

■ **Definition** (Deterministic Finite Automaton). A DFA is a structure $\mathcal{A} = (Q, \Sigma, \delta)$ consisting of the following:

- a finite set of states Q
- a finite *input alphabet* Σ
- a transition function $\delta : Q \times \Sigma \rightarrow Q$ which is total

we don't have to worry.

There are different states which represent the different states that the system can be in. The transition function takes a state and takes a letter from the alphabet and goes to another state. For example,

```
In[2]:= MyCartesianProduct[listsoflists_List] :=
  If[Length[listsoflists] == 1, listsoflists[[1]], If[Length[listsoflists] == 2,
    MyCartesianProduct[listsoflists[[1]], listsoflists[[2]],
    Map[Flatten[#, 1] &, MyCartesianProduct[Join[{MyCartesianProduct[
      listsoflists[[1]], listsoflists[[2]]}], Drop[listsoflists, 2]]]] ]]
```

```
In[3]:= findstate[DFARule_List, state_, alphabet_] :=
  If[DFARule[[1]][1, 1] == state && DFARule[[1]][1, 2] == alphabet,
    DFARule[[1]][2][1], findstate[Drop[DFARule, 1], state, alphabet]]
```

This function takes a DFA and plots it:

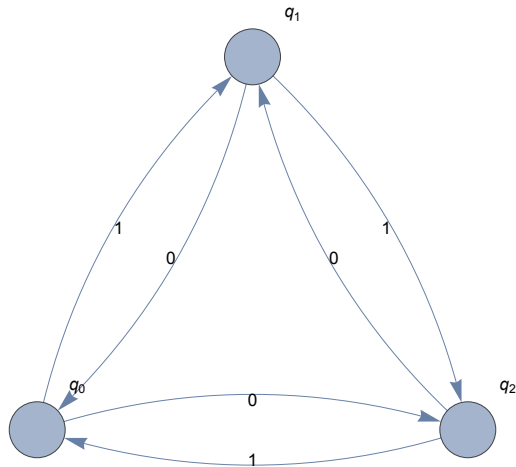
```
In[4]:= DFAPlot[{states_, alphabet_, transition_}] :=
  Module[{}],
  Graph[Flatten@
    Table[Table[Labeled[s → transition[s, a], a], {a, alphabet}], {s, states}],
    VertexLabels → Table[s → qs, {s, states}], VertexSize → 0.13]
  ]
```

```
In[5]:= DFAPlot[{states_, alphabet_, transition_}, bool_] :=
  Module[{}],
  If[bool == True,
    Graph[Flatten@
      Table[Table[Labeled[s → transition[s, a], a], {a, alphabet}], {s, states}],
      VertexLabels → Table[s → qs, {s, states}], VertexSize → 0.13],
    Graph[Flatten@Table[Table[s → transition[s, a], {a, alphabet}], {s, states}],
      VertexLabels → None, VertexSize → 0.13]
  ]
  ]
```

We can specify a DFA using a list of transition functions. For example, this DFA

```
In[*]:= {{0, 1} → {1}, {1, 1} → {2}, {2, 1} → {0}, {1, 0} → {0}, {2, 0} → {1}, {0, 0} → {2}}
Out[*]:= {{0, 1} → {1}, {1, 1} → {2}, {2, 1} → {0}, {1, 0} → {0}, {2, 0} → {1}, {0, 0} → {2}}
```

represents the DFA:



For instance, $\{0, 1\} \rightarrow \{1\}$ means $\delta(0, 1) = 1$ (so state $q_0 \rightarrow q_1$ under the letter 1). The following takes a DFA and converts into the list of transitions

```

In[6]:= DFAtoRule[{states_, alphabet_, transition_}] :=
  Module[{},
    Flatten[Table[Table[{s, a} → {transition[s, a]}, {a, alphabet}], {s, states}], 1]
  ]

```

this goes the opposite way

```

In[7]:= RuleToDFA[DFA_List] :=
  Module[{states, alphabet},
    states = DeleteDuplicates[ (#[[1]] [[1]]) & /@ DFA ];
    alphabet = DeleteDuplicates[ (#[[1]] [[2]]) & /@ DFA ];
    {states, alphabet, Function[{state, letter}, findstate[DFA, state, letter]]}
  ]

```

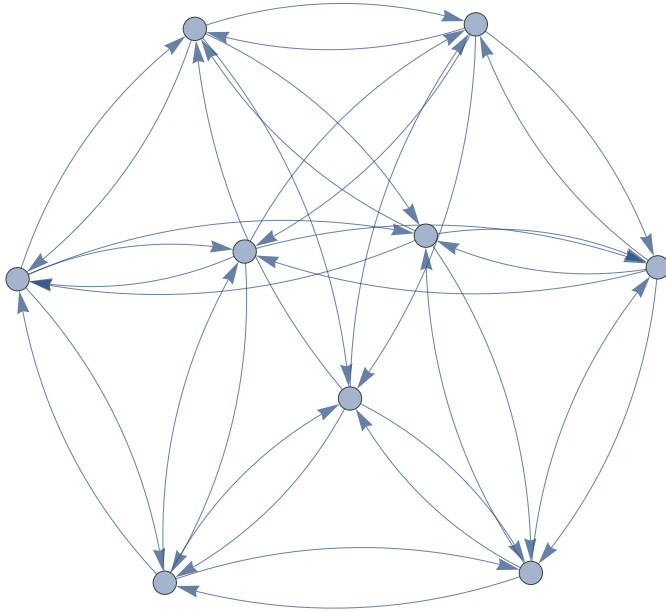
```

In[8]:= DFADirectProduct[DFA1Rule_List, DFA2Rule_List] :=
  Module[{DFA1, DFA2},
    DFA1 = RuleToDFA[DFA1Rule];
    DFA2 = RuleToDFA[DFA2Rule];
    {MyCartesianProduct[DFA1[[1]], DFA2[[1]],
     MyCartesianProduct[DFA1[[2]], DFA2[[2]], Function[{state, letter},
      {DFA1[[3]][state[[1]], letter[[1]], DFA2[[3]][state[[2]], letter[[2]]}]]}
  ]

```

```
In[11]:= DFAPlot[DFADirectProduct[{{0, 1} → {1}, {1, 1} → {2}, {2, 1} → {0},
  {1, 0} → {0}, {2, 0} → {1}, {0, 0} → {2}}, {{0, 1} → {1}, {1, 1} → {2},
  {2, 1} → {0}, {1, 0} → {0}, {2, 0} → {1}, {0, 0} → {2}}, False]
```

Out[11]=



DFA direct product of an arbitrary number of DFAs.

- **Definition** (Direct Product of DFAs). Let $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1)$ be a DFA. Let $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2)$ be another DFA. Then, consider the the DFA $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ is defined by $\mathcal{A} = (Q_1 \times Q_2, \Sigma_1 \times \Sigma_2, \delta)$, where $\delta : (Q_1 \times Q_2) \times (\Sigma_1 \times \Sigma_2) \rightarrow (Q_1 \times Q_2)$ is given by $((q_1, q_2), (\sigma_1, \sigma_2)) \mapsto (\delta_1(q_1, \sigma_1), \delta_2(q_2, \sigma_2))$.
 - It's as if the DFAs are running simultaneously. This can be extended to arbitrary direct products (number of them).
 - Note that the sets don't matter. The "topological" structure of the DFA should be invariant.
- **Definition** (Semigroup of a DFA). Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Then, $S(\mathcal{A})$, the semigroup of \mathcal{A} , is the set $\{\delta_w : Q \rightarrow Q \mid w \in \Sigma^+\}$ with the binary operation as (inverse) function composition $*$.
 - Note that this forms a semigroup because $\delta_{w_1} * \delta_{w_2} = \delta_{w_1 w_2}$. Since concatenation of words is associative, this binary operation is associative.
- **Note.** What's happening here. Can we convert a semigroup into an automata. If we have the multiplication words. But the thing is that this semigroup is a free semigroup over some alphabet. Given the transitions for the individual letters, the semigroup is **freely generated** over these elements. So, if we can find the generators, we can find the DFA.
- **Lemma.** Let $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1)$ be a DFA. Let $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2)$ be another DFA. Then $S(\mathcal{A}_1 \times \mathcal{A}_2) \cong S(\mathcal{A}_1) \times S(\mathcal{A}_2)$
 - *Proof.* The isomorphism takes δ_w , where $w \in (\Sigma_1 \times \Sigma_2)^+$. Assume the words in Σ_1 and Σ_2 are w_1 and w_2 . Then, map δ_w to $(\delta_{w_1}, \delta_{w_2})$. It is easy to see that this is homomorphism. This is injective because if

