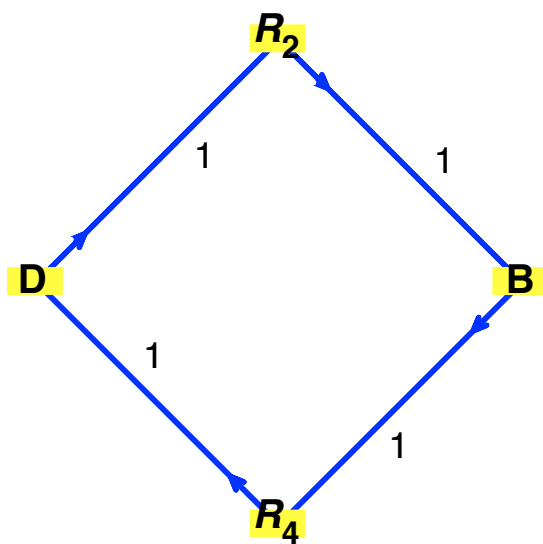
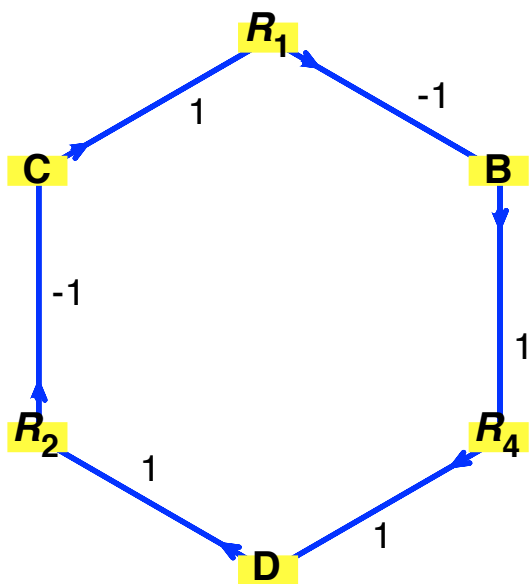
No. 37: injectiveEx3\_2957.csv



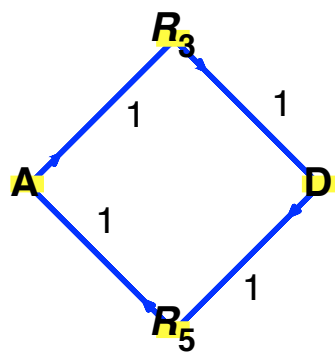
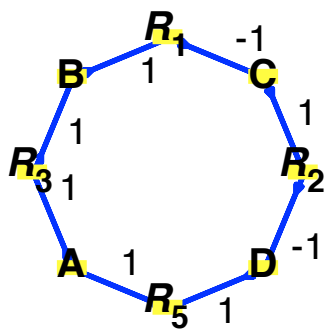
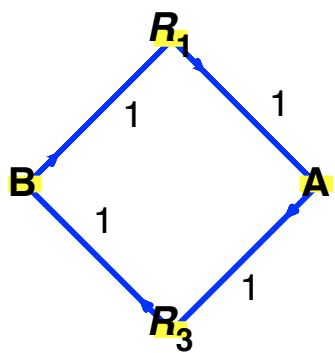


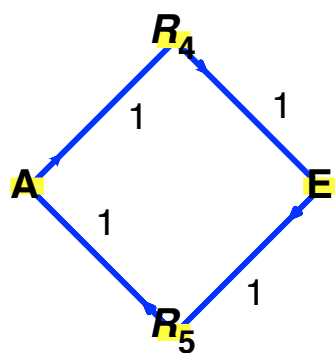
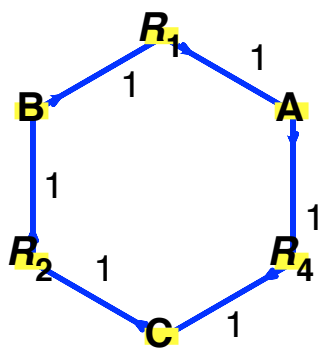
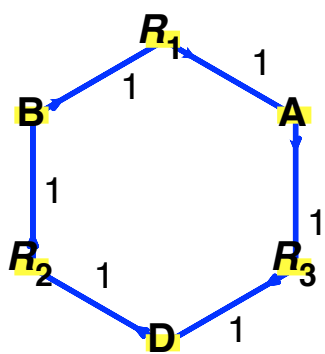
No. 38: injectiveEx3\_2958.csv

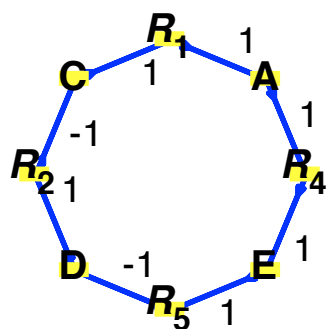
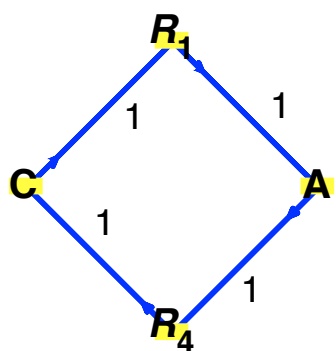




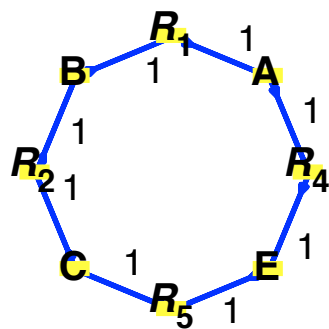
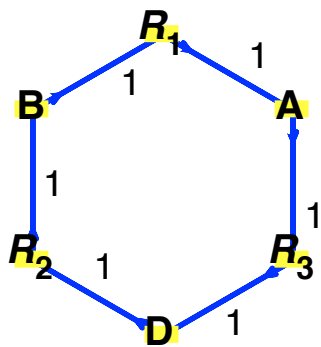
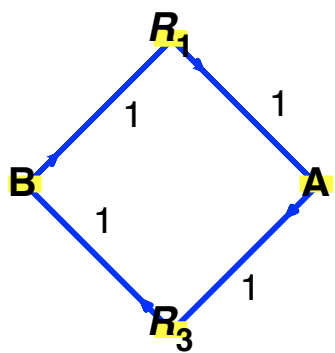
No. 39: injectiveEx3\_2965.csv

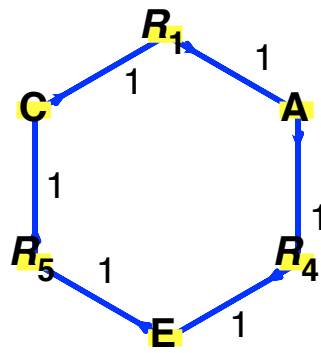
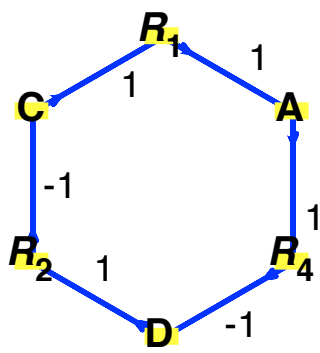
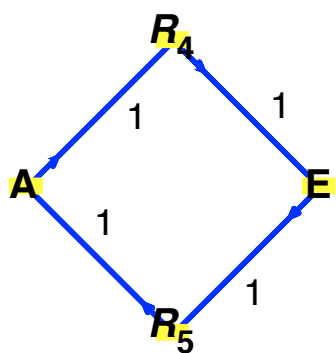




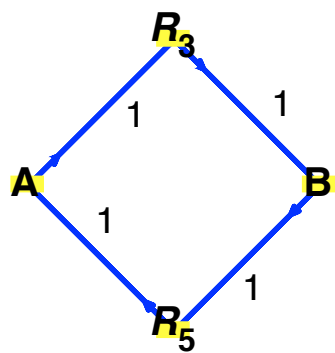
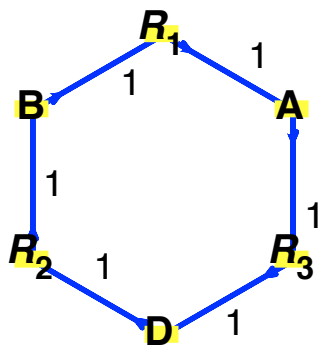
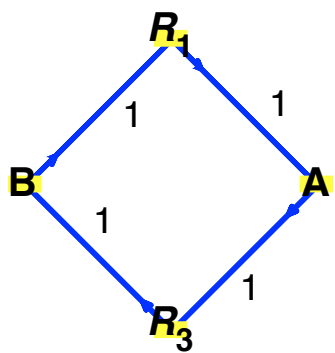


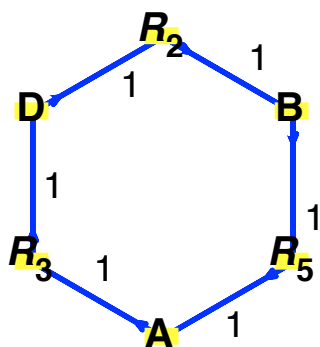
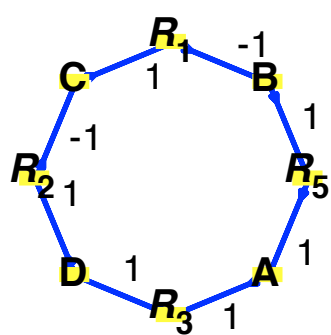
No. 40: injectiveEx3\_2969.csv





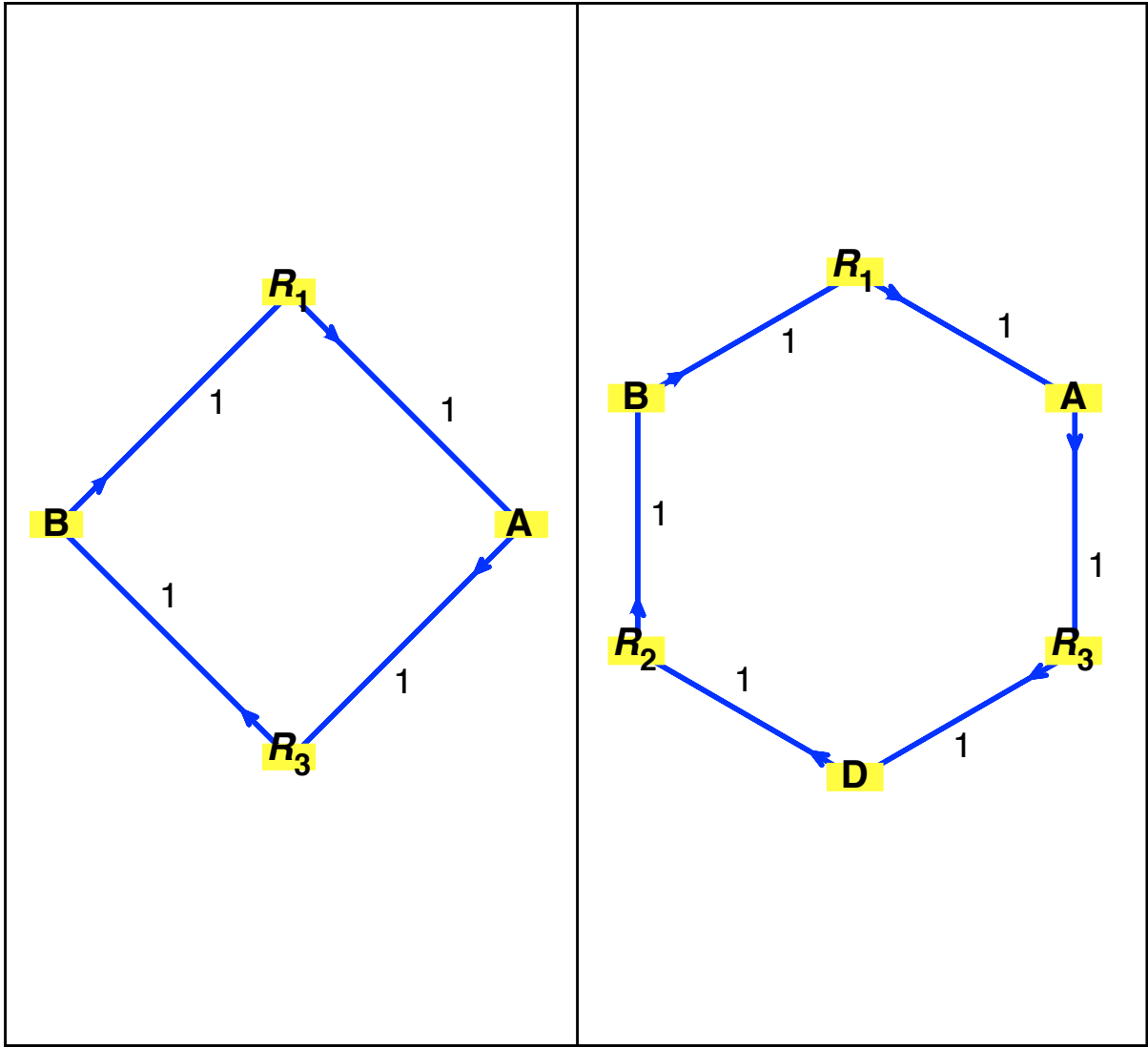
No. 41: injectiveEx3\_2970.csv



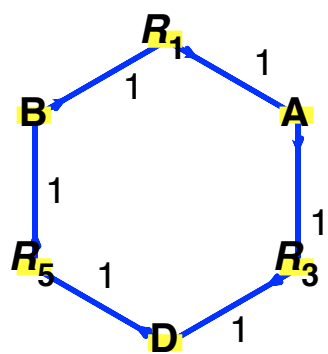
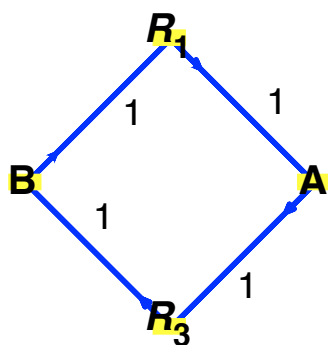
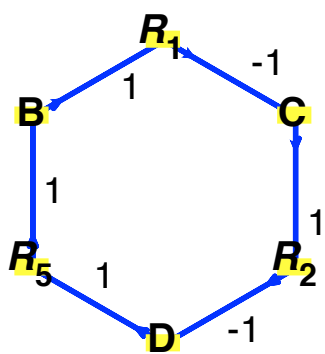


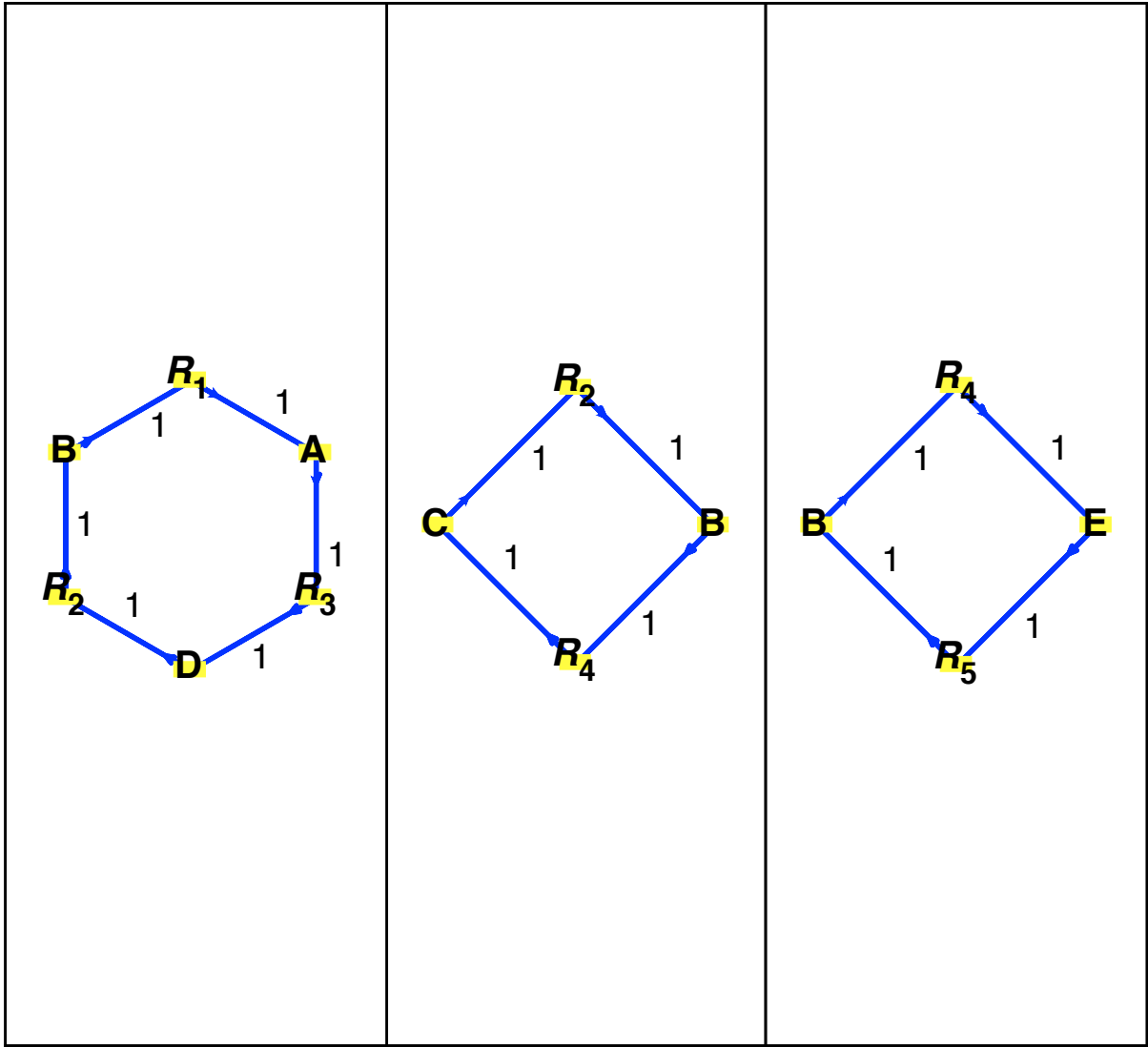
No. 42: injectiveEx3\_2971.csv



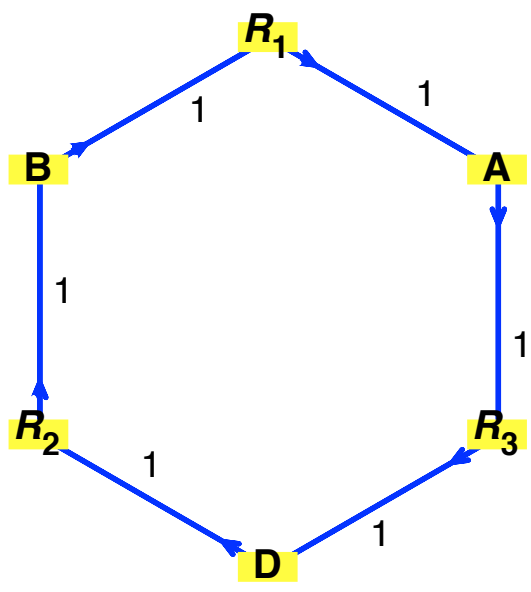
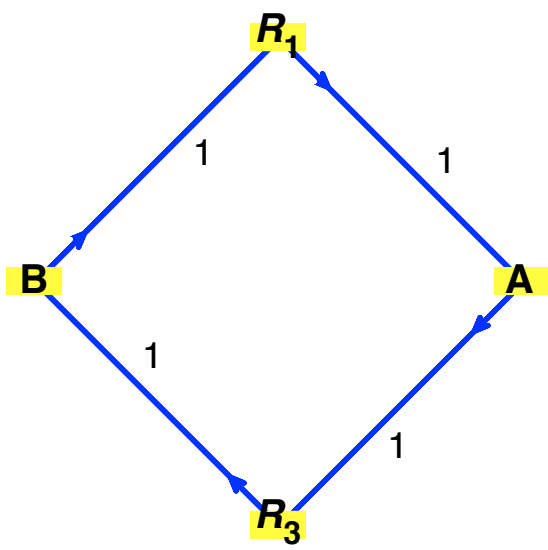


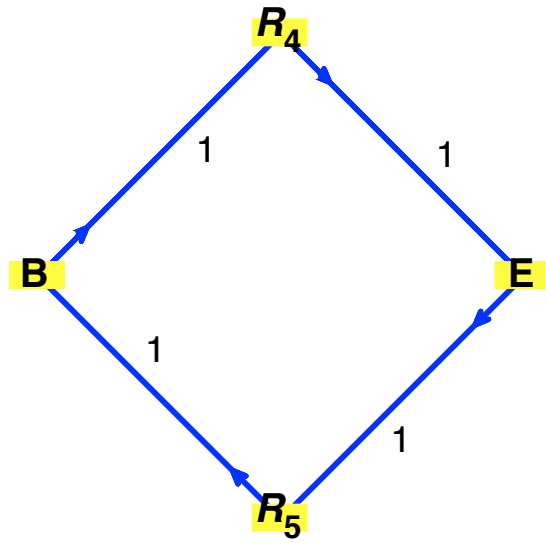
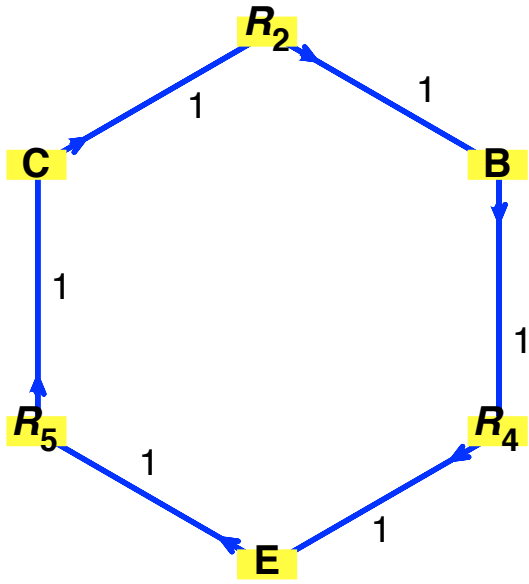
No. 43: injectiveEx3\_2972.csv



