Enumerate reaction networks composed with heteromultimer and single transformations
November 2014

This particular code is to enumerate all possible reaction networks (with certain dimensions, $n \times m$ where n is speciese number m is reaction number) composed of three types of interactions:

$$A + B \rightarrow C$$
$$\mathrm{D} \rightarrow E + F$$
$$G \rightarrow H$$

Those are heterodimerization, disassociation and single transformation.

With those three types of elementary reactions, we could construct a set of reaction networks, then we could use DSR graphs and bipartie to characteristic those networks if they are multistationary and has closed competition loop.

But before to go through such checking, we need to preclude situations that clearly not a complex balanced reaction network, by which mean it obeys the following three constraints:
0. Only allow elementary reactions described above, which is the starting point to construct the matrix;
   (more complex version should be including $I \rightarrow 2\ J$ and $2\ K \rightarrow L$ in future)
   list all possibly combinations and select m of those into matrix (sequence does not matter), then seperate into pos and neg matrices

1. Mass conservation;
   (it's very difficult to check, currently implemented without this checking but manual checking afterwards)

2. Complex banlanced: each species has at least one in flow and one out flow;
   (easy to check)
   Check pos matrix and neg matrix, (there is no zeros in columSum)

Then we need to:
4. check competition: there are at least one species has two -1 and there is another -1 in each of the according reactions;
   (in neg matrix, check if there are any two intersections of colSum and rowSum $\geqslant -2$ in a row are -1)
   find col indices, then get the other two species (competitors)

5. check loop: take indices of competitors, do the network searching, find the loop from one to another and then from the other to this one.

Cluster matrix into four categories: bistable with closed competition loop, bistable without closed competition loop, monostable with closed competition loop, monostable without closed competition loop.

Some functions might need:
sprintf(fmt, x1, ..., xn)
StringTools[Join](stringList, sep)
StringTools[CaseJoin](stringList)

mkdir(dirName)
FileTools[RemoveDirectory](dirName, options)
FileTools[Remove](file, file2, ...)
ListDirectory(dir, opt1, opt2, ...)
ArrayTools[AddAlongDimension](A,dim)

Initializations

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

Functions for multistationality checking (execute before proceeding)

Functions for constructing stoichiomatric matrix and examine the existence of competition and closed loop.

1. To enumerate stoichiomatric vectors (reaction patterns)

List all reaction types based on the number of species. $R_n = \binom{n}{2}\cdot 2 + \binom{n}{3}\cdot 6$

```
> listRs := proc(n)
    local R, r, se, i, j, k, sign, l :
    r := binomial(n, 2)·2 + binomial(n, 3)·6 :
    R := Matrix(r, n) :
    # Now construct the reaction pattern matrix
    i := 1 :

    # here we first consider single transformation
    for se from -1 to 1 by 2 do
      for j from 1 to n by 1 do
        for k from j + 1 to n by 1 do
```

```
            R[i, j] := se :
            R[i, k] := -se :
            i := i + 1 :
          end do:
        end do:
      end do:

      # now consider heterodimerization and disassociation
      for sign from -1 to 1 by 2 do
        for j from 1 to n by 1 do
          for k from j + 1 to n by 1 do
            for l from k + 1 to n by 1 do
              R[i, j] := sign :
              R[i, k] := -sign :
              R[i, l] := -sign :
              i := i + 1 :
              R[i, j] := sign :
              R[i, k] := sign :
              R[i, l] := -sign :
              i := i + 1 :
              R[i, j] := sign :
              R[i, k] := -sign :
              R[i, l] := sign :
              i := i + 1 :
            end do:
          end do:
        end do:
      end do:

      # in this case we don't consider homodimerization and disassociation

      # Here transpose the R
      # R:=Transpose(R) :
       # no need to ranspose, need to assign rows to untransposed A's rows
      return(R) :
    end proc:
```

Here is a long function to enumerate first two reaction pattern (will reduce large mount of symmetric reactions).

```
> listR2 := proc(n)
      local R2, r2, R3, r3, R4, r4 :
      if n ≥ 4 then
        r2 := 29·2 :
        R2 := Matrix(r2, 4) :
```

