

ComputeInvariantDensities

November 8, 2023

Load the badic package:

```
[1]: from badic import *
```

1 Example of dimension 1

```
[2]: a = dag.MatricesGraph([matrix([[1,0],[1,2]]), matrix([[1, 1],[1,0]])])  
print(a.is_complete())  
show(a.alphabet)
```

True

$$\left[\begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \right]$$

```
[3]: d = a.invariant_density()  
print(a.is_invariant_density(d))  
show(d)
```

True

$$\left[\frac{1}{(2x_0 + x_1)(x_0 + x_1)} \right]$$

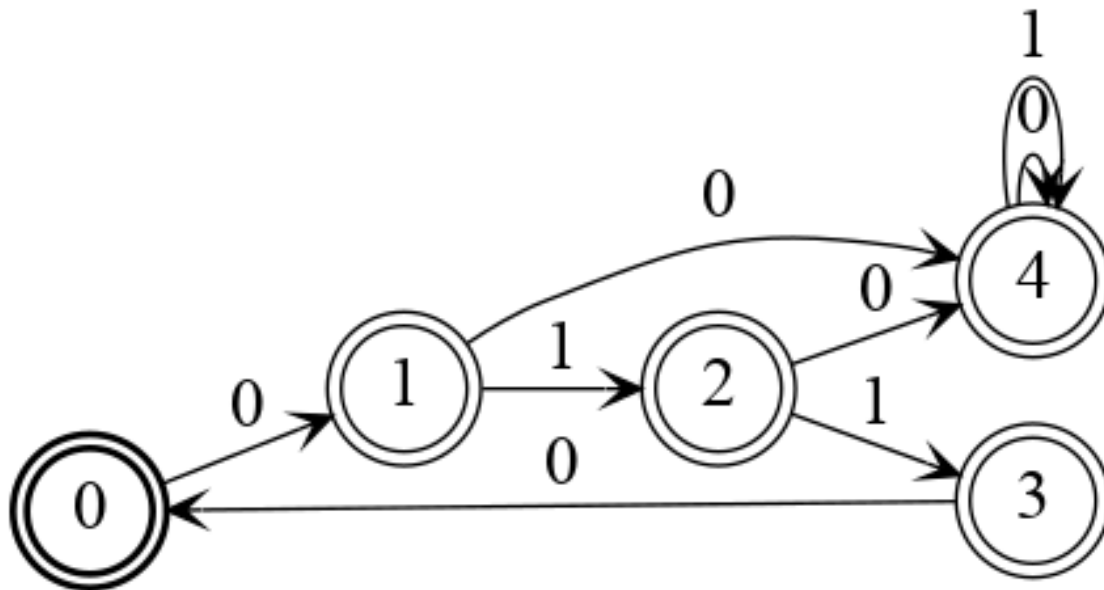
```
[4]: show(a.domains_polyhedra())
```

$$\left[\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \mathbb{R}_+^2 \right]$$

2 Examples with any set of quadratic numbers appearing in densities

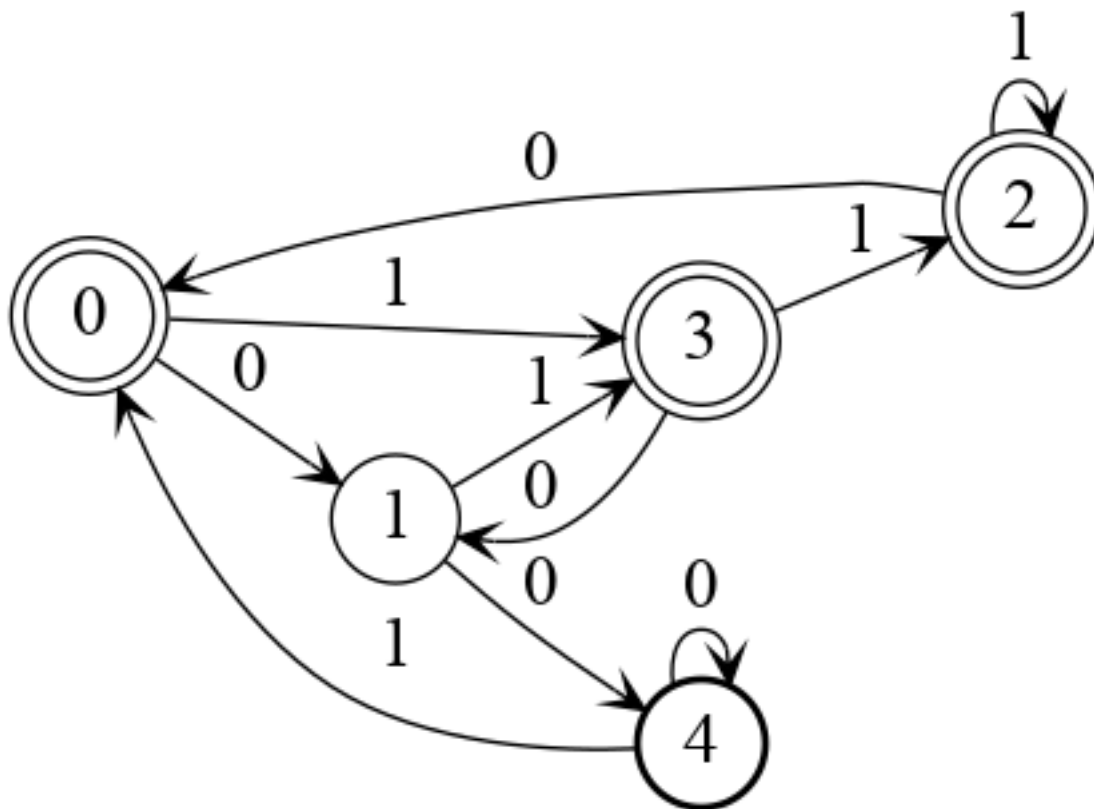
```
[92]: # compute the language  $L_{\sqrt{2}}$  of words less than  $uv^{\infty}$  in lexicographical
      ↪ order,
      # where  $uv^{\infty}$  is the expansion of  $\sqrt{2}$  for the fully subtractive algorithm
      ↪ on two letters
      dag.L(sqrt(2)).plot()
```

[92]:



```
[93]: # construct a win-lose graph from a set of quadratic numbers
      a = dag.winlose_from_set_of_quadratic_numbers([sqrt(2)])
      a.plot()
```

[93]:

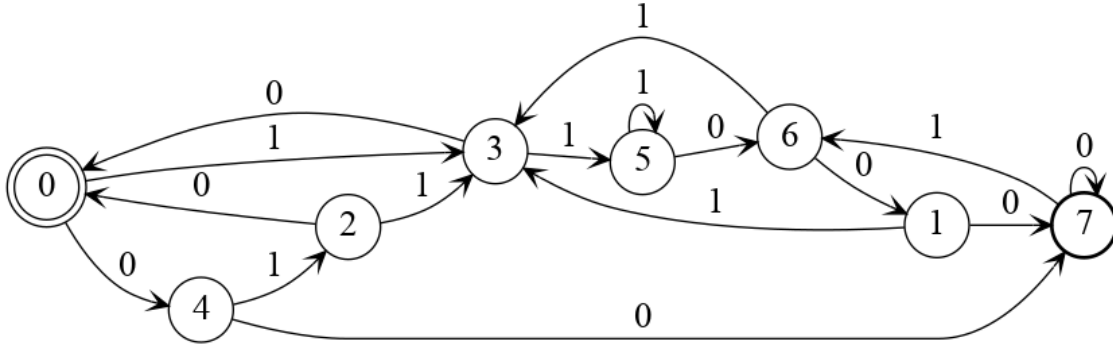


```
[94]: x = sqrt(2)
      K.<b> = NumberField(x.minpoly(), embedding=QQbar(x)) # NumberField to express
      ↪ the result
      show(a.invariant_density(ring=K))
```

$$\left[\frac{2(5\sqrt{2}+7)}{(x_1(3\sqrt{2}+4)+2x_0(2\sqrt{2}+3))(x_0(\sqrt{2}+2)+2x_1(\sqrt{2}+1))}, \frac{12\sqrt{2}+17}{(x_0(7\sqrt{2}+10)+x_1(5\sqrt{2}+7))(x_0(2\sqrt{2}+3)+x_1(3\sqrt{2}+4))} \right]$$

```
[95]: # construct another win-lose graph from another set of quadratic numbers
      a = dag.winlose_from_set_of_quadratic_numbers([sqrt(2), sqrt(3)])
      a.plot()
```

[95]:



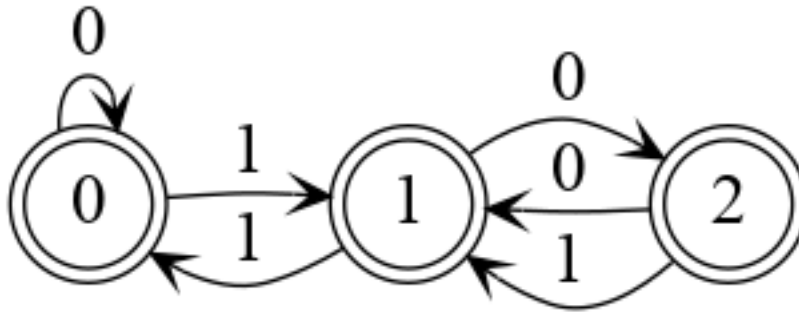
```
[97]: K2.<c> = K.extension(sqrt(3).minpoly(), embedding=QQbar(sqrt(3))) # add sqrt(3)
      ↪to previous field
K3 = K2.absolute_field('d')
K4.<e> = NumberField(K3.gen().minpoly(), embedding=K3.gen().minpoly().
      ↪roots(ring=QQbar)[0][0]) # convert it
show(a.invariant_density(ring=K4))
```

$$\left[\frac{8\sqrt{3}\sqrt{2} + 3\sqrt{20\sqrt{3}\sqrt{2} + 62} + 12}{3\left(x_0(4\sqrt{3} + 7) + x_1\left(7\sqrt{\frac{1}{3}} + 4\right)\right)\left(x_1(3\sqrt{2} + 4) + 2x_0(2\sqrt{2} + 3)\right)}, \frac{\sqrt{6} + \sqrt{8\sqrt{6} + 50} + 6}{(x_0(5\sqrt{3} + 9) + x_1(3\sqrt{3} + 5))(x_0(2\sqrt{2} + 3) + x_1(2\sqrt{2} + 3))} \right]$$

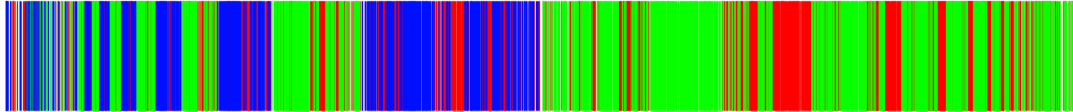
3 Example whose density has a Cantor of singularities

```
[51]: a = DetAutomaton([(0,0,0), (0,1,1), (1,0,1), (1,2,0), (2,1,0), (2,1,1)])
a.plot()
```

[51]:

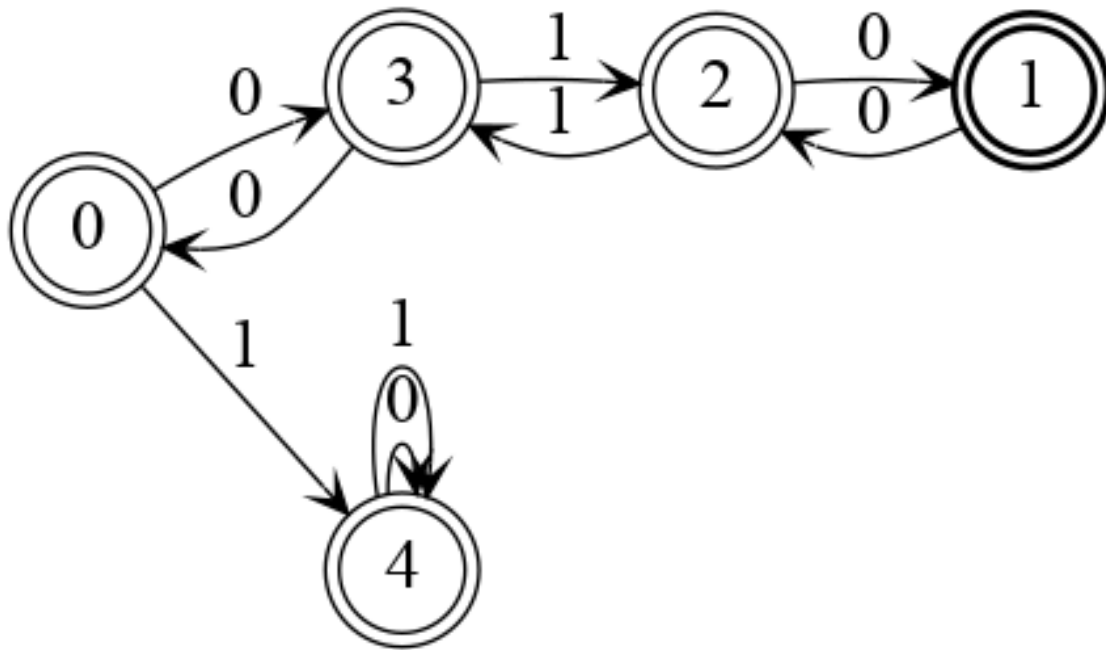


```
[52]: # plot the complicated domain
a.plot_domains(N2=12, colors='hsv').show(figsize=20)
```



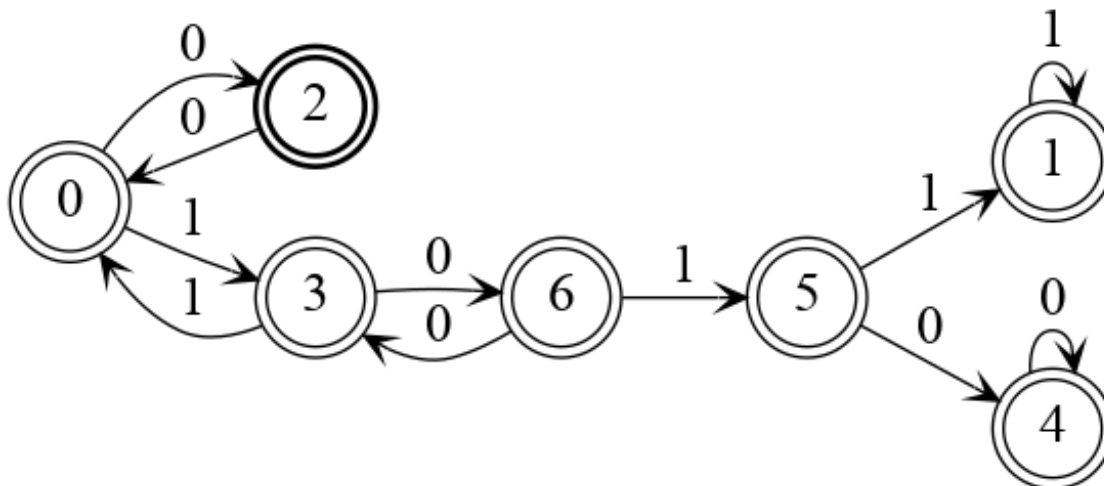
```
[53]: # compute a domain language (but labeled with 0 and 1)
am = a.mirror_languages()[2]
am.plot()
```

[53]:



```
[54]: # compute the boundary language defined in Proposition 5.2
b = am.boundary_two_letters()
b.plot()
```

[54]:



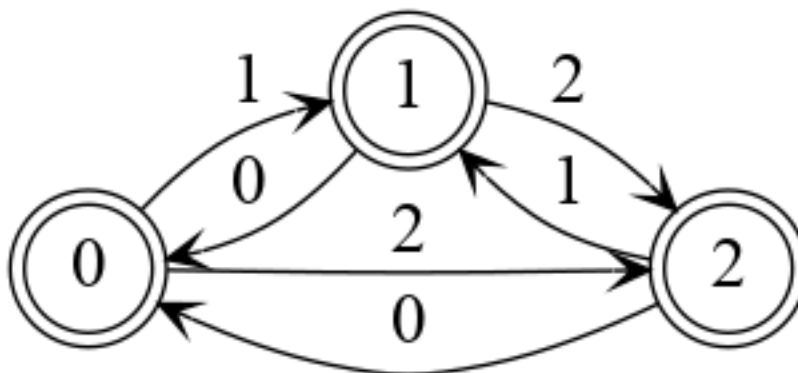
```
[55]: # check that the boundary has no isolated point
b.boundary_two_letters().has_same_language_as(b)
```

[55]: True

4 Cassaigne algorithm as a win-lose graph

```
[56]: a = dag.CassaigneWinLose()
a.plot()
```

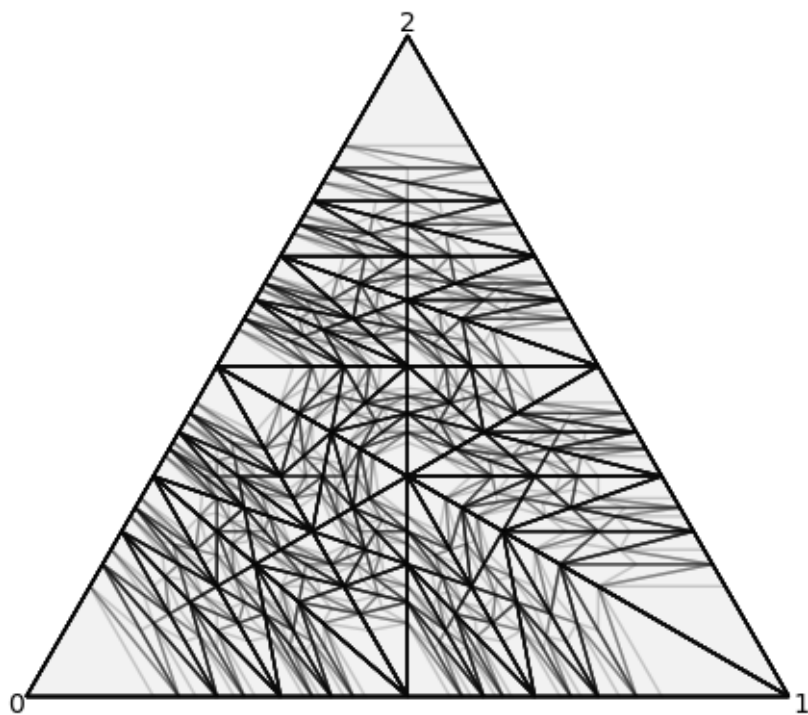
[56]:



Display 10 steps of the algorithm from state 1:

```
[57]: a.plot_algo(1,10)
```

[57]:



Compute an exact invariant density:

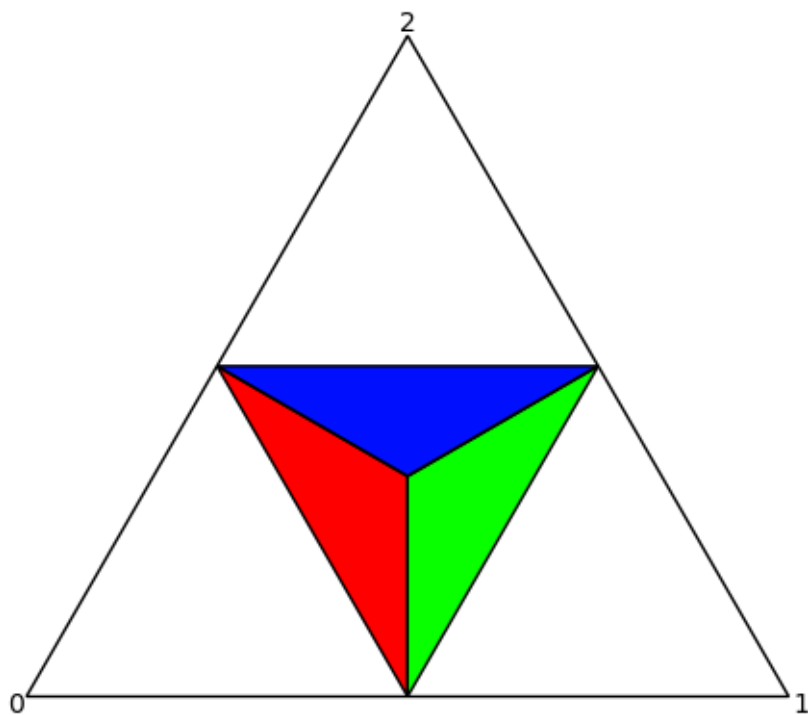
```
[58]: show(a.invariant_density())
```

$$\left[\frac{1}{(x_0 + x_1 + x_2)(x_0 + x_1)(x_0 + x_2)}, \frac{1}{(x_0 + x_1 + x_2)(x_0 + x_1)(x_1 + x_2)}, \frac{1}{(x_0 + x_1 + x_2)(x_0 + x_2)(x_1 + x_2)} \right]$$

Plot domains:

```
[59]: a.plot_domains()
```

```
[59]:
```



Display description of domains:

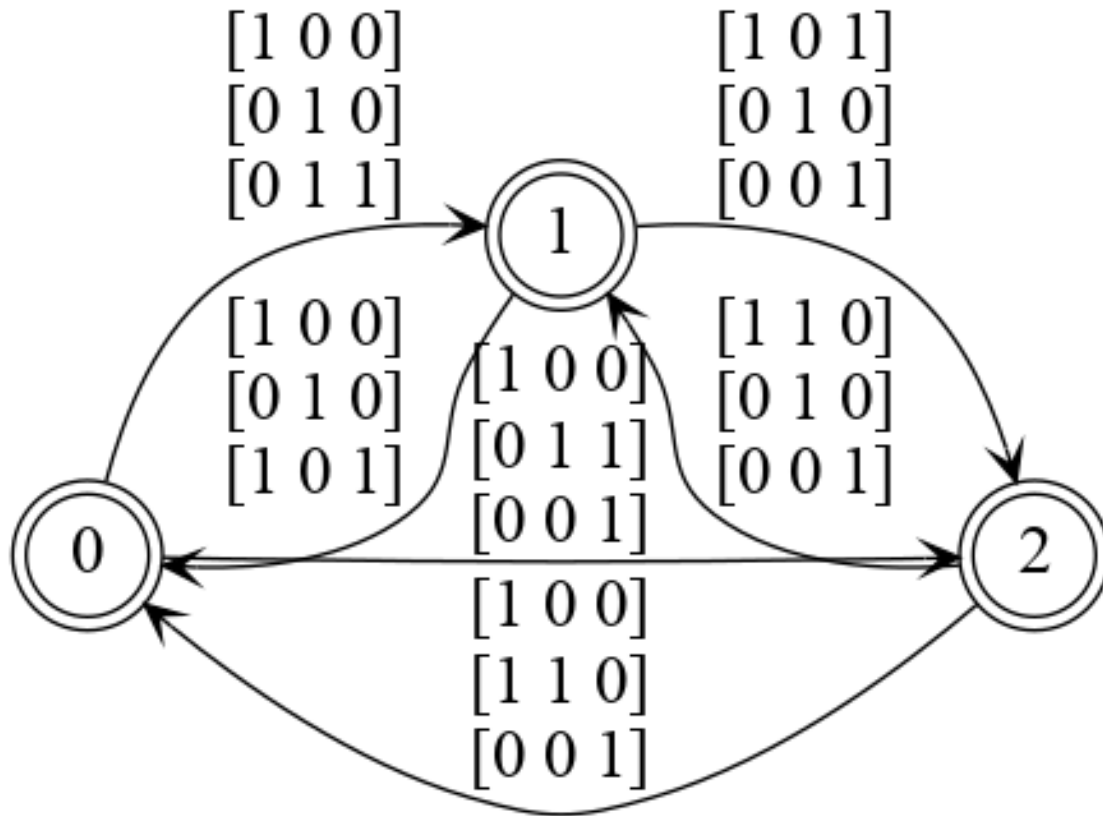
```
[60]: show(a.domains_polyhedra())
```

$$\left[\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3, \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3, \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3 \right]$$

Convert the win-lose graph to a matrices graph:

```
[61]: a.matrices_graph().plot()
```

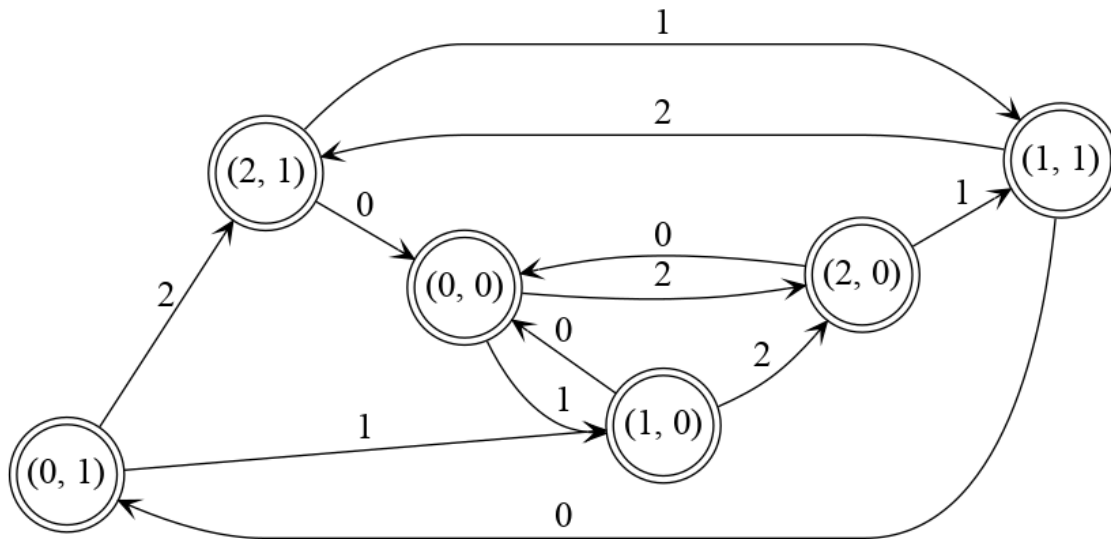
```
[61]:
```



4.0.1 Make an extension of the win-lose graph of the Cassaigne algorithm:

```
[62]: # compute the flip-flop for some set of edges
AS = [[(2,0,0),(0,1,1)],[(2,1,1)]]
a2 = a.flip_flop(AS)
a2.plot()
```

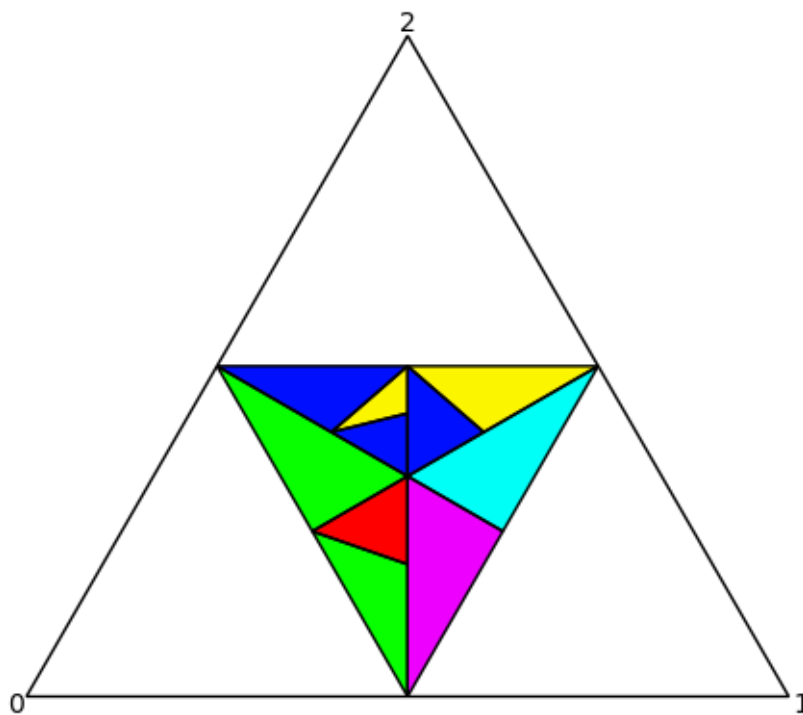
[62]:



Plot domains of the extension:

```
[63]: a2.plot_domains()
```

[63]:



Display corresponding matrices:

```
[64]: ls = a2.domains_polyhedra()
      for d in ls:
          show(d)
```

$$\begin{pmatrix} 1 & 2 & 2 \\ 1 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 2 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 1 & 2 & 2 \\ 1 & 1 & 2 \\ 2 & 2 & 3 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 2 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 2 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 2 & 1 & 2 \\ 1 & 1 & 2 \\ 2 & 1 & 3 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3$$

Compute an exact invariant density:

```
[65]: ld = a2.invariant_density()
      print(a2.is_invariant_density(ld))
      ld
```

True

```
[65]: [1/((2*x0 + 2*x1 + x2)*(2*x0 + x1 + x2)*(x0 + x1 + x2)),
      1/((2*x0 + 2*x1 + 3*x2)*(2*x0 + x1 + 2*x2)*(x0 + x1 + 2*x2)) + 1/((x0 + 2*x1 +
      2*x2)*(x0 + x1 + 2*x2)*(x1 + x2)),
      1/((2*x0 + 2*x1 + x2)*(2*x0 + x1 + x2)*(x0 + x1)) + 1/((2*x0 + x1 + x2)*(x0 +
      x1 + x2)*(x0 + x2)),
      1/((x0 + 2*x1 + x2)*(x0 + x1 + x2)*(x1 + x2)),
      1/((2*x0 + 2*x1 + 3*x2)*(2*x0 + x1 + 2*x2)*(x0 + x1 + x2)) + 1/((x0 + 2*x1 +
      2*x2)*(x0 + x1 + 2*x2)*(x0 + x1 + x2)) + 1/((2*x0 + x1 + 2*x2)*(x0 + x1 +
      2*x2)*(x0 + x2)),
      1/((x0 + 2*x1 + x2)*(x0 + x1 + x2)*(x0 + x1))]
```

```
[66]: for d in ld:
      show(d)
```

$$\frac{1}{(2x_0 + 2x_1 + x_2)(2x_0 + x_1 + x_2)(x_0 + x_1 + x_2)}$$

$$\frac{1}{(2x_0 + 2x_1 + 3x_2)(2x_0 + x_1 + 2x_2)(x_0 + x_1 + 2x_2)} + \frac{1}{(x_0 + 2x_1 + 2x_2)(x_0 + x_1 + 2x_2)(x_1 + x_2)}$$

$$\frac{1}{(2x_0 + 2x_1 + x_2)(2x_0 + x_1 + x_2)(x_0 + x_1)} + \frac{1}{(2x_0 + x_1 + x_2)(x_0 + x_1 + x_2)(x_0 + x_2)}$$

$$\frac{1}{(x_0 + 2x_1 + x_2)(x_0 + x_1 + x_2)(x_1 + x_2)}$$

$$\frac{1}{(2x_0 + 2x_1 + 3x_2)(2x_0 + x_1 + 2x_2)(x_0 + x_1 + x_2)} + \frac{1}{(x_0 + 2x_1 + 2x_2)(x_0 + x_1 + 2x_2)(x_0 + x_1 + x_2)} + \frac{1}{(2x_0 + x_1 + x_2)(x_0 + x_1 + x_2)(x_1 + x_2)}$$

$$\frac{1}{(x_0 + 2x_1 + x_2)(x_0 + x_1 + x_2)(x_0 + x_1)}$$

Check that it is indeed an invariant density:

```
[67]: a2.is_invariant_density(ld)
```

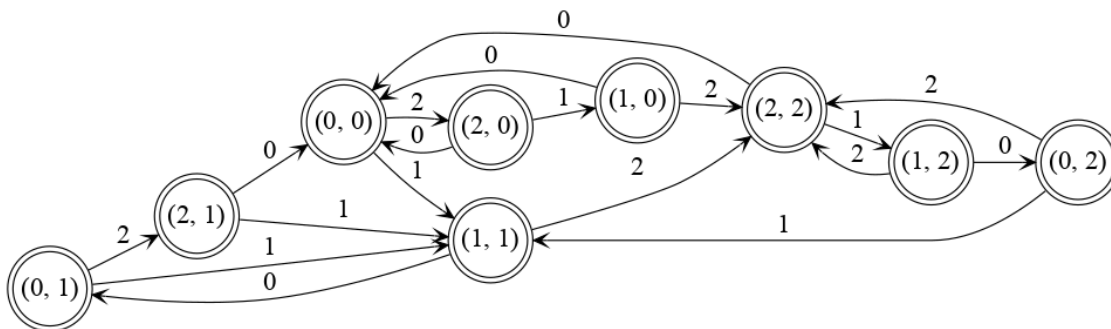
[67]: True

4.0.2 Make another extension with infinite sum

```
[68]: # compute the flip-flop for another set of edges
AS = [[(2,0,0)],[(0,1,1)],[(1,2,2)]]
a2 = a.flip_flop(AS)
cc = a2.strongly_connected_components()
print(cc)
a2 = a2.sub_automaton(max(cc, key=lambda x:len(x)))
a2.plot()
```

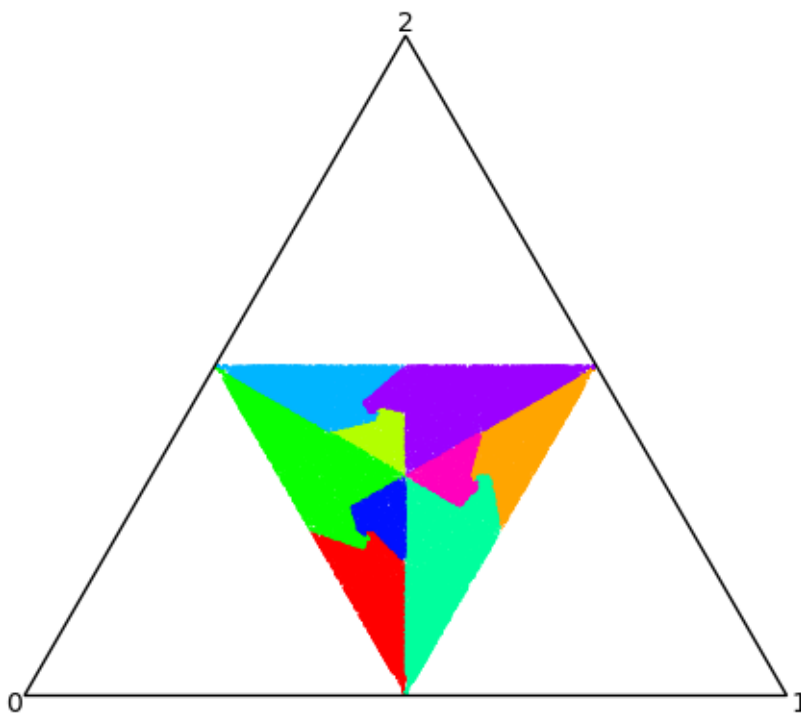
```
dict_values([[0, 1, 2, 3, 4, 5, 6, 7, 8]])
```

[68]:



```
[18]: a2.plot_domains()
```

```
[18]:
```



```
[69]: for d in a2.domains_polyhedra():
       show(d)
```

$$\bigcup_{j \in \mathbb{N}} \left(\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \right)^j \begin{pmatrix} 2 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}^j \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}^j \begin{pmatrix} 2 & 1 & 2 \\ 1 & 1 & 2 \\ 2 & 1 & 3 \end{pmatrix} \mathbb{R}_+^3$$

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}^j \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 2 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

```
[70]: # this computation works, but the result is too big
      #a2.invariant_density()
```

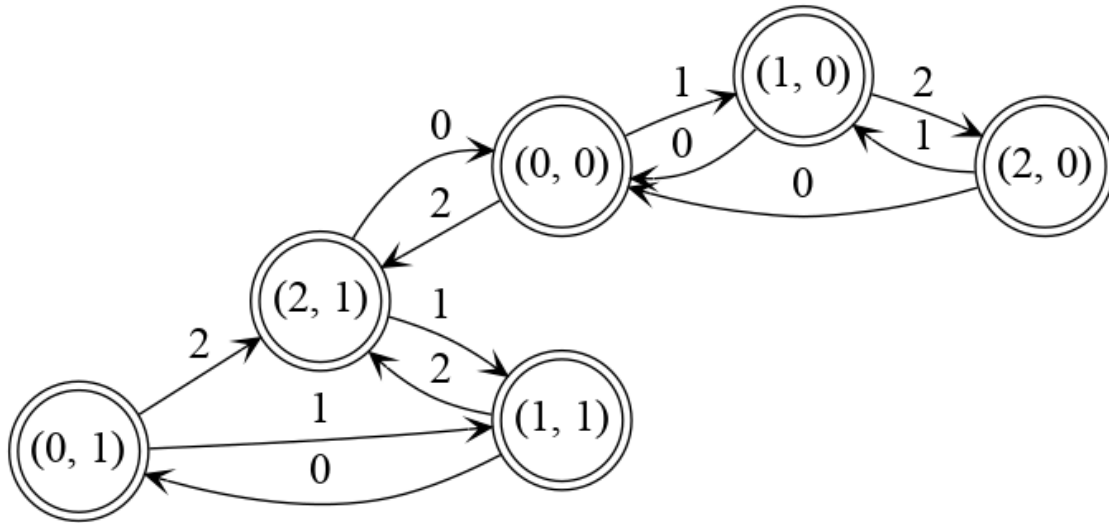
4.0.3 Make another extension with more complicated domains

```
[71]: # compute the flip-flop for another set of edges
      #AS = [[(2,0,0)],[(0,1,1)],[(1,2,2)]]
      AS = [[(2,0,0)],[(0,2,2)]]
      a2 = a.flip_flop(AS)
      cc = a2.strongly_connected_components()
      print(cc)
      a2 = a2.sub_automaton(max(cc, key=lambda x:len(x)))
```

```
a2.plot()
```

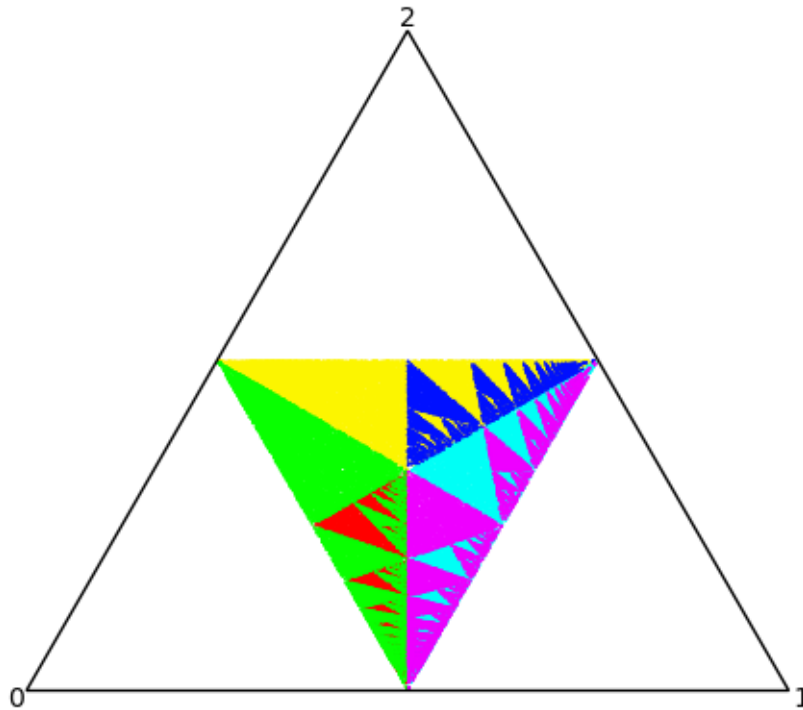
```
dict_values([[0, 1, 2, 3, 4, 5]])
```

[71]:



[72]: `a2.plot_domains()`

[72]:



```
[73]: for d in a2.domains_polyhedra():
      show(d)
```

$\bigcup_{m \in L} m\mathbb{R}_+^3$, where L is recognized by a DetAutomaton with 5 states and an alphabet of 6 letters

$\bigcup_{m \in L} m\mathbb{R}_+^3$, where L is recognized by a DetAutomaton with 5 states and an alphabet of 6 letters

$\bigcup_{m \in L} m\mathbb{R}_+^3$, where L is recognized by a DetAutomaton with 5 states and an alphabet of 6 letters

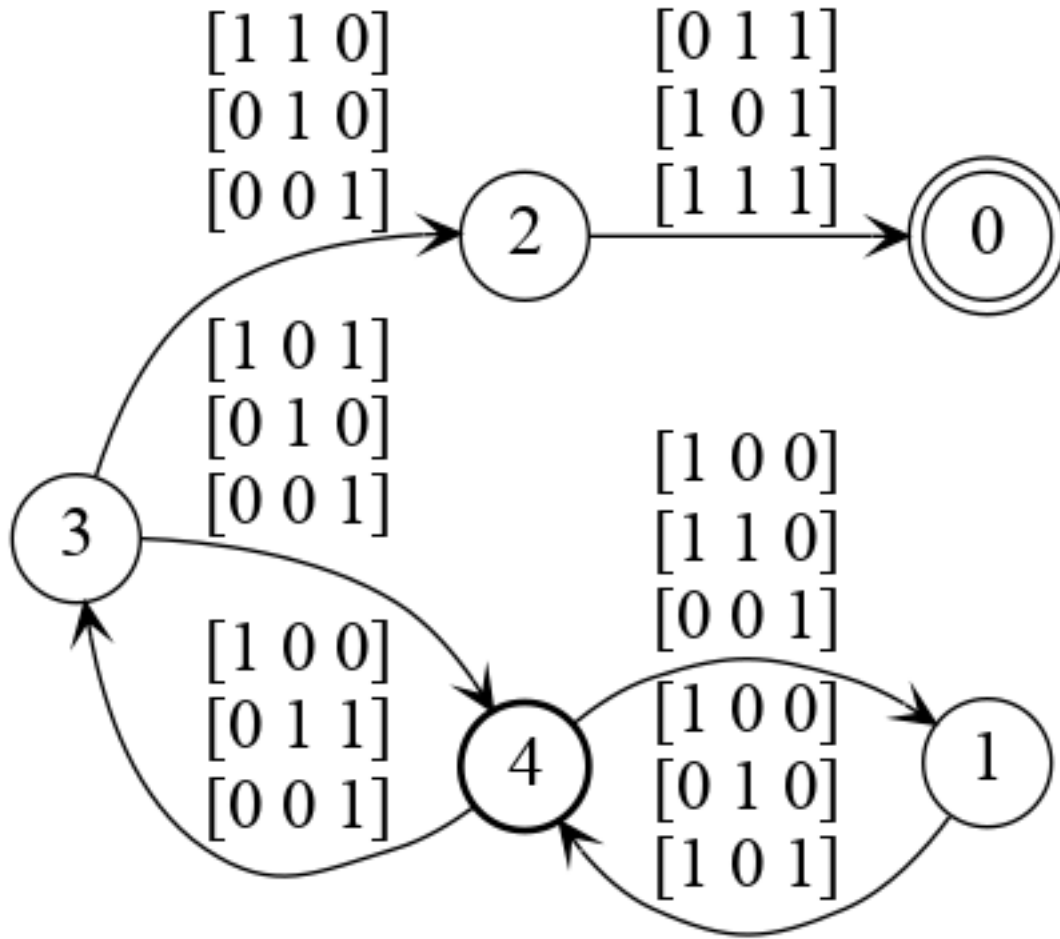
$\bigcup_{m \in L} m\mathbb{R}_+^3$, where L is recognized by a DetAutomaton with 5 states and an alphabet of 6 letters