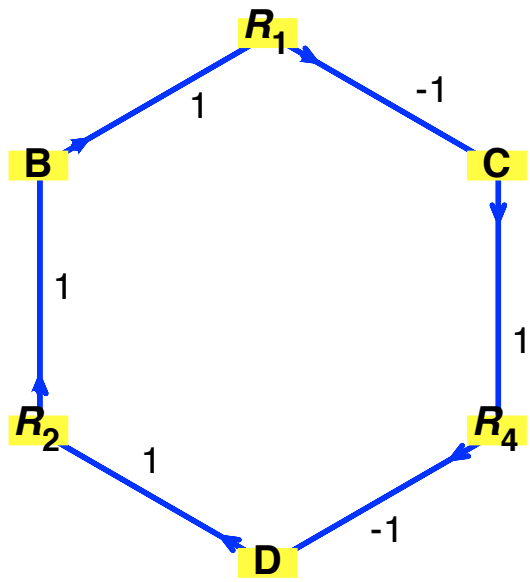
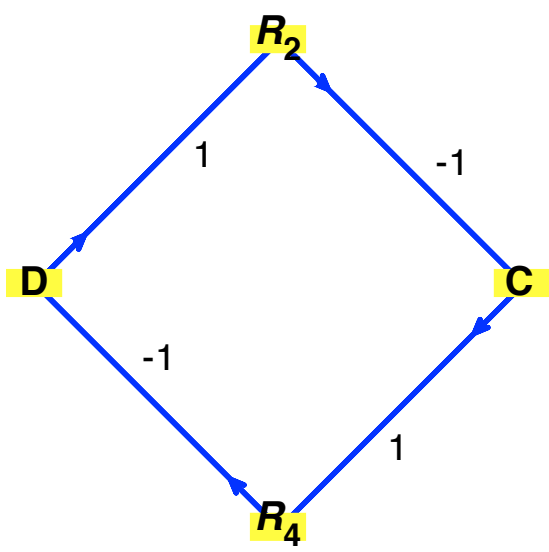


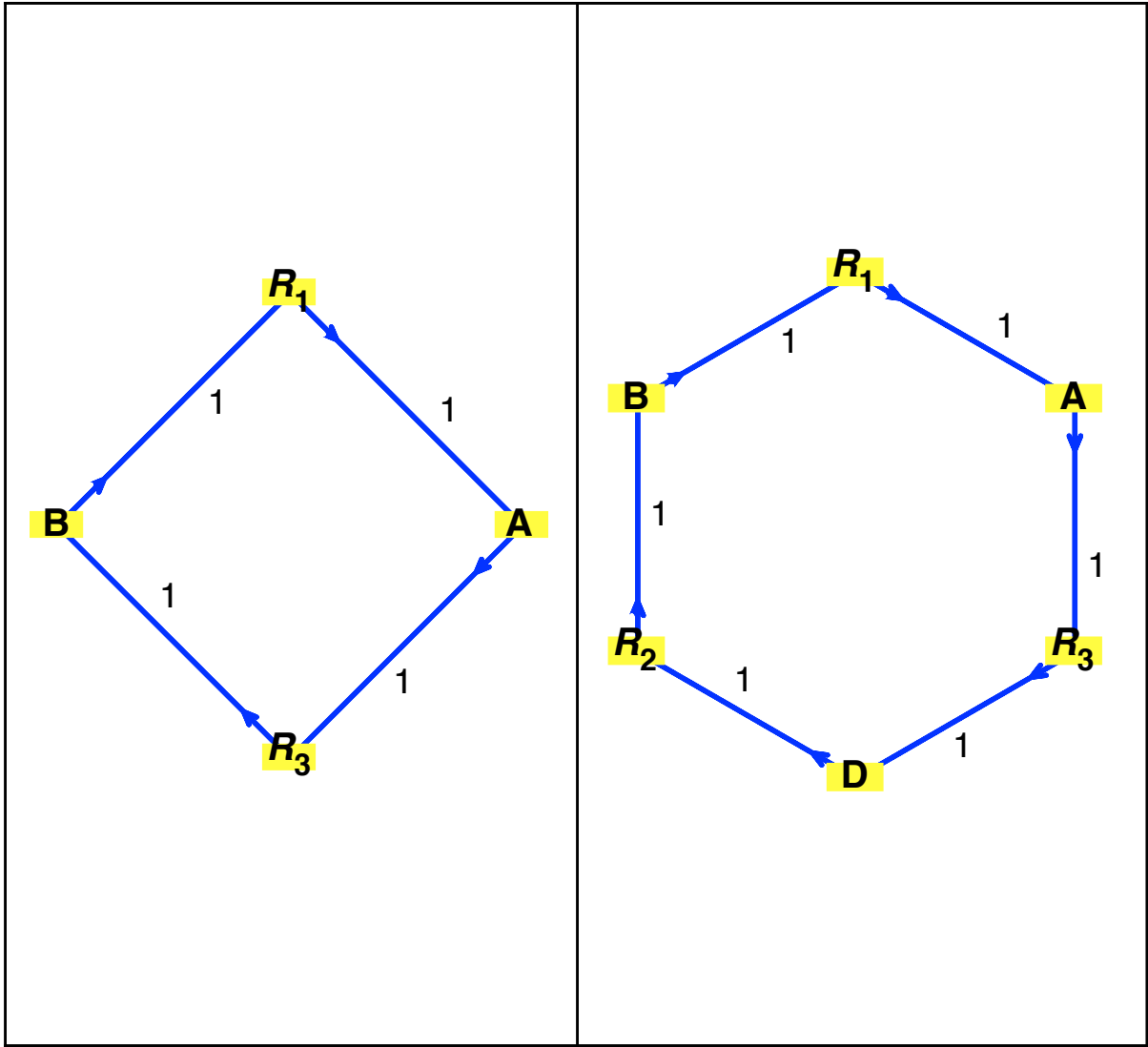
No. 44: injectiveEx3\_2974.csv



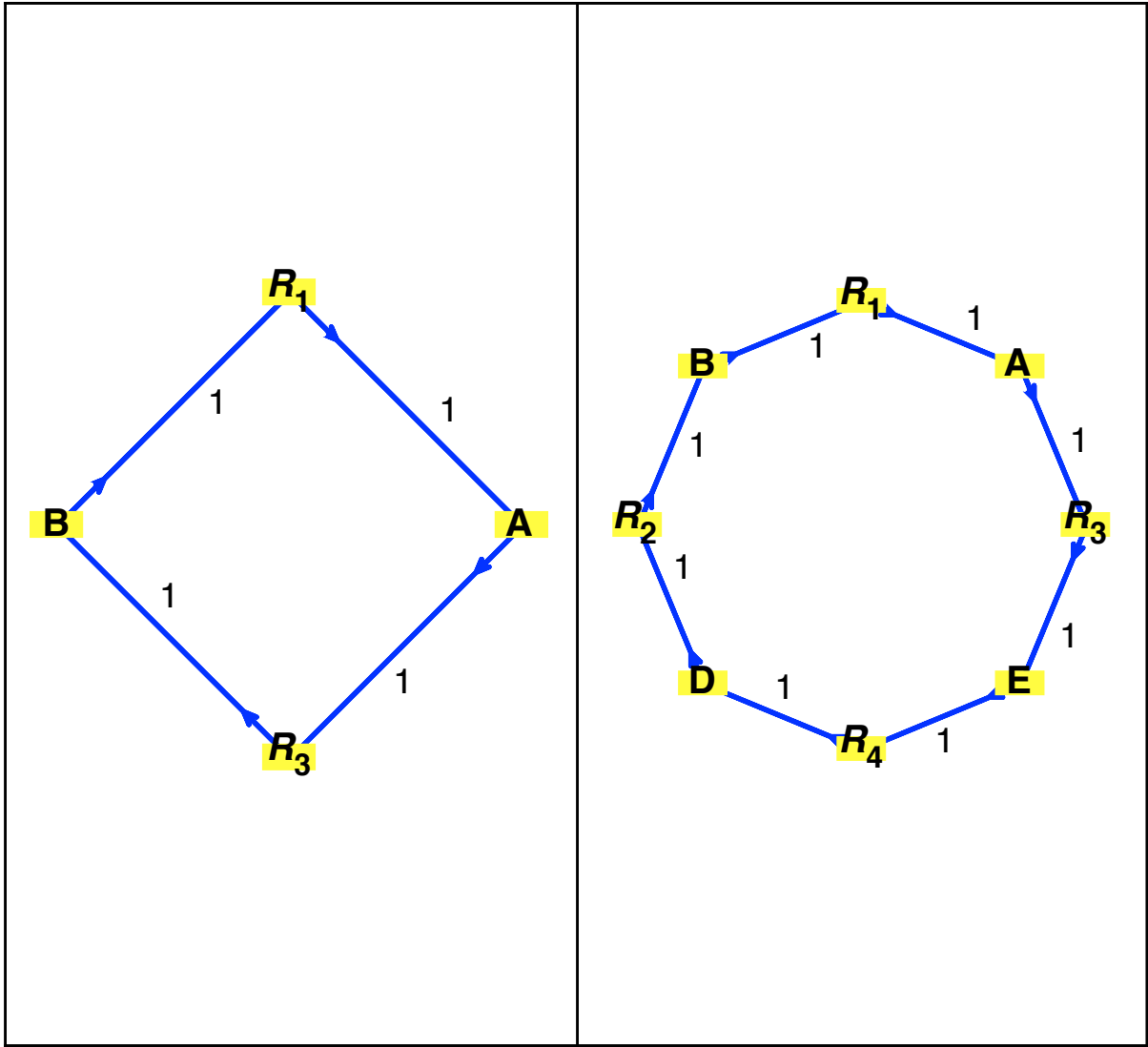


No. 45: injectiveEx3\_2975.csv



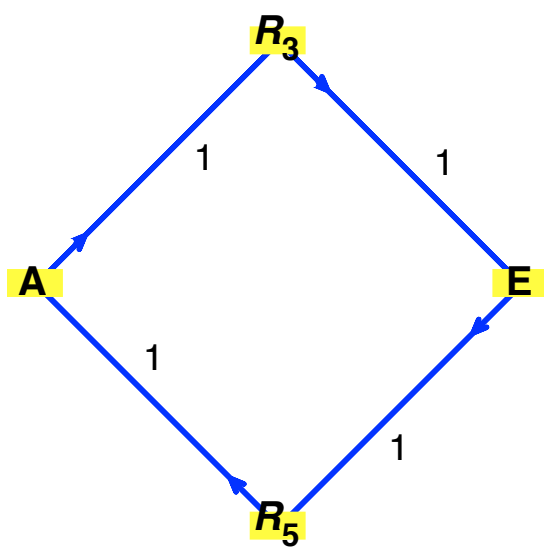
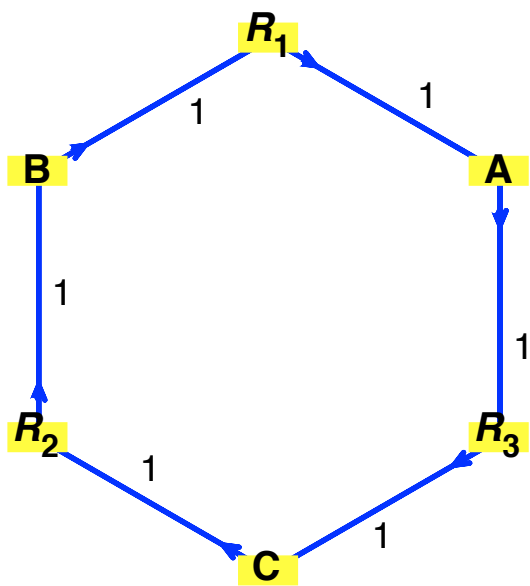


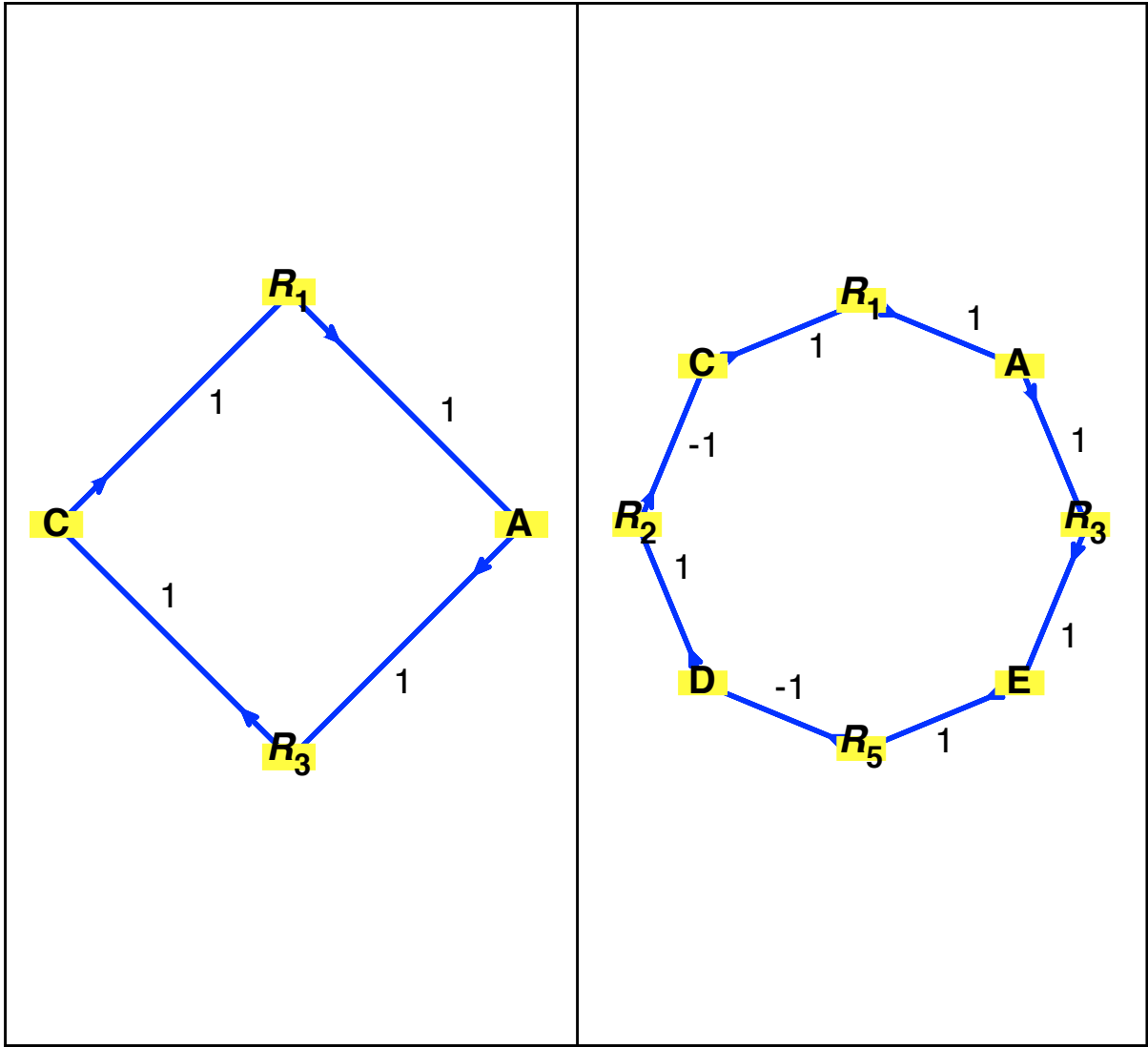
No. 46: injectiveEx3\_2980.csv



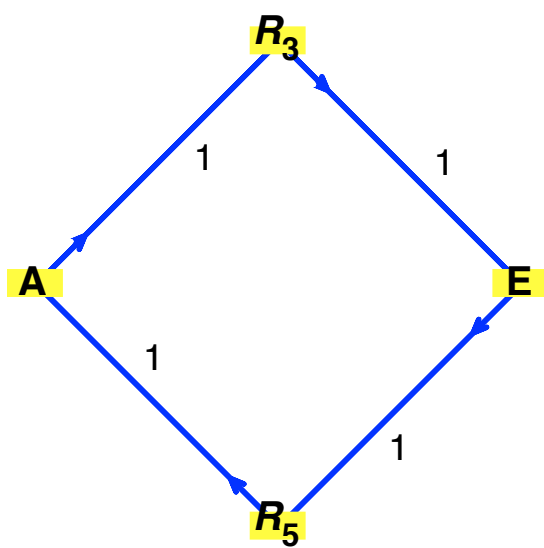
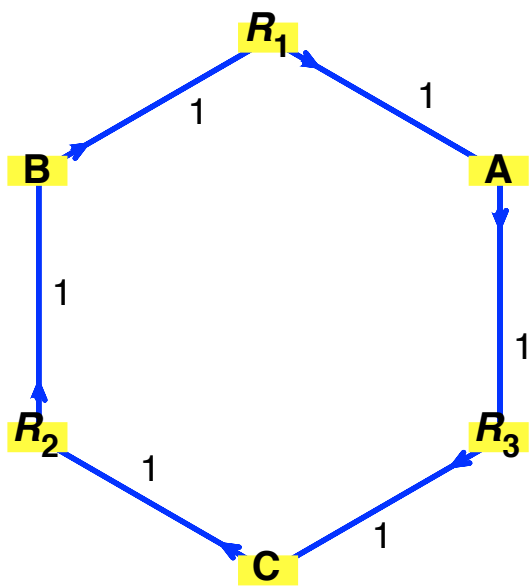
No. 47: injectiveEx3\_2983.csv

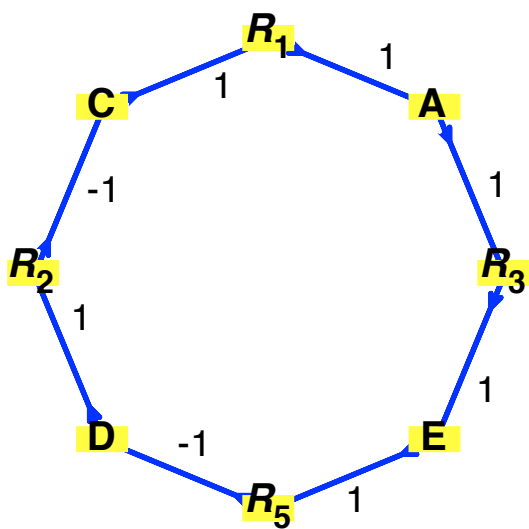
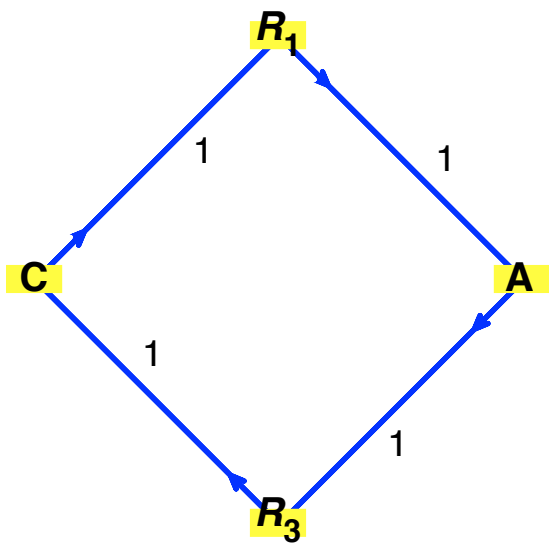




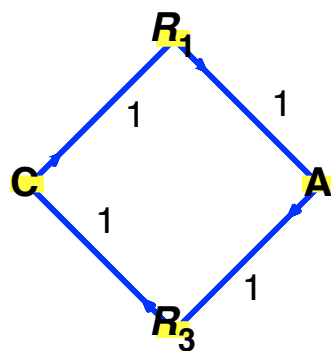
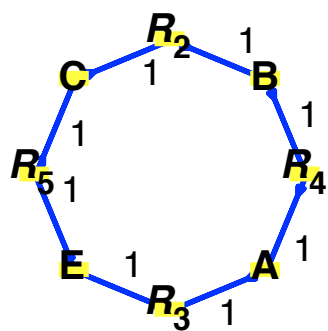
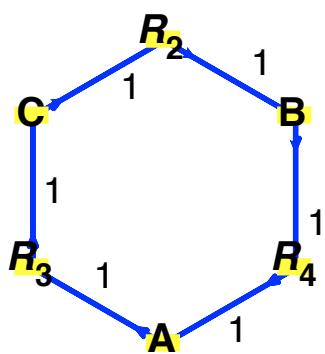


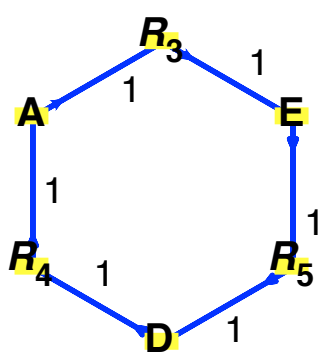
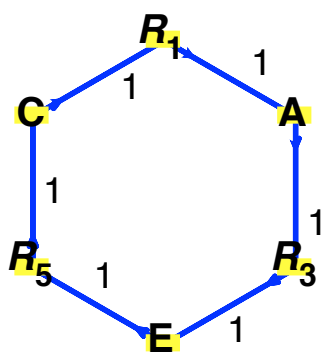
No. 48: injectiveEx3\_2986.csv



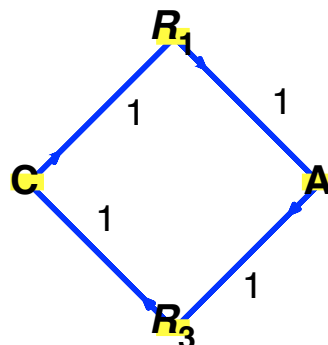
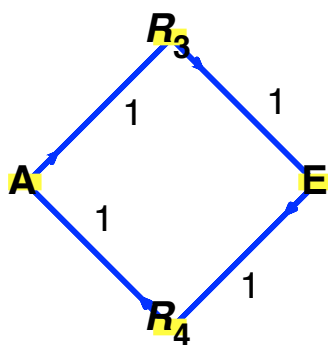
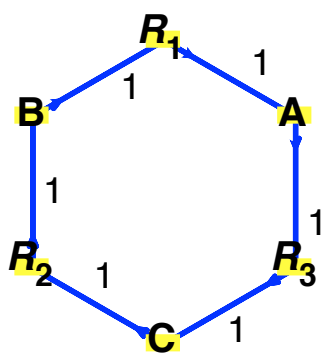