 # in this case we don't consider homodimerization and disassociation

now we construct the reaction pattern with n = 4

$R2[1] := \langle 1, -1 \rangle : R2[2] := \langle 0, 0, 1, -1 \rangle :$
$R2[3] := \langle 1, -1 \rangle : R2[4] := \langle 0, 1, -1 \rangle :$
$R2[5] := \langle 1, -1 \rangle : R2[6] := \langle 1, 0, -1 \rangle :$
$R2[7] := \langle 1, -1 \rangle : R2[8] := \langle 0, -1, 1 \rangle :$
$R2[9] := \langle 1, -1 \rangle : R2[10] := \langle -1, 0, 1 \rangle :$
$R2[11] := \langle 1, -1 \rangle : R2[12] := \langle -1, 1 \rangle :$
$R2[13] := \langle 1, -1 \rangle : R2[14] := \langle 0, 1, -1, -1 \rangle :$
$R2[15] := \langle 1, -1 \rangle : R2[16] := \langle 1, 0, -1, -1 \rangle :$
$R2[17] := \langle 1, -1 \rangle : R2[18] := \langle 0, -1, 1, -1 \rangle :$
$R2[19] := \langle 1, -1 \rangle : R2[20] := \langle -1, 0, 1, -1 \rangle :$
$R2[21] := \langle 1, -1 \rangle : R2[22] := \langle 0, 1, 1, -1 \rangle :$
$R2[23] := \langle 1, -1 \rangle : R2[24] := \langle 1, 0, 1, -1 \rangle :$
$R2[25] := \langle 1, -1 \rangle : R2[26] := \langle 0, -1, 1, 1 \rangle :$
$R2[27] := \langle 1, -1 \rangle : R2[28] := \langle -1, 0, 1, 1 \rangle :$
$R2[29] := \langle 1, -1 \rangle : R2[30] := \langle -1, -1, 1 \rangle :$
$R2[31] := \langle 1, -1 \rangle : R2[32] := \langle 1, 1, -1 \rangle :$
$R2[33] := \langle 1, -1, -1 \rangle : R2[34] := \langle 0, 1, -1, -1 \rangle :$
$R2[35] := \langle 1, -1, -1 \rangle : R2[36] := \langle 1, 0, -1, -1 \rangle :$
$R2[37] := \langle 1, -1, -1 \rangle : R2[38] := \langle 0, -1, -1, 1 \rangle :$
$R2[39] := \langle 1, -1, -1 \rangle : R2[40] := \langle -1, -1, 0, 1 \rangle :$
$R2[41] := \langle 1, -1, -1 \rangle : R2[42] := \langle 0, 1, 1, -1 \rangle :$
$R2[43] := \langle 1, -1, -1 \rangle : R2[44] := \langle 1, 1, 0, -1 \rangle :$
$R2[45] := \langle 1, -1, -1 \rangle : R2[46] := \langle 0, 1, -1, 1 \rangle :$
$R2[47] := \langle 1, -1, -1 \rangle : R2[48] := \langle -1, 1, 0, 1 \rangle :$
$R2[49] := \langle 1, -1, -1 \rangle : R2[50] := \langle -1, 1, 1 \rangle :$
$R2[51] := \langle 1, 1, -1 \rangle : R2[52] := \langle 0, 1, 1, -1 \rangle :$
$R2[53] := \langle 1, 1, -1 \rangle : R2[54] := \langle 1, 1, 0, -1 \rangle :$
$R2[55] := \langle 1, 1, -1 \rangle : R2[56] := \langle 0, 1, -1, 1 \rangle :$
$R2[57] := \langle 1, 1, -1 \rangle : R2[58] := \langle 1, -1, 0, 1 \rangle :$

if $n \geq 5$ **then**

now we add the reaction pattern with n = 5

$r3 := 43 \cdot 2 :$
$R3 := Matrix(r3, 5) :$
$R3[1..r2] := R2[\ ] :$

$R3[59] := \langle 1, -1 \rangle : R3[60] := \langle 0, 0, 1, -1, -1 \rangle :$
$R3[61] := \langle 1, -1 \rangle : R3[62] := \langle 0, 0, 1, 1, -1 \rangle :$
$R3[63] := \langle 1, -1, -1 \rangle : R3[64] := \langle 0, 0, 1, -1, -1 \rangle :$
$R3[65] := \langle 1, -1, -1 \rangle : R3[66] := \langle 1, 0, 0, -1, -1 \rangle :$
$R3[67] := \langle 1, -1, -1 \rangle : R3[68] := \langle 0, 0, -1, 1, -1 \rangle :$
$R3[69] := \langle 1, -1, -1 \rangle : R3[70] := \langle -1, 0, 0, 1, -1 \rangle :$
$R3[71] := \langle 1, -1, -1 \rangle : R3[72] := \langle 0, 0, 1, 1, -1 \rangle :$

```
R3[73] := ⟨1,-1,-1⟩ : R3[74] := ⟨1, 0, 0, 1,-1⟩ :
R3[75] := ⟨1,-1,-1⟩ : R3[76] := ⟨0, 0,-1, 1, 1⟩ :
R3[77] := ⟨1,-1,-1⟩ : R3[78] := ⟨-1, 0, 0, 1, 1⟩ :
R3[79] := ⟨1, 1,-1⟩ : R3[80] := ⟨0, 0, 1, 1,-1⟩ :
R3[81] := ⟨1, 1,-1⟩ : R3[82] := ⟨1, 0, 0, 1,-1⟩ :
R3[83] := ⟨1, 1,-1⟩ : R3[84] := ⟨0, 0,-1, 1, 1⟩ :
R3[85] := ⟨1, 1,-1⟩ : R3[86] := ⟨-1, 0, 0, 1, 1⟩ :

if n ≥ 6 then
  ## now we add the reaction pattern with n = 6

  r4 := 46·2 :
  R4 := Matrix(r4, n) :
  R4[1 ..r3] := R3[ ] :

  R4[87] := ⟨1,-1,-1⟩ : R4[88] := ⟨0, 0, 0, 1,-1,-1⟩ :
  R4[89] := ⟨1,-1,-1⟩ : R4[90] := ⟨0, 0, 0, 1, 1,-1⟩ :
  R4[91] := ⟨1, 1,-1⟩ : R4[92] := ⟨0, 0, 0, 1, 1,-1⟩ :

  return(R4) :
else
  return(R3) :
end if:
        else
          return(R2) :
        end if:
      else
        error "ERROR: n is smaller than 4"
      end if:

    end proc:
```

2. Now we can construct stoichiomatric matrix based on the reaction patterns, and examine their properties.

Here, we construct the stoichiomatric matrices.

The total number of stoichiomatric matrices is $\binom{R_n}{m}$, which is still a huge number. But currently there seems no other better options.

We only consider when $m \leq 6$.

The function(s) to examine existence of competition and loops a stoichiomatric matrix.