$\bigcup_{m \in L} m\mathbb{R}_+^3$, where L is recognized by a DetAutomaton with 5 states and an alphabet of 6 letters

$\bigcup_{m \in L} m\mathbb{R}_+^3$, where L is recognized by a DetAutomaton with 5 states and an alphabet of 6 letters

```
[74]: # automaton describing the infinite union of triangles for the domain of one
      ↪ state
      d.a.plot()
```

[74]:



[75]: `a2.invariant_density()`

[75]: [Sum of $|\det(m)|/\text{prod}(m \ x)$ for m recognized by a DetAutomaton with 5 states and an alphabet of 6 letters,
Sum of $|\det(m)|/\text{prod}(m \ x)$ for m recognized by a DetAutomaton with 5 states and an alphabet of 6 letters,
Sum of $|\det(m)|/\text{prod}(m \ x)$ for m recognized by a DetAutomaton with 5 states and an alphabet of 6 letters,
Sum of $|\det(m)|/\text{prod}(m \ x)$ for m recognized by a DetAutomaton with 5 states and an alphabet of 6 letters,
Sum of $|\det(m)|/\text{prod}(m \ x)$ for m recognized by a DetAutomaton with 5 states and an alphabet of 6 letters,
Sum of $|\det(m)|/\text{prod}(m \ x)$ for m recognized by a DetAutomaton with 5 states and an alphabet of 6 letters]

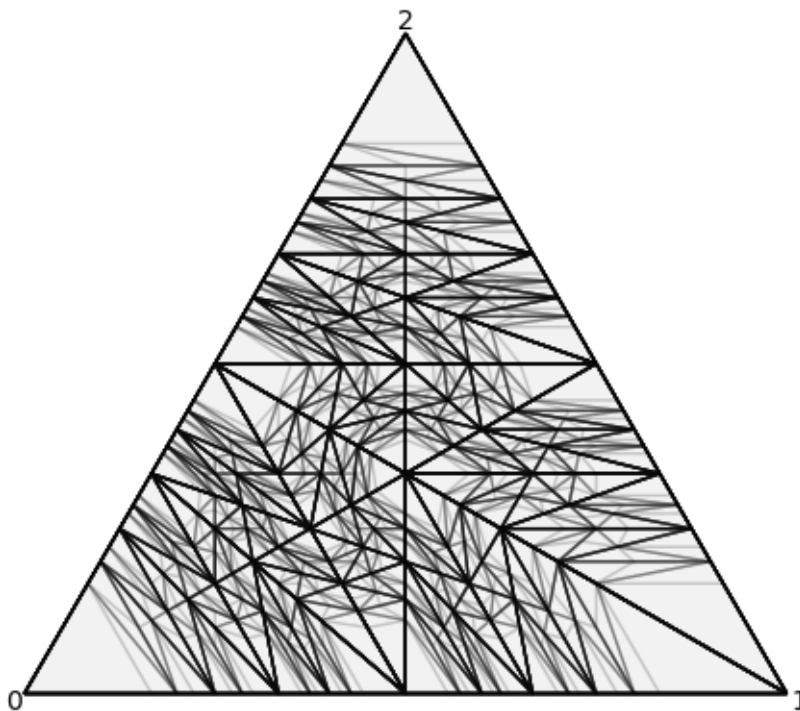
5 Cassaigne algorithm

```
[76]: # Cassaigne as full-shift with matrices
a = dag.Cassaigne()
a
```

[76]: DetAutomaton with 1 state and an alphabet of 2 letters

```
[77]: a.plot_algo(a.initial_state, 10)
```

[77]:



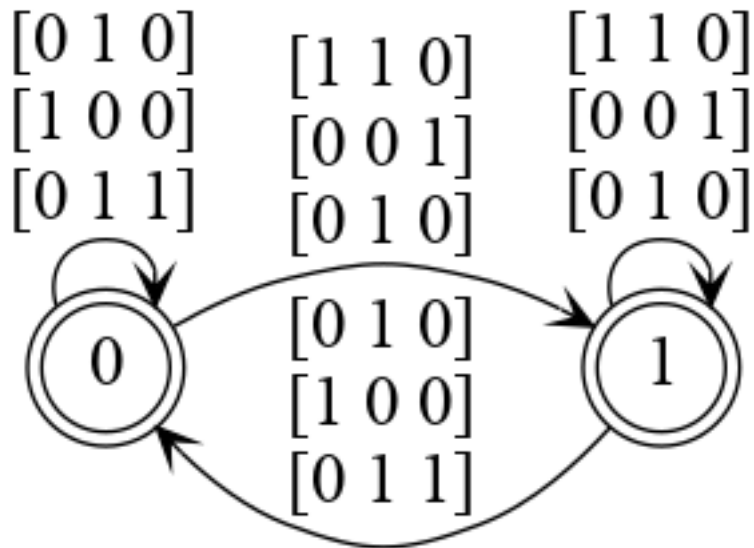
```
[78]: show(a.invariant_density())
```

$$\left[\frac{1}{(x_0 + x_1 + x_2)(x_0 + x_1)(x_1 + x_2)} \right]$$

5.0.1 Make an extension:

```
[79]: # extension
lm = a.alphabet
a = DetAutomaton([(0, lm[0], 0), (0, lm[1], 1), (1, lm[0], 0), (1, lm[1], 1)], □
    ↪avoidDiGraph=True)
a.plot()
```

[79]:



```
[80]: ld = a.invariant_density()
      show(ld)
```

$$\left[\frac{1}{(x_0 + 2x_1 + x_2)(x_0 + x_1 + x_2)(x_0 + x_1)}, \frac{1}{(x_0 + 2x_1 + x_2)(x_0 + x_1 + x_2)(x_1 + x_2)} \right]$$

```
[81]: a.is_invariant_density(ld)
```

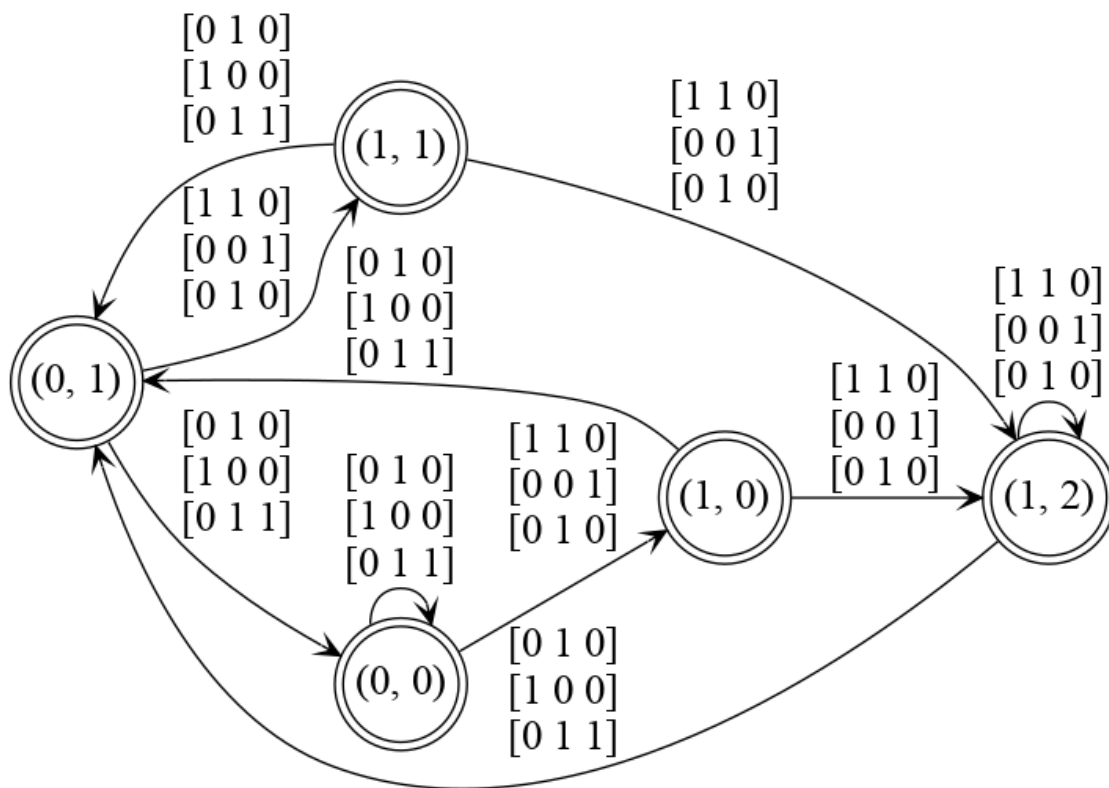
```
[81]: True
```

5.0.2 Make an extension of the extension:

```
[82]: # flip-flop
      #AS = [[(0, lm[0], 0)], [(1, lm[1], 1)]]
      AS = [[(0, lm[0], 0)], [(1, lm[0], 0)], [(1, lm[1], 1)]]
      a2 = a.flip_flop(AS)
      cc = a2.strongly_connected_components()
      print(cc)
      a2 = a2.sub_automaton(max(cc, key=lambda x: len(x)))
      a2.plot()
```

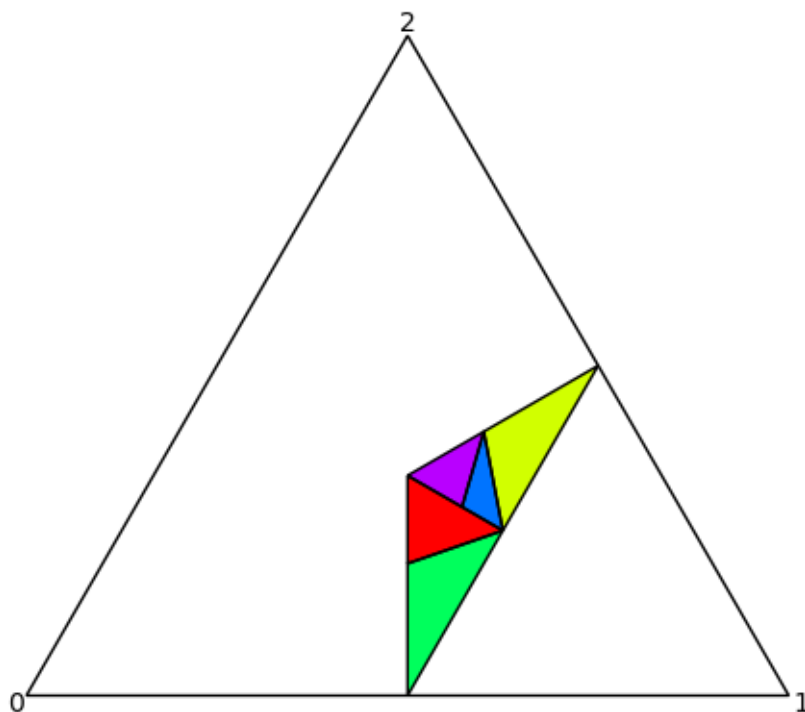
```
dict_values([[0, 1, 2, 3, 5], [4]])
```

```
[82]:
```



```
[83]: a2.plot_domains()
```

```
[83]:
```



```
[84]: ld = a2.invariant_density()
      for d in ld:
          show(d)
```

$$\frac{1}{(2x_0 + 2x_1 + x_2)(x_0 + 2x_1 + x_2)(x_0 + x_1 + x_2)}$$

$$\frac{1}{(x_0 + 2x_1 + 2x_2)(x_0 + 2x_1 + x_2)(x_1 + x_2)}$$

$$\frac{1}{(2x_0 + 2x_1 + x_2)(x_0 + 2x_1 + x_2)(x_0 + x_1)}$$

$$\frac{1}{(2x_0 + 3x_1 + 2x_2)(x_0 + 2x_1 + 2x_2)(x_0 + 2x_1 + x_2)}$$

$$\frac{1}{(2x_0 + 3x_1 + 2x_2)(x_0 + 2x_1 + 2x_2)(x_0 + x_1 + x_2)}$$

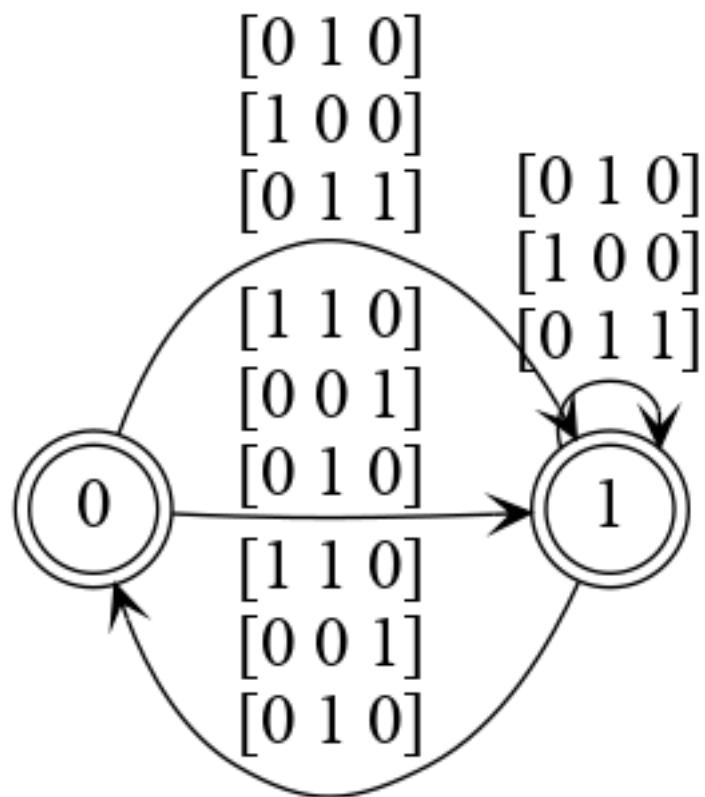
```
[85]: a2.is_invariant_density(ld)
```

```
[85]: True
```

5.0.3 Another extension of Cassaigne algo with infinite union of triangles

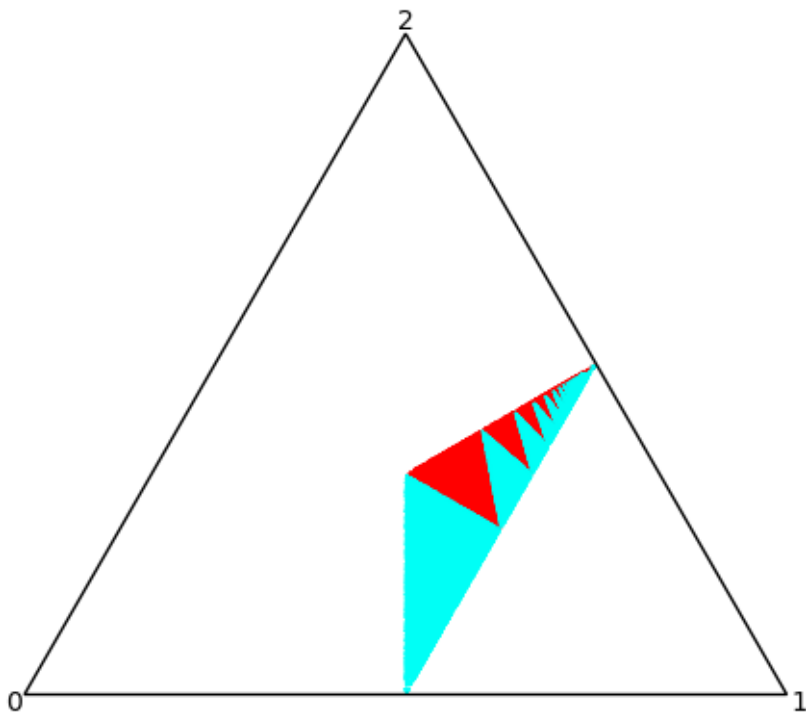
```
[86]: A = dag.Cassaigne().alphabet
      a = DetAutomaton([(0,1,A[0]),(0,1,A[1]),(1,1,A[0]),(1,0,A[1])])
      a.plot()
```

```
[86]:
```



```
[87]: a.plot_domains()
```

```
[87]:
```



```
[88]: for d in a.domains_polyhedra():
      show(d)
```

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3$$

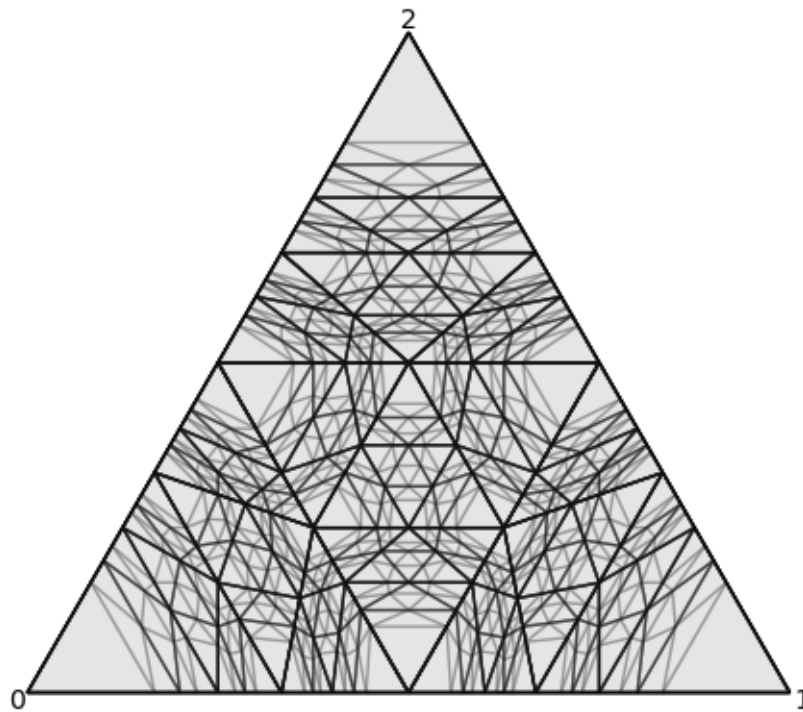
```
[89]: show(a.invariant_density())
```

$$\left[\sum_{j \in \mathbb{N}} \frac{1}{((j+2)x_1 + (j+2)x_2 + x_0)((j+2)x_1 + (j+1)x_2 + x_0)((j+1)x_1 + (j+1)x_2 + x_0)}, \sum_{j \in \mathbb{N}} \frac{1}{((j+2)x_1 + (j+1)x_2 + x_0)} \right]$$

6 Reverse algorithm

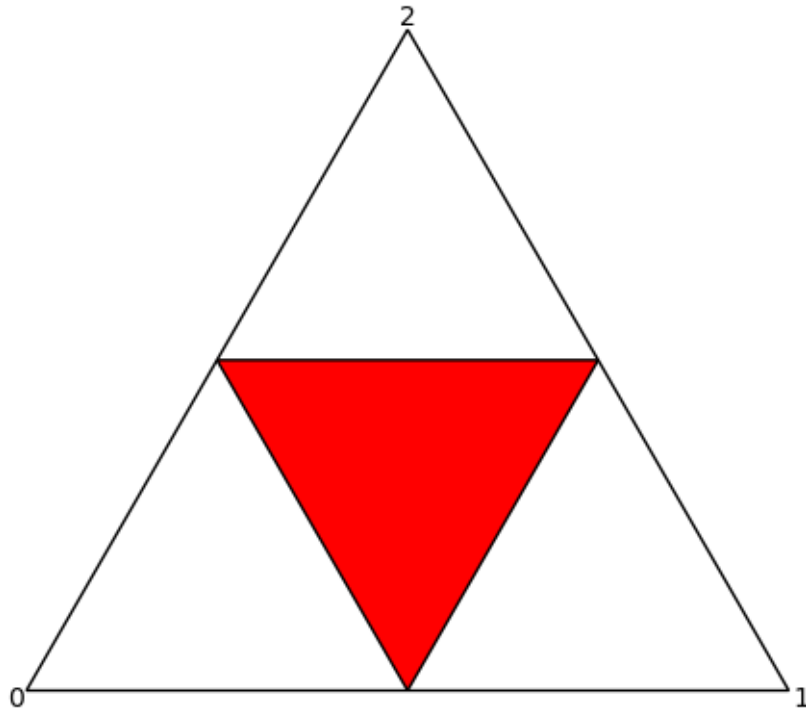
```
[46]: # reverse algorithm  
a = dag.Reverse()  
a.plot_algo(0,5)
```

[46]:



```
[47]: a.plot_domains()
```

[47]:



```
[48]: ld = a.invariant_density()
      show(ld)
      a.is_invariant_density(ld)
```