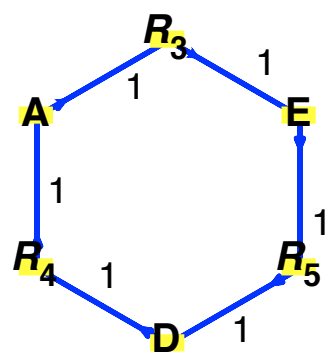
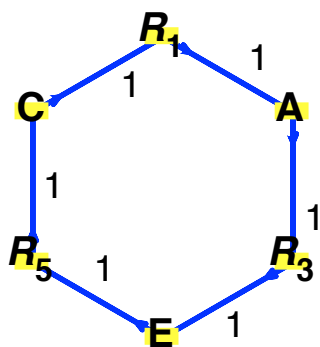
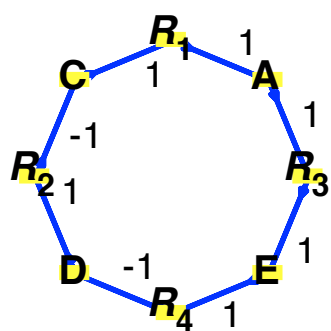
No. 49: injectiveEx3\_2990.csv





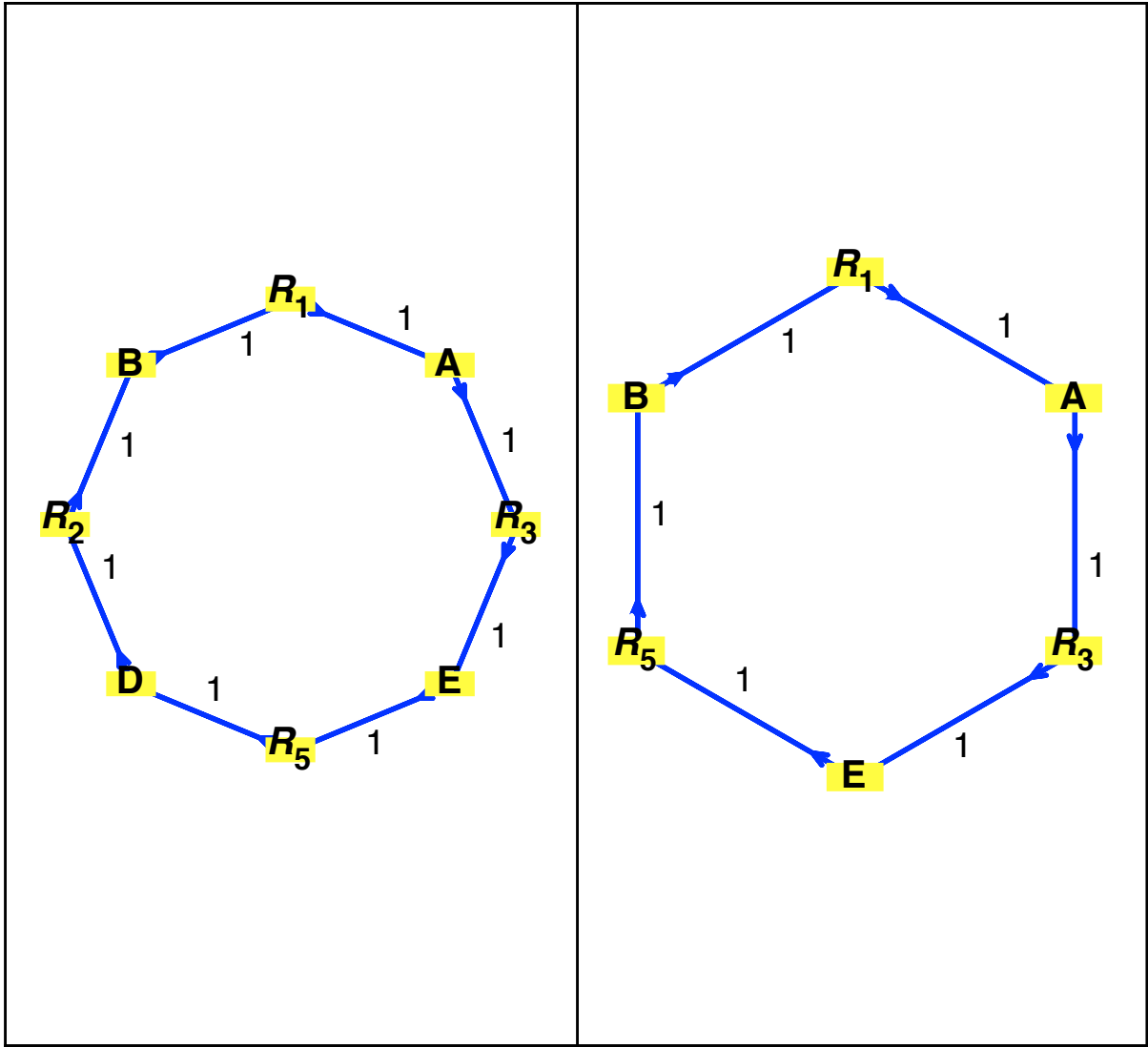
No. 50: injectiveEx3\_2992.csv



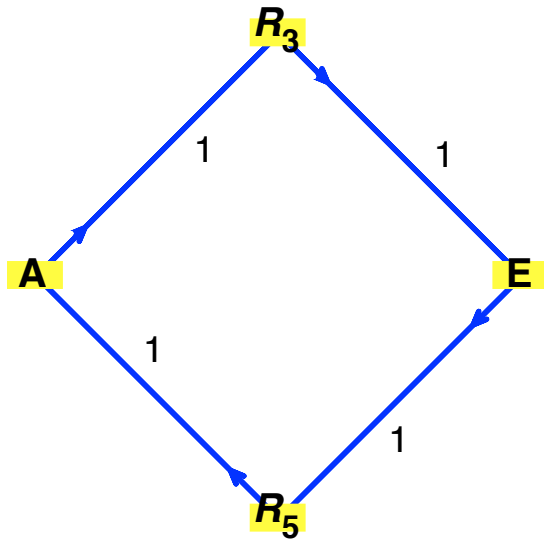
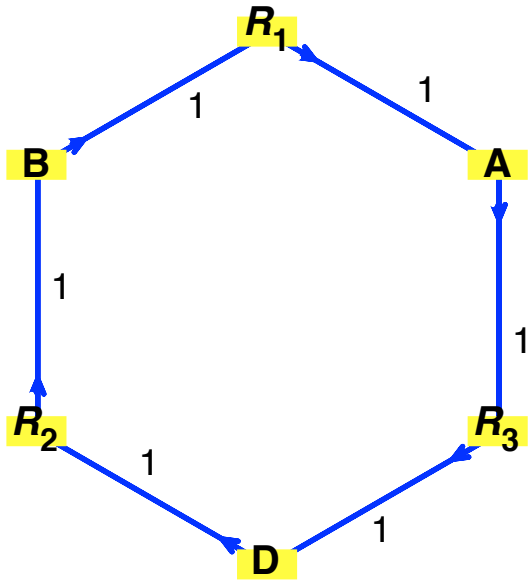


No. 51: injectiveEx3\_2996.csv

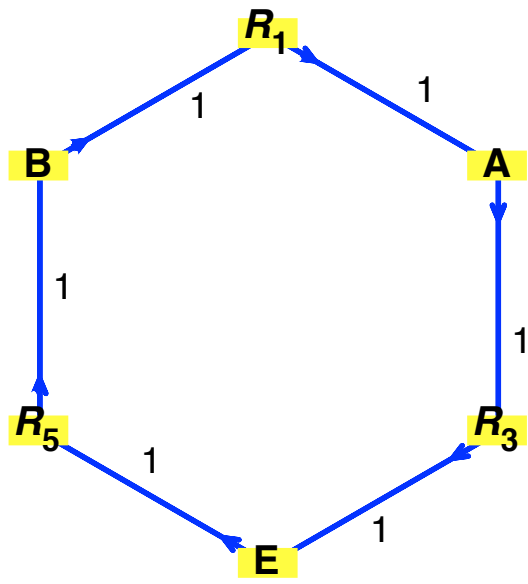
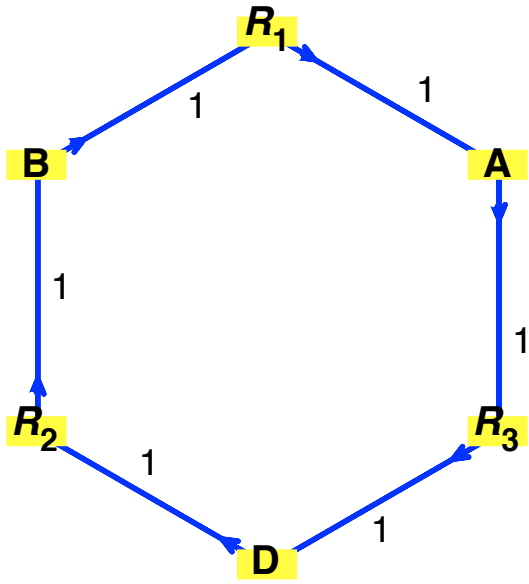




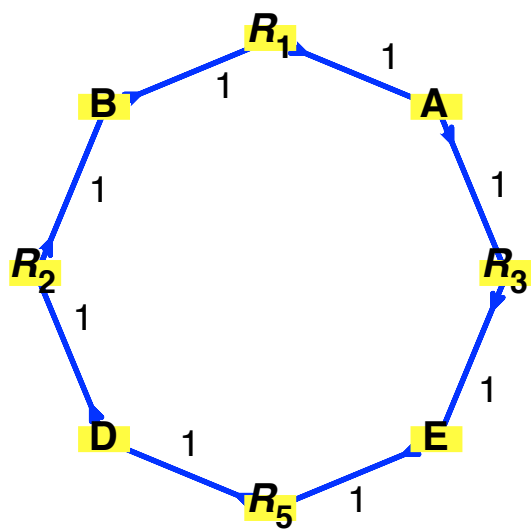
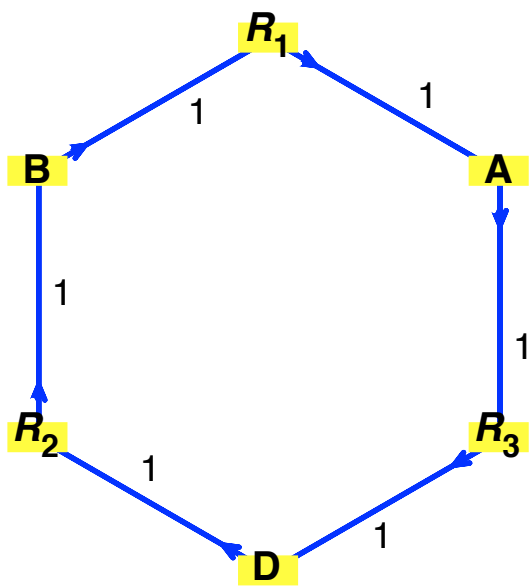
No. 52: injectiveEx3\_3000.csv

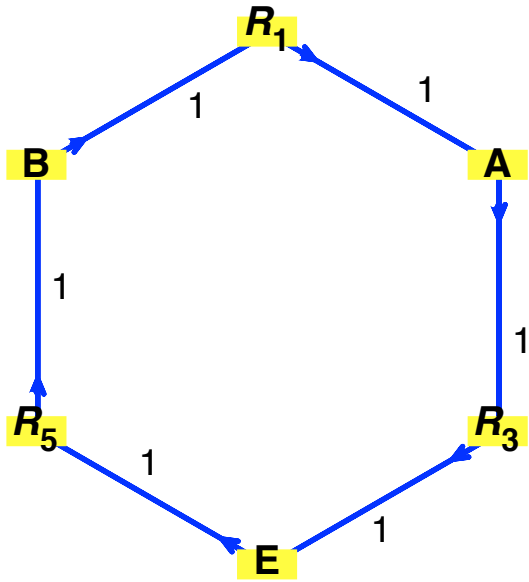


No. 53: injectiveEx3\_3002.csv

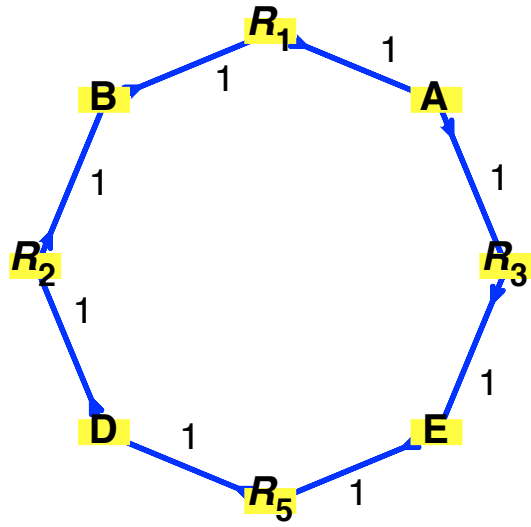
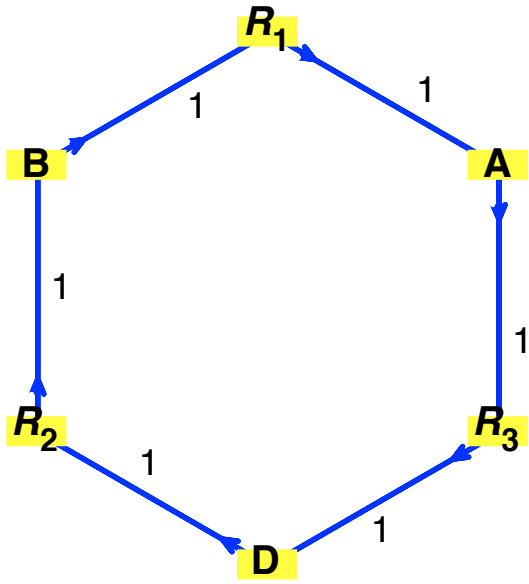


No. 54: injectiveEx3\_3003.csv

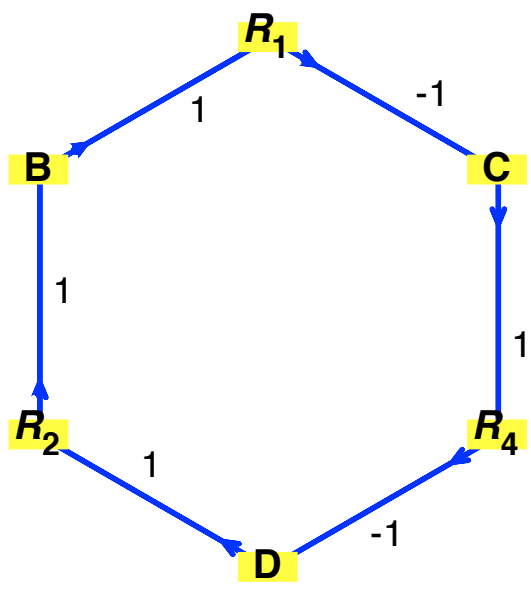
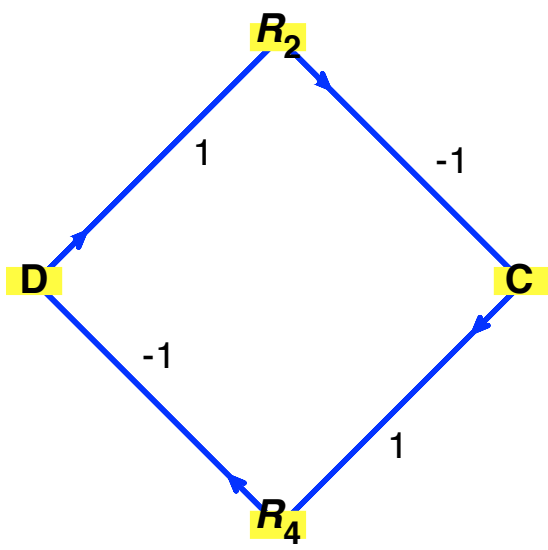


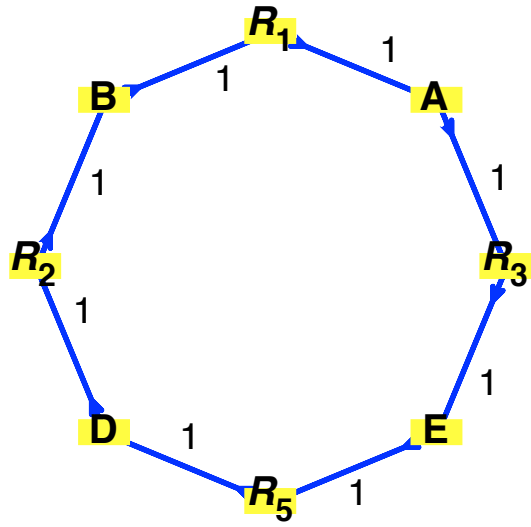
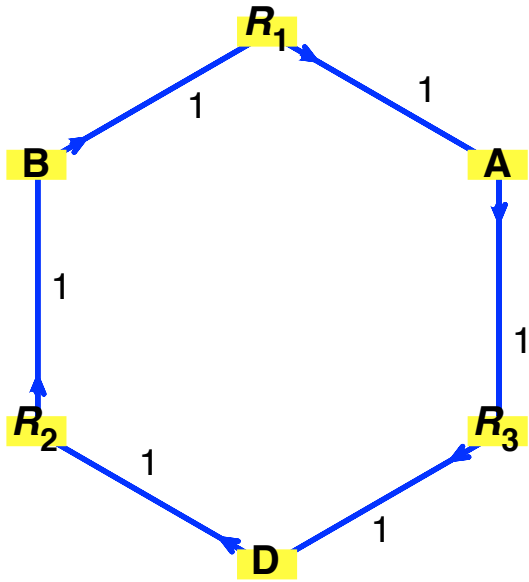


No. 55: injectiveEx3\_3004.csv



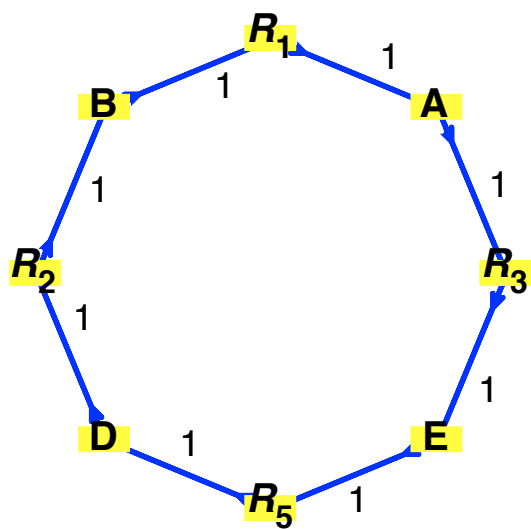
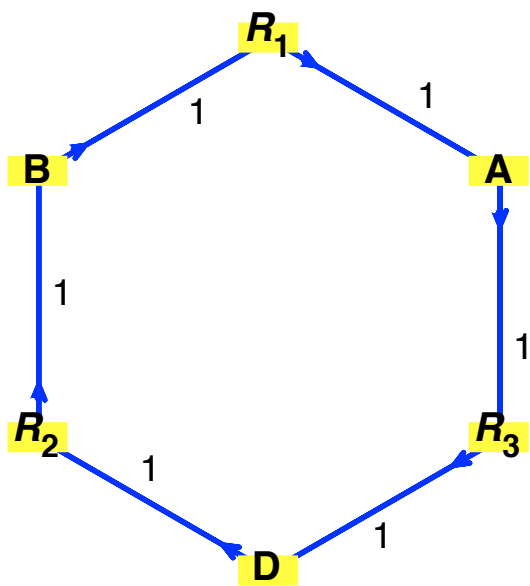
No. 56: injectiveEx3\_3007.csv

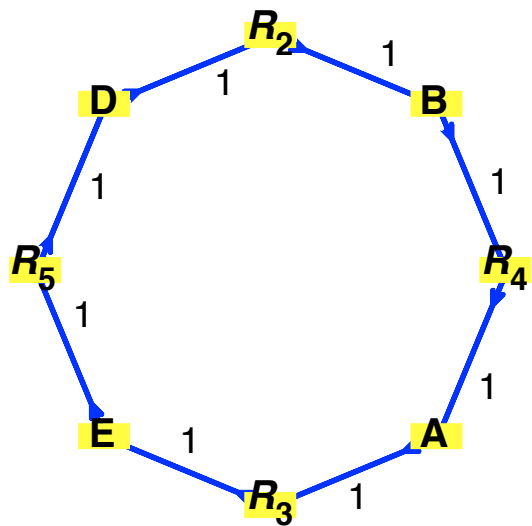
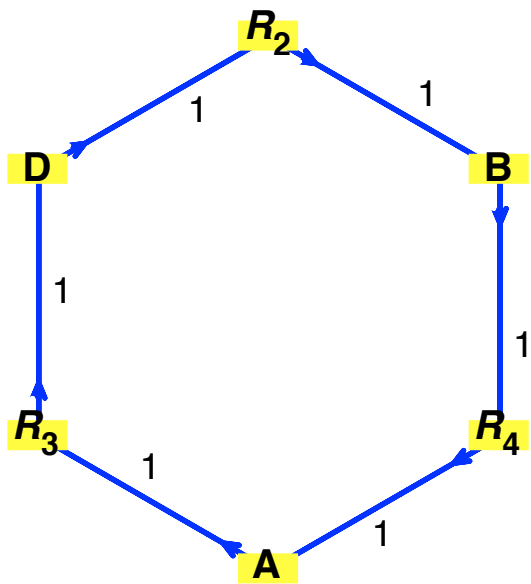