This function is used to check the existence of competition in the system

```
> existcompetition := proc(A)
```

```
    local i, j, m, n, count, check, An, Rs, Cs, Checks :
    An := (A-|A|)/2 :
    Rs := AddAlongDimension(A, 2) :
    Cs := AddAlongDimension(An, 1) :
    m := Dimension(A)[1] : n := Dimension(A)[2] :

    count := 0 :
    for j from 1 to n by 1 do
      if Cs[j] ≤ -2 then
        check := 0 :
        for i from 1 to m by 1 do
          if Rs[i] = -1 and A[i, j] ≤ -1 then
            check := check + 1 :
          end if:
        end do:

        if check ≥ 2 then
          count := count + 1 :
          return(count) :
        end if:
      end if:
    end do:
    return(count) :
  end proc:
```

This function is to check the existence of closed positive feedback loop with competition. It returns
a number if 0 then no competition, if 1 then only competition no loop, if 2 then with competition loop no switching (back), if 3 then with competition loop and switching (back).

```
> existcompetitionloop := proc(A)
    local i, j, m, n, count, check, An, Rs, Cs, Checks, exist, k, l, loops, switches, comps, p,
     q, v, r, s :
    An := (A-|A|)/2 :
    Rs := AddAlongDimension(A, 2) :
    Cs := AddAlongDimension(An, 1) :
    m := Dimension(A)[1] : n := Dimension(A)[2] :

    count := 0 : exist := 0 : loops := 0 : switches := 0 :
    #print(Rs);
    #print(Cs);
    for j from 1 to n by 1 do
      if Cs[j] ≤ -2 then
        check := 0 :
        Checks := Array( ) :
        k := 0 :
```

```
for i from 1 to m by 1 do
  if Rs[i] = -1 and A[i, j] ≤ -1 then
    k := k + 1 :
    check := check + 1 :
    Checks(k) := i :
  end if:
end do:
#print(check);
if check ≥ 2 then
  count := count + 1 :
  #print(count);
  comps := Size(Checks, 2) :
  v := Vector(5) :
  v[3] := j :
  for p from 1 to comps by 1 do
    v[1] := Checks(p) :
    if A[Checks[p], j] = -1 then
      for r from 1 to n by 1 do
        if A[Checks[p], r] = -1 and r ≠ j then
          v[4] := r :
        end if:
      end do:
      if v[4] = 0 then
        error "Can not find the other reactant" :
      end if:
    elif A[Checks[p], j] = -2 then
      v[4] := j :
    else
      error "The reactant is neither -1 nor -2" :
    end if:
    for q from p + 1 to comps by 1 do
      v[2] := Checks(q) :
      if A[Checks[q], j] = -1 then
        for r from 1 to n by 1 do
          if A[Checks[q], r] = -1 and r ≠ j then
            v[5] := r :
          end if:
        end do:
        if v[5] = 0 then
          error "Can not find the other reactant for second reaction" :
        end if:
      elif A[Checks[q], j] = -2 then
        v[5] := j :
      else
        error "The reactant is neither -1 nor -2 in second reaction" :
      end if:
      #print(v);
      if v[4] ≠ v[5] then
```

```
                    l := 0 :
                    l := checkloop(A, v) :
                    if l = 1 then
                      loops := loops + 1 :
                      #print(loops);
                    elif l = 2 then
                      switches := switches + 1 :
                      exist := 3 :
                      return(exist) :
                    end if:
                  end if:
                end do:
              end do:
            end if:
          end if:
        end do:

        if count ≥ 1 then
          if loops ≥ 1 then
            exist := 2 :
          else
            exist := 1 :
          end if:
        end if:
        return(exist) :
      end proc:
```

This function is used to check the competition loop (return 1) and switches (return 2), if no exist any of those return 0.

```
> checkloop := proc(A, v)
    local exist, loop, switch, Q, i, j, k, m, n, visited :
    m := Dimension(A)[1] : n := Dimension(A)[2] :
    loop := 0 :
    switch := 0 :
    exist := 0 :
    Q := queue[new]( ) :
    visited := Vector(n) :
    queue[enqueue](Q, v[4]) :
    while not queue[empty](Q) do
      j := queue[dequeue](Q) :
      for i from 1 to m do
        if i ≠ v[1] and i ≠ v[2] then
          if A[i, j] ≤ -1 then
            for k from 1 to n do
              if A[i, k] ≥ 1 then
                if k = v[5] then
```

```
                    loop := loop + 1 :
                    break:
                  elif k ≠ v[4] then
                    if visited[k] = 0 then
                      queue[enqueue](Q, k) :
                      visited[k] := 1 :
                    end if:
                  end if:
                end if:
              end do:
              if loop ≥ 1 then
                break:
              end if:
            end if:
          end if:
        end do:
        if loop ≥ 1 then
          queue[clear](Q) :
          break:
        end if:
      end do:

      queue[clear](Q) :
      visited := Vector(n) :
      if loop ≥ 1 then
        exist := 1 :
        for j from 1 to n do
          if j = v[4] then
            switch := switch + 1 :
            exist := 2 :
            return(exist) :
          elif A[v[2], j] ≥ 1 and visited[j] = 0 then
            queue[enqueue](Q, j) :
            visited[j] := 1 :
          end if:
        end do:

        while not queue[empty](Q) do
          j := queue[dequeue](Q) :
          for i from 1 to m do
            if i ≠ v[1] and i ≠ v[2] then
              if A[i, j] ≤ -1 then
                for k from 1 to n do
                  if A[i, k] ≥ 1 then
                    if k = v[4] then
                      switch := switch + 1 :
                      exist := 2 :
                      return(exist) :
```

```
                    elif k ≠ v[5] and visited[k] = 0 then
                        queue[enqueue](Q, k) :
                        visited[k] := 1 :
                    end if:
                end if:
            end do:
        end if:
    end if:
  end do:
end do:
end if:

loop := 0 :
queue[clear](Q) :
queue[enqueue](Q, v[5]) :
visited := Vector(n) :
while not queue[empty](Q) do
  j := queue[dequeue](Q) :
  for i from 1 to m do
    if i ≠ v[2] and i ≠ v[1] then
      if A[i, j] ≤ -1 then
        for k from 1 to n do
          if A[i, k] ≥ 1 then
            if k = v[4] then
              loop := loop + 1 :
              break:
            elif k ≠ v[5] and visited[k] = 0 then
              queue[enqueue](Q, k) :
              visited[k] := 1 :
            end if:
          end if:
        end do:
        if loop ≥ 1 then
          break:
        end if:
      end if:
    end if:
  end do:
  if loop ≥ 1 then
    queue[clear](Q) :
    break:
  end if:
end do:

queue[clear](Q) :
visited := Vector(n) :
if loop ≥ 1 then
  exist := 1 :
```

```
        for j from 1 to n do
          if j = v[5] then
            switch := switch + 1 :
            exist := 2 :
            return(exist) :
          elif A[v[1], j] ≥ 1 and visited[j] = 0 then
            queue[enqueue](Q, j) :
            visited[j] := 1 :
          end if:
        end do:

        while not queue[empty](Q) do
          j := queue[dequeue](Q) :
          for i from 1 to m do
            if i ≠ v[1] and i ≠ v[2] then
              if A[i, j] ≤ -1 then
                for k from 1 to n do
                  if A[i, k] ≥ 1 then
                    if k = v[5] then
                      switch := switch + 1 :
                      exist := 2 :
                      return(exist) :
                    elif k ≠ v[4] and visited[k] = 0 then
                      queue[enqueue](Q, k) :
                      visited[k] := 1 :
                    end if:
                  end if:
                end do:
              end if:
            end if:
          end do:
        end do:
      end if:

      return(exist) :
    end proc:
```