$(\delta_{w_1}, \delta_{w_2}) = (\delta_{w_1'}, \delta_{w_2'})$, then it's easy to see. Yeah. It is surjective as well. Given $(\delta_{w_1}, \delta_{w_2})$, if w_2 is longer than w_1 , we can consider the word w where we add the extra epsilons. Yeah, then we will get the same thing. So yeah!

```
In[*]:= SetPower[A_List, B_List] := Function[b, #[[Position[B, b][[1, 1]]]] & /@
  MyCartesianProduct[Table[A, {i, 1, Length[B]}]]

In[*]:= DFAWreathProduct[DFA1Rule_List, DFA2Rule_List] :=
  Module[{DFA1, DFA2},
    DFA1 = RuleToDFA[DFA1Rule];
    DFA2 = RuleToDFA[DFA2Rule];
    {MyCartesianProduct[DFA1[[1]], DFA2[[1]]], MyCartesianProduct[
      SetPower[DFA1[[2]], DFA2[[1]]], DFA2[[2]], Function[{state, letter},
        {DFA1[[3]][state[[1]], letter[[1]][state[[2]]], DFA2[[3]][state[[2]], letter[[2]]]}]}
  ]

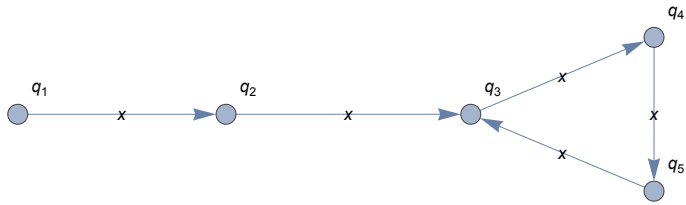
In[*]:= DFAWreathProduct[
  {{0, 1} -> {1}, {1, 1} -> {2}, {2, 1} -> {0}, {1, 0} -> {0}, {2, 0} -> {1}, {0, 0} -> {2}},
  {{0, 1} -> {1}, {1, 1} -> {2}, {2, 1} -> {0}, {1, 0} -> {0}, {2, 0} -> {1}, {0, 0} -> {2}}]

Out[*]= {{{0, 0}, {0, 1}, {0, 2}, {1, 0}, {1, 1}, {1, 2}, {2, 0}, {2, 1}, {2, 2}},
  {{Function[b$, {1, 1, 1}[[Position[{0, 1, 2}, b$][[1, 1]]]], 1},
   {Function[b$, {1, 1, 1}[[Position[{0, 1, 2}, b$][[1, 1]]]], 0},
   {Function[b$, {1, 1, 0}[[Position[{0, 1, 2}, b$][[1, 1]]]], 1},
   {Function[b$, {1, 1, 0}[[Position[{0, 1, 2}, b$][[1, 1]]]], 0},
   {Function[b$, {1, 0, 1}[[Position[{0, 1, 2}, b$][[1, 1]]]], 1},
   {Function[b$, {1, 0, 1}[[Position[{0, 1, 2}, b$][[1, 1]]]], 0},
   {Function[b$, {1, 0, 0}[[Position[{0, 1, 2}, b$][[1, 1]]]], 1},
   {Function[b$, {1, 0, 0}[[Position[{0, 1, 2}, b$][[1, 1]]]], 0},
   {Function[b$, {0, 1, 1}[[Position[{0, 1, 2}, b$][[1, 1]]]], 1},
   {Function[b$, {0, 1, 1}[[Position[{0, 1, 2}, b$][[1, 1]]]], 0},
   {Function[b$, {0, 1, 0}[[Position[{0, 1, 2}, b$][[1, 1]]]], 1},
   {Function[b$, {0, 1, 0}[[Position[{0, 1, 2}, b$][[1, 1]]]], 0},
   {Function[b$, {0, 0, 1}[[Position[{0, 1, 2}, b$][[1, 1]]]], 1},
   {Function[b$, {0, 0, 1}[[Position[{0, 1, 2}, b$][[1, 1]]]], 0},
   {Function[b$, {0, 0, 0}[[Position[{0, 1, 2}, b$][[1, 1]]]], 1},
   {Function[b$, {0, 0, 0}[[Position[{0, 1, 2}, b$][[1, 1]]]], 0}},
  Function[{state$, letter$}, {DFA1$44167[[3]][state$[[1]], letter$[[1]][state$[[2]]]],
    DFA2$44167[[3]][state$[[2]], letter$[[2]]]}]}
```

```
In[*]:= DFAPlot[
```

```
  RuleToDFA[{{1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3}}]]
```

```
Out[*]:=
```



```
In[*]:= DFAWreathProduct[
```

```
  {{1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3}},
```

```
  {{1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3}}]
```

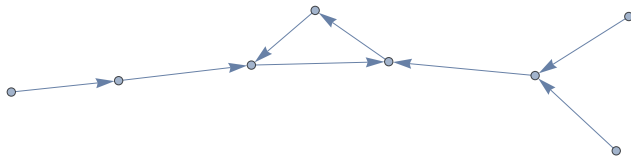
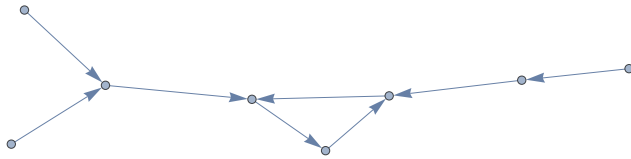
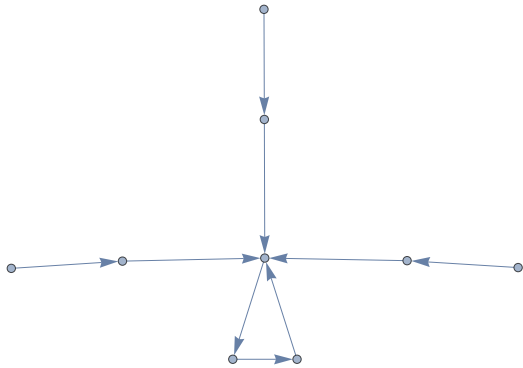
```
Out[*]:=
```

```

{{{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 1}, {2, 2},
 {2, 3}, {2, 4}, {2, 5}, {3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {4, 1},
 {4, 2}, {4, 3}, {4, 4}, {4, 5}, {5, 1}, {5, 2}, {5, 3}, {5, 4}, {5, 5}},
 {{Function[b$, {x, x, x, x, x}][Position[{1, 2, 3, 4, 5}, b$][[1, 1]]], x}},
 Function[{state$, letter$}, {DFA1$62616[[3]][state$[[1]], letter$[[1]][state$[[2]]],
   DFA2$62616[[3]][state$[[2]], letter$[[2]]]}]}]
```

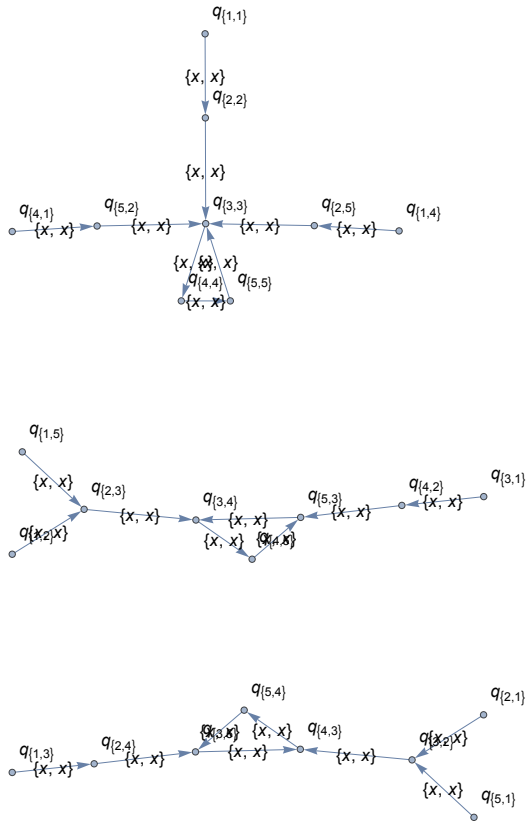
```
In[•]:= DFAPlot[%85, False]
```

Out[•]=



```
In[8] := DFAPlot[%88]
```

```
Out[8] :=
```

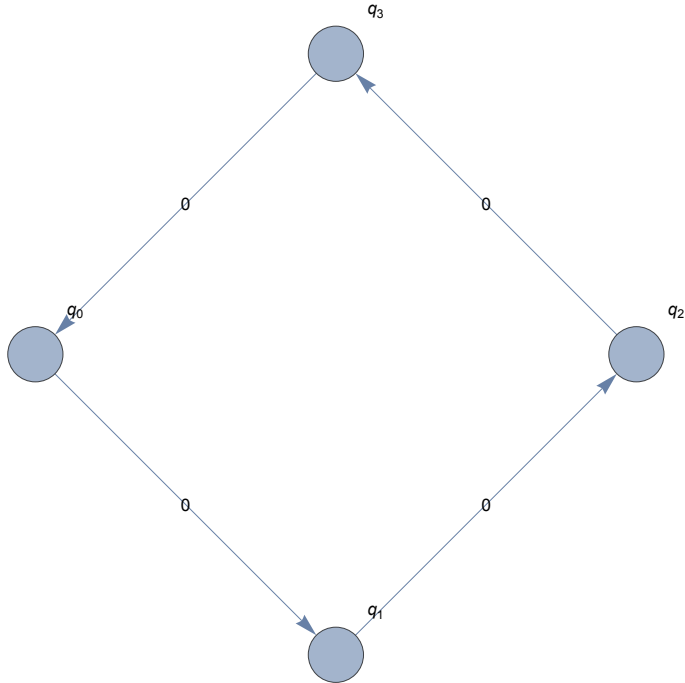


```
In[13] := CKcounter[k_] :=
```

```
{Table[i, {i, 0, k - 1}], {0}, Function[{s, l}, If[l == 0, Mod[s + 1, k] ]]}
```

```
In[ ]:= DFAPlot[CKcounter[4]]
```

```
Out[ ]:=
```



```
In[ ]:= Map[DFAtRule, {CKcounter[10], CKcounter[4]}]
```

```
Out[ ]:=
```

```

{{{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {3}, {3, 0} -> {4}, {4, 0} -> {5},
 {5, 0} -> {6}, {6, 0} -> {7}, {7, 0} -> {8}, {8, 0} -> {9}, {9, 0} -> {0}},
 {{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {3}, {3, 0} -> {0}}}