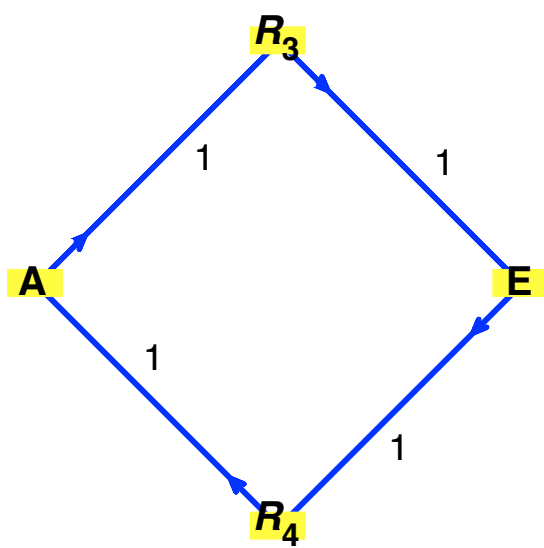
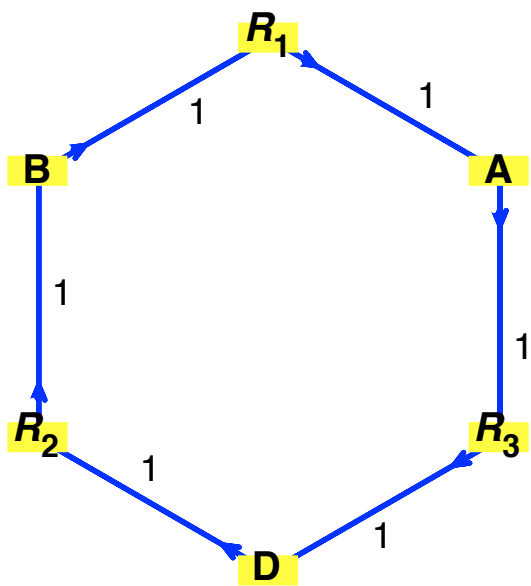
No. 57: injectiveEx3\_3009.csv

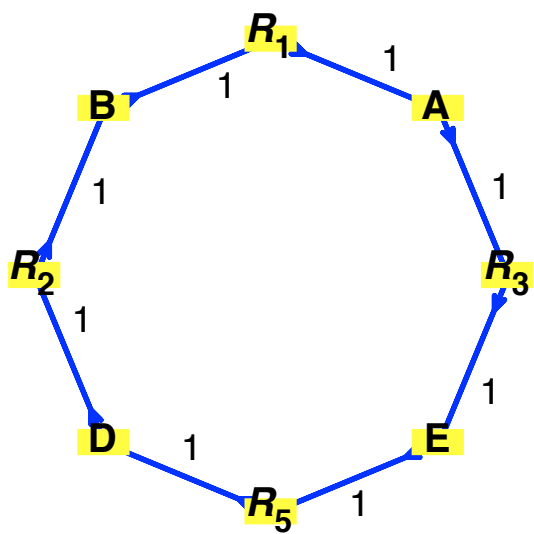




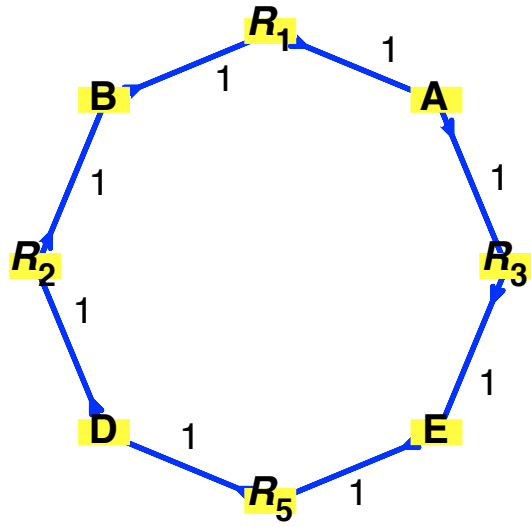
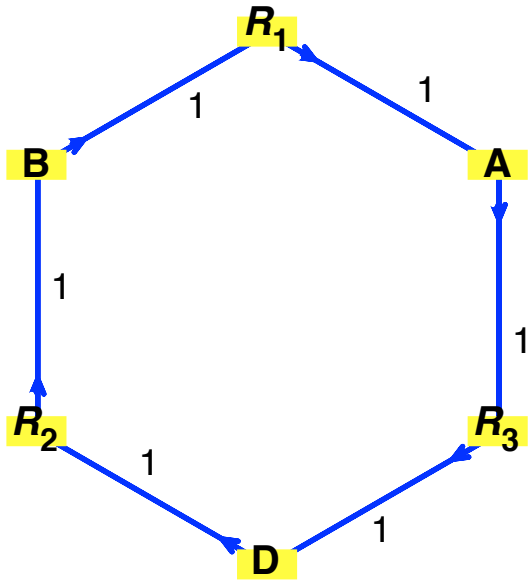


No. 58: injectiveEx3\_3011.csv

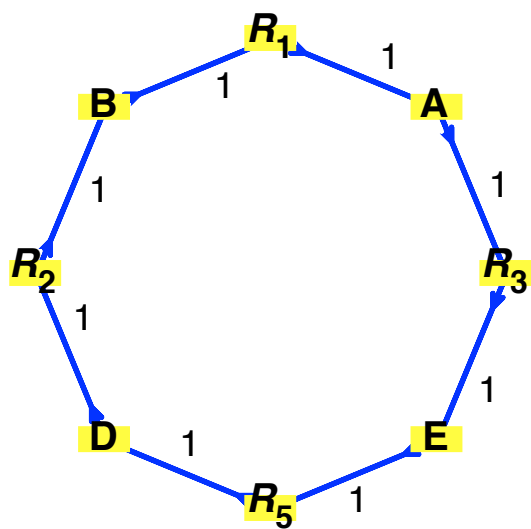
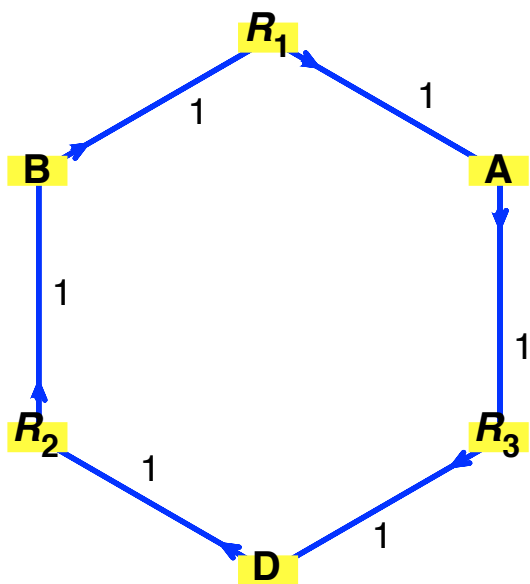


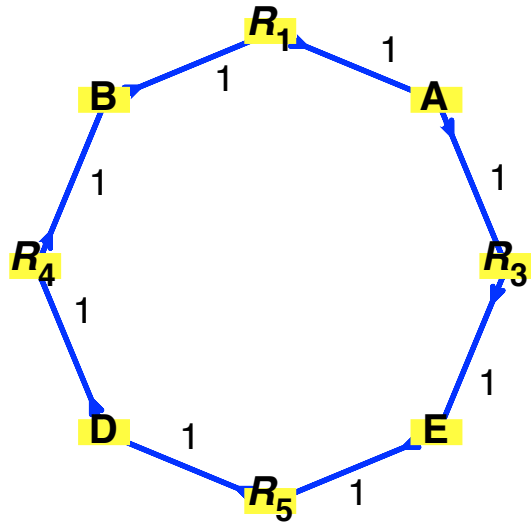
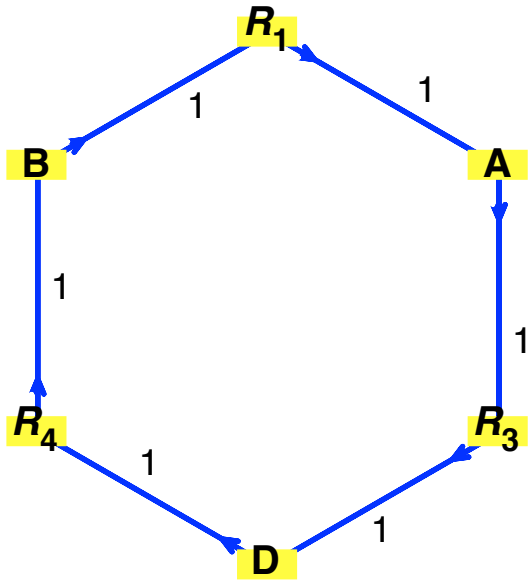


No. 59: injectiveEx3\_3012.csv

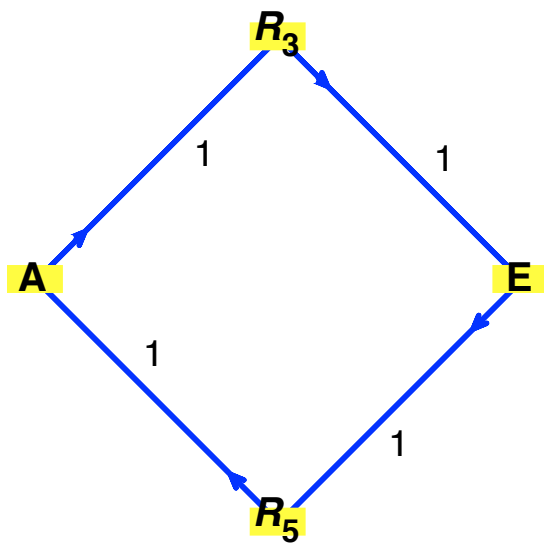
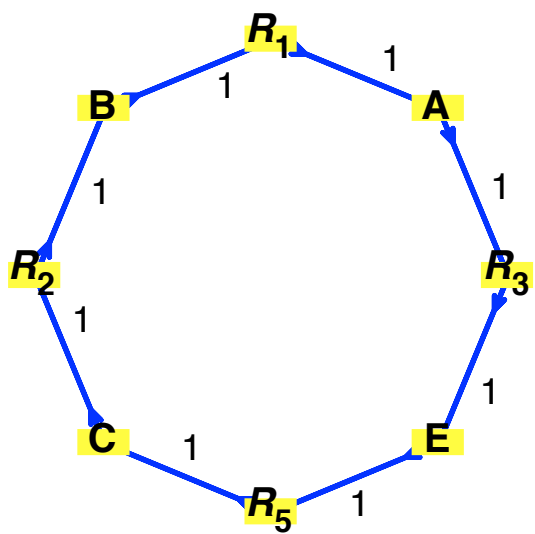


No. 60: injectiveEx3\_3014.csv

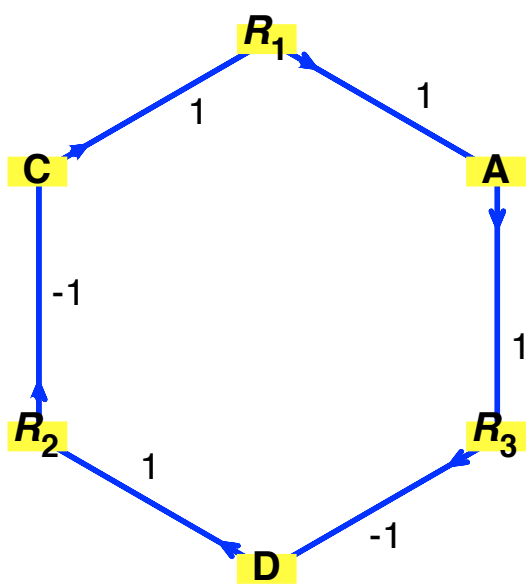
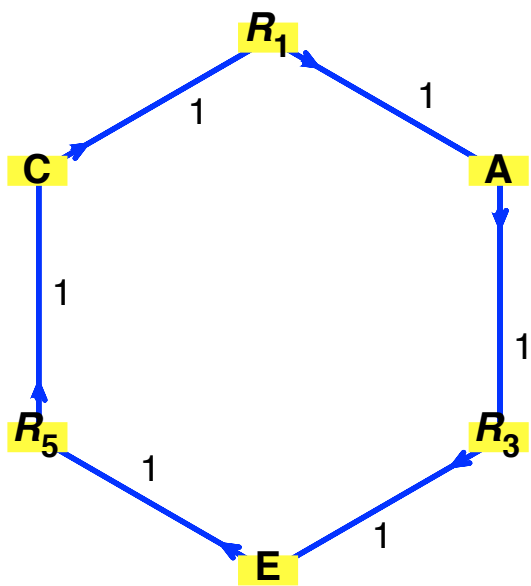




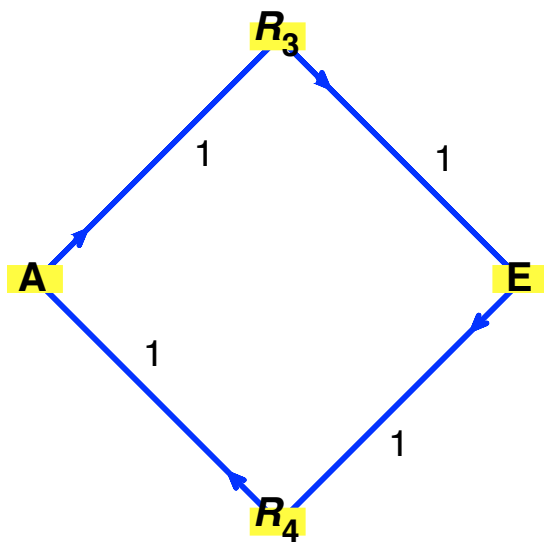
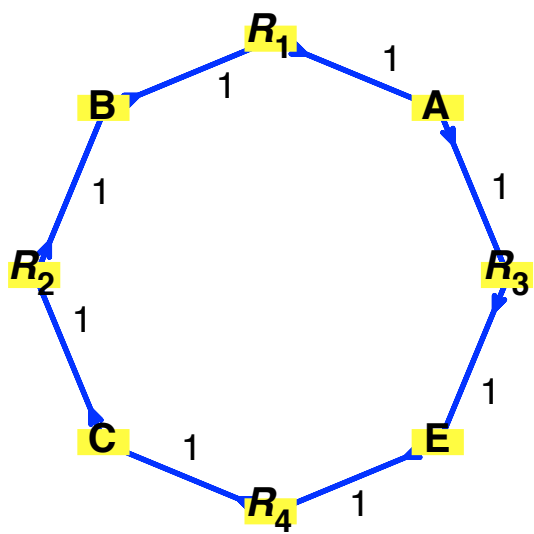
No. 61: injectiveEx3\_3015.csv

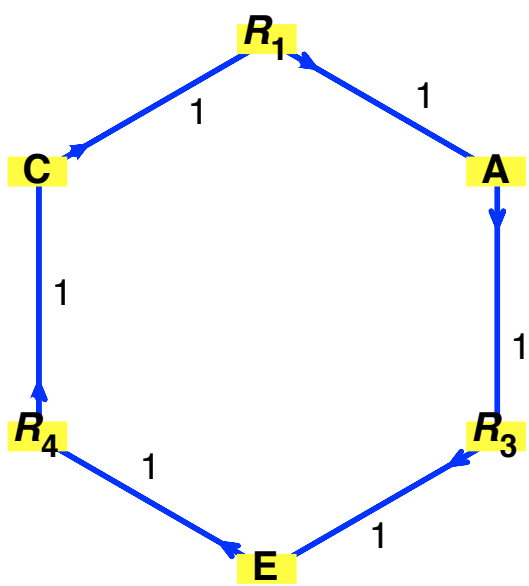
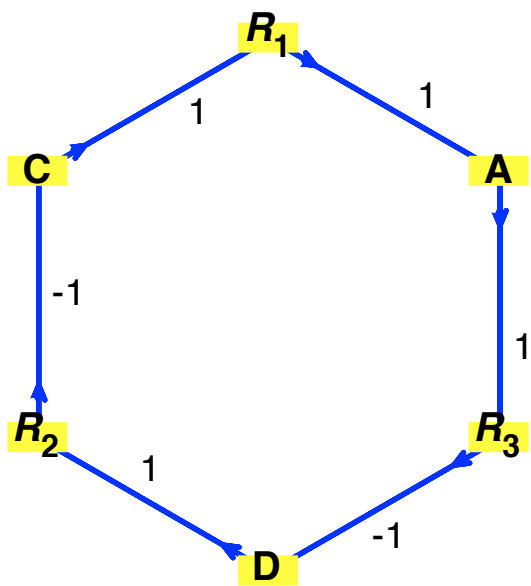




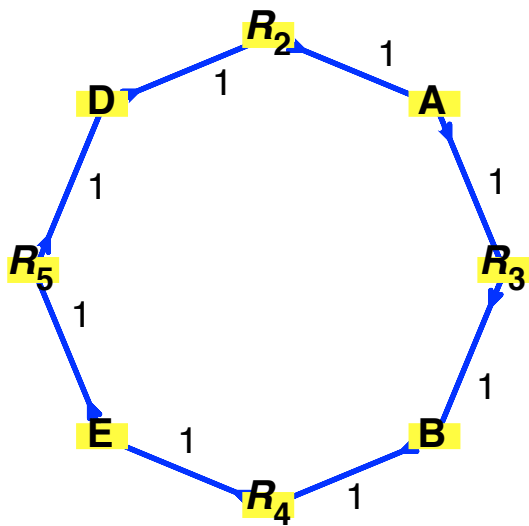
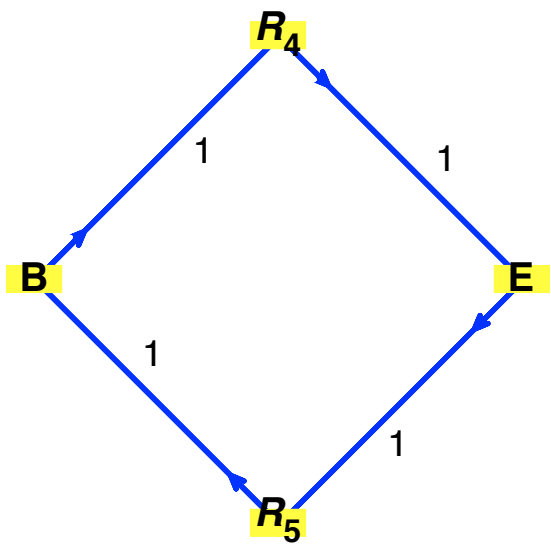


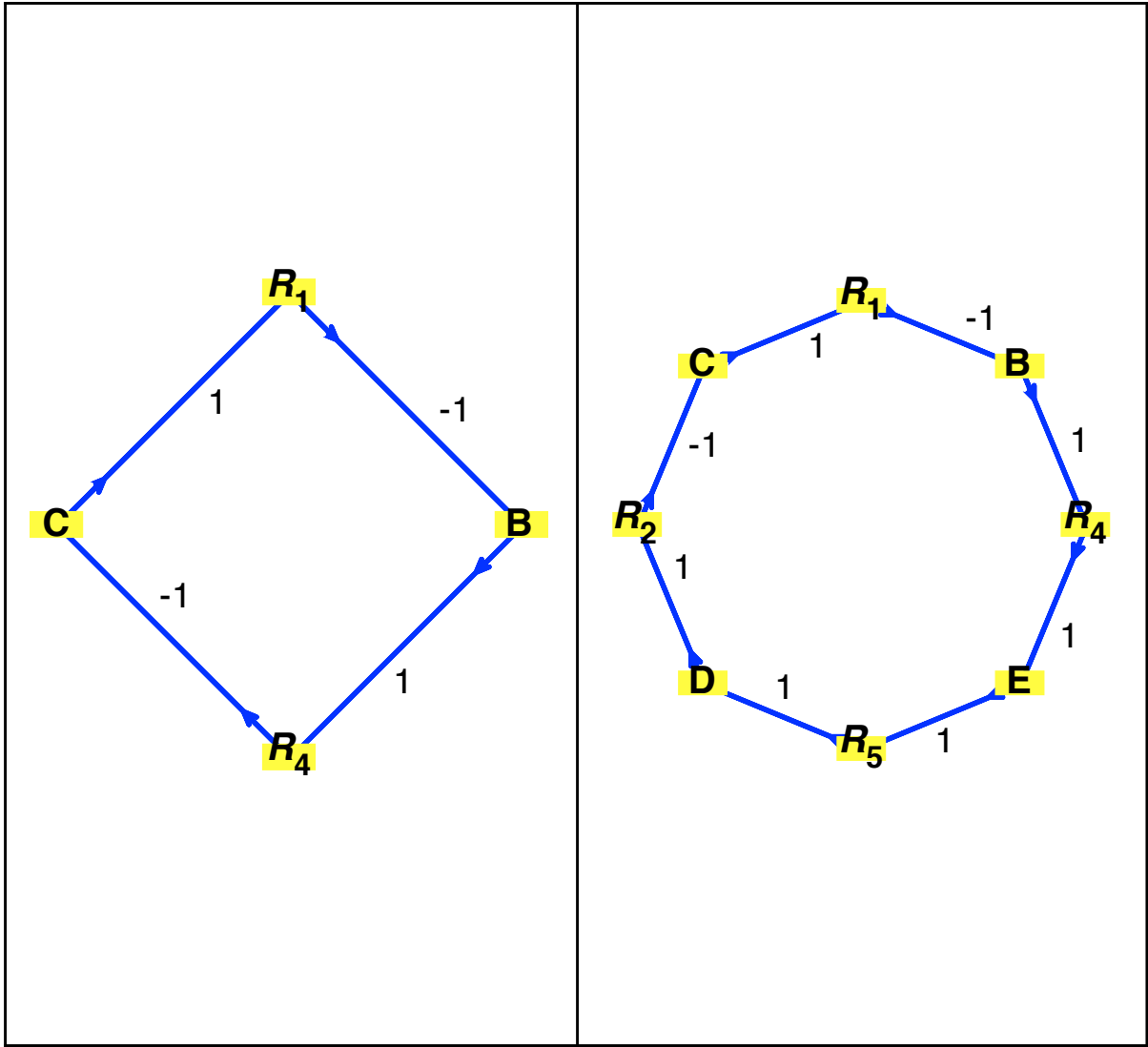
No. 62: injectiveEx3\_3017.csv



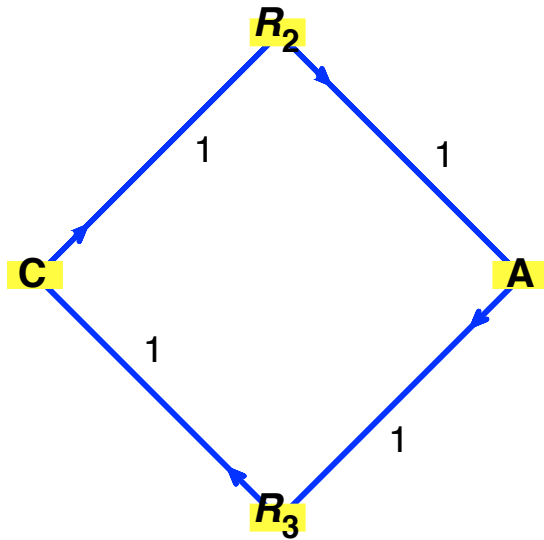
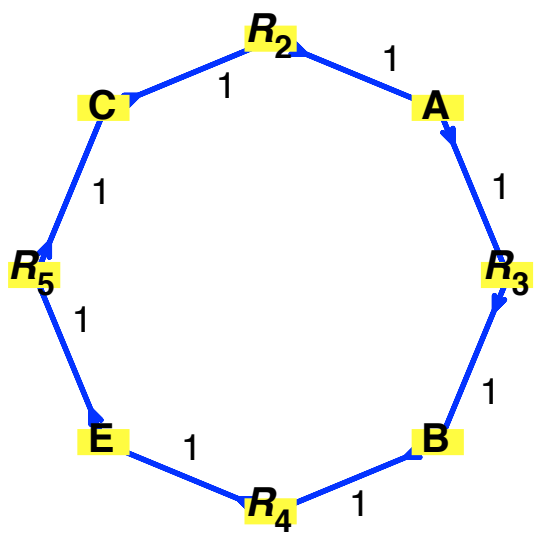