The function to check if the stoichiomatric matrix is mass conserved.

First check the passed matrix $A_{m \times n}$ (must be in a consistent form)

```
> ismassconserved := proc(A)
    local R, N, NS, m, n, absAdd, Add, x, y, z, e, i, nsAdd, nsAbsAdd, a, b, c :
    m := 0 :
    n := Dimension(A)[2] :
    absAdd := AddAlongDimension(|A|, 1) :
    z := Search(0, absAdd) :
    if z = 0 then
      Add := AddAlongDimension(A, 1) :
```

```
      x := Search(0, VectorAdd(absAdd, Add, 1, -1)) :
      if x = 0 then
        y := Search(0, VectorAdd(absAdd, Add, 1, 1)) :
        if y = 0 then
          N := NullSpace(A) :
          e := numelems(N) :
          if e > 0 then
            NS := Matrix(e, n) :
            for i from 1 to e by 1 do
              NS[i] := N[i] :
            end do:
            nsAbsAdd := AddAlongDimension(|NS|, 1) :
            a := Search(0, nsAbsAdd) :
            if a = 0 then
              nsAdd := AddAlongDimension(NS, 1) :
              b := Search(0, VectorAdd(nsAbsAdd, nsAdd, 1, 1)) :
              if b = 0 then
                m := 1 :
              end if:
            end if:
          end if:
        end if:
      end if:
    end if:
    return(m) :
  end proc:
```