```

```

In[ ]:= DFADirectProduct[{{{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {3}, {3, 0} -> {4}, {4, 0} -> {5},
 {5, 0} -> {6}, {6, 0} -> {7}, {7, 0} -> {8}, {8, 0} -> {9}, {9, 0} -> {0}},
 {{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {3}, {3, 0} -> {0}}}]

```

```
Out[ ]:=
```

```

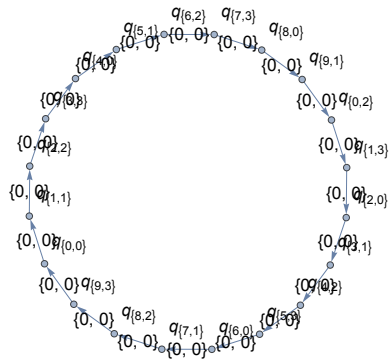
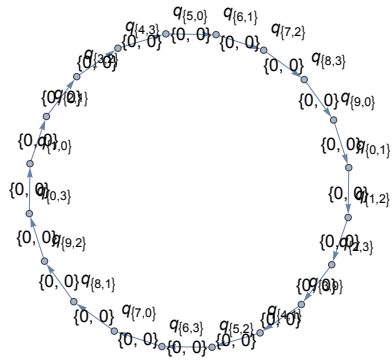
{{{0, 0}, {0, 1}, {0, 2}, {0, 3}, {1, 0}, {1, 1}, {1, 2}, {1, 3}, {2, 0}, {2, 1},
 {2, 2}, {2, 3}, {3, 0}, {3, 1}, {3, 2}, {3, 3}, {4, 0}, {4, 1}, {4, 2}, {4, 3},
 {5, 0}, {5, 1}, {5, 2}, {5, 3}, {6, 0}, {6, 1}, {6, 2}, {6, 3}, {7, 0}, {7, 1},
 {7, 2}, {7, 3}, {8, 0}, {8, 1}, {8, 2}, {8, 3}, {9, 0}, {9, 1}, {9, 2}, {9, 3}},
 {{0, 0}}, Function[{s$, l$}, Table[dfalist$45069[[i]][3][s$[[i]], l$[[i]],
 {i, Length[{{{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {3}, {3, 0} -> {4}, {4, 0} -> {5},
 {5, 0} -> {6}, {6, 0} -> {7}, {7, 0} -> {8}, {8, 0} -> {9}, {9, 0} -> {0}},
 {{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {3}, {3, 0} -> {0}}}}]]]]]

```



```
In[ ]:= DFAPlot[%60]
```

```
Out[ ]=
```



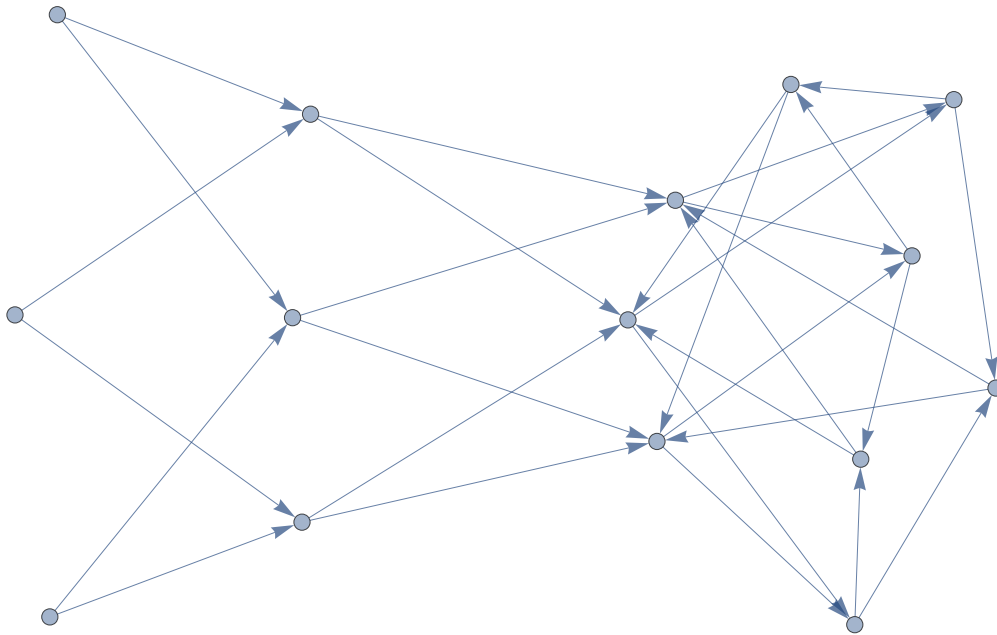
```
In[ ]:= DFAWreathProduct[
```

```
{ {1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3} },
{ {0, 1} → {1}, {1, 1} → {2}, {2, 1} → {0}, {1, 0} → {0}, {2, 0} → {1}, {0, 0} → {2} }]
```

```
Out[ ]=
```

```
{{ {1, 0}, {1, 1}, {1, 2}, {2, 0}, {2, 1}, {2, 2}, {3, 0},
{3, 1}, {3, 2}, {4, 0}, {4, 1}, {4, 2}, {5, 0}, {5, 1}, {5, 2} },
{ {Function[b$, {x, x, x}][Position[{0, 1, 2}, b$][[1, 1]]], 1},
{Function[b$, {x, x, x}][Position[{0, 1, 2}, b$][[1, 1]]], 0} },
Function[{state$, letter$}, {DFA1$50897[[3]][state$[[1]], letter$[[1]][state$[[2]]],
DFA2$50897[[3]][state$[[2]], letter$[[2]]]} ] }
```

```
In[*]:= DFAPlot[%69, False]
Out[*]=
```



So they run in “parallel”.

```
In[*]:= Map[DFAtoRule, {CKcounter[10], CKcounter[4], CKcounter[3]}]
Out[*]=
```

$$\begin{aligned} & \{ \{ \{0, 0\} \rightarrow \{1\}, \{1, 0\} \rightarrow \{2\}, \{2, 0\} \rightarrow \{3\}, \{3, 0\} \rightarrow \{4\}, \{4, 0\} \rightarrow \{5\}, \\ & \quad \{5, 0\} \rightarrow \{6\}, \{6, 0\} \rightarrow \{7\}, \{7, 0\} \rightarrow \{8\}, \{8, 0\} \rightarrow \{9\}, \{9, 0\} \rightarrow \{0\} \}, \\ & \{ \{0, 0\} \rightarrow \{1\}, \{1, 0\} \rightarrow \{2\}, \{2, 0\} \rightarrow \{3\}, \{3, 0\} \rightarrow \{0\} \}, \\ & \{ \{0, 0\} \rightarrow \{1\}, \{1, 0\} \rightarrow \{2\}, \{2, 0\} \rightarrow \{0\} \} \} \end{aligned}$$