No. 63: injectiveEx3\_3110.csv

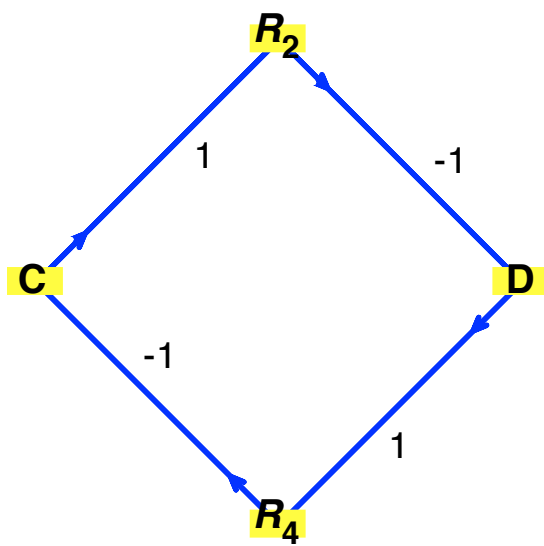
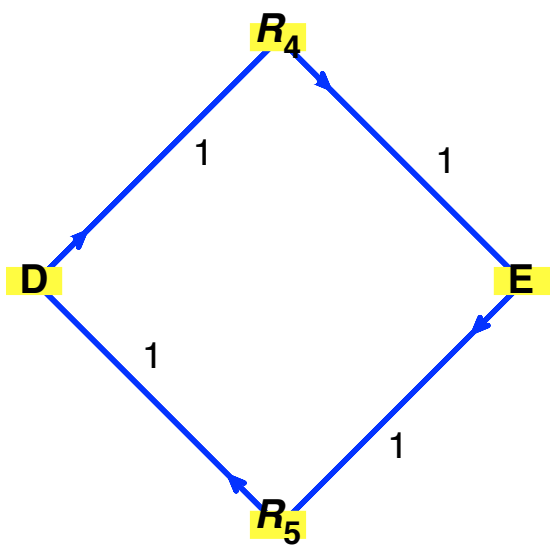


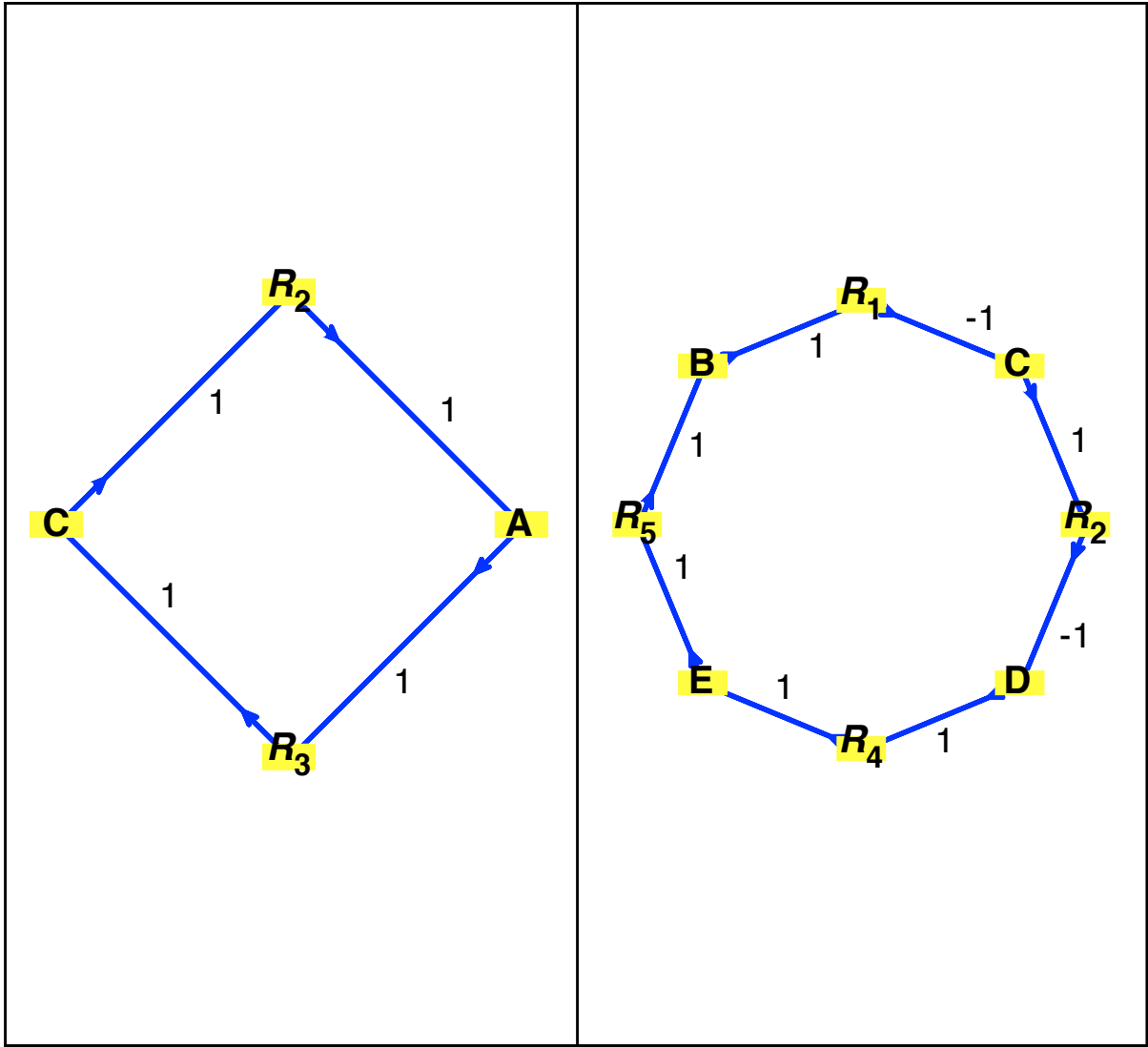


No. 64: injectiveEx3\_3114.csv



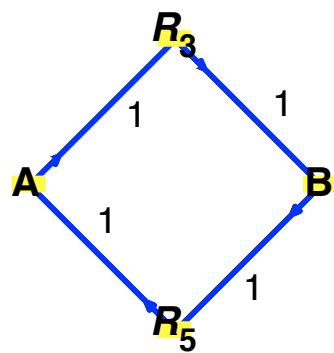
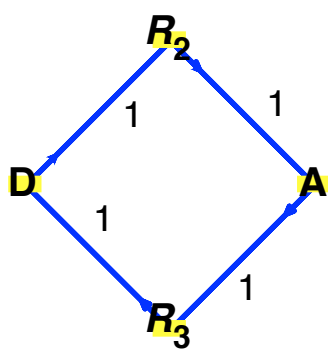
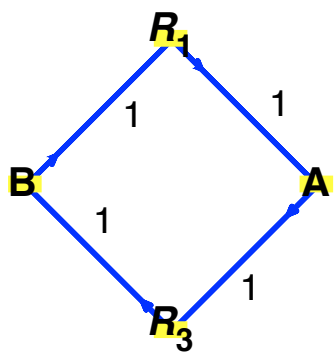
No. 65: injectiveEx3\_3118.csv

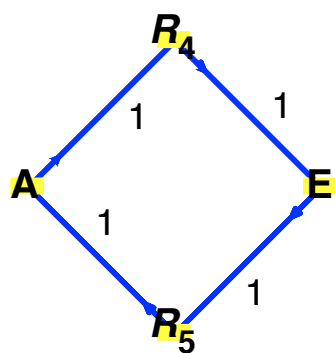
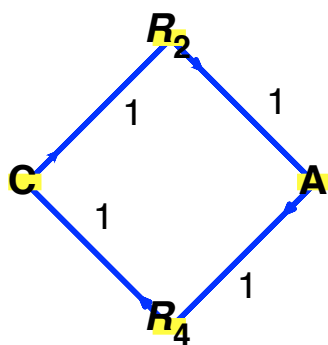
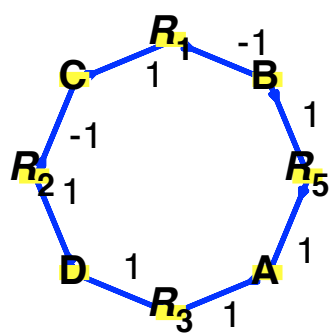


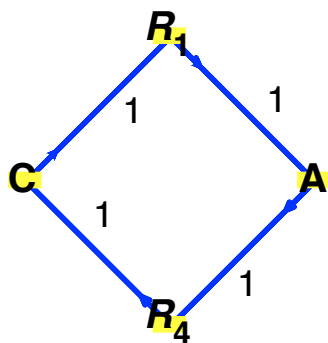
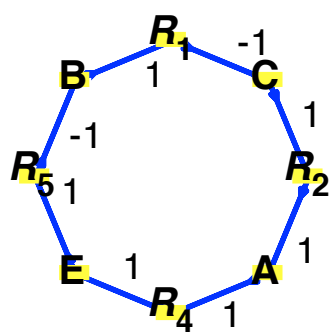


No. 66: injectiveEx3\_3123.csv

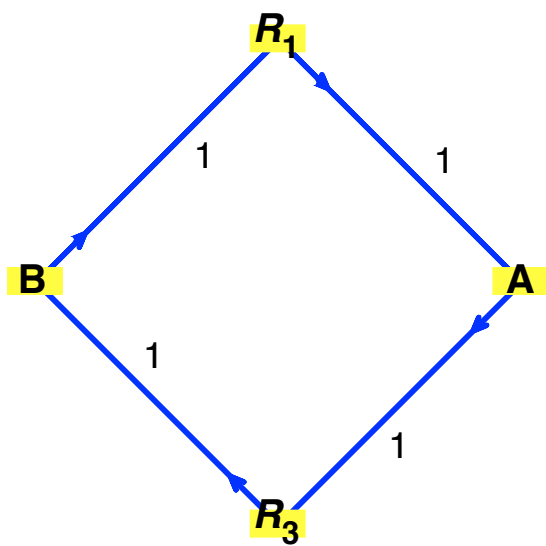
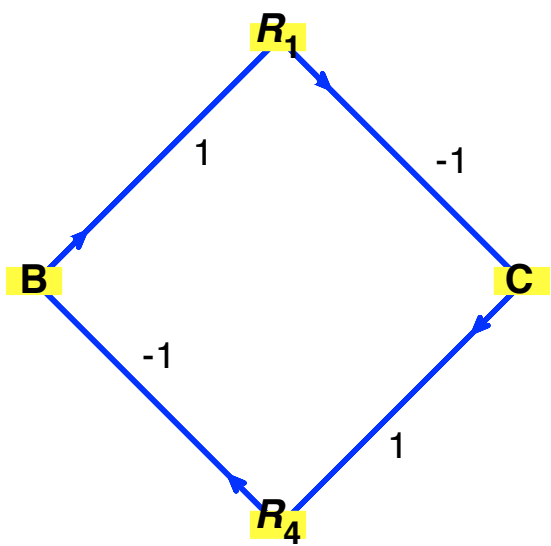


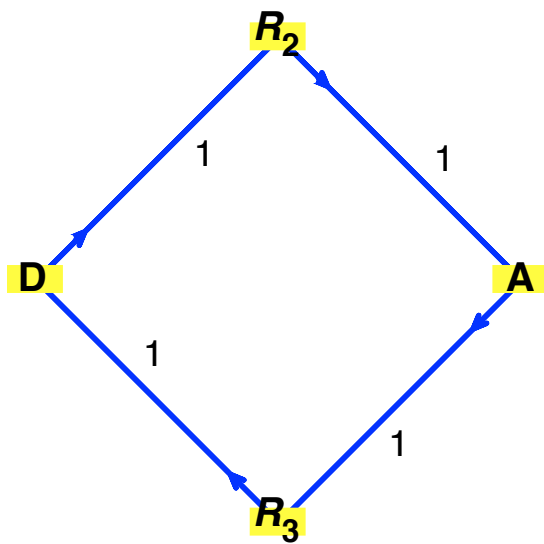
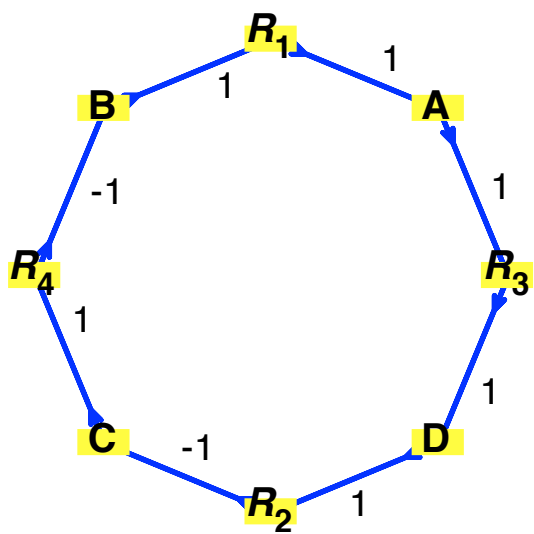




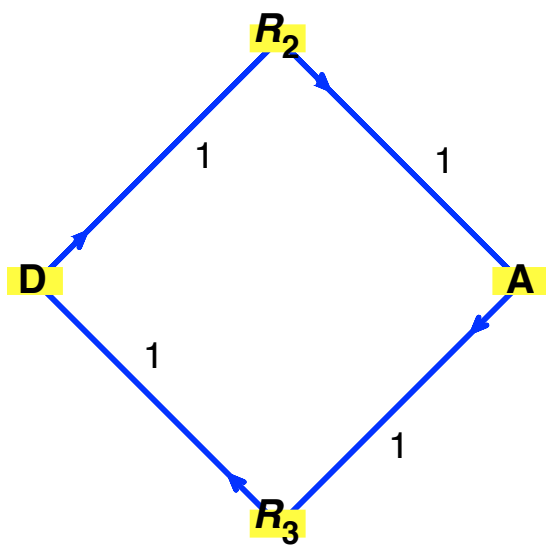
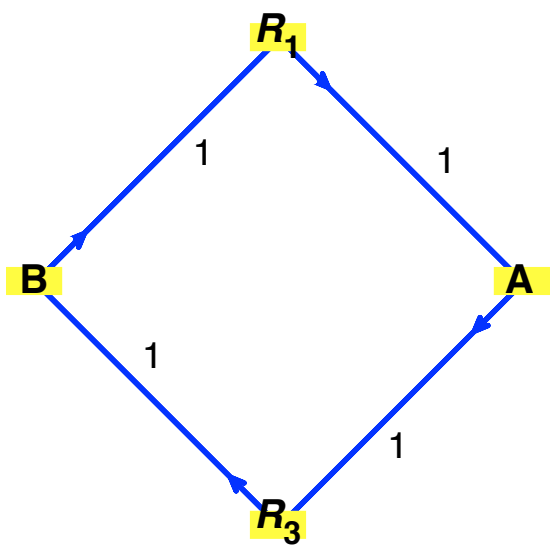


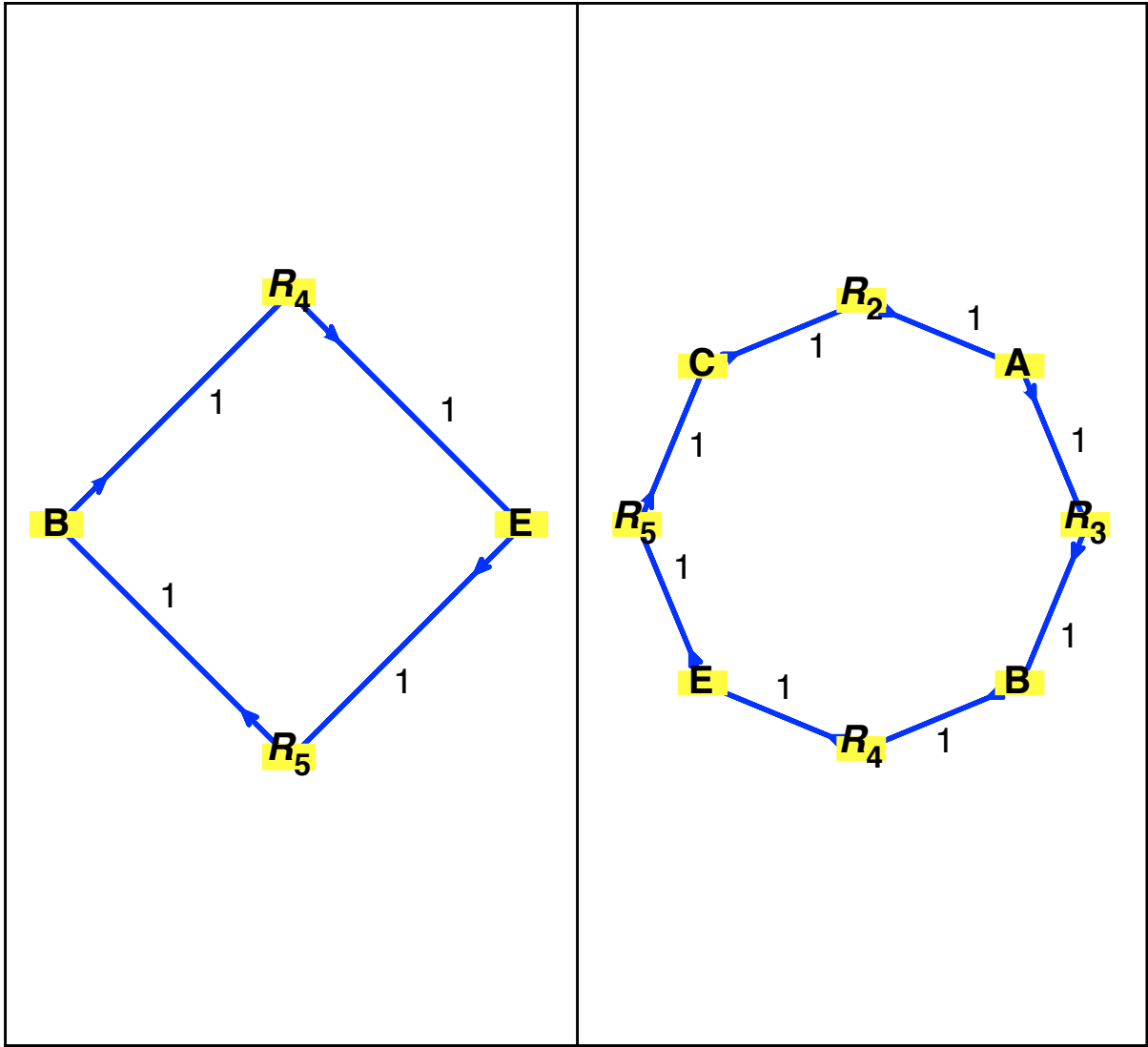
No. 67: [injectiveEx3\\_3128.csv](#)



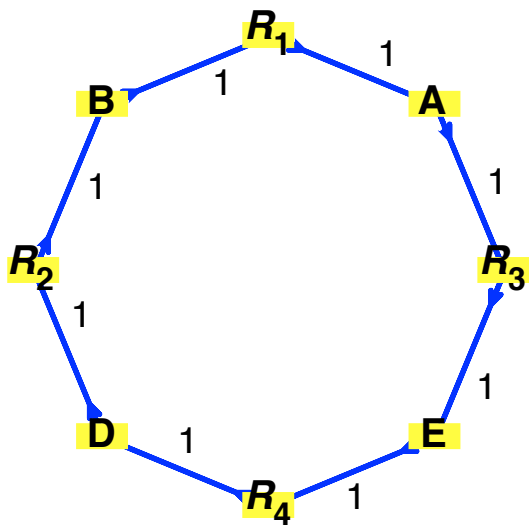
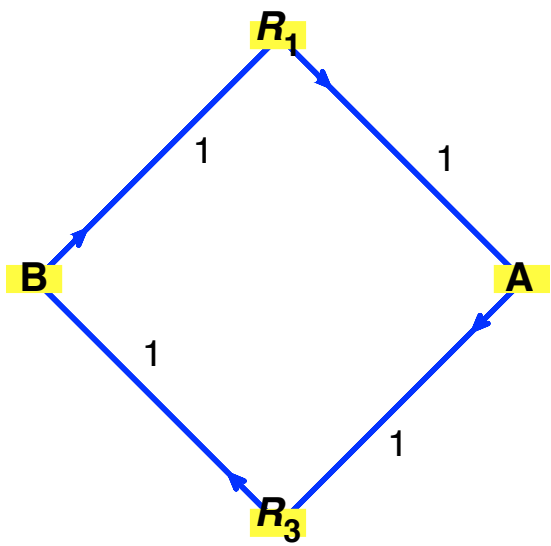