Construct and examine the properties of all stoichiomatric matrices.

```
> constrM := proc(n, m)
    local R, tA, A, iA, Z, r, g, h, i, j, k, l, total, right, mc, V, R2, r2, injective, injective0,
     injective1, injectiveEx, fileName, matrixData, interV, comp, injectiveEx0,
     injectiveEx1, injectiveEx2, injectiveEx3, s, selected, pfloops, unique, pfintersect,
     pfcount, f :
    r := binomial(n, 2)·2 + binomial(n, 3)·6 :

    A := Matrix(m, n) :

    R := listRs(n) :
    R2 := listR2(n) :

    if n = 4  then  r2 := 29·2 :  end if:
    if n = 5  then  r2 := 43·2 :  end if:
    if n ≥ 6  then  r2 := 46·2 :  end if:
    if n < 4 then error "ERROR: n is smaller than 4" end if:
```

```
total := r2/2 · binomial(r, m − 2) :

# here we use some variable to count how many reactions are correct.
right := 0 : injective0 := 0 : injective1 := 0 :
#injectiveEx0:=0:injectiveEx1:=0:injectiveEx2:=0: injectiveEx3:=0:
for l from 1 to r2 − 1 by 2 do
  A[1] := R2[l] :
  A[2] := R2[l + 1] :
  for g from 1 to r by 1 do
    # here we should check whether this is duplicate of the fixed two reactions.
    #######
    A[3] := R[g] :
    for h from g + 1 to r by 1 do
      A[4] := R[h] :
      for i from h + 1 to r by 1 do
        A[5] := R[i] :
        #for j from i+1 to r by 1 do
          #A[6] := R[j] :
          # now we have matrix A, we need to exam A with constraints.
          mc := ismassconserved(A) :
          if mc = 1 then
            right := right + 1 :

            # check the existence of compeition and competition loop.
            comp := existcompetitionloop(A) :

            tA := Transpose(A) :

            # check the existence of intersecting positive feedback loops
            Z := findZ(tA) : s := Rank(tA) : selected := findloops(tA, Z) :
            pfloops := numelems(selected) :
            pfintersect := 0 :
            if pfloops ≥ 2 then
              pfcount := 0 :
              for f from 1 to numelems(selected) by 1 do
                pfcount := pfcount + numelems(selected[f]) :
              end do:
              unique := [ ] :
              for f from 1 to numelems(selected) by 1 do
                unique := [op(unique), op(selected[f])] :
              end do:
              if pfcount > numelems(MakeUnique(unique)) then
                pfintersect := 1 :
              end if:
            end if:

            iA := Transpose(A) :
```

```
# simple injectivity check
injective := isinjective(iA) :
if injective = 0 then
    injective0 := injective0 + 1 :
    injectiveEx := isinjectiveextended(iA) :
    if injectiveEx = 1 or injectiveEx = 3 then
        if comp = 3 then
            if pfintersect = 1 then
                fileName
 := sprintf(
"%1dspecies/nonmultistationary/competitionloop_intersectingloops/injectiveEx%1d_\
%d.csv", n, injectiveEx, right) :
                ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
            else
                fileName
 := sprintf(
"%1dspecies/nonmultistationary/competitionloop_nointersectingloops/injectiveEx%1\
d_%d.csv", n, injectiveEx, right) :
                ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
            end if:
        else
            if pfintersect = 1 then
                fileName
 := sprintf(
"%1dspecies/nonmultistationary/nocompetitionloop_intersectingloops/injectiveEx%1\
d_%d.csv", n, injectiveEx, right) :
                ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
            else
                fileName
 := sprintf(
"%1dspecies/nonmultistationary/nocompetitionloop_nointersectingloops/injectiveEx\
%1d_%d.csv", n, injectiveEx, right) :
                ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
            end if:
        end if:
    elif injectiveEx = 0 or injectiveEx = 2 then
        if comp = 3 then
            if pfintersect = 1 then
                fileName
 := sprintf(
"%1dspecies/multistationary/competitionloop_intersectingloops/injectiveEx%1d_%d.
csv", n, injectiveEx, right) :
                ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
```

```
                              else
                                fileName
 := sprintf(
"%1dspecies/multistationary/competitionloop_nointersectingloops/injectiveEx%1d_%\
d.csv", n, injectiveEx, right) :
                                ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
                              end if:
                            else
                              if pfintersect = 1 then
                                fileName
 := sprintf(
"%1dspecies/multistationary/nocompetitionloop_intersectingloops/injectiveEx%1d_%\
d.csv", n, injectiveEx, right) :
                                ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
                              else
                                fileName
 := sprintf(
"%1dspecies/multistationary/nocompetitionloop_nointersectingloops/injectiveEx%1d\
_%d.csv", n, injectiveEx, right) :
                                ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
                              end if:
                            end if:
                          else
                            error "ERROR: injectivity extended of A is not any of 0 to 3."
                          end if:
                        elif injective = 1 then
                          injective1 := injective1 + 1 :
                          if comp = 3 then
                            if pfintersect = 1 then
                              fileName
 := sprintf(
"%1dspecies/nonmultistationary/competitionloop_intersectingloops/injective%1d_%d\
.csv", n, injective, right) :
                              ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
                            else
                              fileName
 := sprintf(
"%1dspecies/nonmultistationary/competitionloop_nointersectingloops/injective%1d_\
%d.csv", n, injective, right) :
                              ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
                            end if:
                          else
                            if pfintersect = 1 then
```

```
                    fileName
 := sprintf(
"%1dspecies/nonmultistationary/nocompetitionloop_intersectingloops/injective%1d_\
%d.csv", n, injective, right) :
                    ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
                  else
                    fileName
 := sprintf(
"%1dspecies/nonmultistationary/nocompetitionloop_nointersectingloops/injective%1\
d_%d.csv", n, injective, right) :
                    ExportMatrix(fileName, iA, target = csv, format
 = rectangular, mode = ascii) :
                  end if:
                end if:
              else
                error "ERROR: the injectivity of iA is neither 0 nor 1."
              end if:
            end if:
          #end do:
        end do:
      end do:
    end do:
  end do:

  V := [injective0, injective1, right, total, r, r2] :
  return(V) :
end proc:
```

Testing

Here we test all functions:

```
> V := constrM(5, 5) # just count right matrices ~ 2360s (40 mins) (9.29s to 2359.41s)
```

$V := [0, 0, 0, 0, 0, 0, 9229, 3532880, 80, 86]$ **(3.1)**

```
> V := constrM(5, 5)
    # also count injective extended matrices and export bistable matrices ~ 2716s
                                                    #(45 mins) (4621.73 s
    to 7337.76 s)
```

$V := [65, 578, 87, 3977, 4707, 4522, 9229, 3532880, 80, 86]$ **(3.2)**

```
> V := constrM(5, 5)
    # both check injectivity extended and competition loop ~2763.17s (46 mins)  (15s to
```

2778.17s)

$$V := [65, 578, 87, 3977, 4707, 4522, 9229, 3532880, 80, 86] \quad \textbf{(3.3)}$$

> $V := constrM(5, 5)$ # *check competition loop and intersecting loops ~ (36s to)*

$$V := [4707, 4522, 9229, 3532880, 80, 86] \quad \textbf{(3.4)}$$

> $A := ImportMatrix($"5species/injective/right_1479_injective1.csv"$)$

$$A := \begin{bmatrix} 1 & 0 & -1 & -1 & 0 \\ -1 & -1 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & -1 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix} \quad \textbf{(3.5)}$$

> $Z := findZ(A) : s := Rank(A) : selected := findloops(A, Z)$

$$selected := [\ ] \quad \textbf{(3.6)}$$

> $isinjectiveextended(A)$

$$3 \quad \textbf{(3.7)}$$

> A

$$\begin{bmatrix} 1 & 0 & -1 & -1 & 0 \\ -1 & -1 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & -1 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix} \quad \textbf{(3.8)}$$

> $tA := ImportMatrix($"5species/bistability/bistable_1.csv"$)$

$$tA := \begin{bmatrix} 1 & 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & -1 \end{bmatrix} \quad \textbf{(3.9)}$$