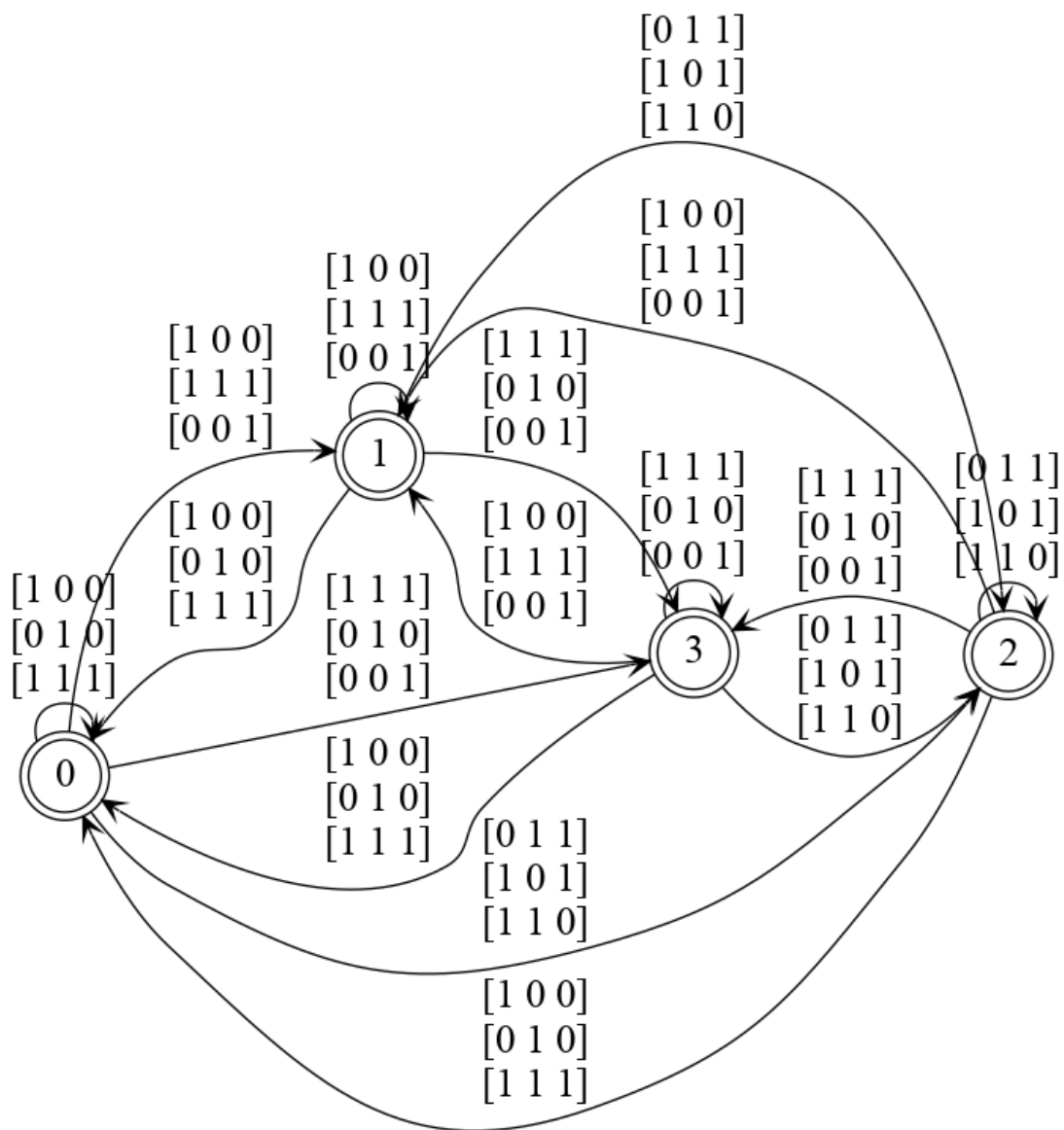
$$\left[\frac{2}{(x_0 + x_1)(x_0 + x_2)(x_1 + x_2)} \right]$$

[48]: True

6.0.1 Make an extension of the reverse algorithm:

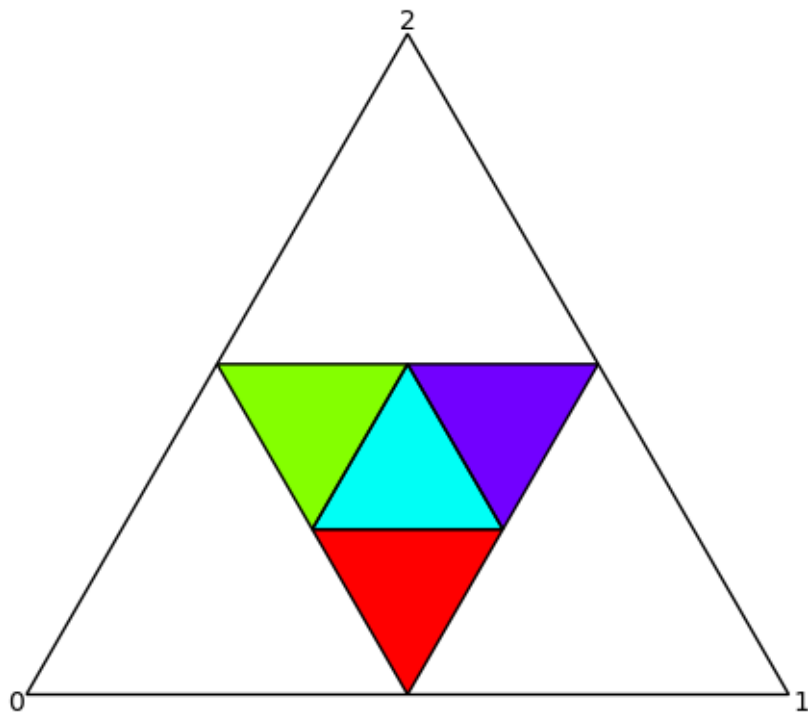
```
[49]: # extension
      lm = a.alphabet
      a = DetAutomaton([(i, lm[j], j) for i in range(len(lm)) for j in range(len(lm))],
      ↪ avoidDiGraph=True)
      a.plot()
```

[49]:



```
[50]: a.plot_domains()
```

```
[50]:
```



```
[51]: ld = a.invariant_density()
      for d in ld:
          show(d)
```

$$\frac{2}{(2x_0 + x_1 + x_2)(x_0 + 2x_1 + x_2)(x_0 + x_1)}$$

$$\frac{2}{(2x_0 + x_1 + x_2)(x_0 + x_1 + 2x_2)(x_0 + x_2)}$$

$$\frac{4}{(2x_0 + x_1 + x_2)(x_0 + 2x_1 + x_2)(x_0 + x_1 + 2x_2)}$$

$$\frac{2}{(x_0 + 2x_1 + x_2)(x_0 + x_1 + 2x_2)(x_1 + x_2)}$$

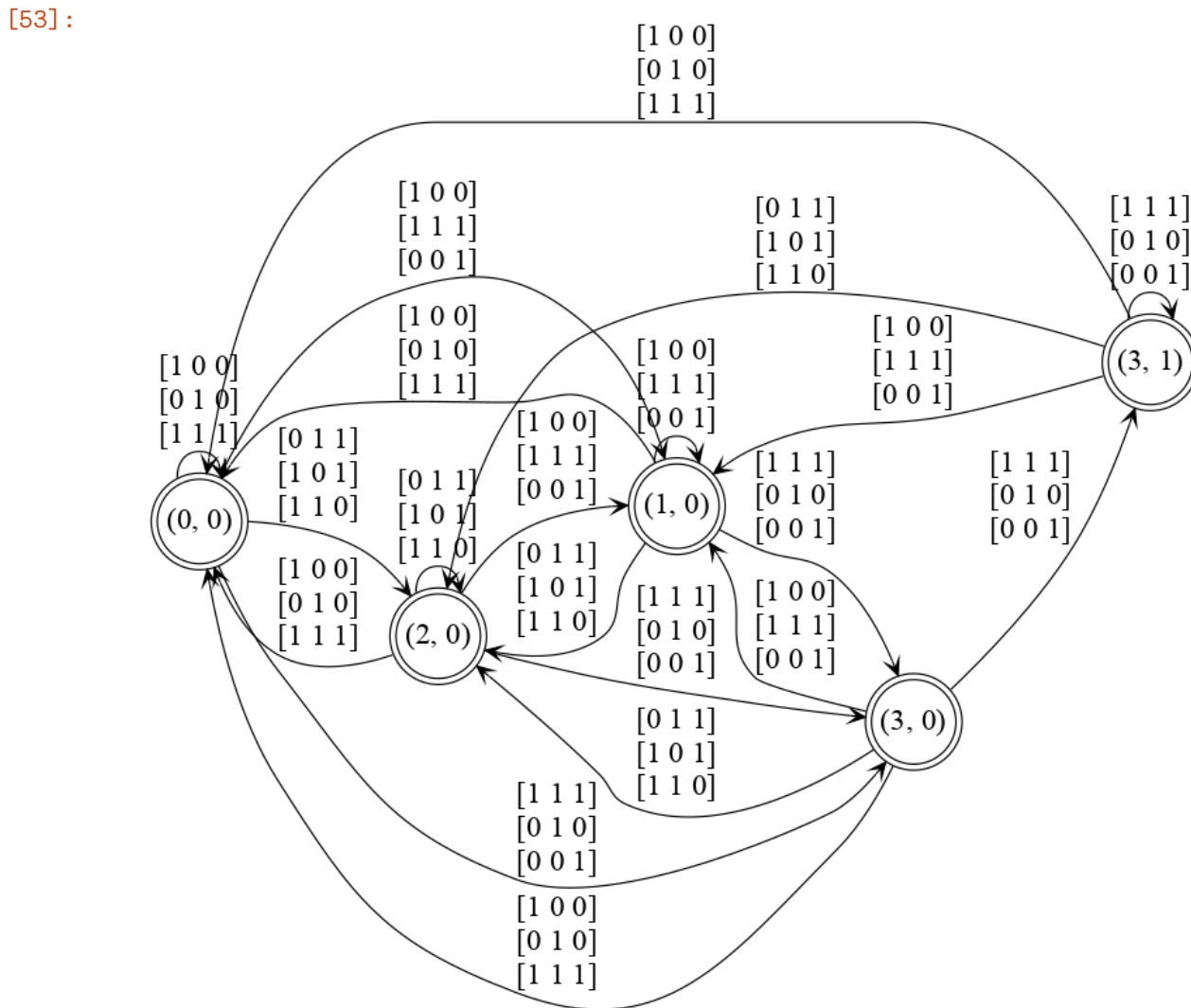
```
[52]: a.is_invariant_density(ld)
```

```
[52]: True
```

6.0.2 Make an extension of the extension:

```
[53]: # flip-flop
AS = [[(3,lm[0],0),(3,lm[1],1),(3,lm[2],2)],[(3,lm[3],3)]]
a2 = a.flip_flop(AS)
cc = a2.strongly_connected_components()
print(cc)
a2 = a2.sub_automaton(max(cc, key=lambda x:len(x)))
a2.plot()
```

```
dict_values([[0, 1, 4], [2, 3, 5, 6, 7]])
```



```
[54]: ld = a2.invariant_density()
print(a2.is_invariant_density(ld))
for d in ld:
    show(d)
```

True

$$\frac{2}{(2x_0 + x_1 + x_2)(x_0 + 2x_1 + x_2)(x_0 + x_1)}$$

$$\frac{2}{(x_0 + 3x_1 + 2x_2)(x_0 + 2x_1 + 3x_2)(x_1 + x_2)}$$

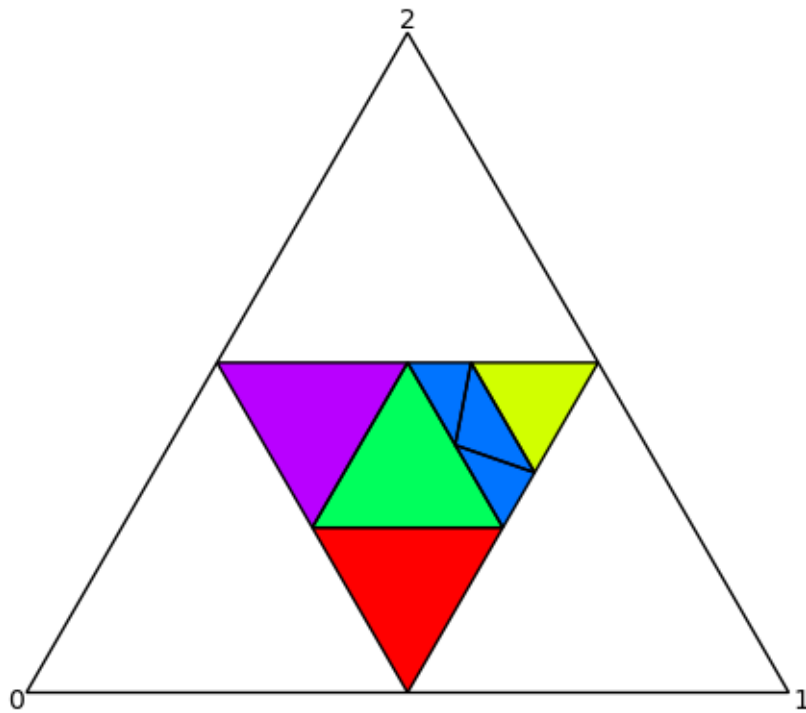
$$\frac{4}{(2x_0 + x_1 + x_2)(x_0 + 2x_1 + x_2)(x_0 + x_1 + 2x_2)}$$

$$\frac{4}{(2x_0 + 3x_1 + 3x_2)(x_0 + 3x_1 + 2x_2)(x_0 + 2x_1 + 3x_2)} + \frac{2}{(2x_0 + 3x_1 + 3x_2)(x_0 + 3x_1 + 2x_2)(x_0 + 2x_1 + x_2)} + \frac{2}{(2x_0 + 3x_1 + 3x_2)(x_0 + 2x_1 + x_2)(x_0 + x_1 + 2x_2)}$$

$$\frac{2}{(2x_0 + x_1 + x_2)(x_0 + x_1 + 2x_2)(x_0 + x_2)}$$

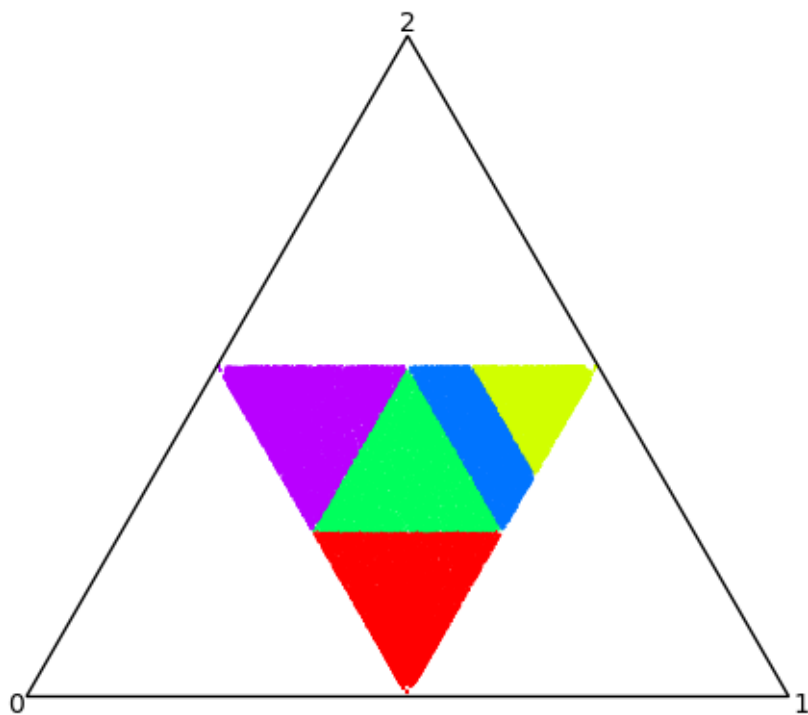
[55]: a2.plot_domains()

[55]:



[93]: a2.plot_domains(exact=False)

[93]:



```
[56]: ls = a2.domains_polyhedra()
      for d in ls:
          show(d)
```

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 2 & 1 & 1 \\ 3 & 3 & 2 \\ 3 & 2 & 3 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 1 & 2 & 1 \\ 3 & 3 & 2 \\ 2 & 3 & 1 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 3 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3$$

```
[57]: ls[3].e.l[0][0].det()
```

```
[57]: 4
```

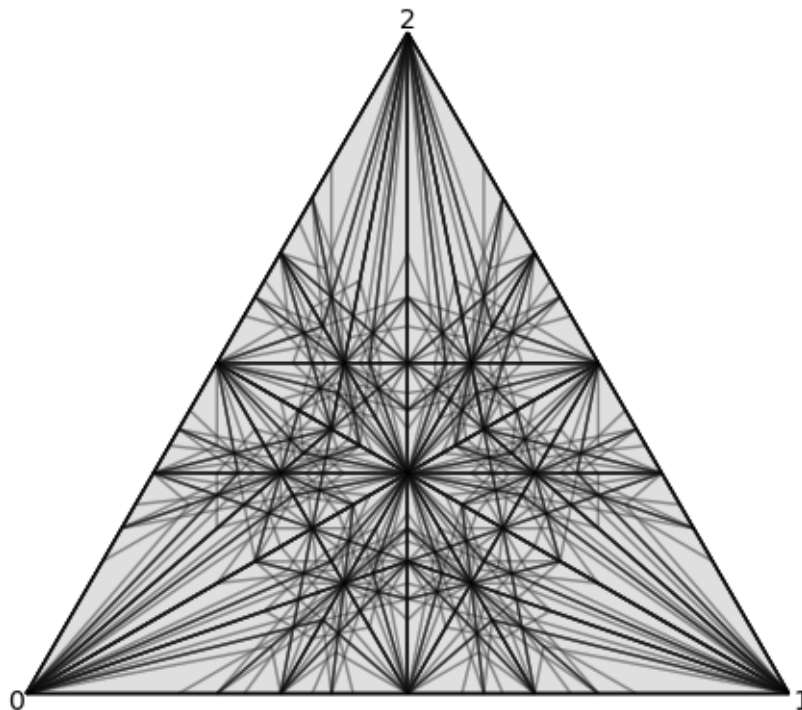
7 Poincaré

```
[58]: # Poincare
a = dag.Poincare()
a
```

```
[58]: DetAutomaton with 1 state and an alphabet of 6 letters
```

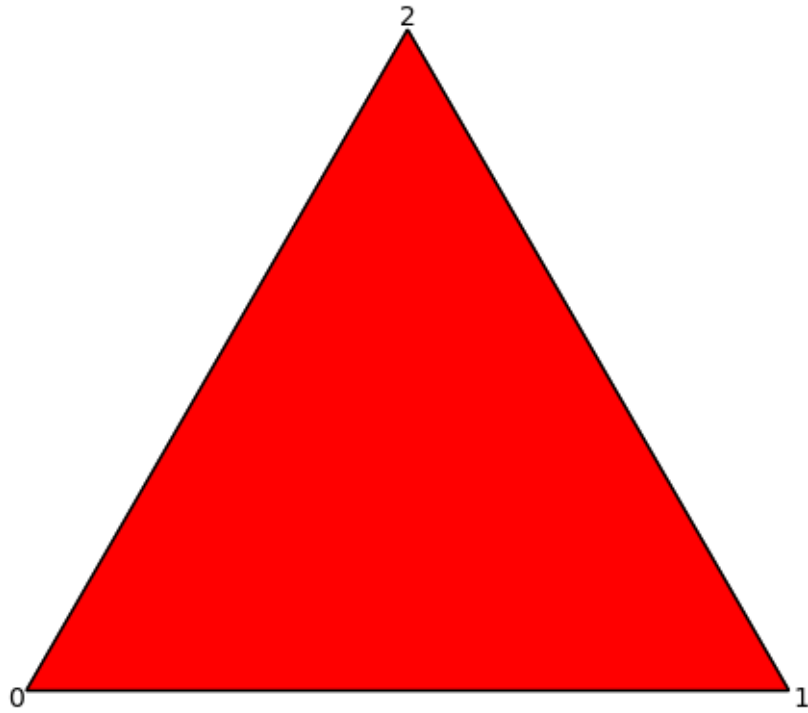
```
[59]: a.plot_algo(0,4)
```

```
[59]:
```



```
[60]: a.plot_domains()
```

```
[60]:
```



```
[61]: ld = a.invariant_density()
      print(a.is_invariant_density(ld))
      show(ld)
```