No. 68: injectiveEx3\_3130.csv

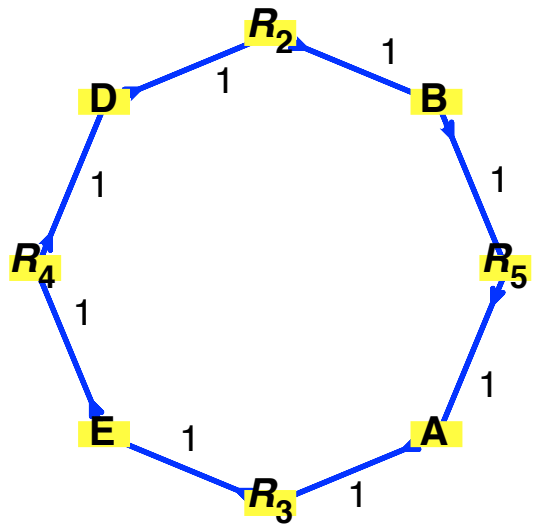
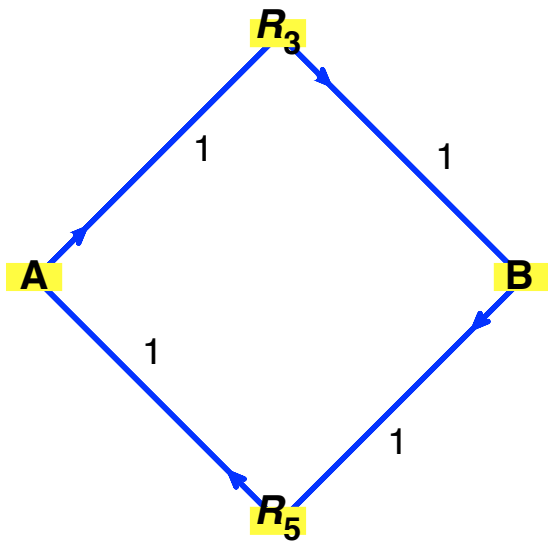




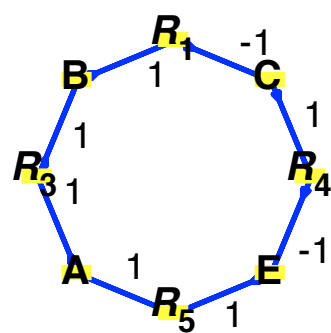
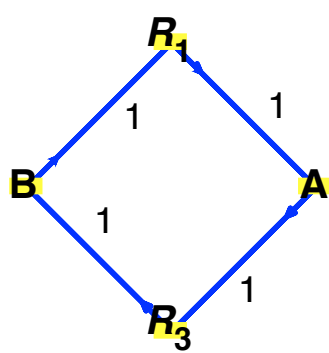
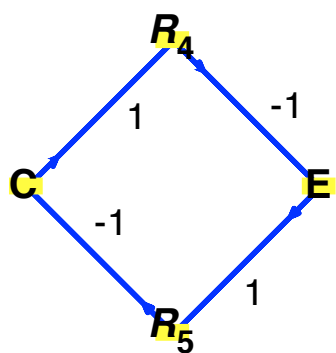
No. 69: injectiveEx3\_3553.csv

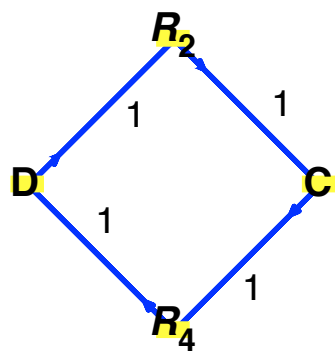
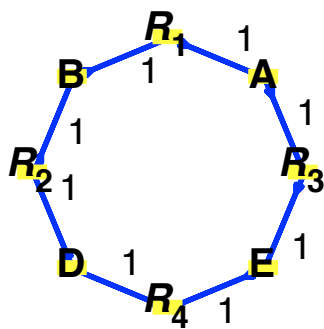
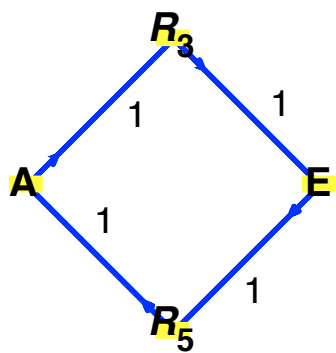




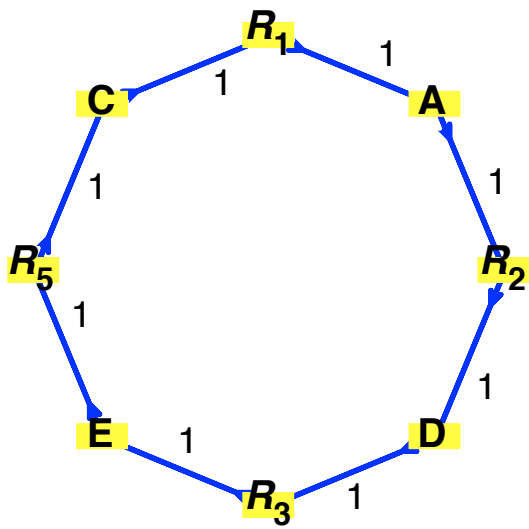
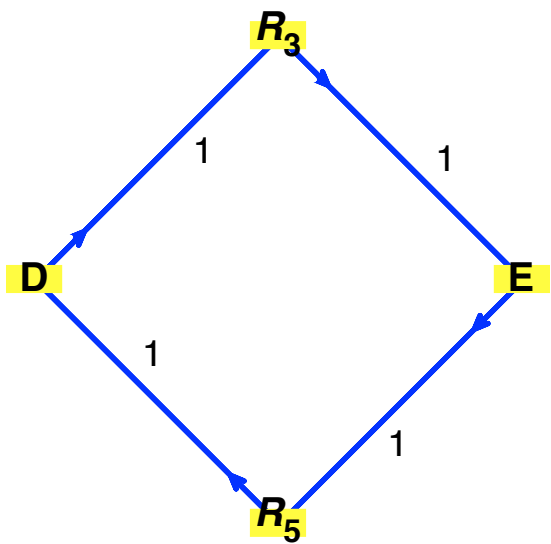


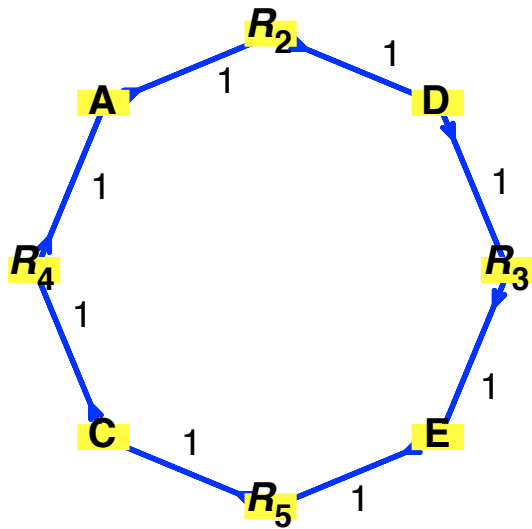
No. 70: injectiveEx3\_3555.csv





No. 71: injectiveEx3\_4324.csv

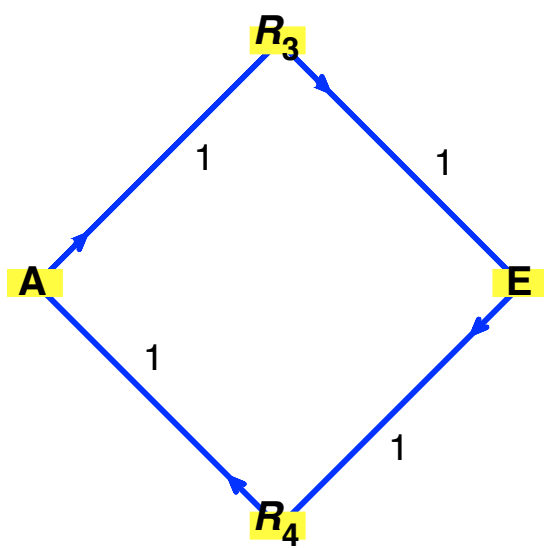
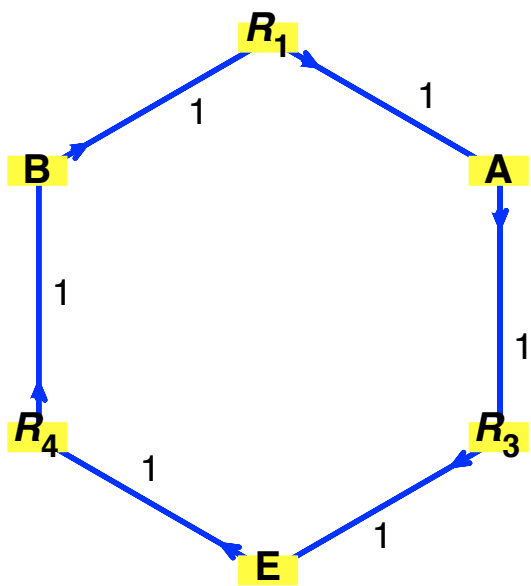


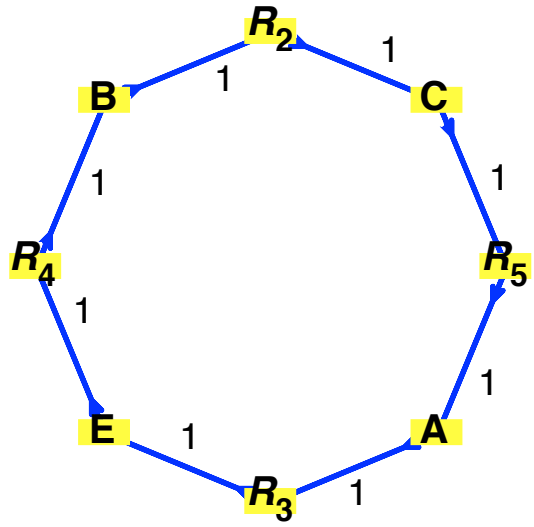
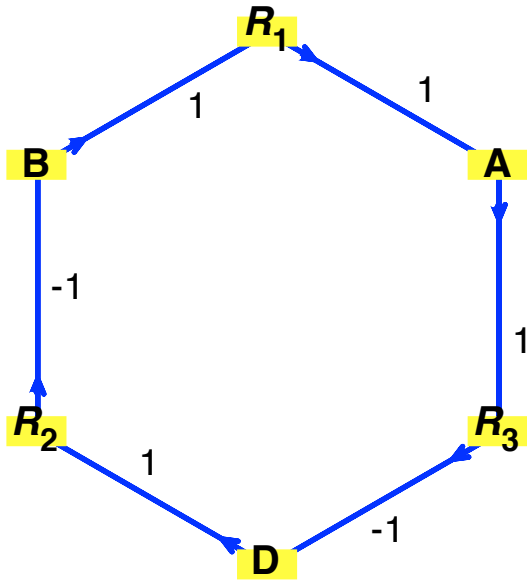


```
[>
[>
```

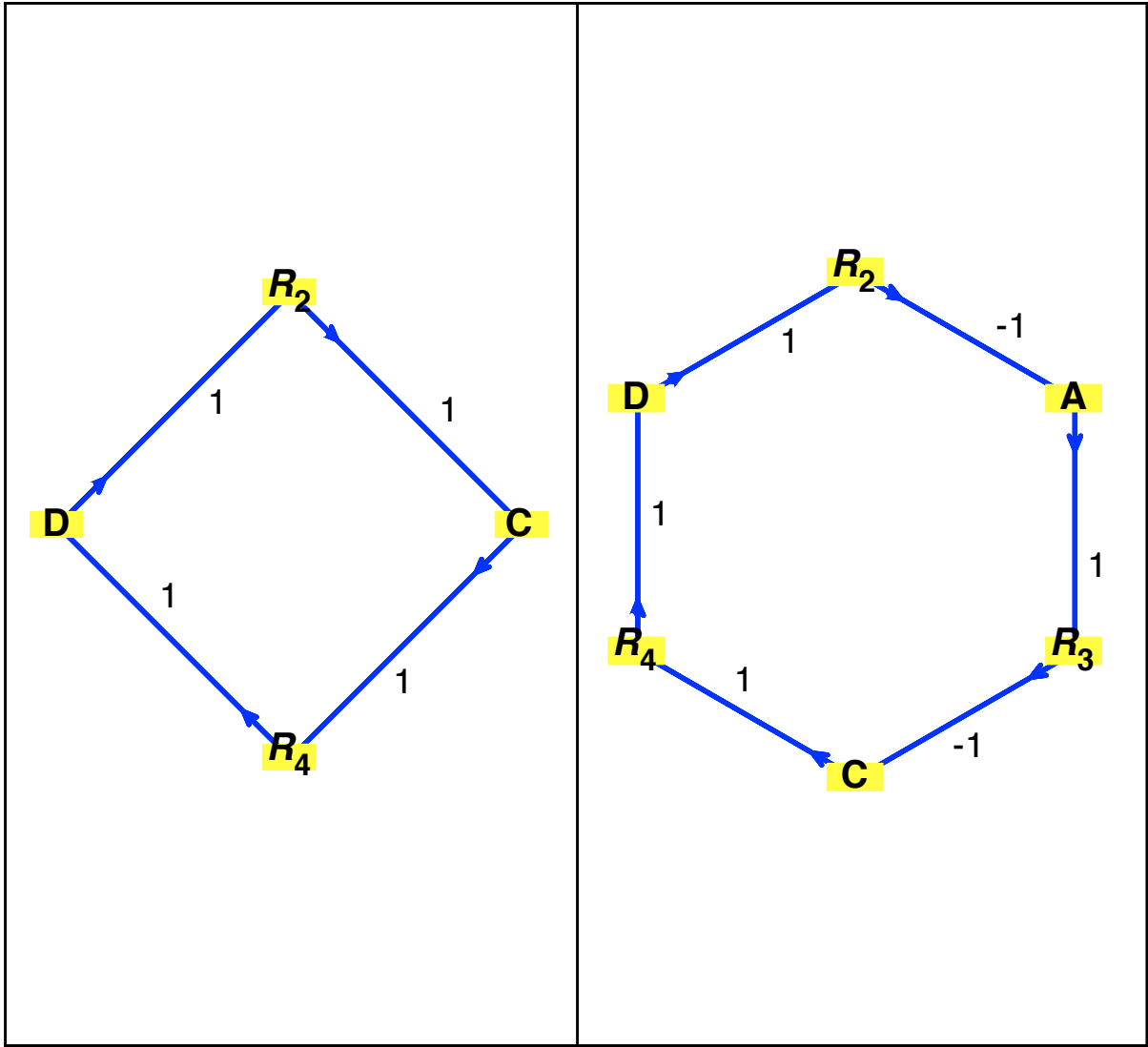
## ▼ Multistationary: with competition loop, with intersecting loops

```
[> 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
```



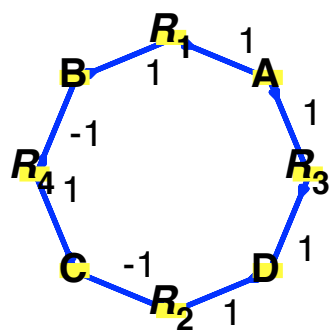
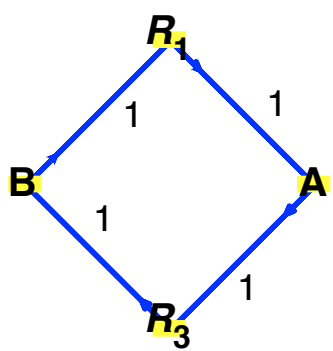
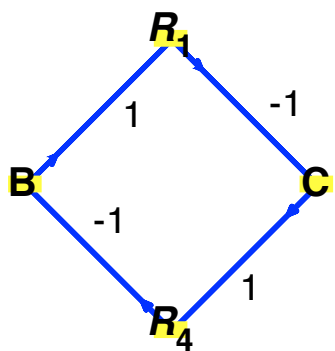


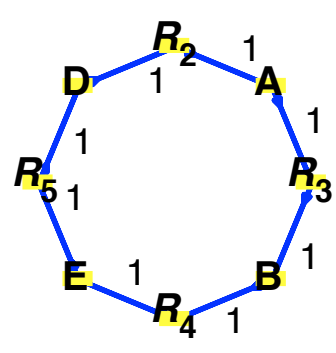
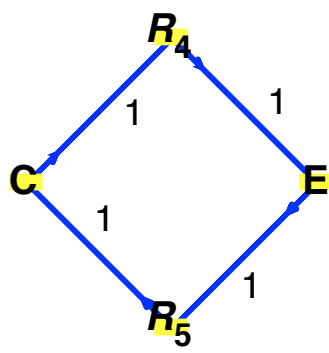
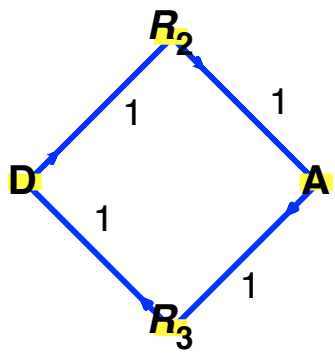
No. 2: injectiveEx0\_1742.csv



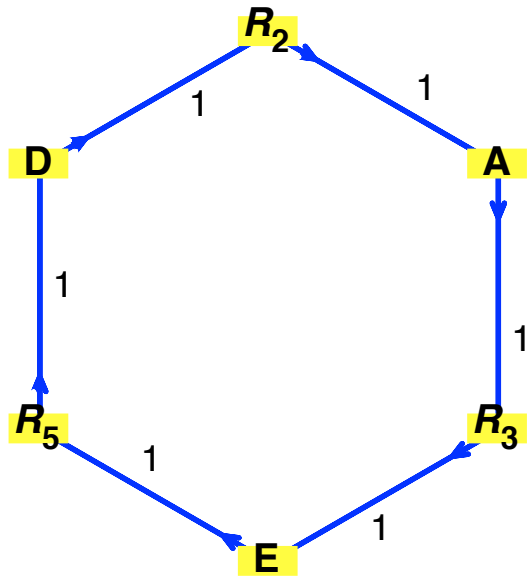
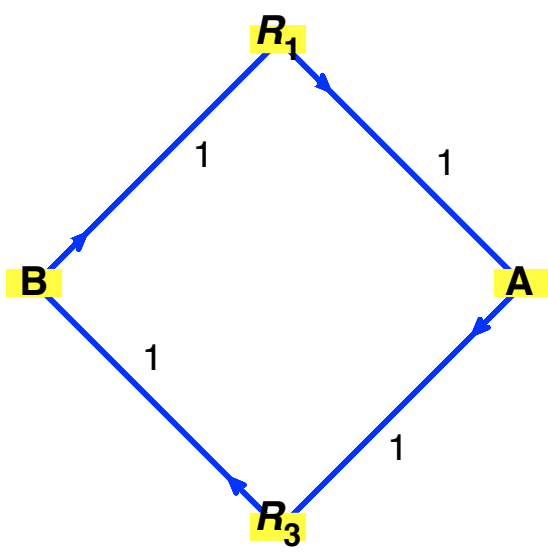
No. 3: injectiveEx0\_3129.csv

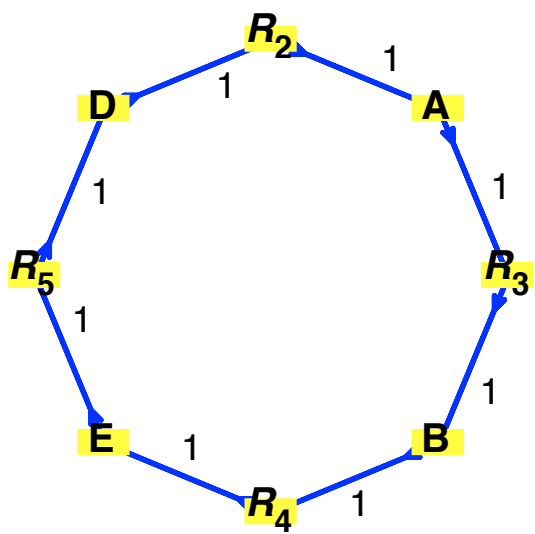




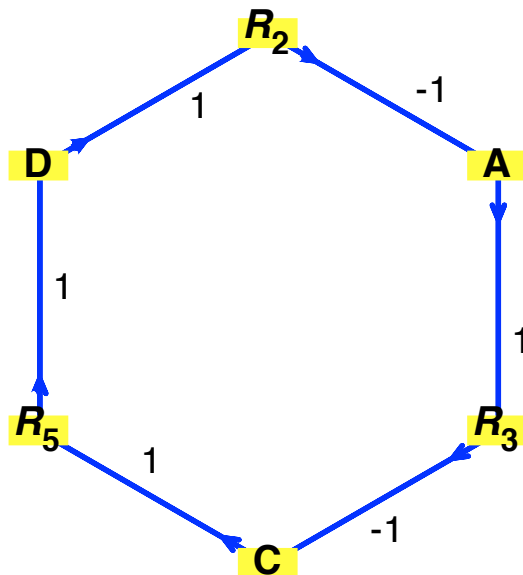
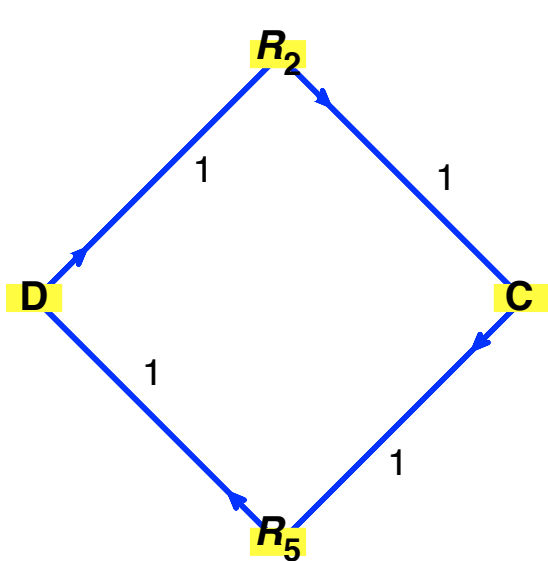