> $Z := findZ(tA) : n := Dimension(tA)[1] : m := Dimension(tA)[2] : s := Rank(tA) :$
> $selected := findloops(tA, Z)$

$$selected := [[[[R_1, S_1], 1], [[S_1, R_3], 1], [[R_3, S_5], 1], [[S_5, R_4], 1], [[R_4, S_2], 1], [[S_2, R_1], 1]], [[[R_3, S_5], 1], [[S_5, R_4], 1], [[R_4, S_1], 1], [[S_1, R_3], 1]], [[[R_2, S_3], 1], [[S_3, R_3], 1], [[R_3, S_5], 1], [[S_5, R_5], 1], [[R_5, S_4], 1], [[S_4, R_2], 1]], [[[R_3, S_5], 1], [[S_5, R_5], 1], [[R_5, S_3], 1], [[S_3, R_3], 1]]] \quad \textbf{(3.10)}$$

> tA

$$\begin{bmatrix} 1 & 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & -1 \end{bmatrix} \quad \textbf{(3.11)}$$

> $isinjectiveextended(tA)$

$$2 \quad \textbf{(3.12)}$$

> $count := 0:$

> **for** i **from** 1 **to** $numelems(selected)$ **by** 1 **do**
> $count := count + numelems(selected[i]):$
> **end do**:

> $unique := MakeUnique(selected)$

$$unique := [[[[R_1, S_1], 1], [[S_1, R_3], 1], [[R_3, S_5], 1], [[S_5, R_4], 1], [[R_4, S_2], 1], [[S_2, R_1], 1]], [[[R_3, S_5], 1], [[S_5, R_4], 1], [[R_4, S_1], 1], [[S_1, R_3], 1]], [[[R_2, S_3], 1], [[S_3, R_3], 1], [[R_3, S_5], 1], [[S_5, R_5], 1], [[R_5, S_4], 1], [[S_4, R_2], 1]], [[[R_3, S_5], 1], [[S_5, R_5], 1], [[R_5, S_3], 1], [[S_3, R_3], 1]]] \quad \textbf{(3.13)}$$

> $unique := [\ ]$

$$unique := [\ ] \quad \textbf{(3.14)}$$

> **for** i **from** 1 **to** $numelems(selected)$ **by** 1 **do**
> $unique := [op(unique), op(selected[i])]:$
> **end do**:

> $unique$

$$[[[R_1, S_1], 1], [[S_1, R_3], 1], [[R_3, S_5], 1], [[S_5, R_4], 1], [[R_4, S_2], 1], [[S_2, R_1], 1], [[R_3, S_5], 1], [[S_5, R_4], 1], [[R_4, S_1], 1], [[S_1, R_3], 1], [[R_2, S_3], 1], [[S_3, R_3], 1], [[R_3, S_5], 1], [[S_5, R_5], 1], [[R_5, S_4], 1], [[S_4, R_2], 1], [[R_3, S_5], 1], [[S_5, R_5], 1], [[R_5, S_3], 1], [[S_3, R_3], 1]] \quad \textbf{(3.15)}$$

> $unique := MakeUnique(unique)$

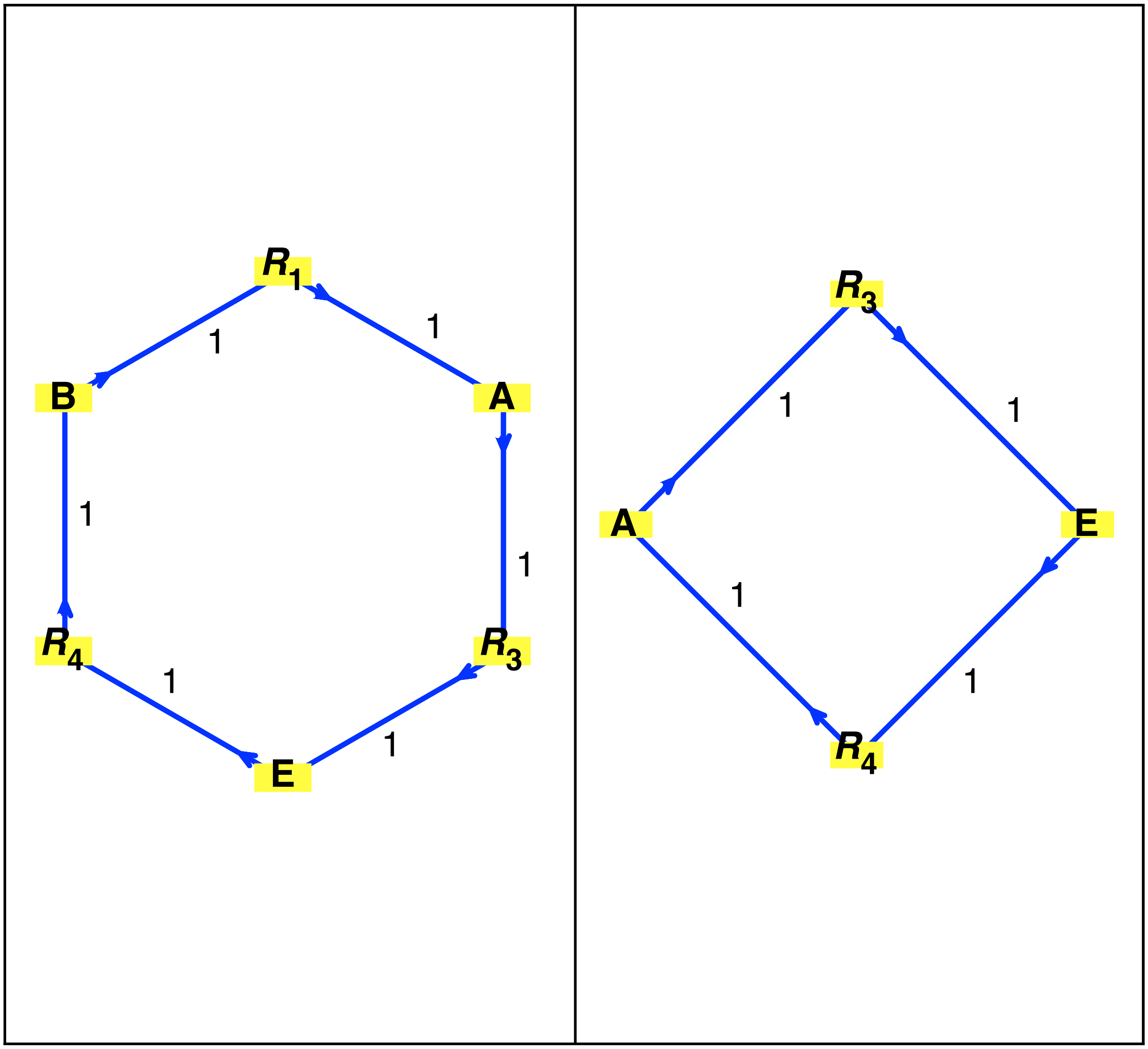
$$unique := [[[R_1, S_1], 1], [[S_1, R_3], 1], [[R_3, S_5], 1], [[S_5, R_4], 1], [[R_4, S_2], 1], [[S_2, R_1], 1], [[R_4, S_1], 1], [[R_2, S_3], 1], [[S_3, R_3], 1], [[S_5, R_5], 1], [[R_5, S_4], 1], [[S_4, R_2], 1], [[R_5, S_3], 1]] \quad \textbf{(3.16)}$$