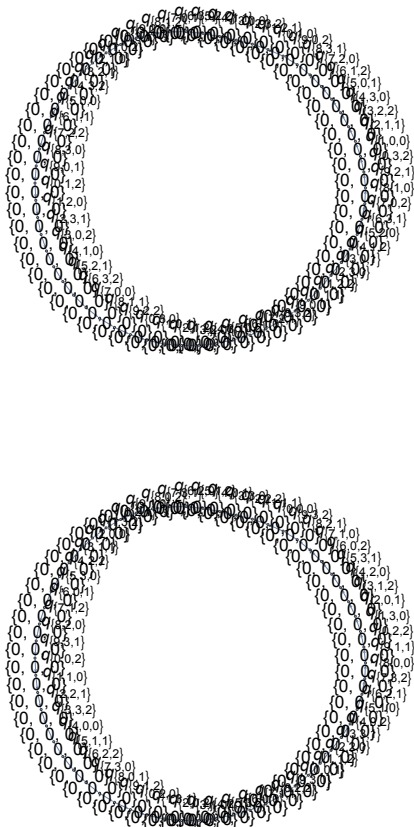
```
In[*]:= DFADirectProduct[Map[DFAtaRule, {CKcounter[10], CKcounter[4], CKcounter[3]}]]
```

```
Out[*]=
```

```
{ {{0, 0, 0}, {0, 0, 1}, {0, 0, 2}, {0, 1, 0}, {0, 1, 1}, {0, 1, 2}, {0, 2, 0}, {0, 2, 1},
  {0, 2, 2}, {0, 3, 0}, {0, 3, 1}, {0, 3, 2}, {1, 0, 0}, {1, 0, 1}, {1, 0, 2}, {1, 1, 0},
  {1, 1, 1}, {1, 1, 2}, {1, 2, 0}, {1, 2, 1}, {1, 2, 2}, {1, 3, 0}, {1, 3, 1}, {1, 3, 2},
  {2, 0, 0}, {2, 0, 1}, {2, 0, 2}, {2, 1, 0}, {2, 1, 1}, {2, 1, 2}, {2, 2, 0}, {2, 2, 1},
  {2, 2, 2}, {2, 3, 0}, {2, 3, 1}, {2, 3, 2}, {3, 0, 0}, {3, 0, 1}, {3, 0, 2}, {3, 1, 0},
  {3, 1, 1}, {3, 1, 2}, {3, 2, 0}, {3, 2, 1}, {3, 2, 2}, {3, 3, 0}, {3, 3, 1}, {3, 3, 2},
  {4, 0, 0}, {4, 0, 1}, {4, 0, 2}, {4, 1, 0}, {4, 1, 1}, {4, 1, 2}, {4, 2, 0}, {4, 2, 1},
  {4, 2, 2}, {4, 3, 0}, {4, 3, 1}, {4, 3, 2}, {5, 0, 0}, {5, 0, 1}, {5, 0, 2}, {5, 1, 0},
  {5, 1, 1}, {5, 1, 2}, {5, 2, 0}, {5, 2, 1}, {5, 2, 2}, {5, 3, 0}, {5, 3, 1},
  {5, 3, 2}, {6, 0, 0}, {6, 0, 1}, {6, 0, 2}, {6, 1, 0}, {6, 1, 1}, {6, 1, 2},
  {6, 2, 0}, {6, 2, 1}, {6, 2, 2}, {6, 3, 0}, {6, 3, 1}, {6, 3, 2}, {7, 0, 0},
  {7, 0, 1}, {7, 0, 2}, {7, 1, 0}, {7, 1, 1}, {7, 1, 2}, {7, 2, 0}, {7, 2, 1},
  {7, 2, 2}, {7, 3, 0}, {7, 3, 1}, {7, 3, 2}, {8, 0, 0}, {8, 0, 1}, {8, 0, 2},
  {8, 1, 0}, {8, 1, 1}, {8, 1, 2}, {8, 2, 0}, {8, 2, 1}, {8, 2, 2}, {8, 3, 0},
  {8, 3, 1}, {8, 3, 2}, {9, 0, 0}, {9, 0, 1}, {9, 0, 2}, {9, 1, 0}, {9, 1, 1},
  {9, 1, 2}, {9, 2, 0}, {9, 2, 1}, {9, 2, 2}, {9, 3, 0}, {9, 3, 1}, {9, 3, 2}},
  {{0, 0, 0}}, Function[{s$, l$}, Table[dfalist$46286[[i]][3][s$[[i]], l$[[i]],
    {i, Length[{{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {3}, {3, 0} -> {4}, {4, 0} -> {5},
      {5, 0} -> {6}, {6, 0} -> {7}, {7, 0} -> {8}, {8, 0} -> {9}, {9, 0} -> {0}},
    {{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {3}, {3, 0} -> {0}},
    {{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {0}}]}]]]
```

```
In[ ]:= DFAPlot[%63]
```

```
Out[ ]:=
```



we get something weird. but the thing is they run in parallel.

CloudObject : Unable to authenticate with Wolfram Cloud server. Please try authenticating again.

Let's define the wreath product. That will be nice. Some category theory will be nice.

- **Definition** (Wreath product). The *wreath product* of transformation semigroups (X, S) and (Y, T) , denoted $(X, S) \wr (Y, T)$ is the transformation semigroup $(X \times Y, W)$, where

$W = \{(f, t) \mid f : Y \rightarrow S, t \in T\} = S^Y \times T$, and the binary operation on the semigroup is defined by $(f, t) \cdot (g, t') = (h_{f,g}, t t')$, where $h_{f,g} : Y \rightarrow S$ takes $y \mapsto f(y) g(y \cdot t)$, and the action is defined as $\alpha((x, y), (f, t)) = (x \cdot f(y), y \cdot t)$.

- This is different from the direct product above! In the direct product, a state (x, y) is transformed to $(x \cdot s, y \cdot t)$ when we specify that the event (s, t) is acting on the state (x, y) . s and t act independently of each other in the sense that they act on their individual elements. However, in the wreath product, to specify an action on a state (x, y) , we need to specify an action on y first (which is t) and then y determines the action on x . This is because there is a map $f : Y \rightarrow S$ which specifies how to act on $x \in X$ given an element of y . In this way, the wreath product represents a **feed-forward** system. There is a *flow of information from (Y, T) to (X, S)* .
- **Definition** (S^λ). Let S be a semigroup. Then S^λ be the least monoid containing S .
- **Definition** (Semigroup automaton). Given a semigroup S , then the semigroup automaton of S is given by $\mathcal{A}_S = (S^\lambda, S, \delta_S)$ be the automaton with $\delta_S(s_1, s_2) = s_1 s_2$ where $s_1 \in S$.

This is an injective transformation from semigroup to automaton. However, the 2 different automata can correspond to the same semigroup under the canonical transformation from automata to semigroups. This means that the canonical transformation is not injective. However, this does not span all automata. So this means that the semigroup has less information than the automata. What about the generating sets. Well, the semigroup of an automata has a generating set. Given a semigroup, if we know the generators, we would only know how the generators **interact**. There is no canonical way of choosing the state set.

- **Lemma 2.39.** Let (X, S) and (X', S') be transformation semigroups. Let $\chi = (X, S, \delta_\chi)$ and $\chi' = (X', S', \delta')$ be the corresponding automata. Then (X, S) divides (X', S') according to the definition of transformation semigroups iff χ divides χ' according to the definition of division for automata.
- Ok. Now, let's try to define wreath products of DFAs. We can convert them to semigroups....but we want to know what the **actual** DFA represents. Ok. So let's do this
- **Definition** (Wreath Product of DFAs). $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1)$ be a DFA. Let $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2)$ be another DFA. Then we can define their wreath product in the following way. $\mathcal{A}_1 \wr \mathcal{A}_2 = (Q_1 \times Q_2, \Sigma_1^{Q_2} \times \Sigma_2, \delta)$ where $\delta((q_1, q_2), (f, \sigma)) = (\delta_1(q_1, f(q_2)), \delta_2(q_2, \sigma))$.
- **Theorem.** Let $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1)$ be a DFA. Let $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2)$. Then let $(Q_1, S(\mathcal{A}_1))$ be the transformation semigroup of \mathcal{A}_1 and $(Q_2, S(\mathcal{A}_2))$ be the transformation semigroup of \mathcal{A}_2 . Then $(Q_1 \times Q_2, S(\mathcal{A}_1 \wr \mathcal{A}_2)) \cong (Q_1, S(\mathcal{A}_1)) \wr (Q_2, S(\mathcal{A}_2))$.
 - *Proof.* Note that $(Q_1, S(\mathcal{A}_1)) \wr (Q_2, S(\mathcal{A}_2))$ is the transformation semigroup $(Q_1 \times Q_2, S(\mathcal{A}_1)^{Q_2} \times S(\mathcal{A}_2))$. This is isomorphism of semigroups. An element of $S(\mathcal{A}_1 \wr \mathcal{A}_2)$ is described by $\delta_w, w \in (\Sigma_1^{Q_2} \times \Sigma_2)^+$ i.e. $w = (f_1, \sigma_1) \dots (f_n, \sigma_n)$. This will act on $Q_1 \times Q_2$. By definition, $\delta((q_1, q_2), (f, \sigma)) = (\delta_1(q_1, f(q_2)), \delta_2(q_2, \sigma))$. Then, we will get $(\delta_1(q_1, f(q_2)), \delta_2(\delta_2(q_2, \sigma_1), \sigma_2))$. So we map this to $\delta_{\sigma_1 \dots \sigma_n}$ in $S(\mathcal{A}_2)$. Well, we can easily create the thing. Let's create $f : Q_2 \rightarrow S(\mathcal{A}_1)$ as $f_1(q_2) f_2(\delta_2(q_2, \sigma_1)), \dots$ yeah...it probably is an isomorphism. Ok. iDC.

Here, we are assuming the the DFAs are finite.