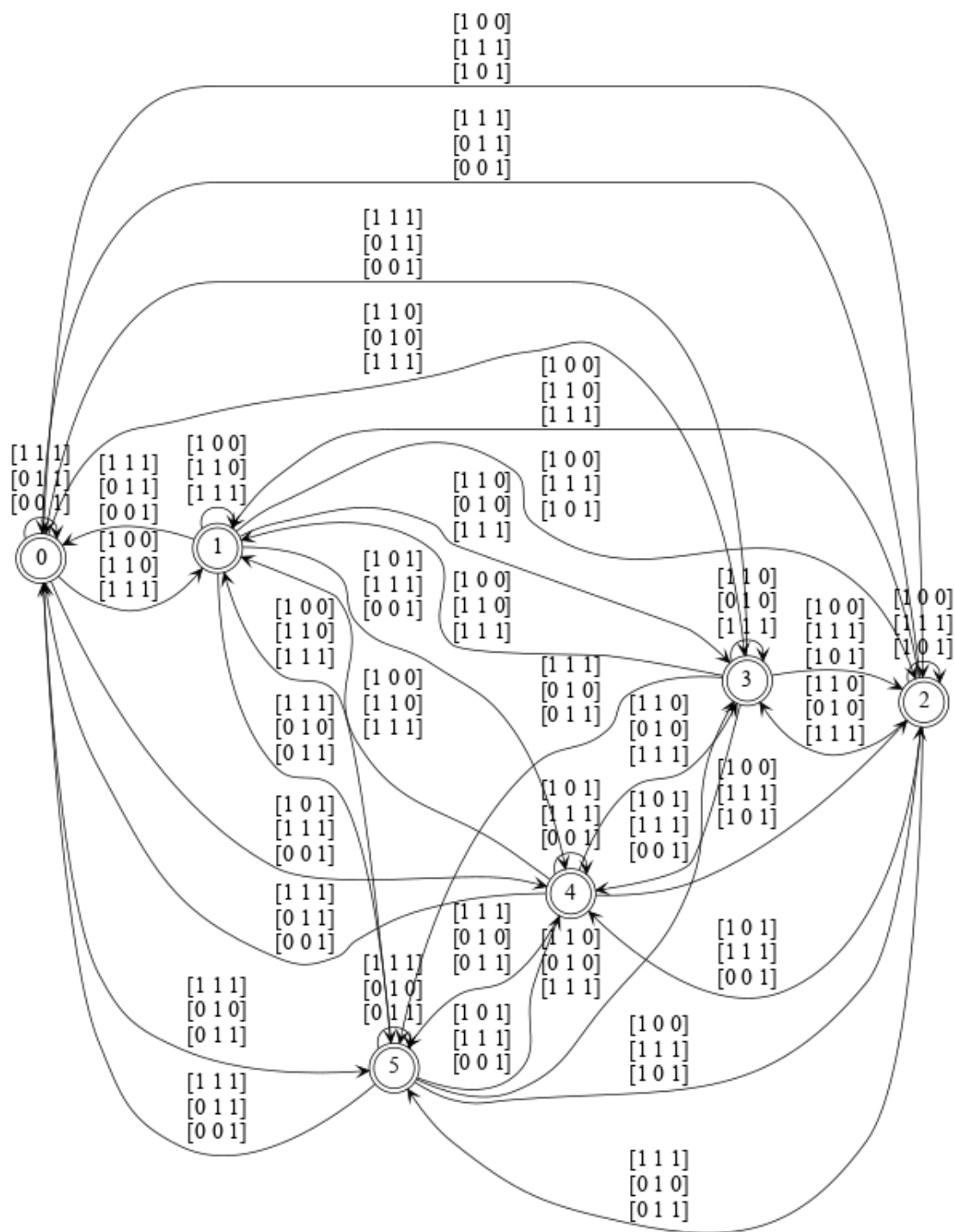
True

$$\left[\frac{1}{x_0 x_1 x_2} \right]$$

7.0.1 Make an extension of Poincaré algorithm:

```
[62]: # extension
      lm = a.alphabet
      a = DetAutomaton([(i,lm[j],j) for i in range(len(lm)) for j in range(len(lm))],
      ↪avoidDiGraph=True)
      a.plot()
```

[62]:



```
[63]: ld = a.invariant_density()
print(a.is_invariant_density(ld))
for d in ld:
    show(d)
```

True

$$\frac{1}{(x_0 + x_1 + x_2)(x_1 + x_2)x_2}$$

$$\frac{1}{(x_0 + x_1 + x_2)(x_0 + x_1)x_0}$$

$$\frac{1}{(x_0 + x_1 + x_2)(x_0 + x_2)x_0}$$

$$\frac{1}{(x_0 + x_1 + x_2)(x_0 + x_1)x_1}$$

$$\frac{1}{(x_0 + x_1 + x_2)(x_0 + x_2)x_2}$$

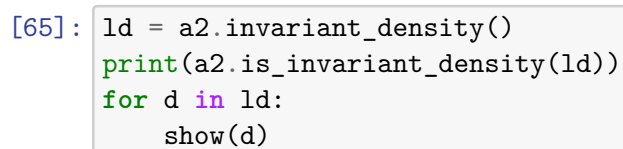
$$\frac{1}{(x_0 + x_1 + x_2)(x_1 + x_2)x_1}$$

7.0.2 Make an extension of the extension:

```
[64]: # flip-flop
lm = a.alphabet
AS = □
    ↪ [[(3,lm[0],0),(3,lm[1],1),(3,lm[2],2),(3,lm[4],4),(3,lm[5],5)],[(3,lm[3],3)]]
a2 = a.flip_flop(AS)
cc = a2.strongly_connected_components()
print(cc)
a2 = a2.sub_automaton(max(cc, key=lambda x:len(x)))
a2.plot()
```

dict_values([[0, 2, 5, 7, 11], [1, 3, 4, 6, 8, 9, 10]])

[64]:


$$\frac{1}{(x_0 + x_1 + x_2)(x_0 + x_2)x_2}$$

$$\frac{1}{(x_0 + x_1 + x_2)(x_1 + x_2)x_2}$$

$$\frac{1}{(2x_0 + 3x_1 + x_2)(x_0 + 2x_1)x_1}$$

$$\frac{1}{(x_0 + x_1 + x_2)(x_0 + x_2)x_0}$$

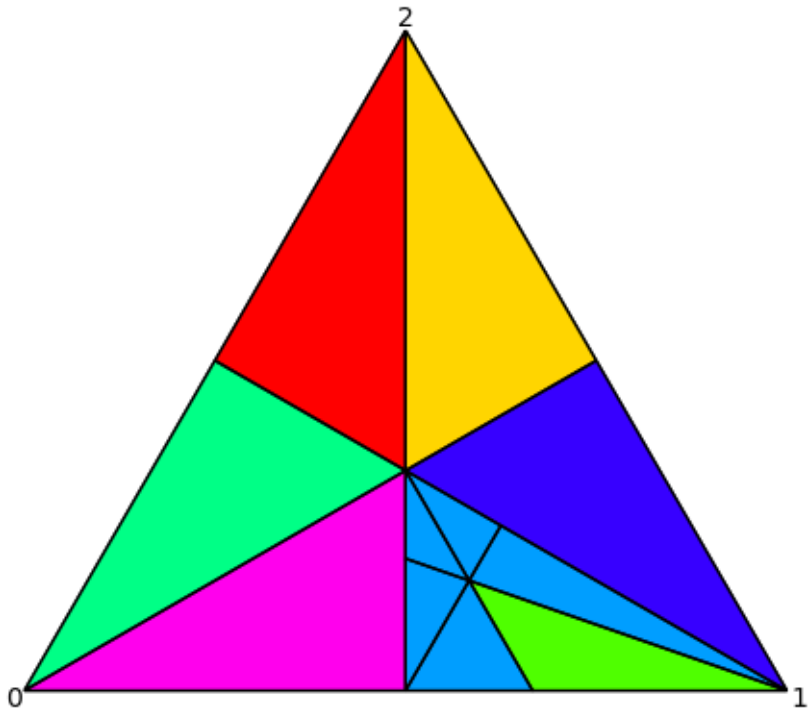
$$\frac{1}{(2x_0 + 3x_1 + x_2)(2x_0 + 2x_1 + x_2)(x_0 + x_1 + x_2)} + \frac{1}{(2x_0 + 3x_1 + x_2)(x_0 + 2x_1 + x_2)(x_0 + x_1 + x_2)} + \frac{1}{(2x_0 + 3x_1 + x_2)(x_0 + x_1 + x_2)x_2}$$

$$\frac{1}{(x_0 + x_1 + x_2)(x_1 + x_2)x_1}$$

$$\frac{1}{(x_0 + x_1 + x_2)(x_0 + x_1)x_0}$$

[66]: `a2.plot_domains()`

[66]:



```
[67]: for d in a2.domains_polyhedra():
      show(d)
```

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 1 & 0 & 2 \\ 2 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 0 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 2 & 1 & 1 \\ 3 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 2 & 0 & 1 \\ 3 & 1 & 2 \\ 1 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 2 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 1 & 2 & 2 \\ 1 & 3 & 2 \\ 0 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3$$

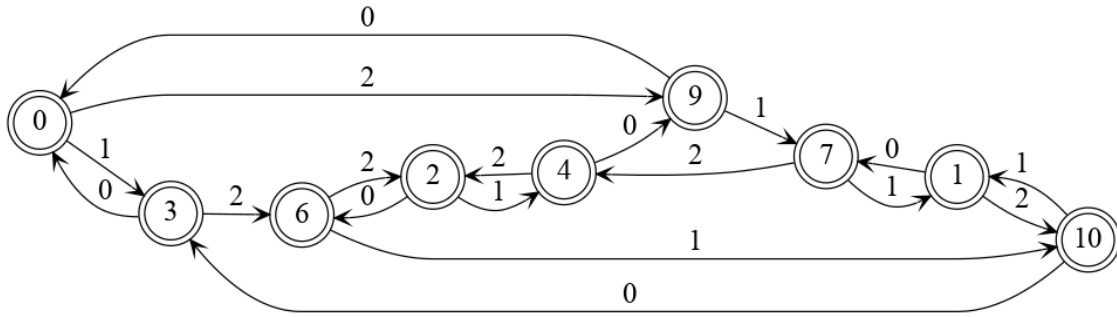
$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \mathbb{R}_+^3$$

8 Brun as a win-lose graph

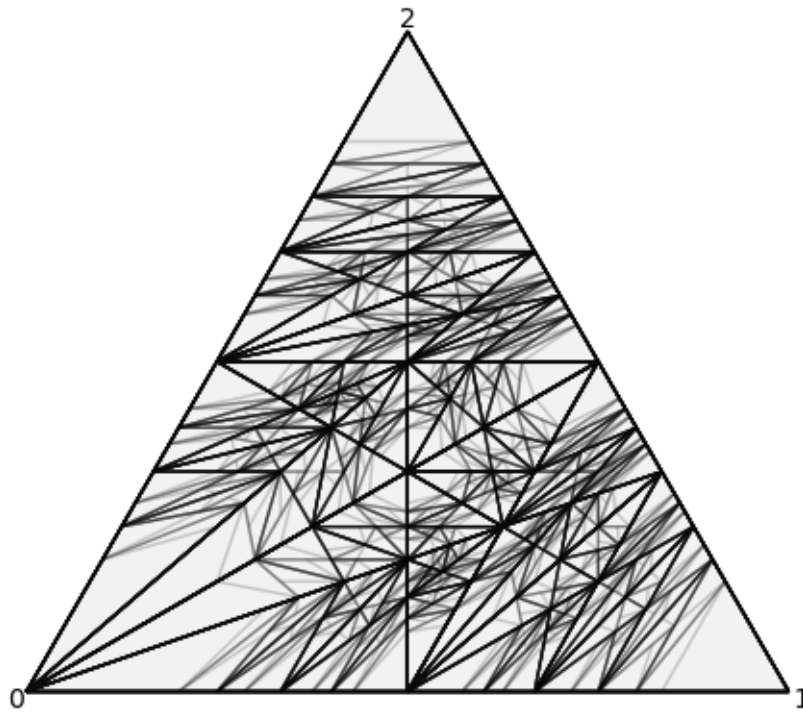
```
[76]: # Brun as win-lose graph
a = dag.Brun().winlose().minimize_winlose() # convert it to a win-lose graph
a = a.sub_automaton(max(a.strongly_connected_components(), key=lambda x:
    ↪len(x))) # keep only the main strongly connected component
a.plot()
```

[76]:



```
[77]: a.plot_algo(0,10)
```

[77]:



```
[78]: ld = a.invariant_density()
print(a.is_invariant_density(ld))
for d in ld:
    show(d)
```

True

$$\frac{1}{(3x_0 + x_1 + x_2)(x_0 + x_1)(x_0 + x_2)}$$

$$\frac{1}{(x_0 + 3x_1 + x_2)(x_0 + x_1)(x_1 + x_2)}$$

$$\frac{1}{(x_0 + x_1 + 3x_2)(x_0 + x_2)(x_1 + x_2)}$$

$$\frac{1}{(2x_0 + x_1 + x_2)(x_0 + x_1 + x_2)(x_0 + x_1)}$$

$$\frac{1}{(x_0 + x_1 + 2x_2)(x_0 + x_1 + x_2)(x_1 + x_2)}$$

$$\frac{1}{(x_0 + x_1 + 2x_2)(x_0 + x_1 + x_2)(x_0 + x_2)}$$

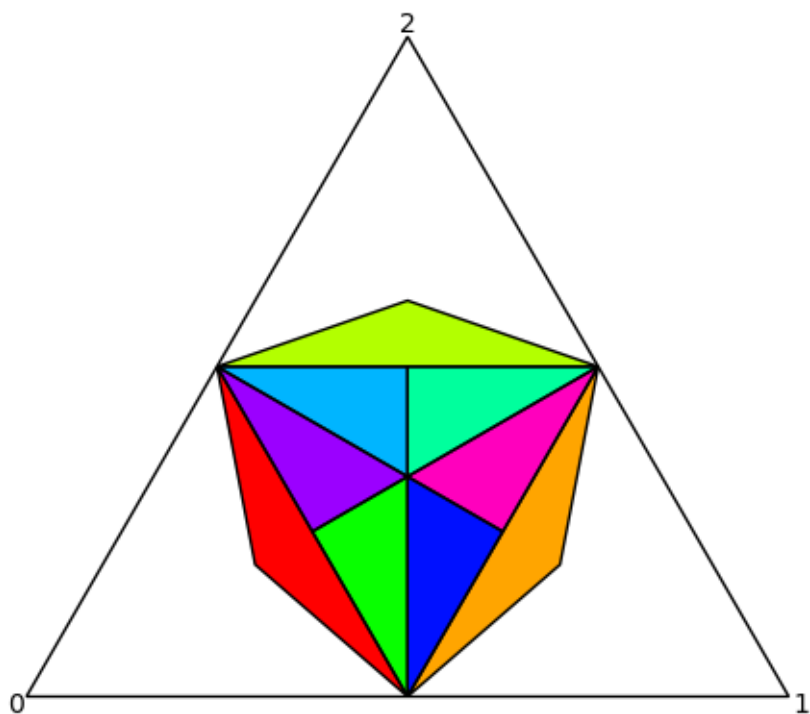
$$\frac{1}{(x_0 + 2x_1 + x_2)(x_0 + x_1 + x_2)(x_0 + x_1)}$$

$$\frac{1}{(2x_0 + x_1 + x_2)(x_0 + x_1 + x_2)(x_0 + x_2)}$$

$$\frac{1}{(x_0 + 2x_1 + x_2)(x_0 + x_1 + x_2)(x_1 + x_2)}$$

```
[79]: a.plot_domains()
```

[79]:



9 Generalization of Cassaigne algorithm

```
[80]: # generalization of Cassaigne algorithm to higher dimension
a = dag.Cassaigne(4)
a
```

[80]: DetAutomaton with 1 state and an alphabet of 2 letters

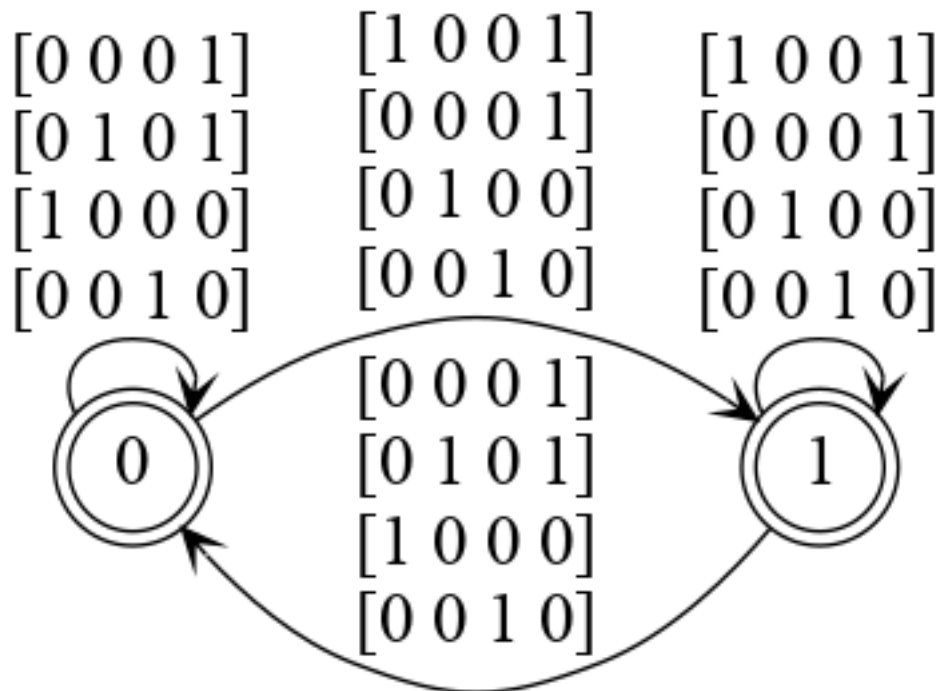
```
[81]: ld = a.invariant_density()
print(a.is_invariant_density(ld))
show(ld)
```

True

$$\left[\frac{1}{(x_0 + x_1 + x_2 + 2x_3)(x_0 + x_1 + x_2 + x_3)(x_0 + x_2 + x_3)(x_1 + x_2 + x_3)} \right]$$

```
[82]: # extension
lm = a.alphabet
a = DetAutomaton([(i, lm[j], j) for i in range(len(lm)) for j in range(len(lm))],
    ↪avoidDiGraph=True)
a.plot()
```

[82]:



```
[83]: ld = a.invariant_density()
print(a.is_invariant_density(ld))
for d in ld:
    show(d)
```

True

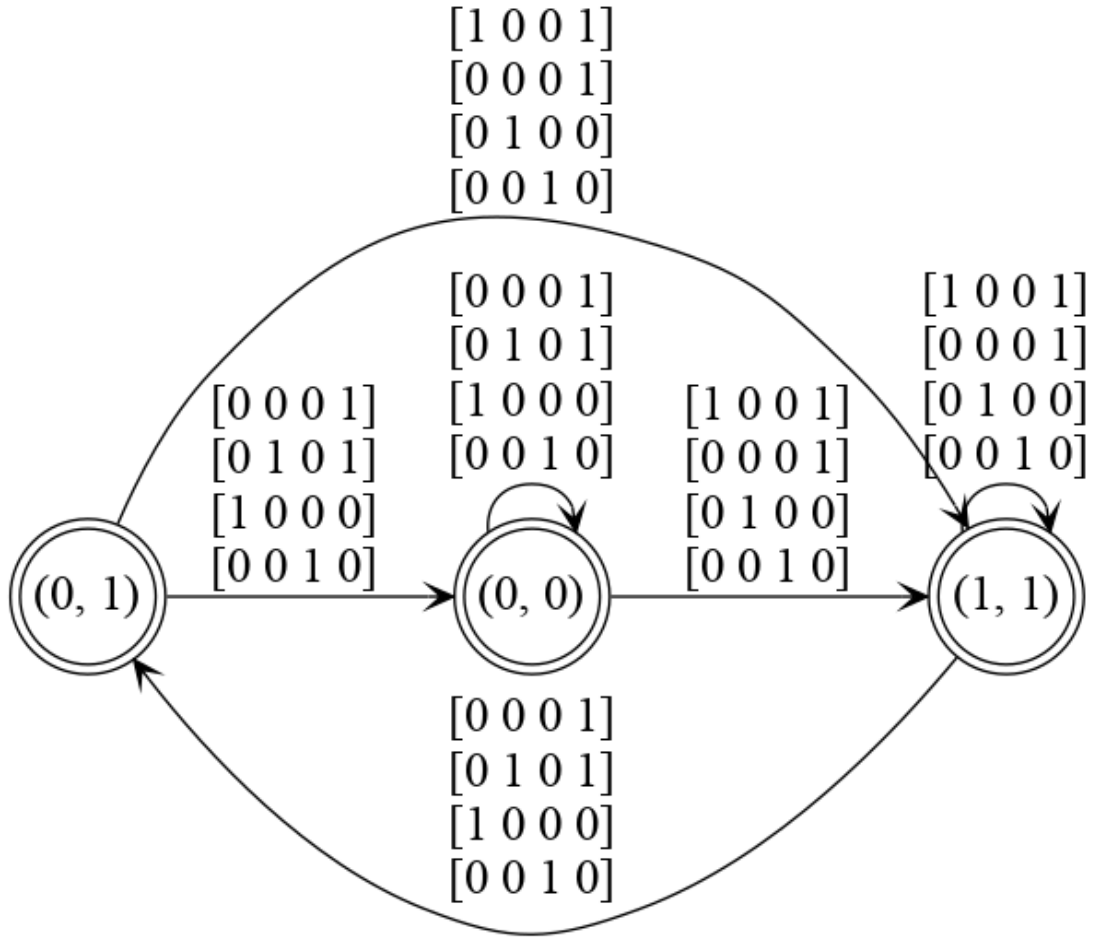
$$\frac{1}{(x_0 + x_1 + 2x_2 + 2x_3)(x_0 + x_1 + x_2 + 2x_3)(x_0 + x_1 + x_2 + x_3)(x_0 + x_2 + x_3)}$$

$$\frac{1}{(x_0 + x_1 + 2x_2 + 2x_3)(x_0 + x_1 + x_2 + 2x_3)(x_0 + x_1 + x_2 + x_3)(x_1 + x_2 + x_3)}$$

```
[84]: # flip-flop
AS = [[(0, lm[0], 0)], [(0, lm[1], 1)]]
a2 = a.flip_flop(AS)
cc = a2.strongly_connected_components()
print(cc)
a2 = a2.sub_automaton(max(cc, key=lambda x: len(x)))
a2.plot()
```

dict_values([[0], [1, 2, 3]])

[84]:



```
[85]: ld = a2.invariant_density()
print(a2.is_invariant_density(ld))
for d in ld:
    show(d)
```