> $numelems(unique)$

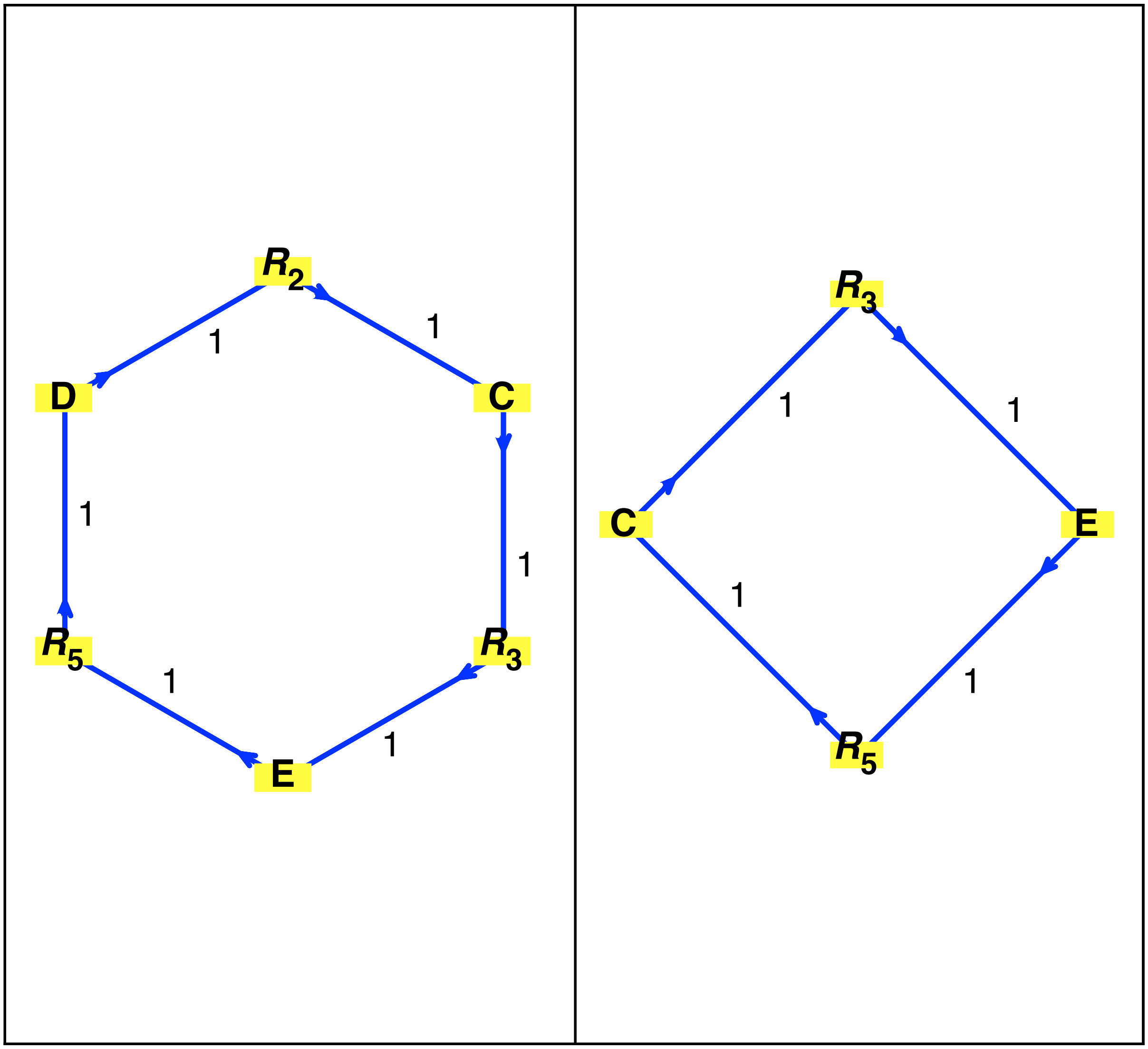
$$13 \quad \textbf{(3.17)}$$

> $speciessord := [\text{"A"}, \text{"B"}, \text{"C"}, \text{"D"}, \text{"E"}]:$

> $drawloops(selected, speciessord)$



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



Here is an example that without two species competing another species that could give rise bistability. It potentially implys that the competition is not necessary happening in between two species but also possible happening between two reactions.