Position

```
In[9]:= SetPower[A_List, B_List] := Function[b, #[[Position[B, b][[1, 1]]]] & /@
      MyCartesianProduct[Table[A, {i, 1, Length[B]}]]

In[10]:= DFAWreathProduct[DFA1Rule_List, DFA2Rule_List] :=
  Module[{DFA1, DFA2},
    DFA1 = RuleToDFA[DFA1Rule];
    DFA2 = RuleToDFA[DFA2Rule];
    {MyCartesianProduct[DFA1[[1]], DFA2[[1]]], MyCartesianProduct[
      SetPower[DFA1[[2]], DFA2[[1]], DFA2[[2]], Function[{state, letter},
        {DFA1[[3]][state[[1]], letter[[1]][state[[2]]]}, DFA2[[3]][state[[2]], letter[[2]]]}]]
  ]
```

■ **Definition** (Chain). Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a DFA. Then, the transformation semigroup of the DFA is $(Q, S(\mathcal{A}))$ where $S(\mathcal{A})$ is the set of functions from Q to Q . So it's already a map. We can consider a chain in $\mathcal{P}(Q)$ as a chain of subsets of Q . Note. An element of $S(\mathcal{A})$ is a poset preserving map...so it maps chains to chains...it's obvious cuz it's a function.

■ **Definition** (C). Let $C = C(X, S)$ denote the set of all maximal chains.

It's kind of impractical.

ImageSet[{}]

-
-
- s a

I can try to write the algorithm. But is it really worth it? Nope. note at all. Can we use some rewriting systems or something to analyse something.

Some examples

Check implemetnation.

```
In[*]:= CKcounter[3]
Out[*]=
{{0, 1, 2}, {0}, Function[{s$, l$}, If[l$ == 0, Mod[s$ + 1, 3]]]}

CkCon

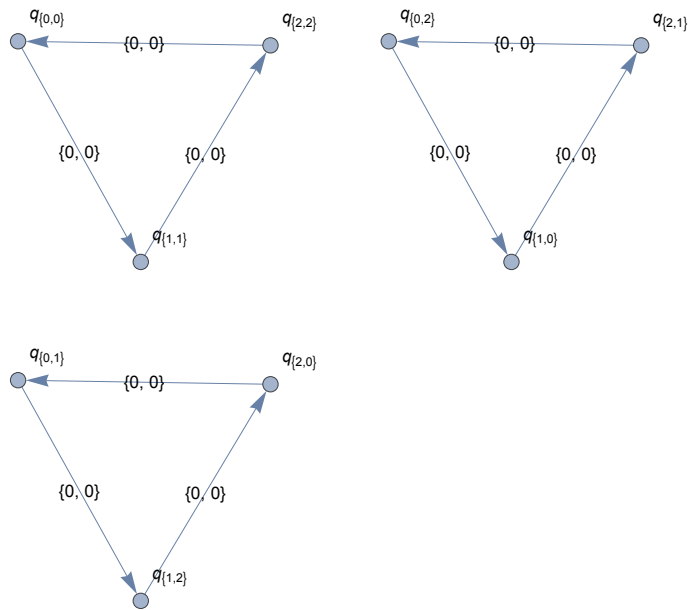
In[*]:= DFAtoRule[CKcounter[3]]
Out[*]=
{{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {0}}

In[*]:= DFAWreathProduct[{{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {0}},
{{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {0}}]
Out[*]=
{{{0, 0}, {0, 1}, {0, 2}, {1, 0}, {1, 1}, {1, 2}, {2, 0}, {2, 1}, {2, 2}},
{{Function[b$, {0, 0, 0}][Position[{0, 1, 2}, b$][1, 1]], 0}},
Function[{state$, letter$}, {DFA1$41812[3][state$[1], letter$[1][state$[2]]],
DFA2$41812[3][state$[2], letter$[2]]}]}}

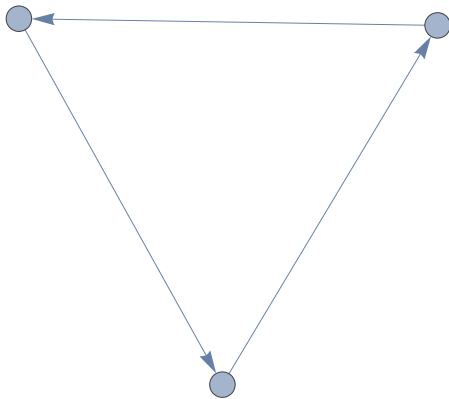
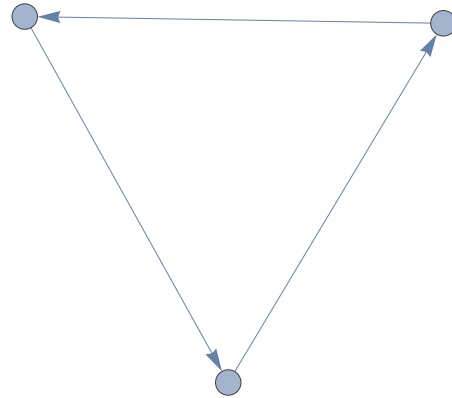
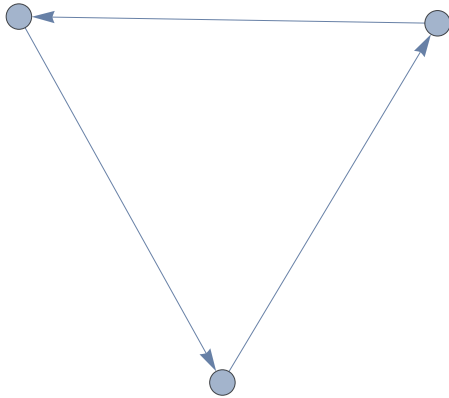
In[*]:= DFADirectProduct[{{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {0}},
{{0, 0} -> {1}, {1, 0} -> {2}, {2, 0} -> {0}}]
Out[*]=
{{{0, 0}, {0, 1}, {0, 2}, {1, 0}, {1, 1}, {1, 2}, {2, 0}, {2, 1}, {2, 2}},
{{0, 0}}, Function[{state$, letter$},
{DFA1$59261[3][state$[1], letter$[1]], DFA2$59261[3][state$[2], letter$[2]]}]}}
```

In[]:= DFAPlot[%83]

Out[]:=



```
In[ ]:= DFAPlot[%51, False]  
Out[ ]=
```



As we can see, the direct product embeds in here.

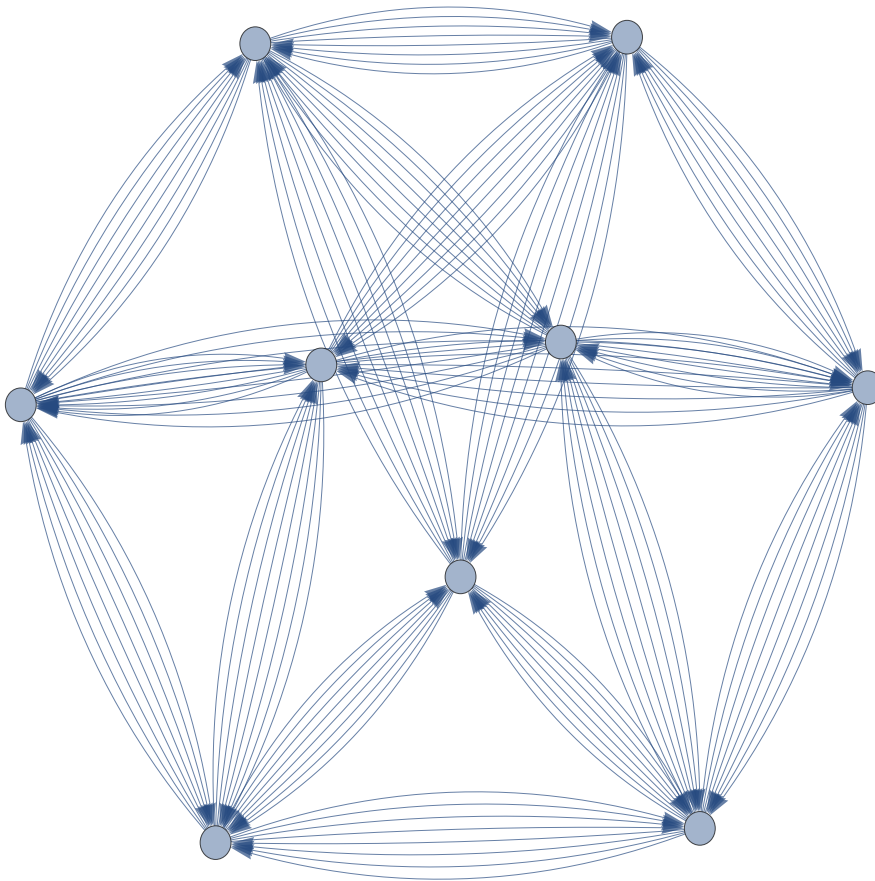

```

In[ ]:= DFAWreathProduct[
  {{0, 1} → {1}, {1, 1} → {2}, {2, 1} → {0}, {1, 0} → {0}, {2, 0} → {1}, {0, 0} → {2}},
  {{0, 1} → {1}, {1, 1} → {2}, {2, 1} → {0}, {1, 0} → {0}, {2, 0} → {1}, {0, 0} → {2}}]

Out[ ]:=
{{{0, 0}, {0, 1}, {0, 2}, {1, 0}, {1, 1}, {1, 2}, {2, 0}, {2, 1}, {2, 2}},
 {{Function[b$, {1, 1, 1}[[Position[{0, 1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {1, 1, 1}[[Position[{0, 1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {1, 1, 0}[[Position[{0, 1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {1, 1, 0}[[Position[{0, 1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {1, 0, 1}[[Position[{0, 1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {1, 0, 1}[[Position[{0, 1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {1, 0, 0}[[Position[{0, 1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {1, 0, 0}[[Position[{0, 1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {0, 1, 1}[[Position[{0, 1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {0, 1, 1}[[Position[{0, 1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {0, 1, 0}[[Position[{0, 1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {0, 1, 0}[[Position[{0, 1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {0, 0, 1}[[Position[{0, 1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {0, 0, 1}[[Position[{0, 1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {0, 0, 0}[[Position[{0, 1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {0, 0, 0}[[Position[{0, 1, 2}, b$][[1, 1]]], 0}},
 Function[{state$, letter$}, {DFA1$44167[[3][state$[[1], letter$[[1][state$[[2]]]],
 DFA2$44167[[3][state$[[2], letter$[[2]]]]}]]}