True

$$\frac{1}{(2x_0 + x_1 + 2x_2 + 2x_3)(x_0 + x_1 + 2x_2 + 2x_3)(x_0 + x_1 + x_2 + 2x_3)(x_0 + x_1 + x_2 + x_3)}$$

$$\frac{1}{(x_0 + x_1 + 2x_2 + 2x_3)(x_0 + x_1 + x_2 + 2x_3)(x_0 + x_1 + x_2 + x_3)(x_1 + x_2 + x_3)}$$

$$\frac{1}{(2x_0 + x_1 + 2x_2 + 2x_3)(x_0 + x_1 + 2x_2 + 2x_3)(x_0 + x_1 + x_2 + 2x_3)(x_0 + x_2 + x_3)}$$

```
[86]: ls = a2.domains_polyhedra()
      from badic.density import formulae_to_matrices
      ls = [formulae_to_matrices(d) for d in ls]
      g = plot_tri(identity_matrix(4))
      for l,color in zip(ls, ["red", "orange","yellow","green", "cyan", "blue",
      ↪ "purple", "magenta"]):
          for m,_ in l:
              g += plot_tri(m, fill=True, color=color, text=False)
      g
```

[86]: Graphics3d Object

```
[87]: show(a2.domains_polyhedra())
```

$$\left[\begin{pmatrix} 1 & 1 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 2 & 2 & 2 \end{pmatrix} \mathbb{R}_+^4, \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 \\ 1 & 1 & 2 & 2 \end{pmatrix} \mathbb{R}_+^4, \begin{pmatrix} 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 2 & 1 & 2 & 2 \end{pmatrix} \mathbb{R}_+^4 \right]$$

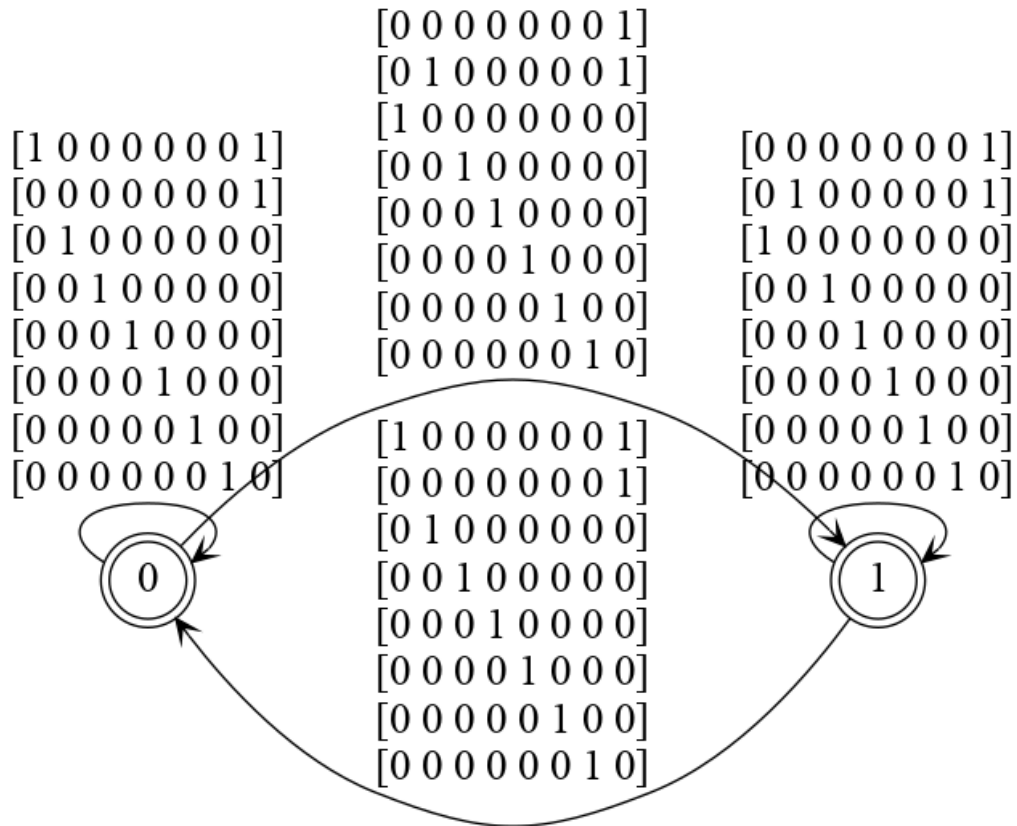
10 Cassaigne with 8 coordinates

```
[88]: # generalization of Cassaigne algorithm to higher dimension
      a = dag.Cassaigne(8)
      a
```

[88]: DetAutomaton with 1 state and an alphabet of 2 letters

```
[89]: # extension
      lm = a.alphabet
      a = DetAutomaton([(i,lm[j],j) for i in range(len(lm)) for j in range(len(lm))],
      ↪ avoidDiGraph=True)
      a.plot()
```

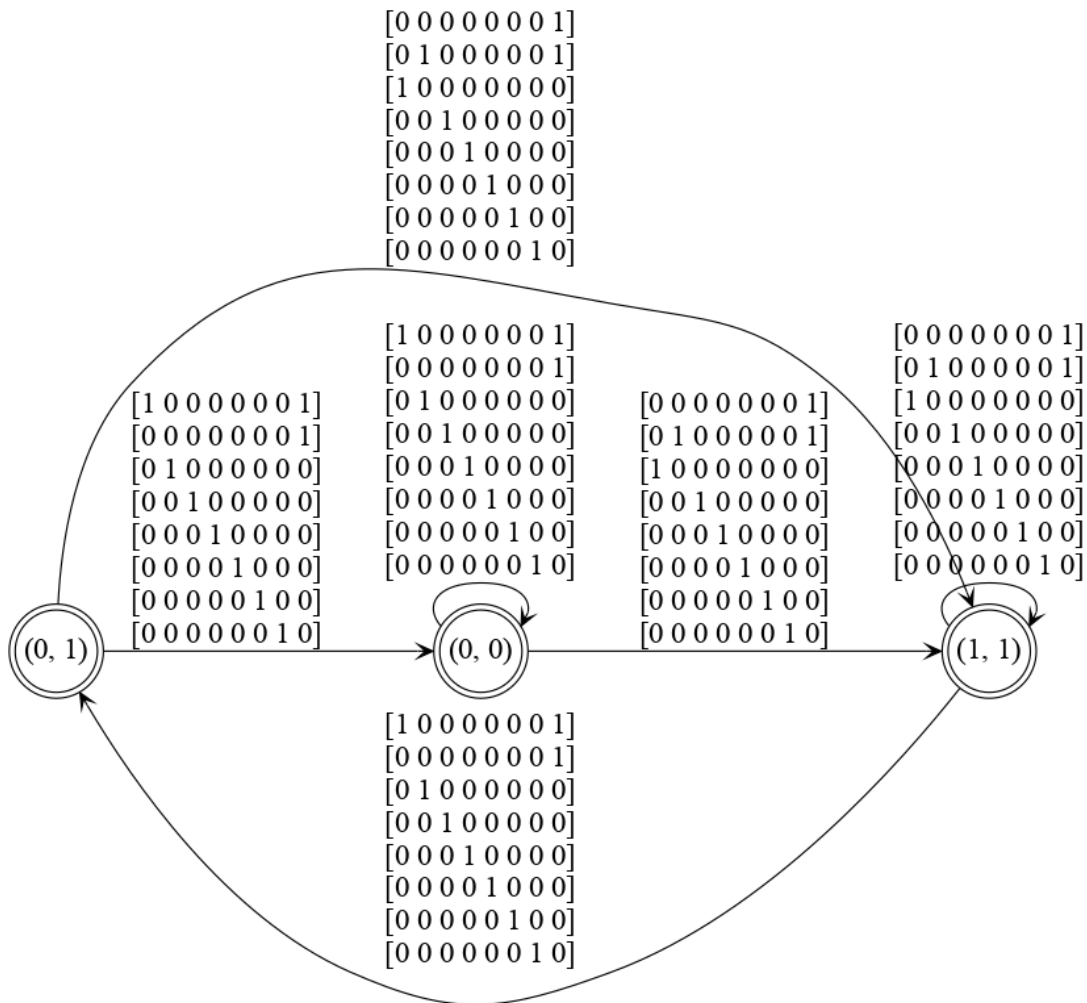
[89]:



```
[90]: # flip-flop
AS = [[(0,lm[0],0)],[(0,lm[1],1)]]
a2 = a.flip_flop(AS)
cc = a2.strongly_connected_components()
print(cc)
a2 = a2.sub_automaton(max(cc, key=lambda x:len(x)))
a2.plot()
```

```
dict_values([[0], [1, 2, 3]])
```

```
[90]:
```



```
[91]: ld = a2.invariant_density(check=False)
      print(a2.is_invariant_density(ld))
      ld
```

True

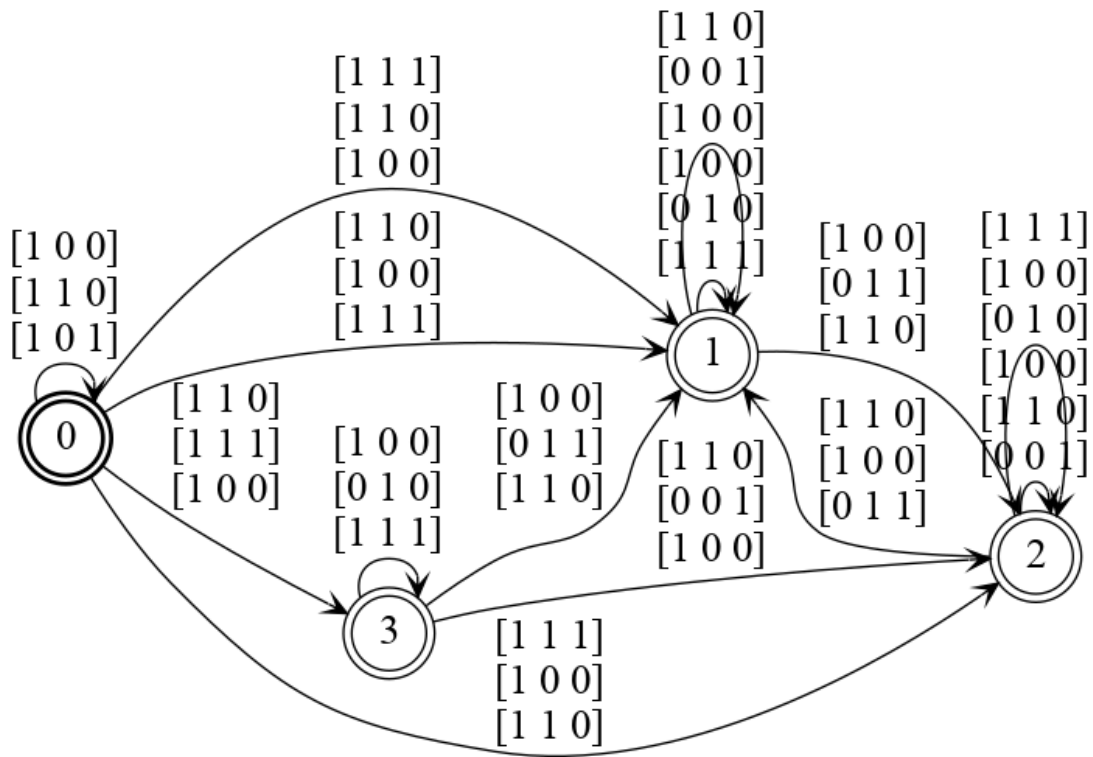
```
[91]: [1/((x0 + 2*x1 + 2*x2 + 2*x3 + 2*x4 + 2*x5 + 2*x6 + 2*x7)*(x0 + x1 + 2*x2 + 2*x3 +
+ 2*x4 + 2*x5 + 2*x6 + 2*x7)*(x0 + x1 + x2 + 2*x3 + 2*x4 + 2*x5 + 2*x6 +
+ 2*x7)*(x0 + x1 + x2 + x3 + 2*x4 + 2*x5 + 2*x6 + 2*x7)*(x0 + x1 + x2 + x3 + x4 +
+ 2*x5 + 2*x6 + 2*x7)*(x0 + x1 + x2 + x3 + x4 + x5 + 2*x6 + 2*x7)*(x0 + x1 + x2 +
+ x3 + x4 + x5 + x6 + 2*x7)*(x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7)),
      1/((x0 + x1 + 2*x2 + 2*x3 + 2*x4 + 2*x5 + 2*x6 + 2*x7)*(x0 + x1 + x2 + 2*x3 +
+ 2*x4 + 2*x5 + 2*x6 + 2*x7)*(x0 + x1 + x2 + x3 + 2*x4 + 2*x5 + 2*x6 + 2*x7)*(x0 +
+ x1 + x2 + x3 + x4 + 2*x5 + 2*x6 + 2*x7)*(x0 + x1 + x2 + x3 + x4 + x5 + 2*x6 +
+ 2*x7)*(x0 + x1 + x2 + x3 + x4 + x5 + x6 + 2*x7)*(x0 + x1 + x2 + x3 + x4 + x5 +
+ x6 + x7)*(x0 + x2 + x3 + x4 + x5 + x6 + x7)),
```

$$1/((x_0 + 2x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 + 2x_6 + 2x_7)(x_0 + x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 + 2x_6 + 2x_7)(x_0 + x_1 + x_2 + 2x_3 + 2x_4 + 2x_5 + 2x_6 + 2x_7)(x_0 + x_1 + x_2 + x_3 + 2x_4 + 2x_5 + 2x_6 + 2x_7)(x_0 + x_1 + x_2 + x_3 + x_4 + 2x_5 + 2x_6 + 2x_7)(x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + 2x_6 + 2x_7)(x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + 2x_7)(x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7))]$$

11 Jacobi-Perron

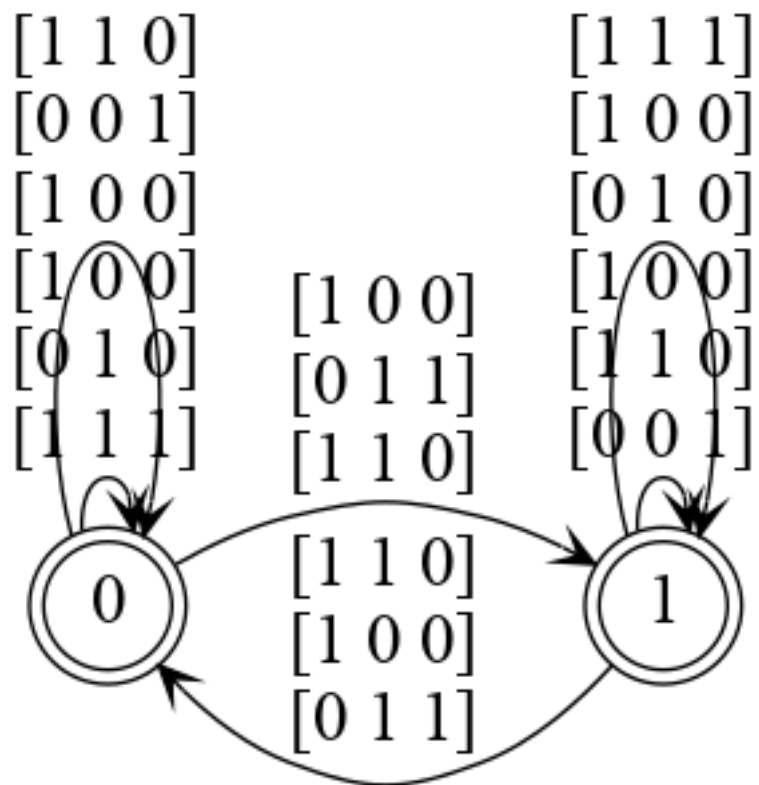
```
[94]: # Jacobi-Perron
a = dag.JacobiPerron().matrices_graph().minimize() # convert it to a matrices_
↳graph
a.plot()
```

[94]:



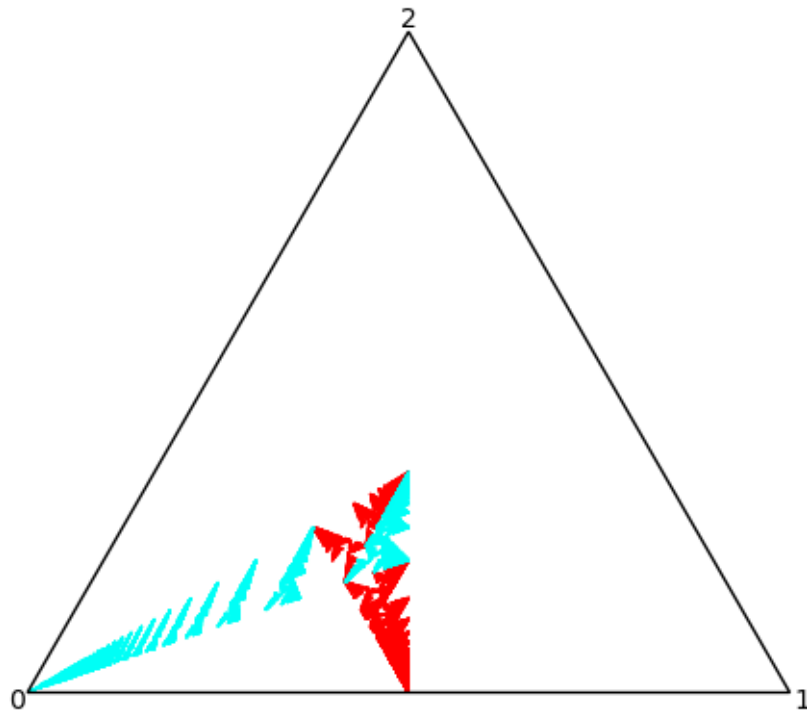
```
[97]: # keep only the interesting strongly connected component
am = a.sub_automaton(max(a.strongly_connected_components(), key=lambda x:
↳len(x)))
am.plot()
```

[97]:



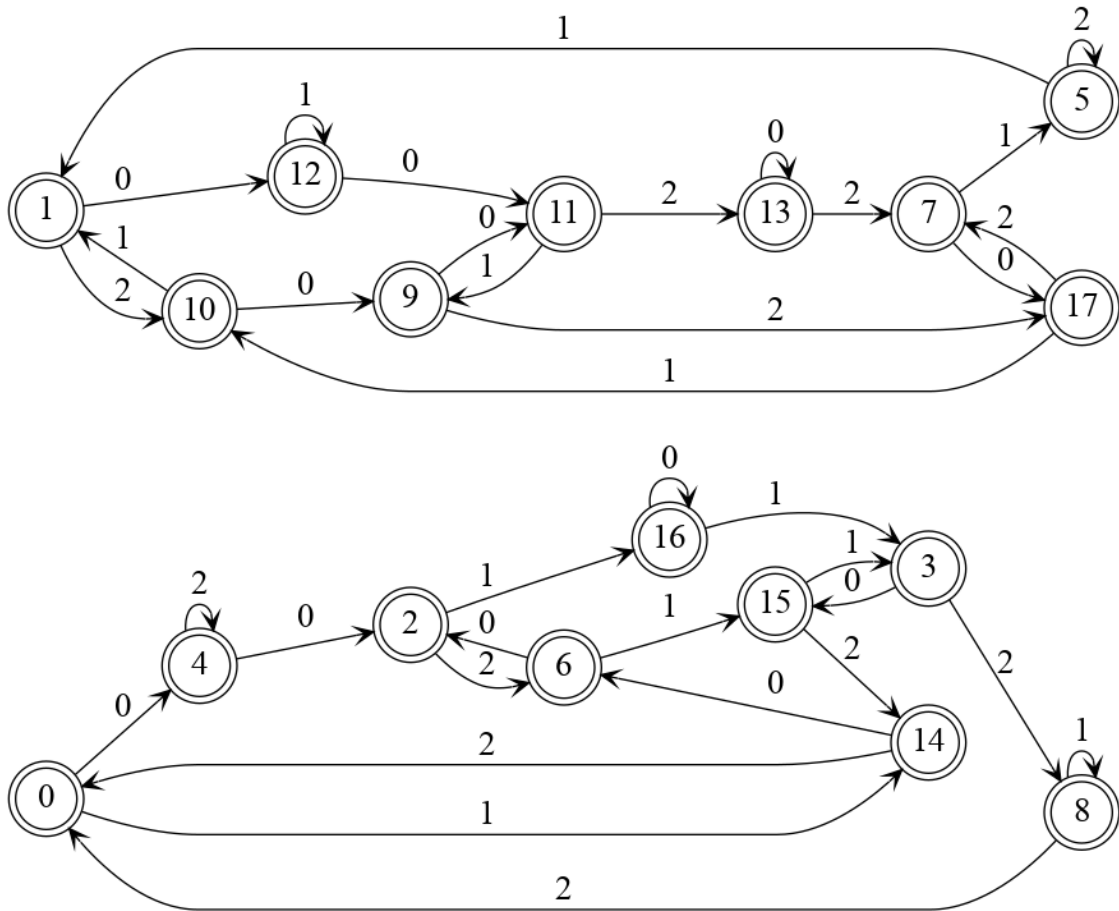
```
[98]: am.plot_domains()
```

```
[98]:
```