No. 4: injectiveEx0\_3141.csv





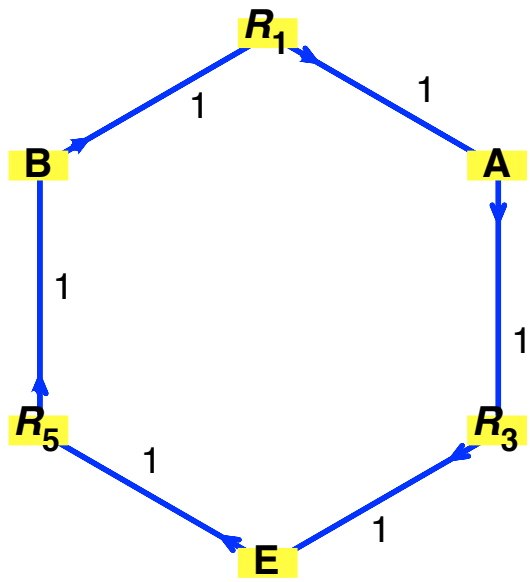
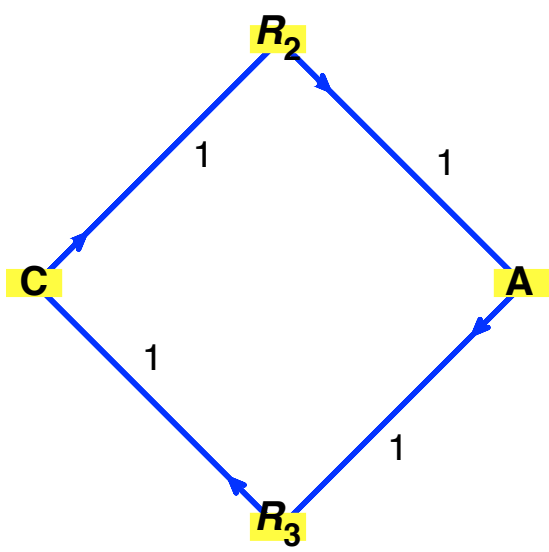
No. 5: injectiveEx2\_1741.csv

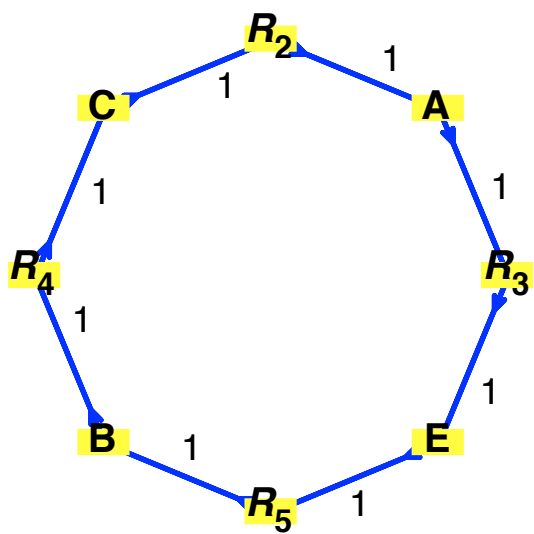


## ▼ Multistationary: no competition loop, with intersecting loops

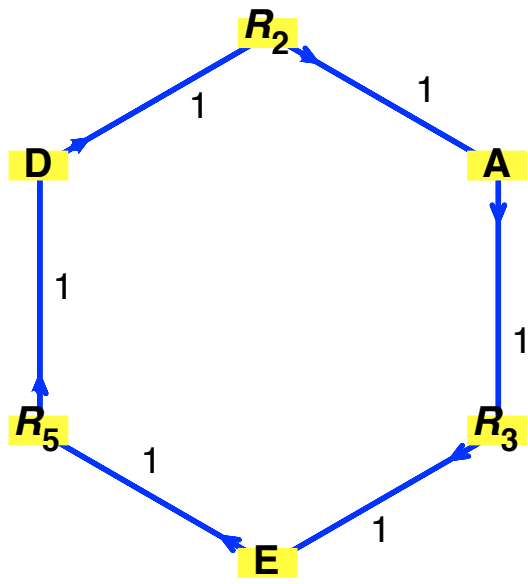
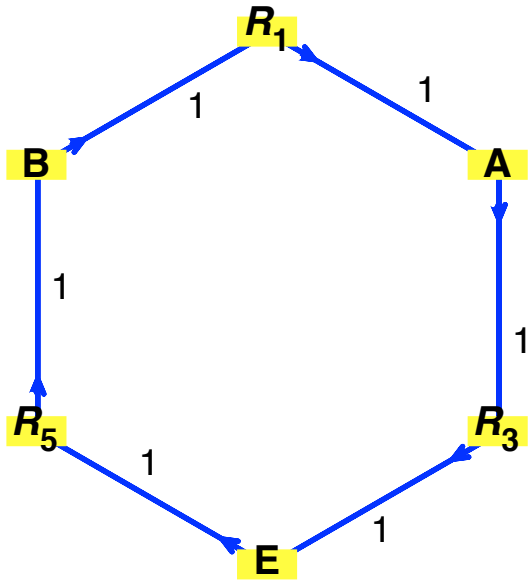
```

[> 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
  
```

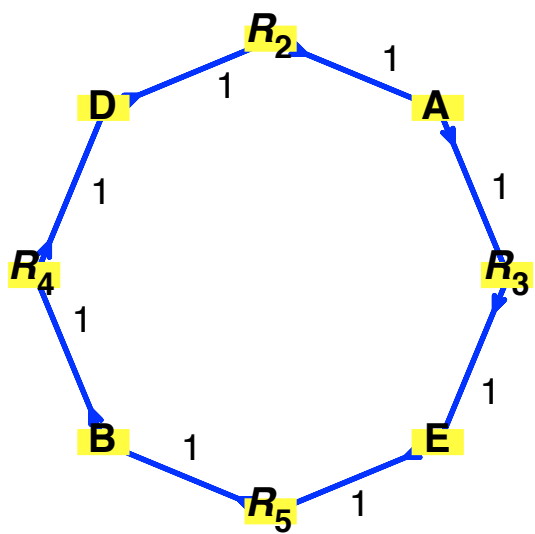




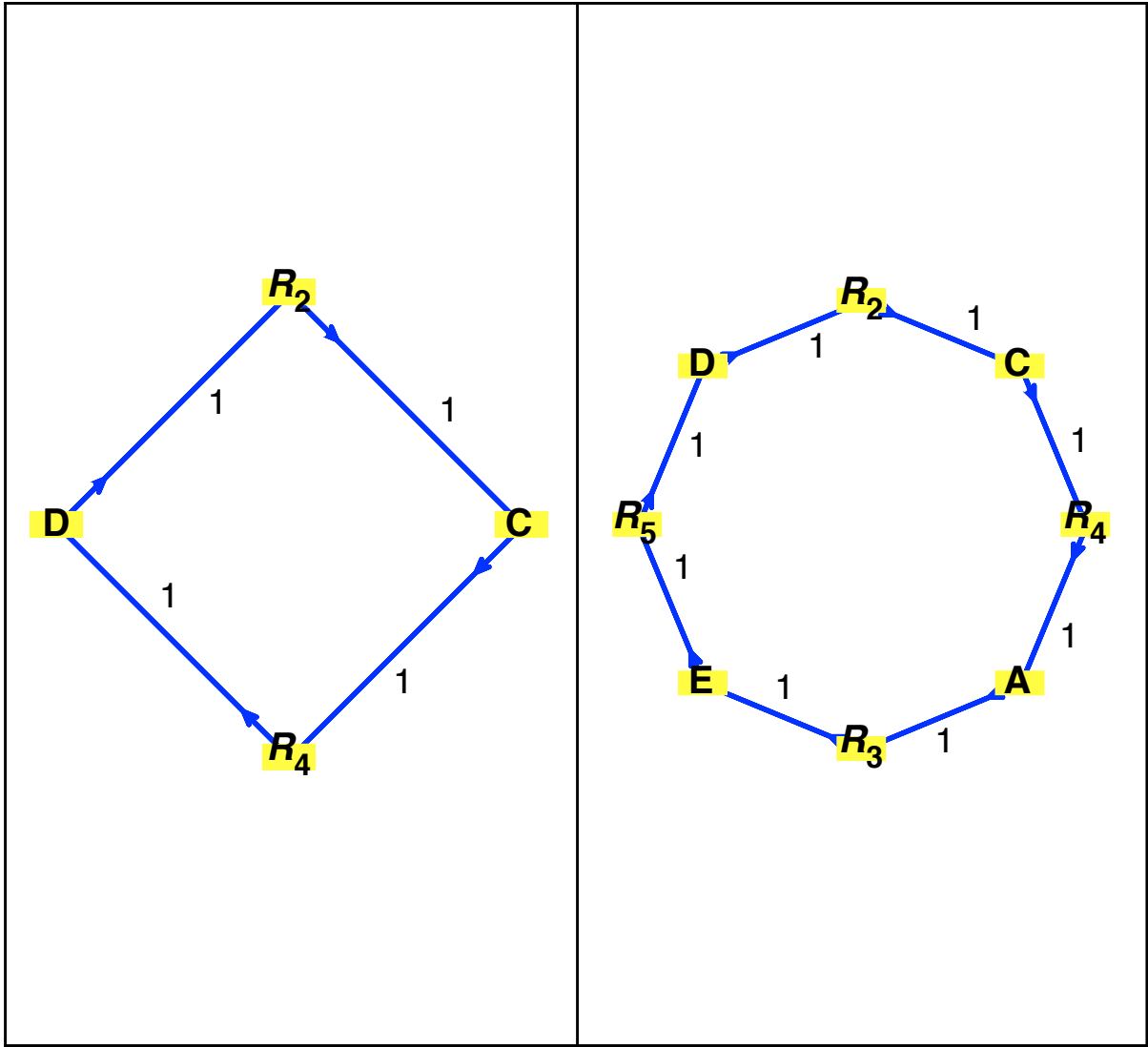
No. 2: injectiveEx0\_1392.csv



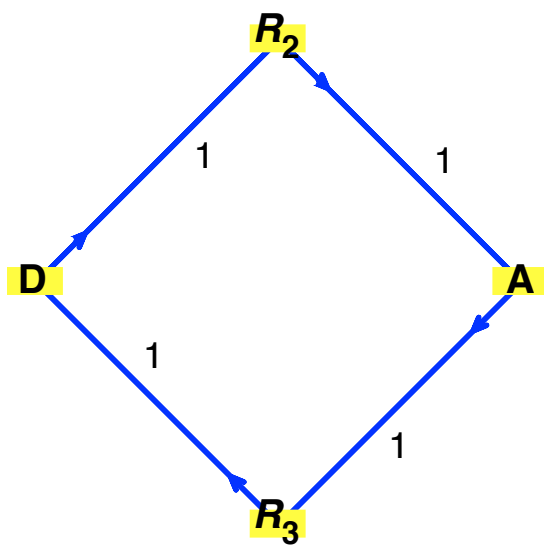
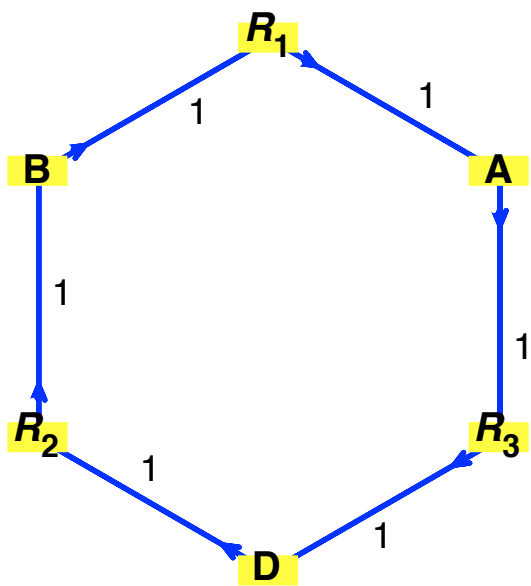


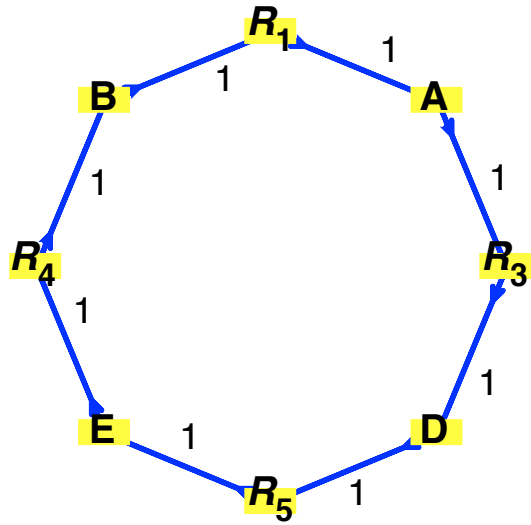
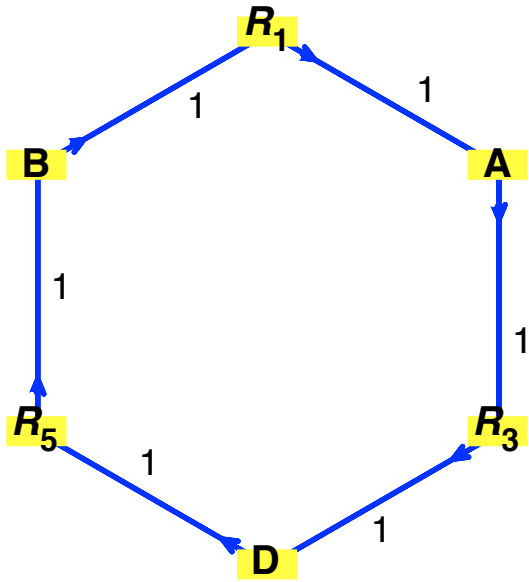


No. 3: injectiveEx0\_1487.csv

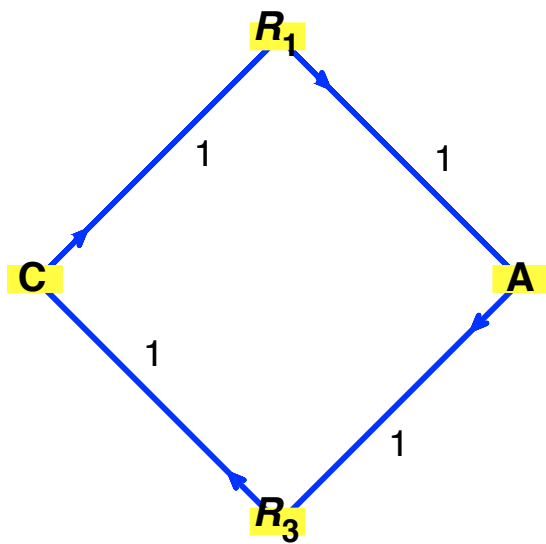
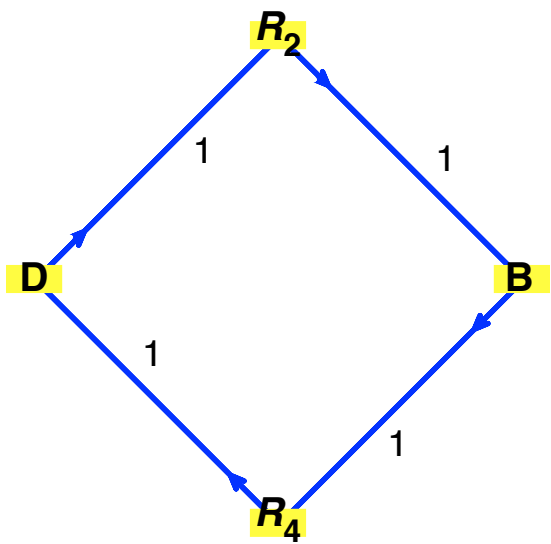


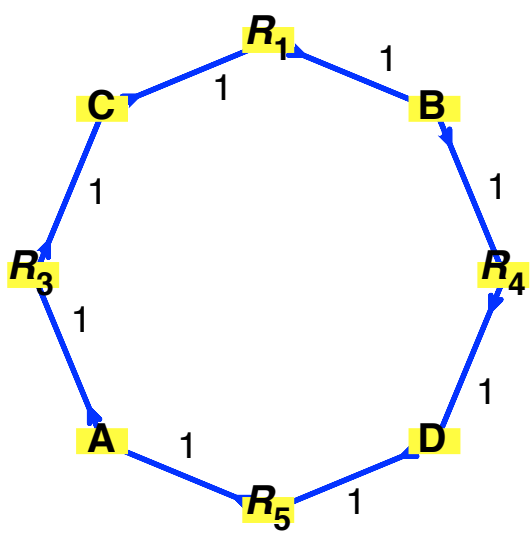
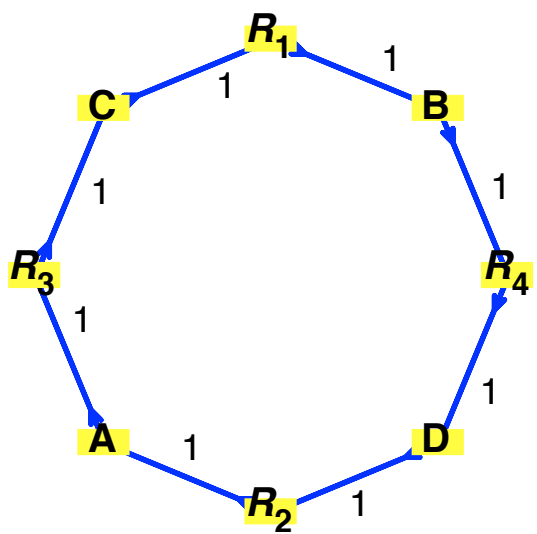
No. 4: injectiveEx0\_3733.csv



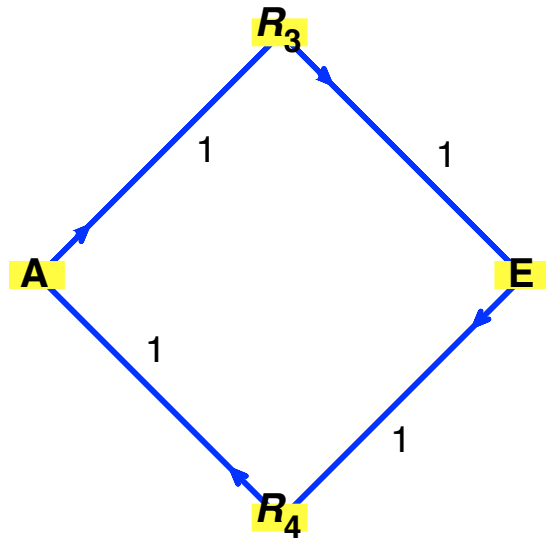
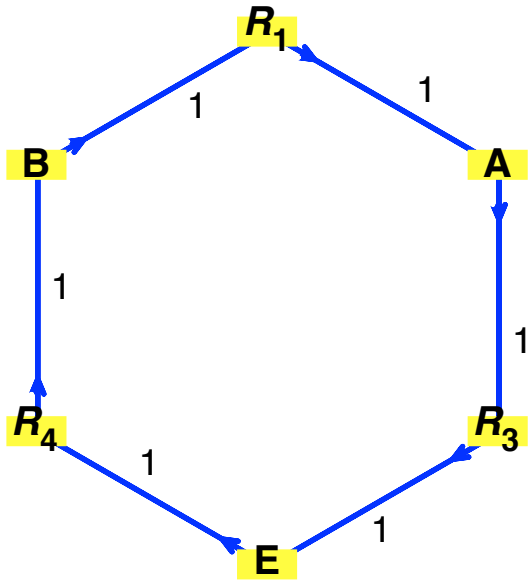


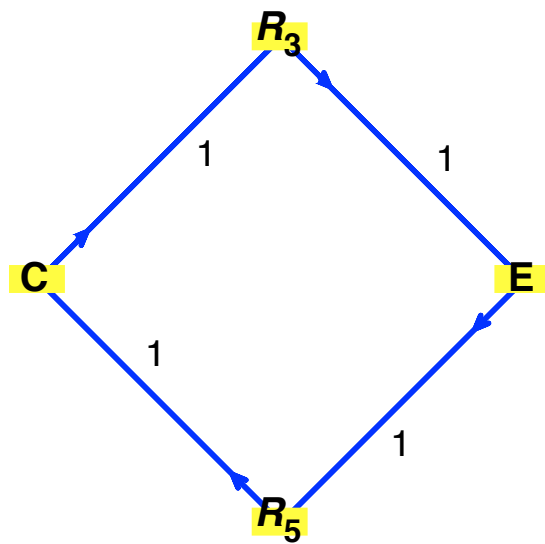
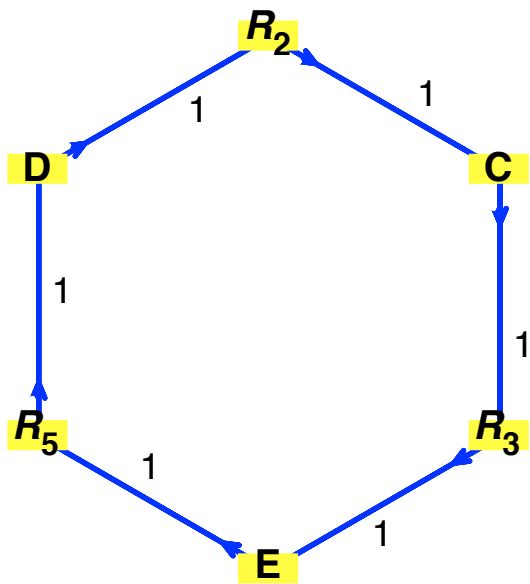
No. 5: injectiveEx0\_5631.csv