```

```
In[ ]:= DFAPlot[%55, False]
Out[ ]=
```



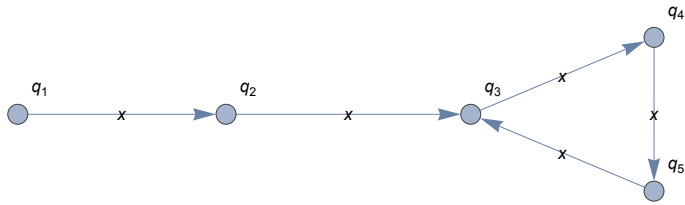
Ok. The direct product embeds in this wreath product. Ok. Cool. Let's make some more interesting pictures.

```
In[ ]:= RuleToDFA[{{1, x} -> {2}, {2, x} -> {3}, {3, x} -> {4}, {4, x} -> {5}, {5, x} -> {3}}]
Out[ ]=
{{1, 2, 3, 4, 5}, {x},
Function[{state$, letter$}, findstate[{{1, x} -> {2}, {2, x} -> {3},
{3, x} -> {4}, {4, x} -> {5}, {5, x} -> {3}}, state$, letter$]]]
```

In[*]:= DFAPlot[

RuleToDFA[{{1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3}}]]

Out[*]=



In[*]:= DFAWreathProduct[

{{1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3}},

{{1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3}}]

Out[*]=

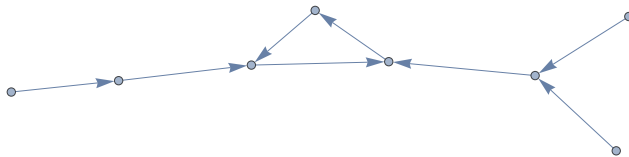
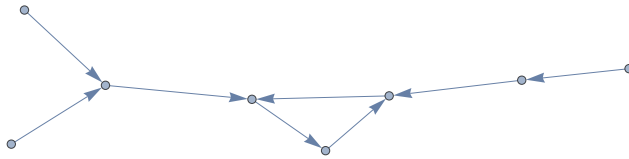
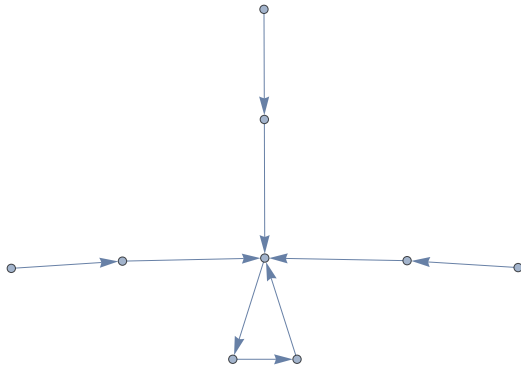
```

{{{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 1}, {2, 2},
 {2, 3}, {2, 4}, {2, 5}, {3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {4, 1},
 {4, 2}, {4, 3}, {4, 4}, {4, 5}, {5, 1}, {5, 2}, {5, 3}, {5, 4}, {5, 5}},
 {{Function[b$, {x, x, x, x, x}][Position[{1, 2, 3, 4, 5}, b$][[1, 1]]], x}},
 Function[{state$, letter$}, {DFA1$62616[[3]][state$[[1]], letter$[[1]][state$[[2]]],
 DFA2$62616[[3]][state$[[2]], letter$[[2]]]}]}

```

```
In[ ]:= DFAPlot[%85, False]
```

```
Out[ ]:=
```



```
In[ ]:= DFADirectProduct[
```

```
{ {1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3} },
```

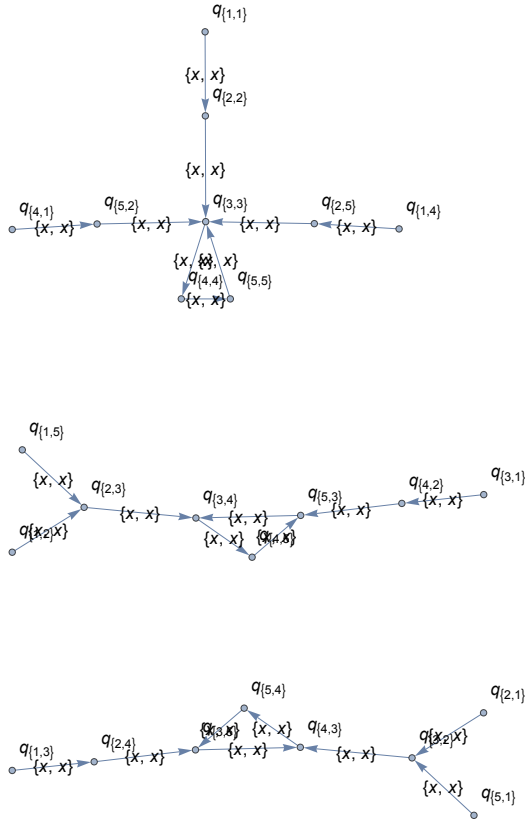
```
{ {1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3} }]
```

```
Out[ ]:=
```

```
{{ {1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 1}, {2, 2}, {2, 3}, {2, 4}, {2, 5},
  {3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {4, 1}, {4, 2}, {4, 3}, {4, 4}, {4, 5},
  {5, 1}, {5, 2}, {5, 3}, {5, 4}, {5, 5} }, { {x, x} }, Function[{state$, letter$},
  {DFA1$61147[[3]][state$[[1]], letter$[[1]], DFA2$61147[[3]][state$[[2]], letter$[[2]]}]] }
```

```
In[ ]:= DFAPlot[%88]
```

```
Out[ ]:=
```



```
In[ ]:= DFAWreathProduct[
```

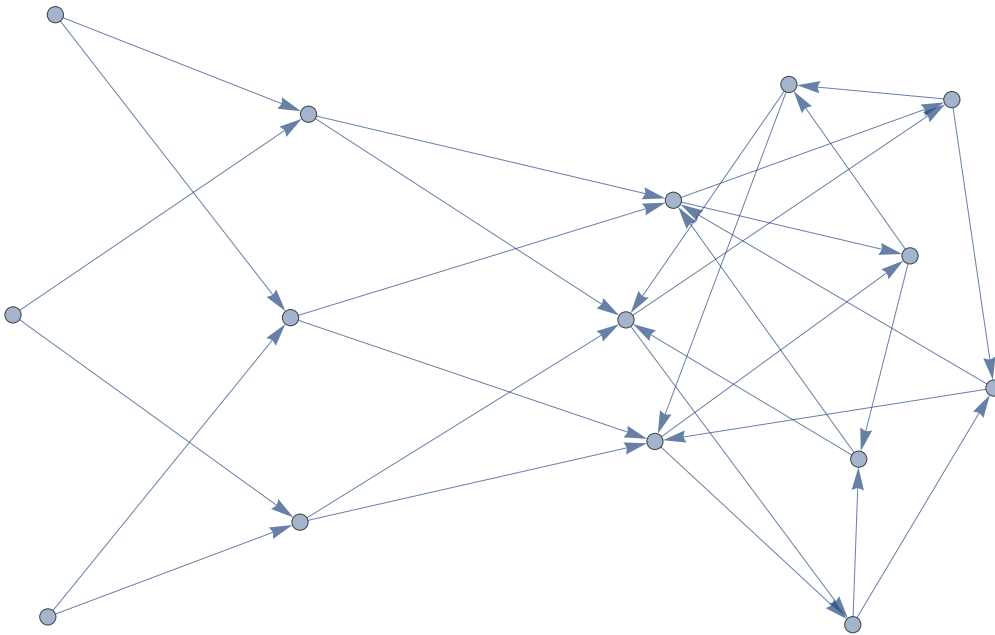
```
{ {1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3} },
{ {0, 1} → {1}, {1, 1} → {2}, {2, 1} → {0}, {1, 0} → {0}, {2, 0} → {1}, {0, 0} → {2} }]
```

```
Out[ ]:=
```

```
{{ {1, 0}, {1, 1}, {1, 2}, {2, 0}, {2, 1}, {2, 2}, {3, 0},
  {3, 1}, {3, 2}, {4, 0}, {4, 1}, {4, 2}, {5, 0}, {5, 1}, {5, 2} },
{ {Function[b$, {x, x, x}][Position[{0, 1, 2}, b$][1, 1]], 1},
  {Function[b$, {x, x, x}][Position[{0, 1, 2}, b$][1, 1]], 0} },
Function[{state$, letter$}, {DFA1$50897[[3]][state$[[1]], letter$[[1]][state$[[2]]],
  DFA2$50897[[3]][state$[[2]], letter$[[2]]]}]
```

```
In[ ]:= DFAPlot[%69, False]
```

```
Out[ ]:=
```



Topological structure invariant! Cool. ZX-diagrams. Looks pretty cool! Why am I getting the same fucking thing? Let's try to find non-isomorphic ones.