> $A := ImportMatrix(\text{"5species/bistability/bistable\_10.csv"})$

$$A := \begin{bmatrix} 1 & -1 & -1 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \\ 0 & 1 & -1 & 1 & -1 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix} \tag{3.18}$$

> $existcompetitionloop(Transpose(A))$

$$3 \tag{3.19}$$

> $Z := findZ(A) : n := Dimension(A)[1] : m := Dimension(A)[2] : s := Rank(A)$

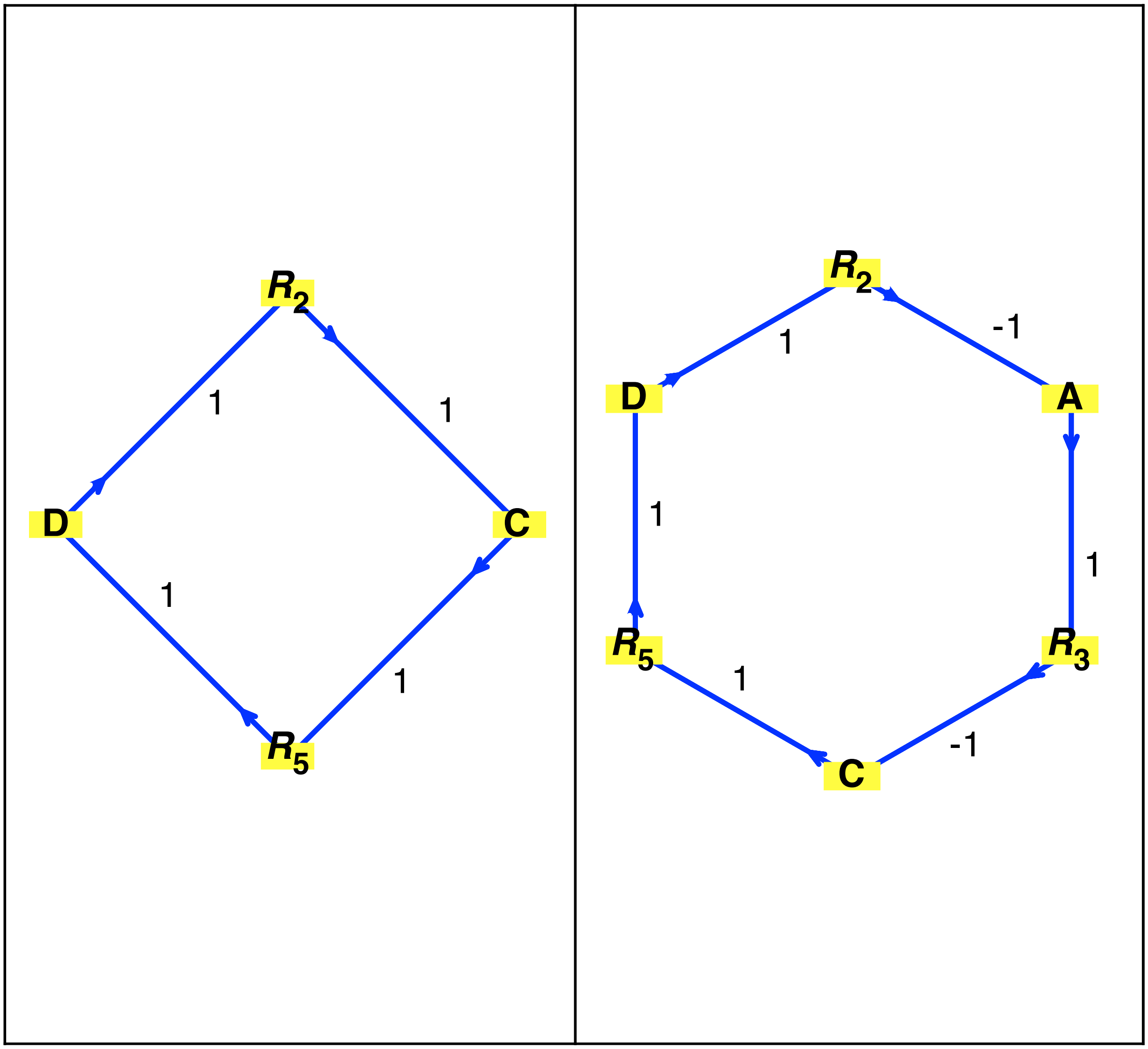
$$s := 3 \tag{3.20}$$

> $selected := findloops(\ )$

$$selected := [[[[R_2, S_3], 1], [[S_3, R_5], 1], [[R_5, S_4], 1], [[S_4, R_2], 1]], [[[R_2, S_1], -1], [[S_1, R_3], 1], [[R_3, S_3], -1], [[S_3, R_5], 1], [[R_5, S_4], 1], [[S_4, R_2], 1]]] \tag{3.21}$$

> $speciessord := ["A", "B", "C", "D", "E"]:$

> $drawloops(selected, speciessord)$



Here is an example with biological/chemical meaning, and also has the competition and closed loop also interchangable loops.

Now we change a little bit to this network:

> $A[1,3] := 0 : A[1,4] := 0 :$

> $A[4,3] := -1 : A[4,4] := 1 :$

> A

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 \\ 0 & 1 & -1 & 1 & -1 \\ 0 & -1 & -1 & 1 & 1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix} \tag{3.22}$$

> $Z := findZ(A) : n := Dimension(A)[1] : m := Dimension(A)[2] : s := Rank(A)$

$$s := 3 \tag{3.23}$$

> $selected := findloops(\,)$

$$selected := [\,] \tag{3.24}$$

> $speciessord := [$"A", "B", "C", "D", "E"$] :$

> $drawloops(selected, speciessord)$

> $isinjectiveextended(A)$

$$1 \tag{3.25}$$