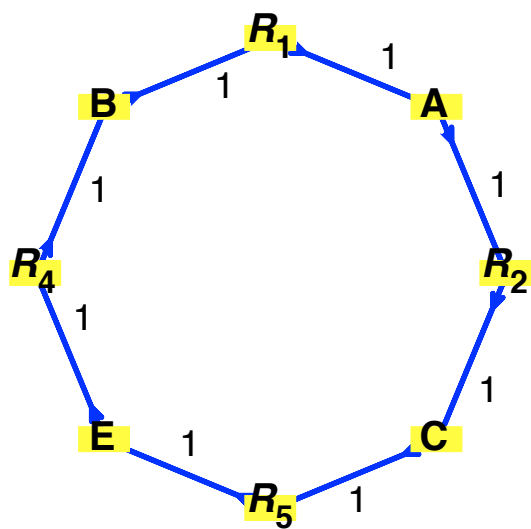
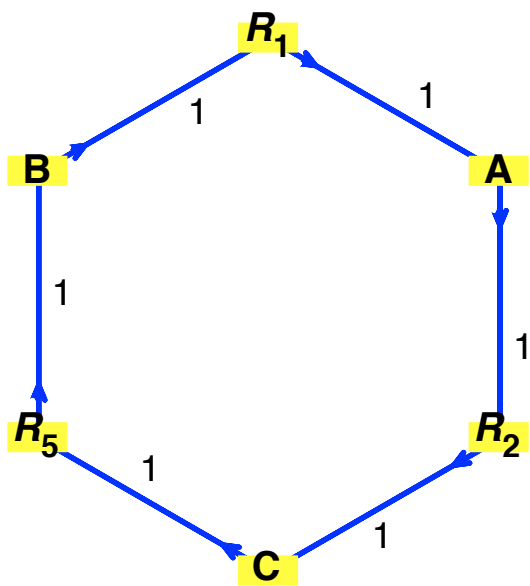
No. 6: injectiveEx2\_146.csv.csv

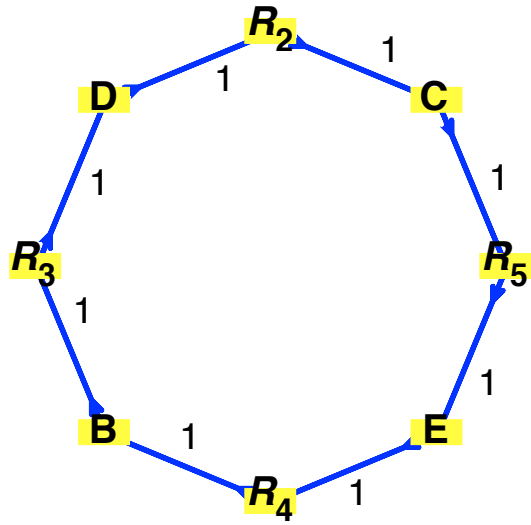
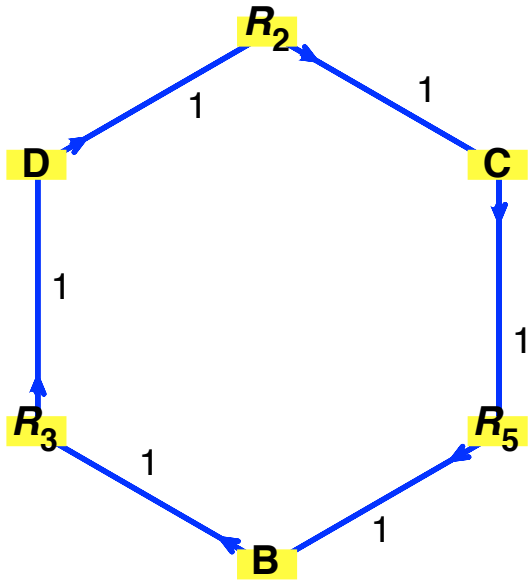




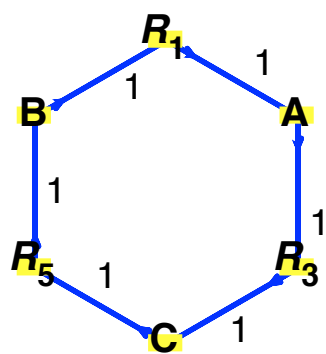
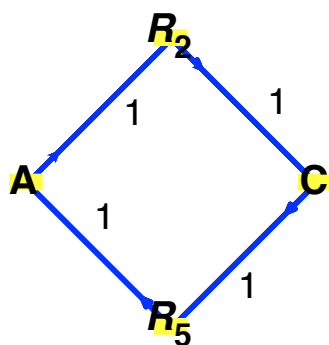
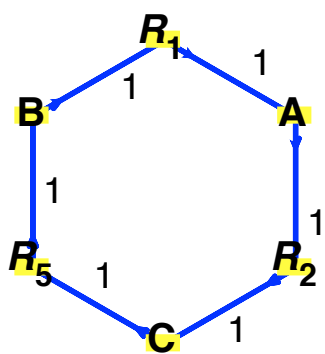
No. 7: injectiveEx2\_1573.csv

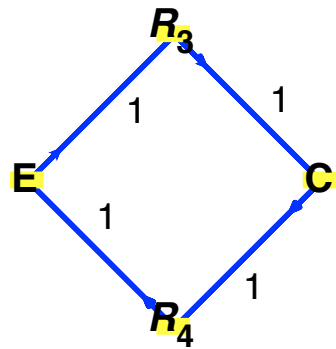
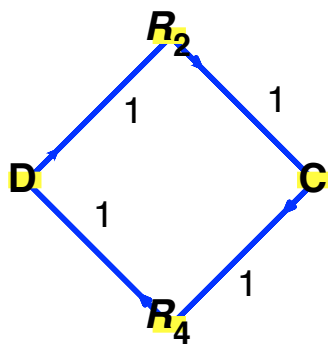
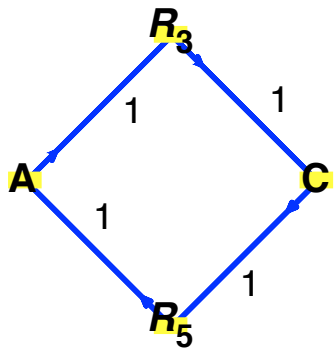




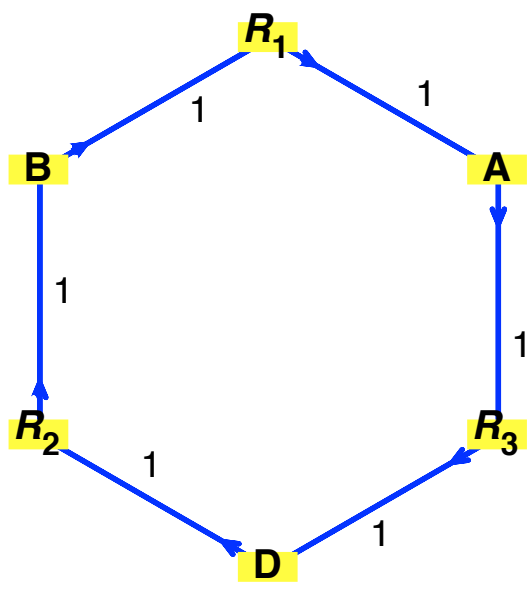
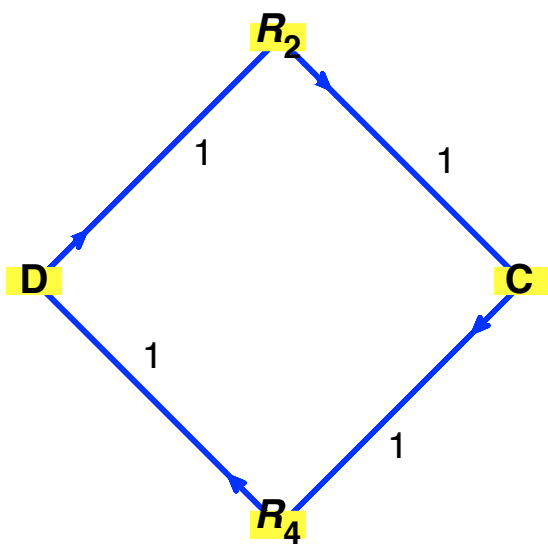


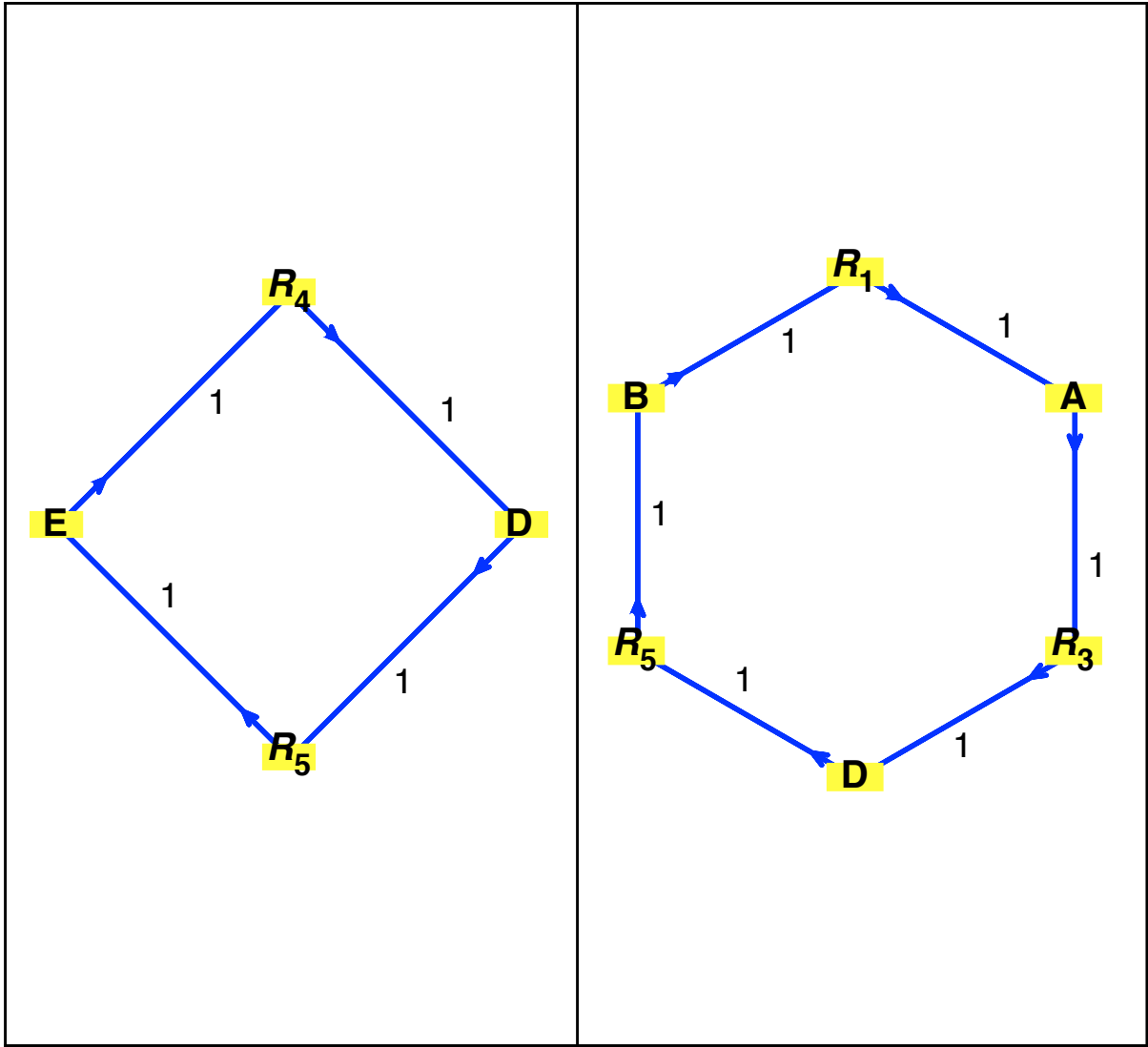
No. 8: injectiveEx2\_1749.csv



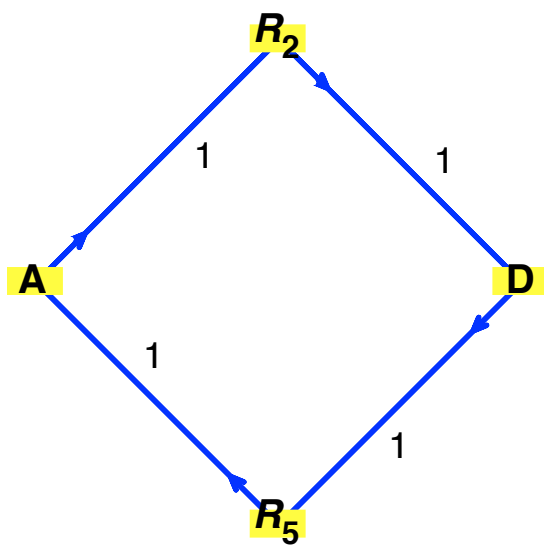
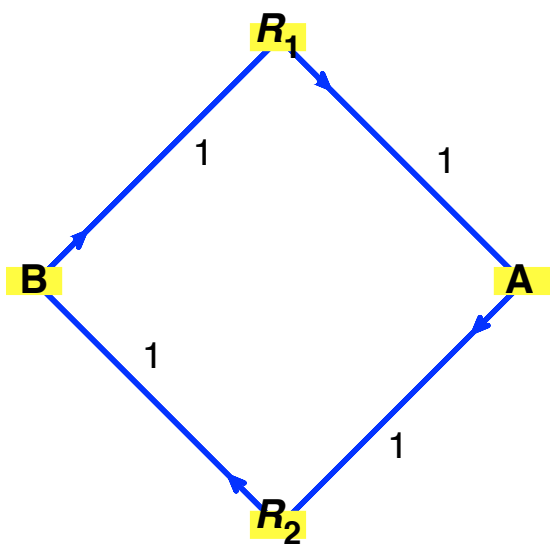


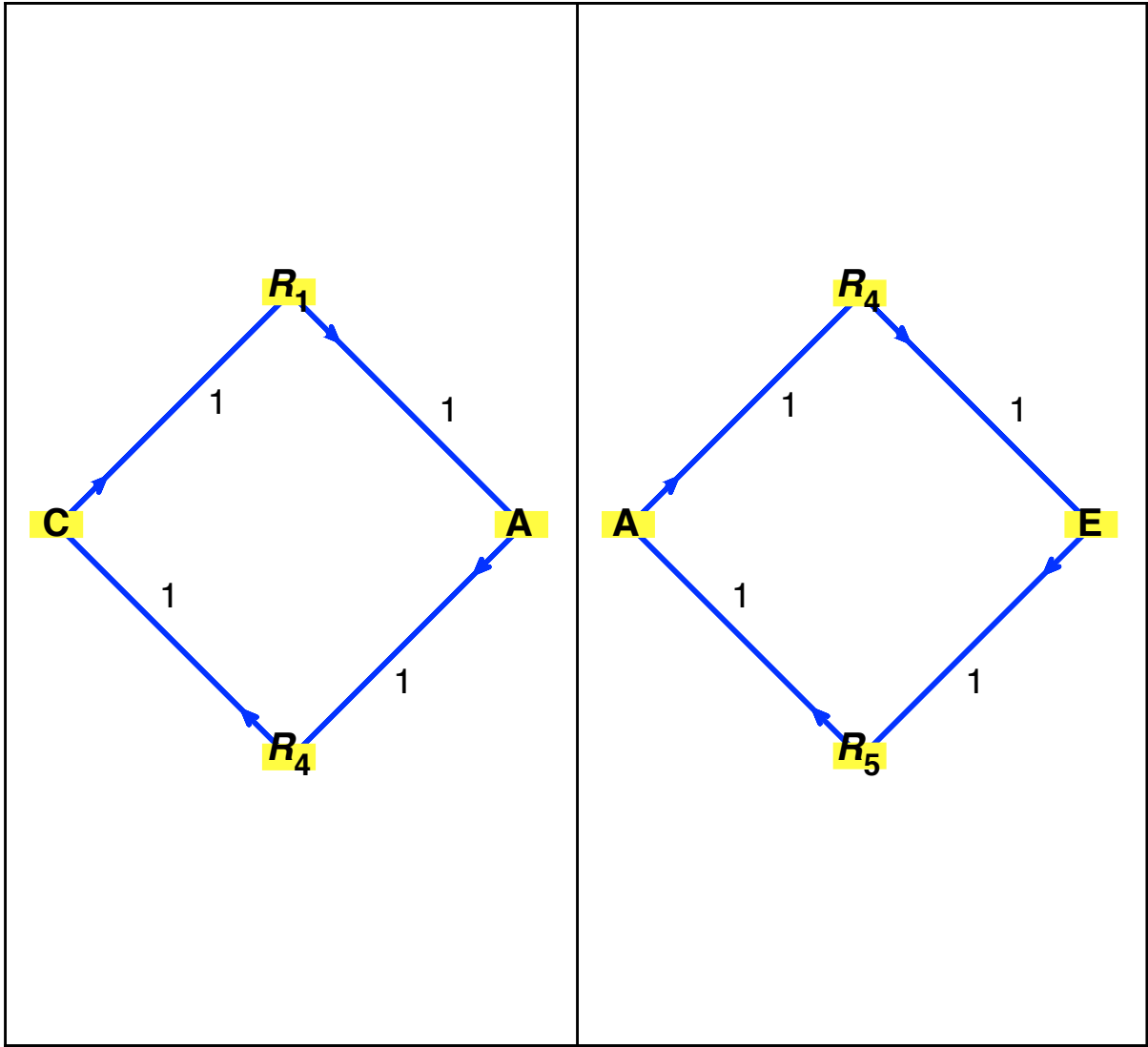
No. 9: injectiveEx2\_2052.csv





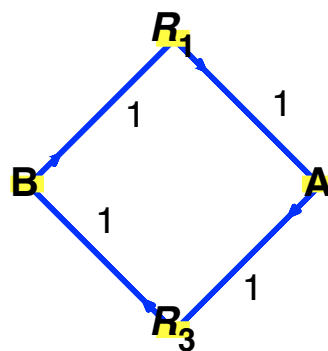
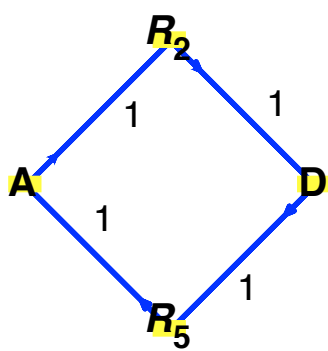
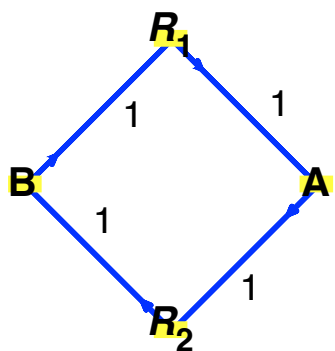
No. 10: injectiveEx2\_4180.csv

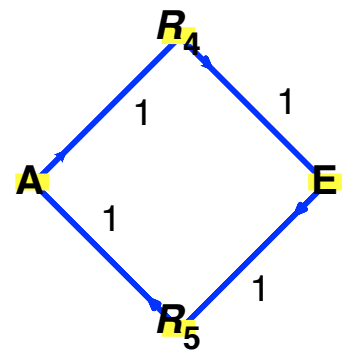
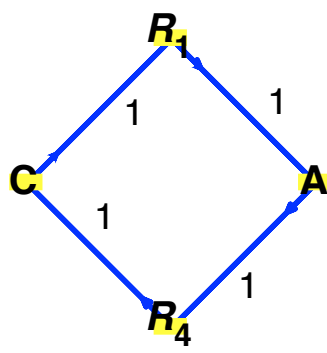
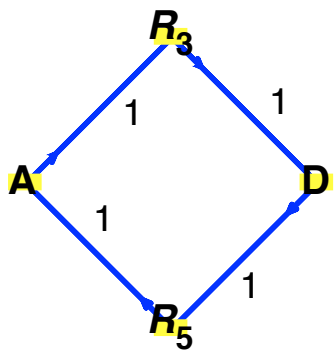




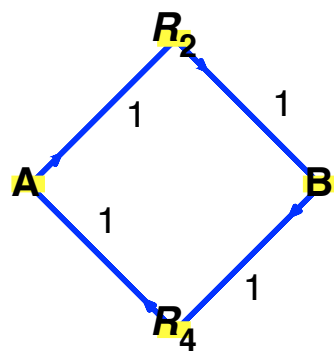
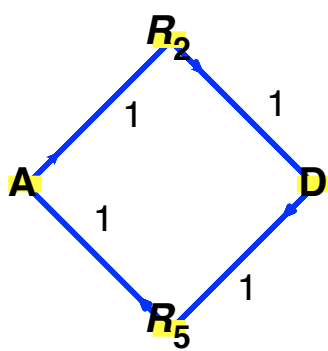
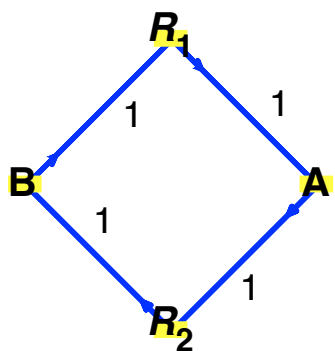
No. 11: injectiveEx2\_4203.csv WRONG.csv

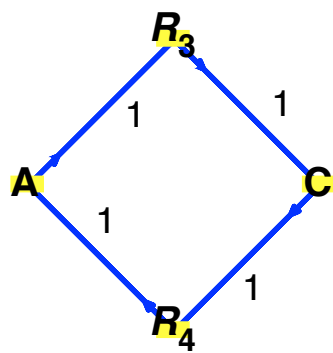
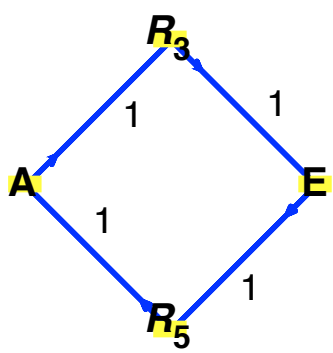
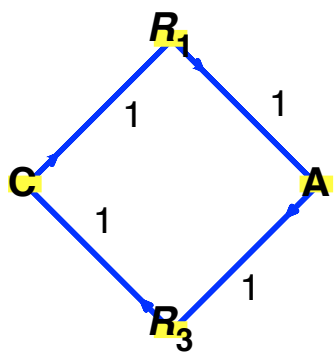






No. 12: injectiveEx2\_4242.csv WRONG.csv





>