Does the direct product embed?

```
In[ ]:= DFADirectProduct[
```

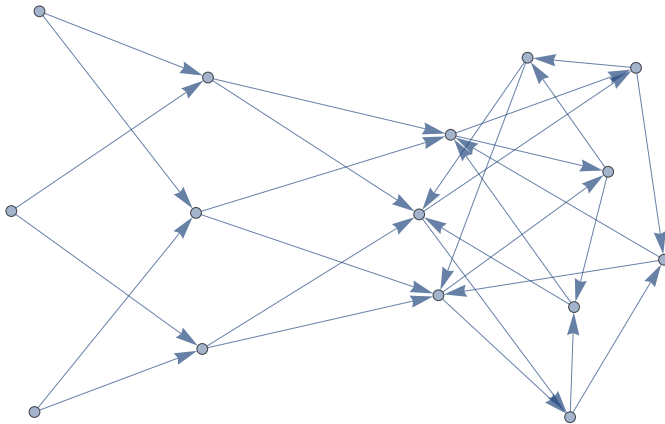
```
{ {1, x} → {2}, {2, x} → {3}, {3, x} → {4}, {4, x} → {5}, {5, x} → {3} },
{ {0, 1} → {1}, {1, 1} → {2}, {2, 1} → {0}, {1, 0} → {0}, {2, 0} → {1}, {0, 0} → {2} }]
```

```
Out[ ]:=
```

```
{{ {1, 0}, {1, 1}, {1, 2}, {2, 0}, {2, 1}, {2, 2}, {3, 0}, {3, 1}, {3, 2}, {4, 0}, {4, 1},
{4, 2}, {5, 0}, {5, 1}, {5, 2} }, {{x, 1}, {x, 0}}, Function[{state$, letter$},
{DFA1$58159[3][state$[1], letter$[1]], DFA2$58159[3][state$[2], letter$[2]]}]]}
```

```
In[ ]:= DFAPlot[%80, False]
```

```
Out[ ]:=
```



```
In[ ]:= IsomorphicGraphQ[
```

```
Out[ ]:=
```

```
True
```

Why is the wreath product here the same as the direct product? That doesn't make sense at all

```
In[ ]:= DFADirectProduct[DFA1Rule_List, DFA2Rule_List] :=
Module[{DFA1, DFA2},
  DFA1 = RuleToDFA[DFA1Rule];
  DFA2 = RuleToDFA[DFA2Rule];
  {MyCartesianProduct[DFA1[[1]], DFA2[[1]]},
  MyCartesianProduct[DFA1[[2]], DFA2[[2]], Function[{state, letter},
    {DFA1[[3]][state[[1]], letter[[1]], DFA2[[3]][state[[2]], letter[[2]]}]]}
]
```

Ok. Taking time off. I get the general idea. Now what. we should talk about some examples or something. how is useful in synbio? We can engineer organisms according to what we **want** them to do. yeah. Okay. there's the wreath product. what about the category theory. What model based stuff...we study the behaviour.

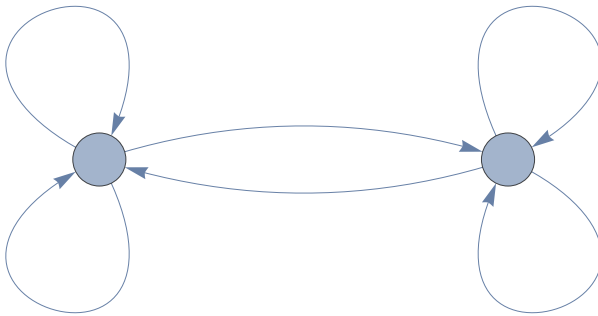
Show example. Analyse using algebra and see the biology. and what's the goal. making quantum circuits---analogous to that. How to make these biocomputer or something. So yeah. Inspiration from ZX-calculus.

```
In[*]:= FlipFlopDFA = RuleToDFA[
  {{1, 0} → {1}, {1, 1} → {1}, {1, 2} → {2}, {2, 0} → {2}, {2, 1} → {1}, {2, 2} → {2}}]
```

```
Out[*]:=
{{1, 2}, {0, 1, 2},
  Function[{state$, letter$}, findstate[{{1, 0} → {1}, {1, 1} → {1}, {1, 2} → {2},
    {2, 0} → {2}, {2, 1} → {1}, {2, 2} → {2}}, state$, letter$]]}
```

```
In[*]:= DFAPlot[%105, False]
```

```
Out[*]:=
```

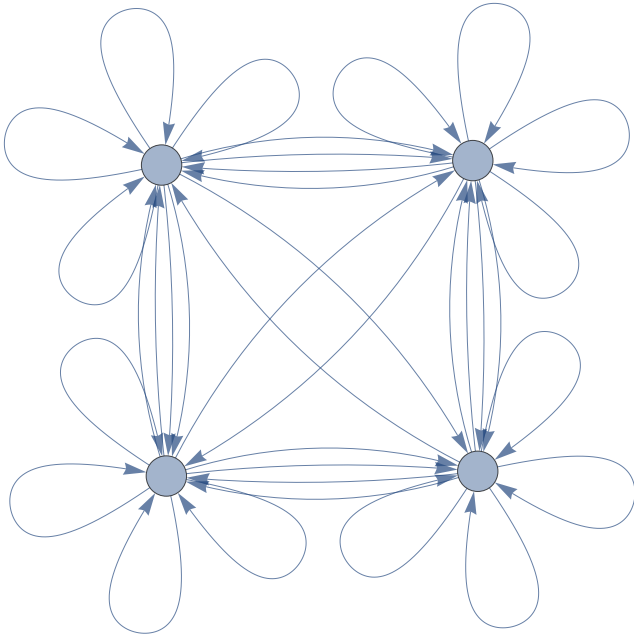


```
In[*]:= DFADirectProduct[
  {{1, 0} → {1}, {1, 1} → {1}, {1, 2} → {2}, {2, 0} → {2}, {2, 1} → {1}, {2, 2} → {2}},
  {{1, 0} → {1}, {1, 1} → {1}, {1, 2} → {2}, {2, 0} → {2}, {2, 1} → {1}, {2, 2} → {2}}]
```

```
Out[*]:=
{{{1, 1}, {1, 2}, {2, 1}, {2, 2}},
 {{0, 0}, {0, 1}, {0, 2}, {1, 0}, {1, 1}, {1, 2}, {2, 0}, {2, 1}, {2, 2}},
  Function[{state$, letter$},
    {DFA1$79781[[3]][state$[[1]], letter$[[1]], DFA2$79781[[3]][state$[[2]], letter$[[2]]]}]]}
```



```
In[ ]:= DFAPlot[%121, False]  
Out[ ]=
```



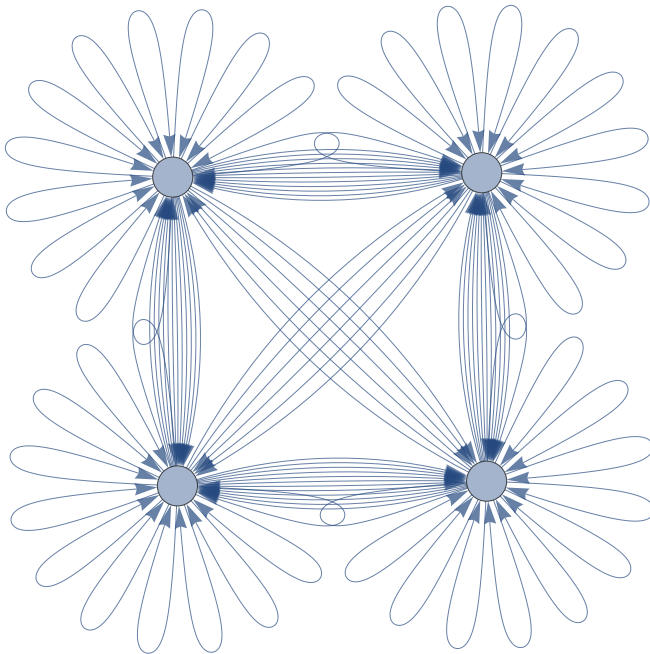
```

In[ ]:= DFAWreathProduct[
  {{1, 0} → {1}, {1, 1} → {1}, {1, 2} → {2}, {2, 0} → {2}, {2, 1} → {1}, {2, 2} → {2}},
  {{1, 0} → {1}, {1, 1} → {1}, {1, 2} → {2}, {2, 0} → {2}, {2, 1} → {1}, {2, 2} → {2}}]

Out[ ]:=
{{{1, 1}, {1, 2}, {2, 1}, {2, 2}},
 {Function[b$, {0, 0}][Position[{1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {0, 0}][Position[{1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {0, 0}][Position[{1, 2}, b$][[1, 1]]], 2},
 {Function[b$, {0, 1}][Position[{1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {0, 1}][Position[{1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {0, 1}][Position[{1, 2}, b$][[1, 1]]], 2},
 {Function[b$, {0, 2}][Position[{1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {0, 2}][Position[{1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {0, 2}][Position[{1, 2}, b$][[1, 1]]], 2},
 {Function[b$, {1, 0}][Position[{1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {1, 0}][Position[{1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {1, 0}][Position[{1, 2}, b$][[1, 1]]], 2},
 {Function[b$, {1, 1}][Position[{1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {1, 1}][Position[{1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {1, 1}][Position[{1, 2}, b$][[1, 1]]], 2},
 {Function[b$, {1, 2}][Position[{1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {1, 2}][Position[{1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {1, 2}][Position[{1, 2}, b$][[1, 1]]], 2},
 {Function[b$, {2, 0}][Position[{1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {2, 0}][Position[{1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {2, 0}][Position[{1, 2}, b$][[1, 1]]], 2},
 {Function[b$, {2, 1}][Position[{1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {2, 1}][Position[{1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {2, 1}][Position[{1, 2}, b$][[1, 1]]], 2},
 {Function[b$, {2, 2}][Position[{1, 2}, b$][[1, 1]]], 0},
 {Function[b$, {2, 2}][Position[{1, 2}, b$][[1, 1]]], 1},
 {Function[b$, {2, 2}][Position[{1, 2}, b$][[1, 1]]], 2}},
 Function[{state$, letter$}, {DFA1$80556[[3]][state$[[1]], letter$[[1]][state$[[2]]],
   DFA2$80556[[3]][state$[[2]], letter$[[2]]]}]}

```

```
In[ ]:= DFAPlot[%123, False]
Out[ ]:=
```



Atleast not isomorphic. Let's check the wreath product again. It's **so fucking weird that** i'm getting the same thing as that. Something that **doesn't embed in the direct product** but does embed in the direct product.

```
In[ ]:= DFAPlot[{states_, alphabet_, transition_}] :=
Module[{},
  Graph[Flatten@
    Table[Table[Labeled[s → transition[s, a], a], {a, alphabet}], {s, states}],
    VertexLabels → Table[s → qs, {s, states}], VertexSize → 0.13]
]

In[ ]:= DFAPlot[{states_, alphabet_, transition_}, bool_] :=
Module[{},
  If[bool == True,
    Graph[Flatten@
      Table[Table[Labeled[s → transition[s, a], a], {a, alphabet}], {s, states}],
      VertexLabels → Table[s → qs, {s, states}], VertexSize → 0.13],
    Graph[Flatten@Table[Table[s → transition[s, a], {a, alphabet}], {s, states}],
      VertexLabels → None, VertexSize → 0.13]
  ]
]
```

What now?

Ok. I get it. So we are clear with the general structure. We have categories and stuff that can help decompose a system into its components. Then, we can engineer somethings from their components

and use them. Essentially, that's what we want to do...focus on the **emergent geometry or something**. That makes sense. There are also some petri nets that we can think about. Yes. There are some things we can do.

Then, we can engineer these genetic networks that can help us. TheNow we want to come up some biological example that we can compare it to something else. Let's come up with a concrete example.

<https://pubmed.ncbi.nlm.nih.gov/21723368/>

<file:///Users/ubajaj/Desktop/p2.pdf>

just read this.

<file:///Users/ubajaj/Downloads/3663-Manuscript%20in%20PDF-1915-1-10-20180614.pdf>

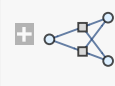
<https://arxiv.org/pdf/1508.06345.pdf>

try to implement holonomy decomposition

We can mention some things. We can analyse some biochemical reaction networks using these petri nets. And then we can analyse this. The point will be to enable the user to analyse these petri nets and carry out the decomposition for them. Yes...algebraic manipulation. We have other models as well...-causal graphs etc. Just finish them.

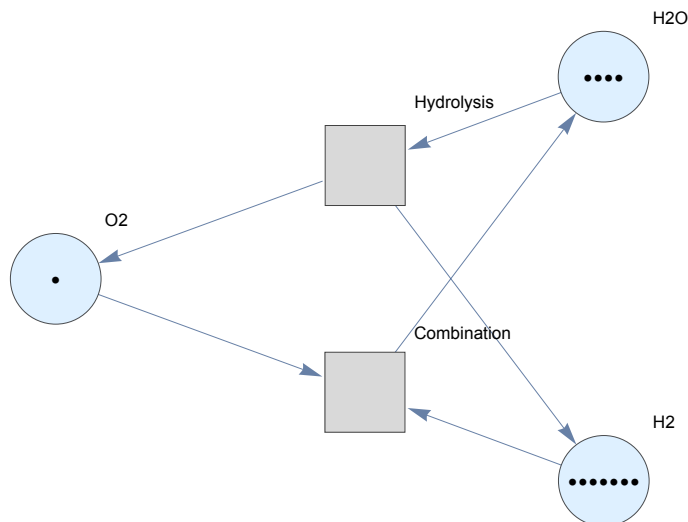
```
In[ ]:= petriNet = ResourceFunction["MakePetriNet"] [ <|
  "Places" -> {"O2", "H2O", "H2"}, "Transitions" -> {"Hydrolysis", "Combination"},
  "Arcs" -> {"Hydrolysis" -> "O2", "O2" -> "Combination", "Combination" -> "H2O",
    "H2O" -> "Hydrolysis", "Hydrolysis" -> "H2", "H2" -> "Combination"} |>, {1, 4, 7}]
```

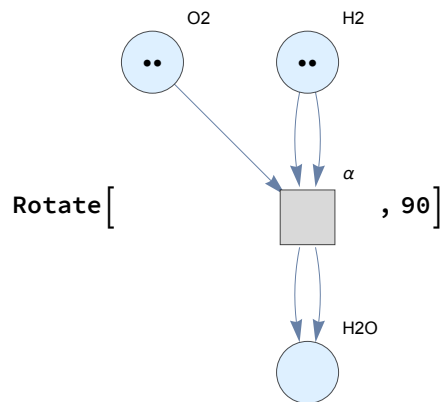
Out[]:=

PetriNetObject[ Places: 3
Transitions: 2
Arcs: 6]

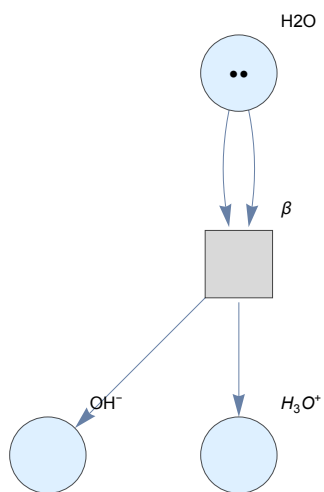
```
In[ ]:= petriNet["LabeledGraph"]
```

Out[]:=

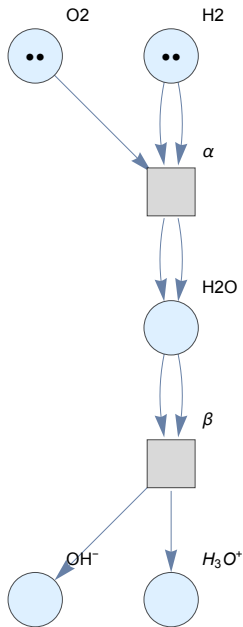




```
In[*]:= HydroniumHydroxidePetriNet["LabeledGraph"]
Out[*]=
```



```
In[*]:= FinalPetriNet["LabeledGraph"]
Out[*]:=
```



Algebraic operation called concatenation. Stack them together. What is the correspondence between these things.

<https://reader.elsevier.com/reader/sd/pii/S0303264711001110?token=417A5BF37FCEDDE3975CB0C9EC5F14F2BEC7014281F67626B94F42942F3D97D8AD0CB4C74C25A6CCF4CFF00298A15282&originRegion=u s-east-1&originCreation=20230325181517>

Designing gene networks

While existing computational approaches lead to models that, to various extent, describe the observed behavior of a gene network under naturally evolved transcription control, there is a lack of formal methods and algorithms for synthesizing external controls to force a given network to exhibit certain behavior as desired. Such methods and algorithms are essential in the design and construction of synthetic gene networks.

In synthetic biology, we **have** to create their genetic regulatory networks. And this algebraic approach will enable biologists to create that

Disentangle a complicated automaton...very complicated...we use holonomy decomposition. incredibly difficult thing.

It provides an algorithm, based on the theory of automata and formal languages, for computing control strategies that guarantee that the system will behave as desired.

For example, each state can be broken down into components called **flip-flops**. It can be emulated by these entirely.

Computational models of gene networks are usually constructed based on the premise that significant activities in gene expression can be considered to occur discretely. In the simplest abstraction, the state of a gene is either **on** or **off**. A gene that has been switched on will start to synthesize its product (i.e. protein).