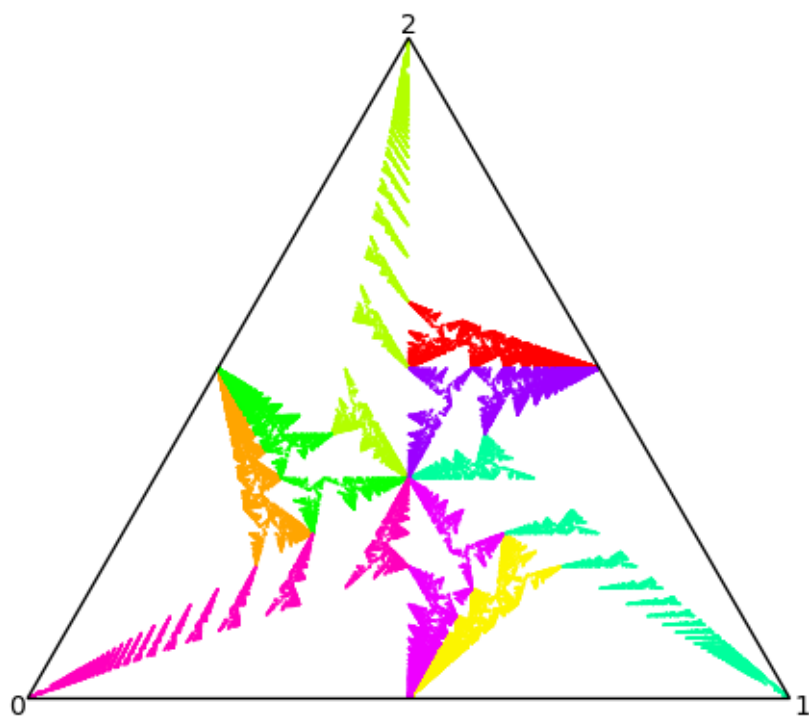
```
[99]: # convert it to a win-lose graph  
b = am.winlose().minimize_winlose()  
b.plot()
```

[99]:



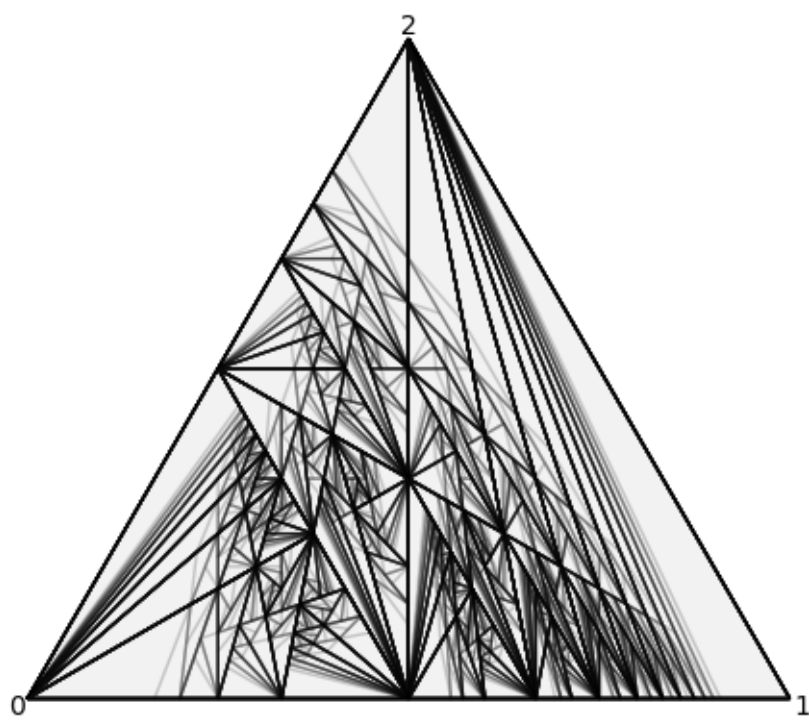
```
[100]: # plot domains of the win-lose graph
b.plot_domains()
```

[100]:



```
[101]: b.plot_algo(16,10)
```

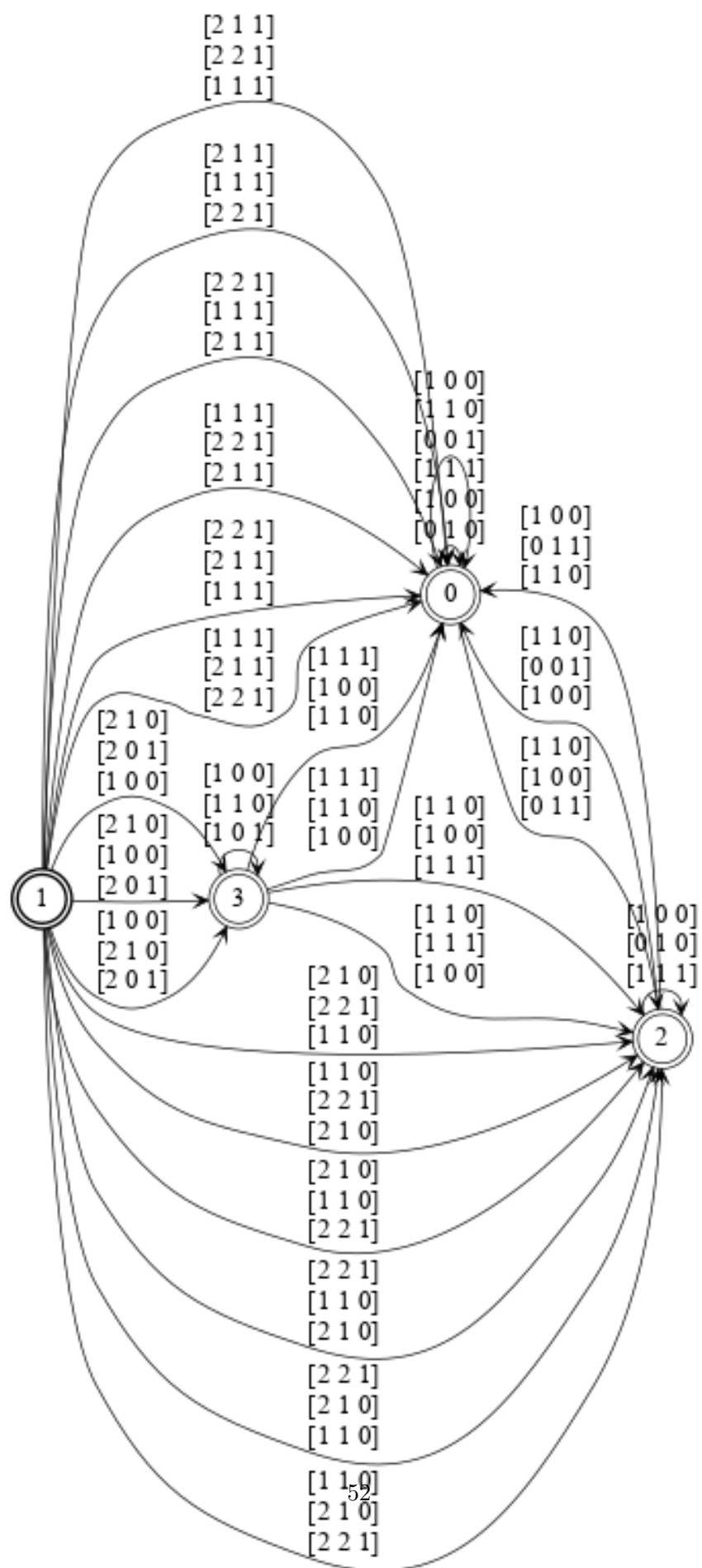
```
[101]:
```



12 Symmetric Jacobi-Perron

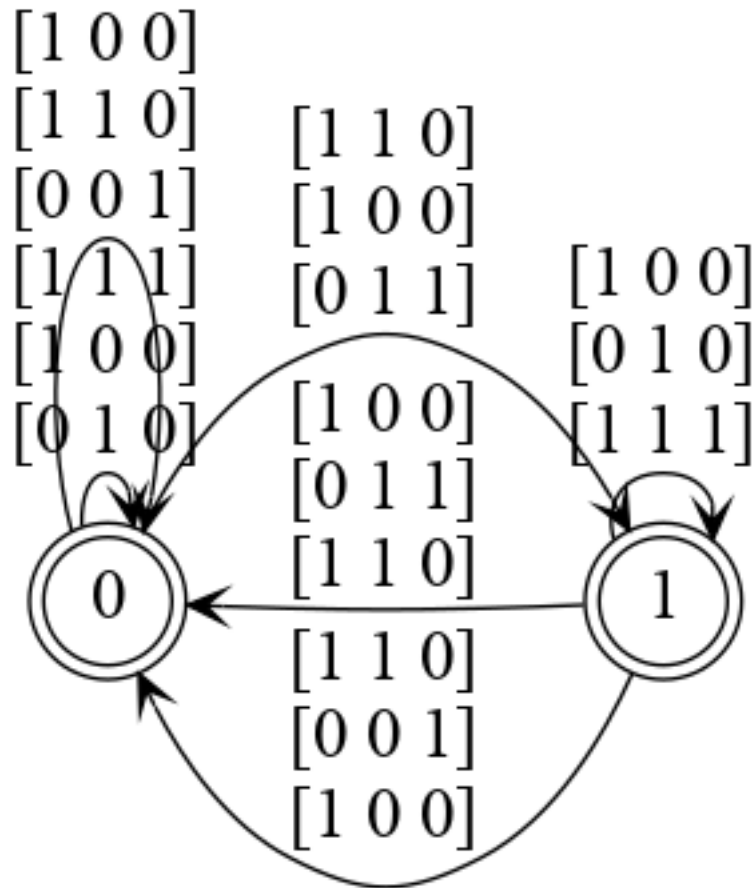
```
[102]: # Symmetric Jacobi-Perron
a = dag.SymmetricJacobiPerron().matrices_graph().minimize() # convert it to a
↪matrices graph
a.plot()
```

[102]:



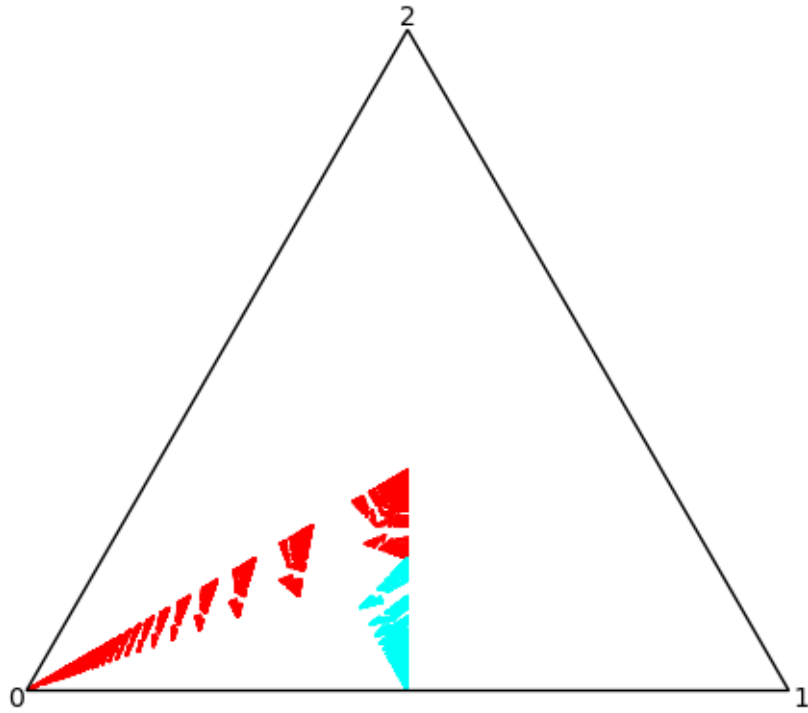
```
[103]: # keep the interesting strongly connected component
am = a.sub_automaton(max(a.strongly_connected_components(), key=lambda x:
↳ len(x)))
am.plot()
```

[103]:

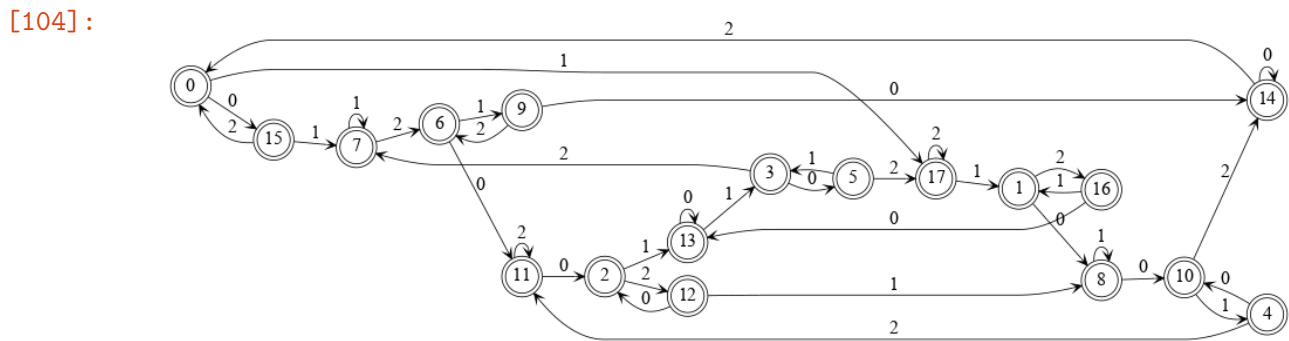


```
[91]: am.plot_domains()
```

[91]:

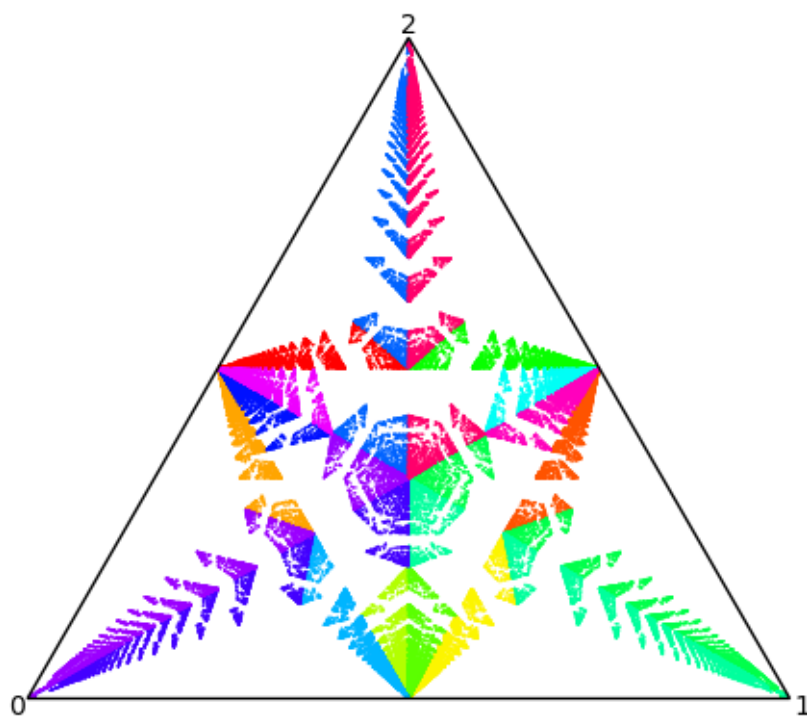


```
[104]: # convert it to a win-lose graph
b = am.winlose().minimize_winlose()
b.plot()
```



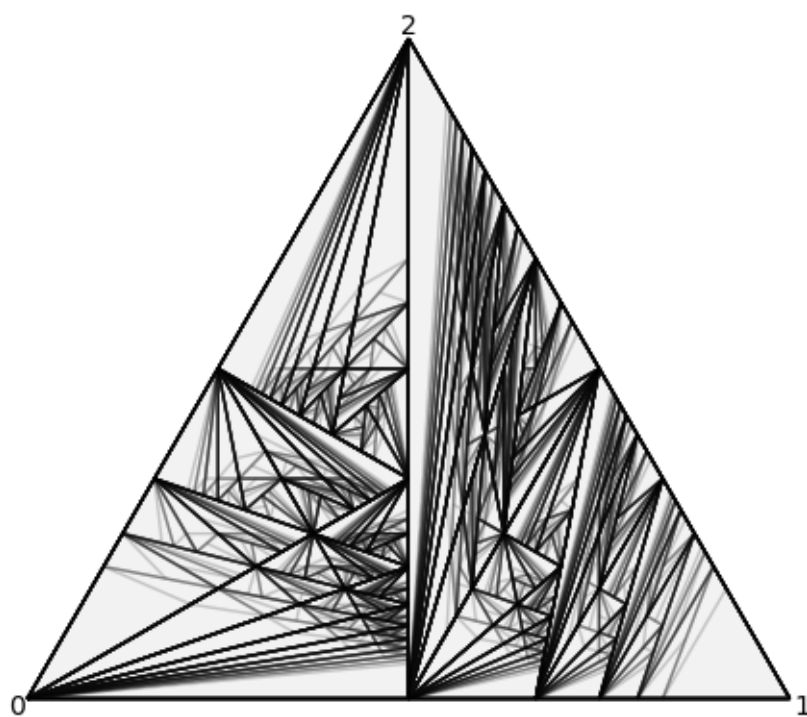
```
[93]: b.plot_domains()
```

[93]:



```
[95]: b.plot_algo(0,10)
```

```
[95]:
```



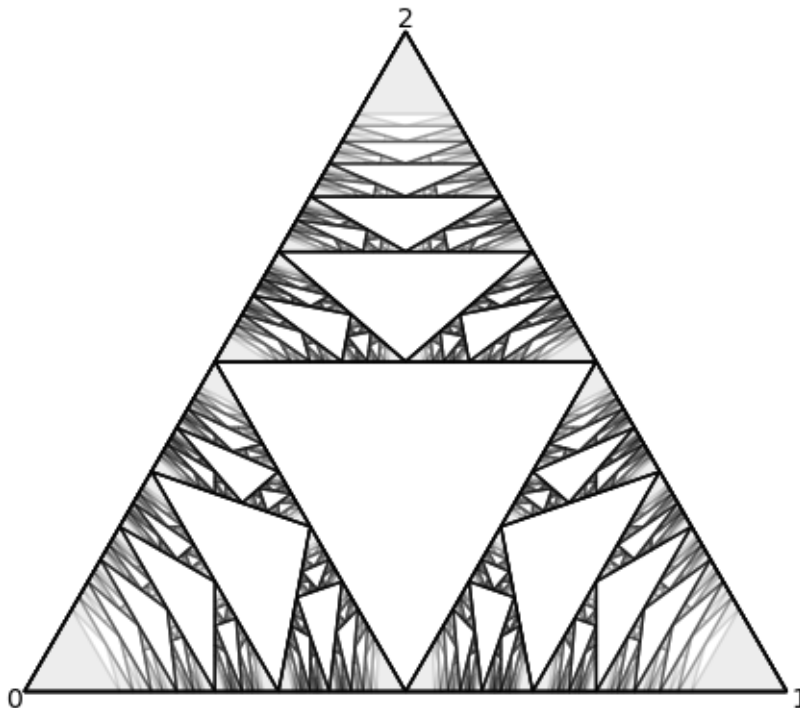
13 Arnoux-Rauzy

```
[105]: # Arnoux-Rauzy
a = dag.ArnouxRauzy()
a
```

[105]: DetAutomaton with 1 state and an alphabet of 3 letters

```
[106]: a.plot_algo(0,7)
```

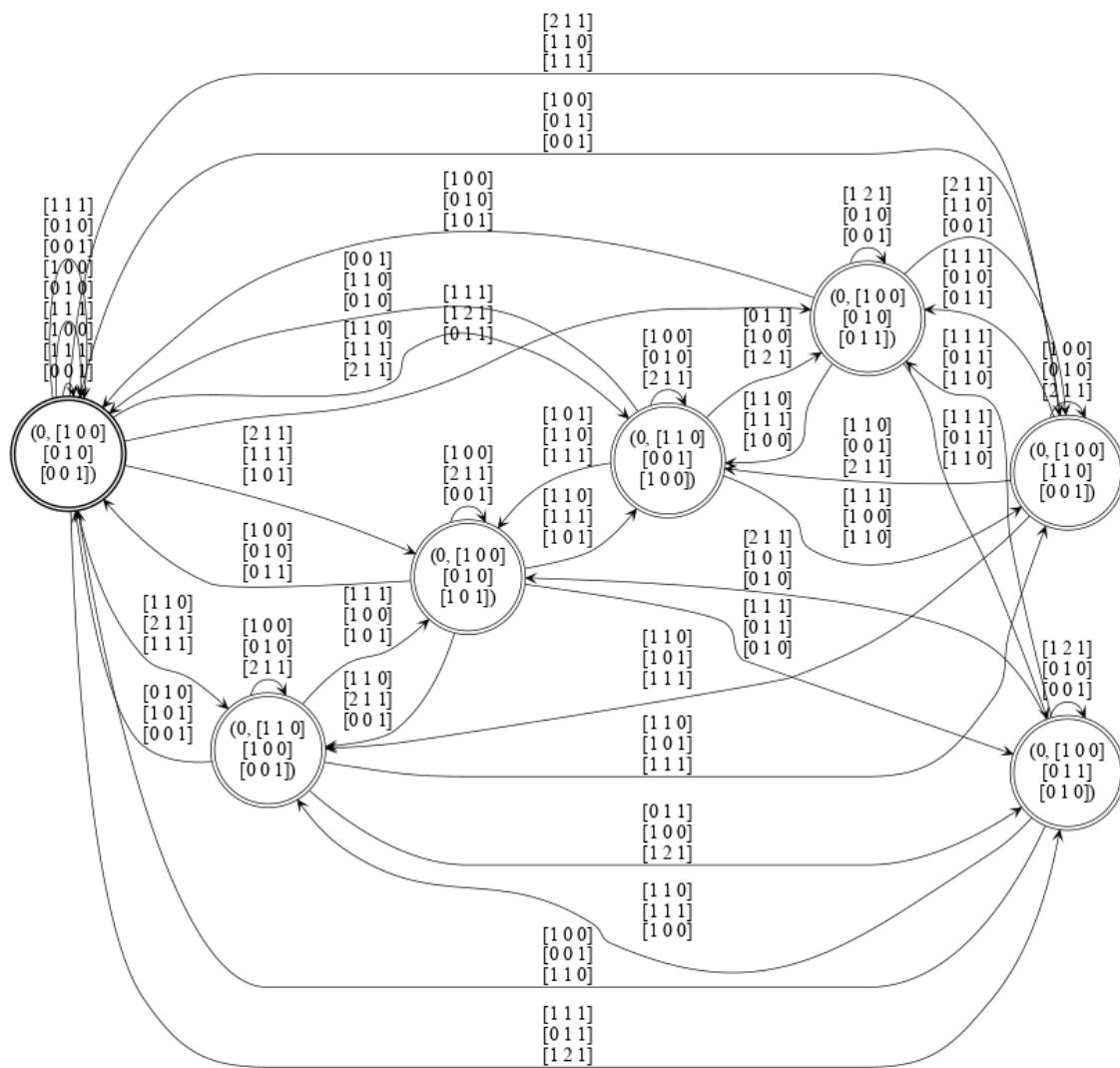
[106]:



14 Arnoux-Rauzy-Poincaré

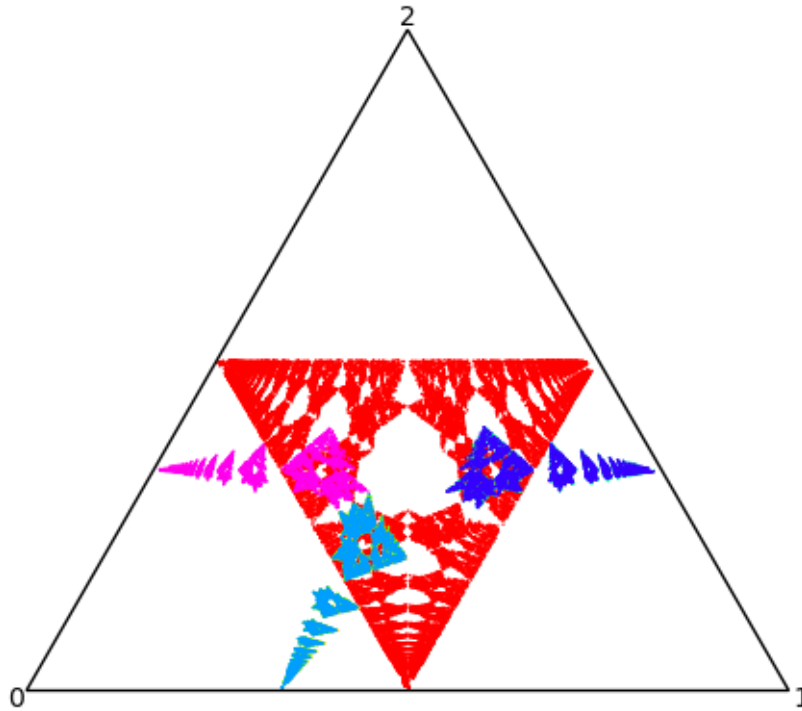
```
[107]: # Arnoux-Rauzy-Poincare
a = dag.ArnouxRauzyPoincare().matrices_graph() # convert it to a matrices graph
a.plot()
```

[107]:



```
[154]: a.plot_domains()
```

```
[154]:
```



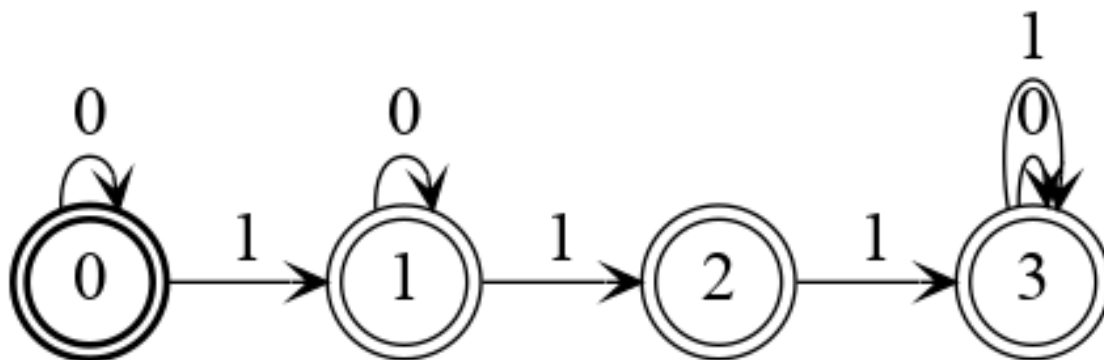
```
[108]: # rational languages describing the domains
a.domains_languages()
```

```
[108]: [DetAutomaton with 7 states and an alphabet of 32 letters,
DetAutomaton with 7 states and an alphabet of 32 letters,
DetAutomaton with 7 states and an alphabet of 32 letters,
DetAutomaton with 7 states and an alphabet of 32 letters,
DetAutomaton with 7 states and an alphabet of 32 letters,
DetAutomaton with 7 states and an alphabet of 32 letters,
DetAutomaton with 7 states and an alphabet of 32 letters]
```

15 Example with a complicated domain

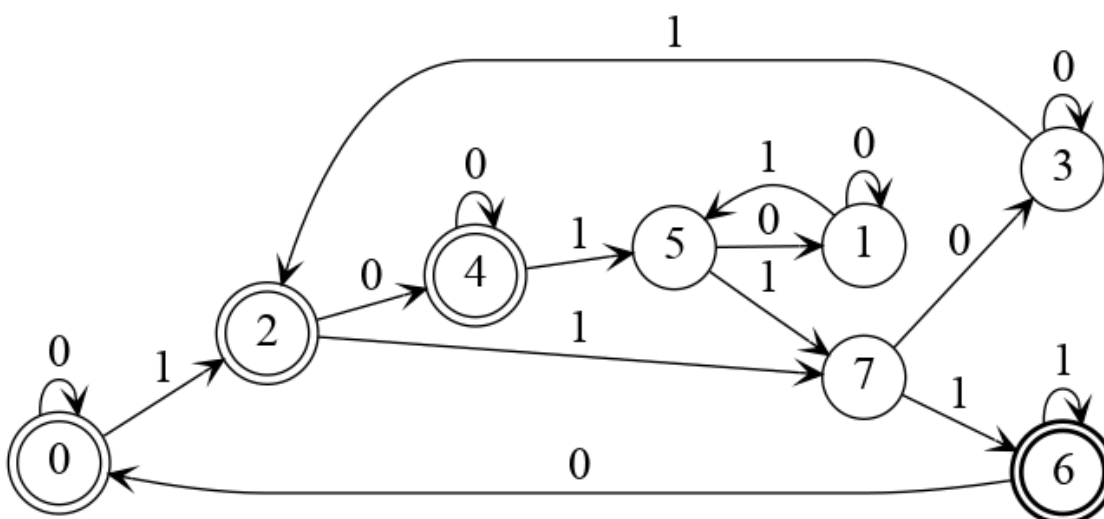
```
[109]: from badic import *
a = DetAutomaton([(0,0,0),(0,1,1),(1,0,1),(1,1,2),(2,1,3),(3,0,3),(3,1,3)],
↳ i=0, avoidDiGraph=True)
a.plot()
```

```
[109]:
```



```
[110]: am = a.mirror().determinize().minimize()
am.plot()
```

[110]:



```
[3]: am.plot_domains()
```

[3]:



```
[111]: # description of domains as unions of simplices
for d in am.domains_polyhedra():
    show(d)
```

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \bigsqcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix} \mathbb{R}_+^2$$

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \bigsqcup_{i \in \mathbb{N}} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^i \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \bigsqcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \mathbb{R}_+^2$$

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \bigsqcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \mathbb{R}_+^2$$

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \bigsqcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix} \mathbb{R}_+^2$$

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \bigsqcup_{i \in \mathbb{N}} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^i \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \bigsqcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \mathbb{R}_+^2$$

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \bigsqcup_{j \in \mathbb{N}} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \mathbb{R}_+^2$$

$$\begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix} \mathbb{R}_+^2$$

$$\begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix} \mathbb{R}_+^2$$

```
[112]: # invariant density
for d in am.invariant_density():
    show(d)
```

$$\sum_{j \in \mathbb{N}} \frac{1}{((3j+4)x_0 + 3x_1)((j+1)x_0 + x_1)}$$

$$\sum_{j, i \in \mathbb{N}} \frac{1}{((i(2j+5) + 4j+8)x_0 + (2j+5)x_1)((i(j+3) + 2j+5)x_0 + (j+3)x_1)}$$

$$\sum_{j \in \mathbb{N}} \frac{1}{((2j+3)x_0 + (2j+5)x_1)((j+1)x_0 + (j+2)x_1)}$$

$$\sum_{j \in \mathbb{N}} \frac{1}{((3j+4)x_0 + 3x_1)((2j+3)x_0 + 2x_1)}$$

$$\sum_{j,i \in \mathbb{N}} \frac{1}{((i(2j+5) + 4j+8)x_0 + (2j+5)x_1)((i(j+2) + 2j+3)x_0 + (j+2)x_1)}$$

$$\sum_{j \in \mathbb{N}} \frac{1}{((2j+3)x_0 + (2j+5)x_1)((j+2)x_0 + (j+3)x_1)}$$

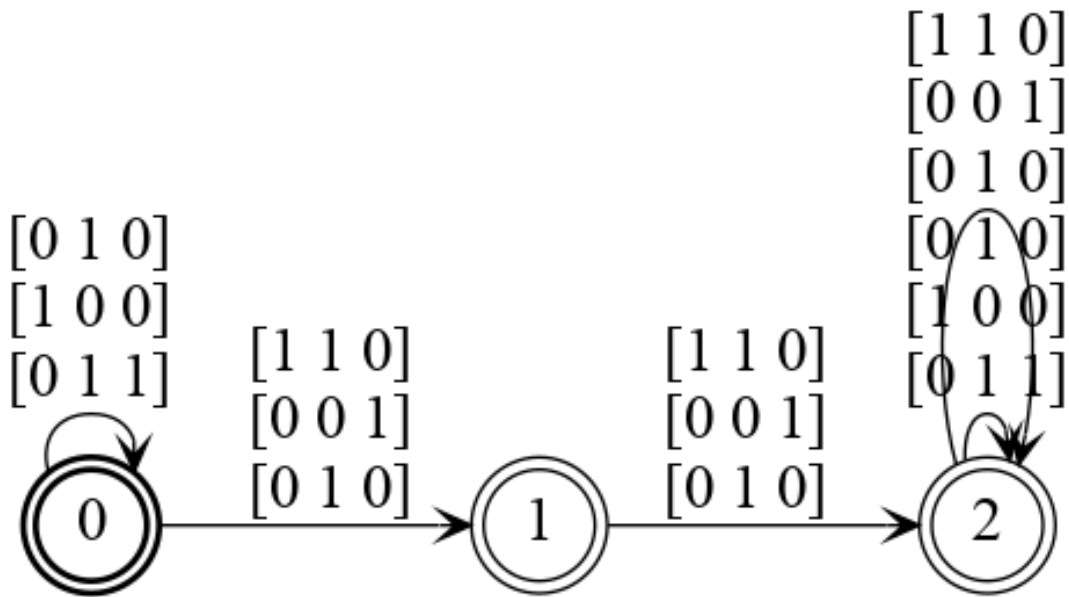
$$\frac{1}{(x_0 + 3x_1)x_1}$$

$$\frac{1}{(x_0 + 3x_1)(x_0 + 2x_1)}$$

16 Other examples with complicated domains

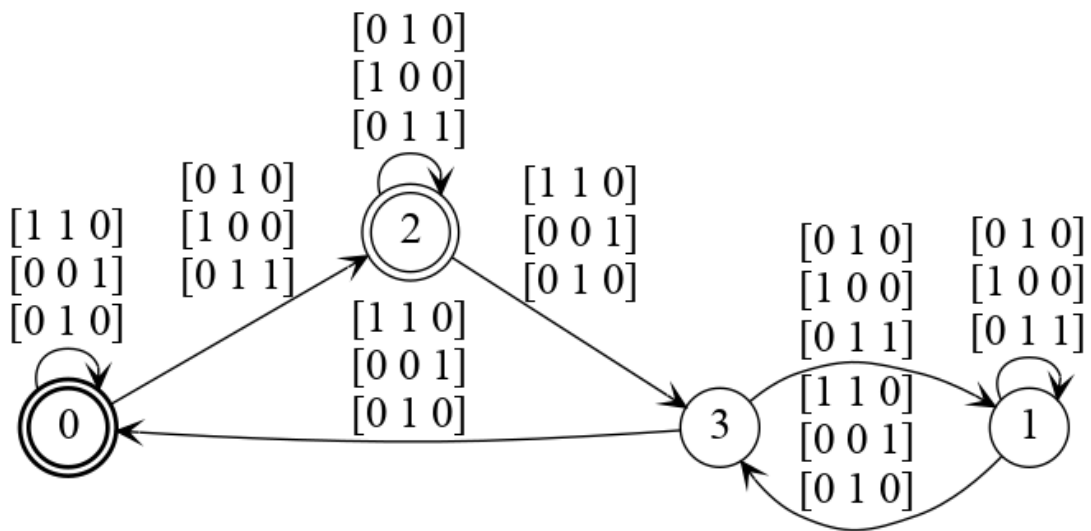
```
[113]: # describe a set with an accumulation points from the Cassaigne algo
from badic import *
a = dag.Cassaigne()
A = a.alphabet
a = DetAutomaton([(0,A[0],0),(0,A[1],1),(1,A[1],2),(2,A[0],2),(2,A[1],2)], i=0,
    ↪avoidDiGraph=True)
a.plot()
```

[113]:

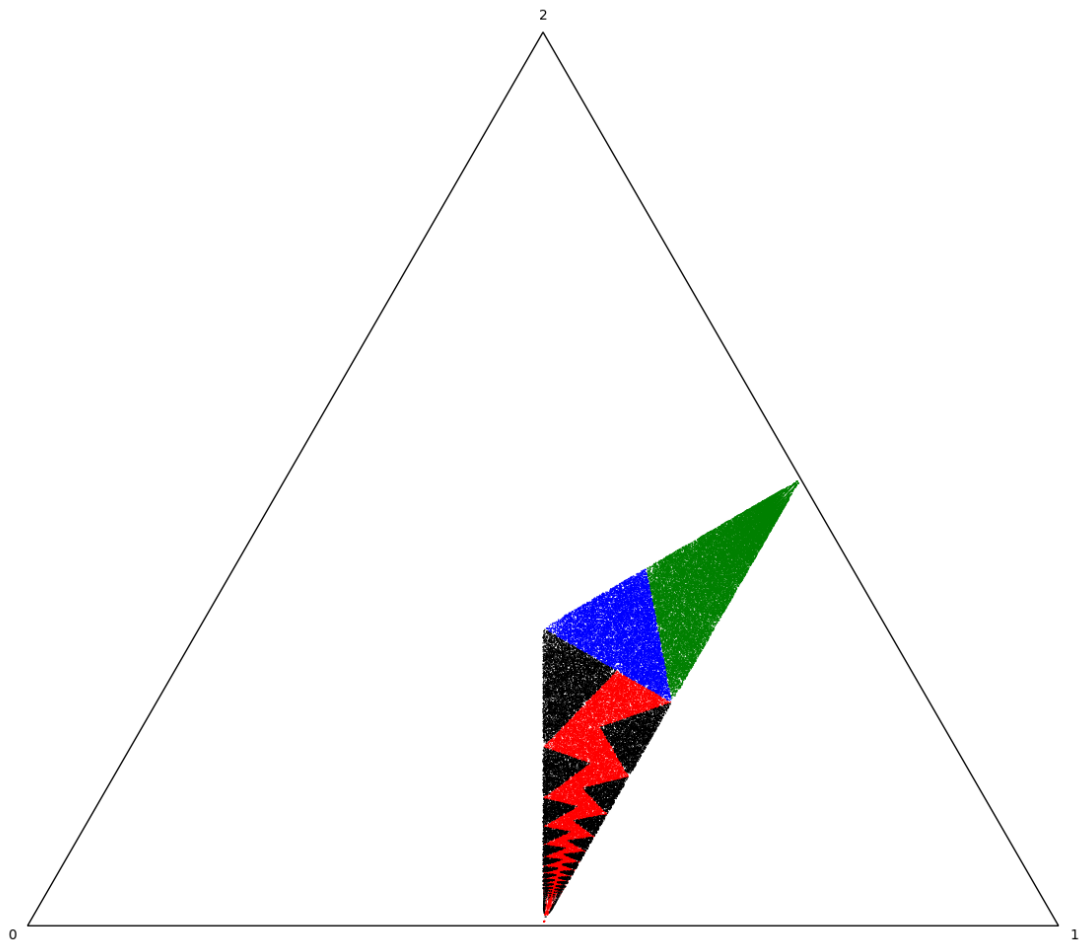


```
[114]: am = a.mirror().determinize().minimize()
am.plot()
```

[114]:



```
[10]: am.plot_domains(colors=["green","red","black","blue"]).show(figsize=15)
```



```
[115]: for d in am.domains_polyhedra():
        show(d)
```

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \bigcup_{j \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \bigcup_{j \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}^j \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

```
[116]: for d in am.invariant_density():
        show(d)
```

$$\frac{1}{(x_0 + 2x_1 + 2x_2)(x_0 + 2x_1 + x_2)(x_1 + x_2)}$$

$$\sum_{j \in \mathbb{N}} - \frac{16}{\left(((-1)^j + 2j + 7)x_0 - ((-1)^j - 2j - 9)x_1 + 4x_2 \right) \left(((-1)^j - 2j - 5)x_0 - ((-1)^j + 2j + 7)x_1 - 4x_2 \right) ((j + 2))}$$

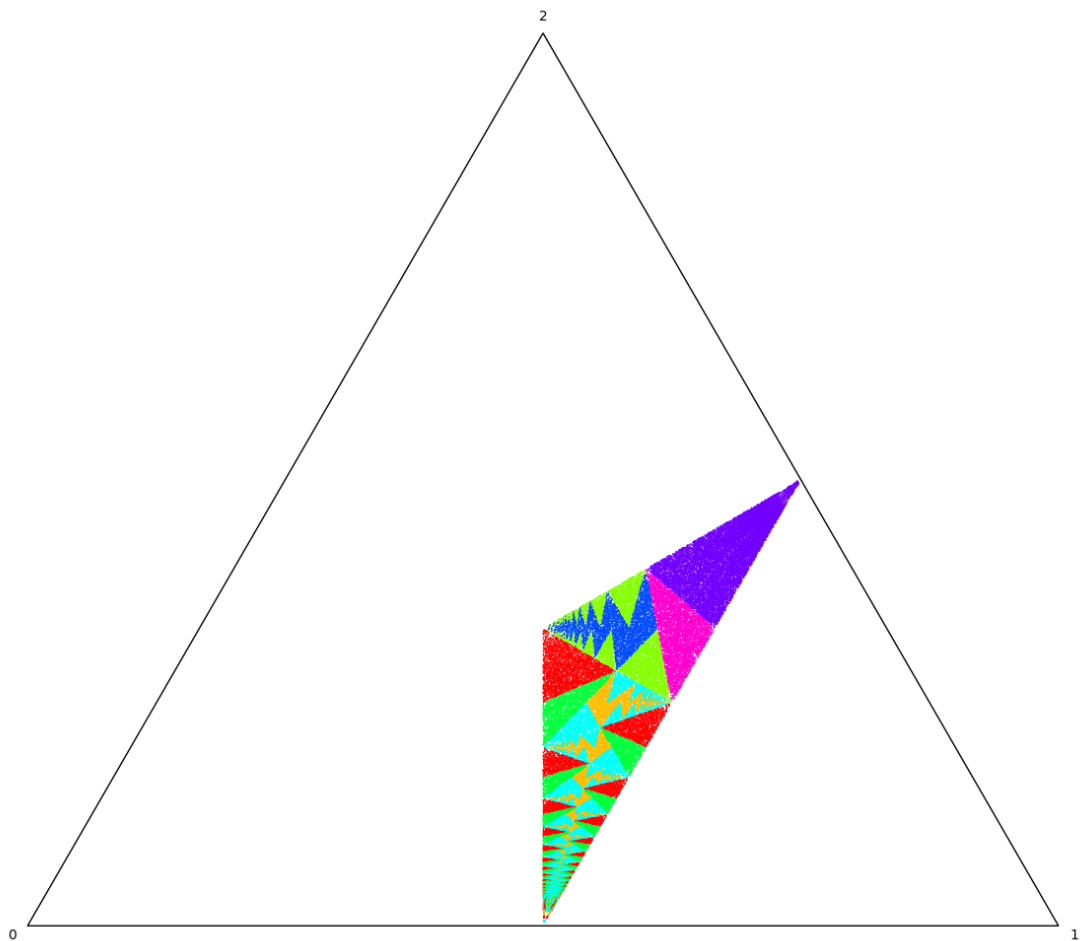
$$\sum_{j \in \mathbb{N}} \frac{16}{\left(((-1)^j + 2j + 7)x_0 - ((-1)^j - 2j - 9)x_1 + 4x_2 \right) \left(((-1)^j + 2j + 3)x_0 - ((-1)^j - 2j - 5)x_1 + 4x_2 \right) ((j + 2))}$$

$$\frac{1}{(x_0 + 2x_1 + 2x_2)(x_0 + 2x_1 + x_2)(x_0 + x_1 + x_2)}$$

```
[117]: # describe a set with infinitely many accumulation points from the Cassaigne
        ↪ algo
a = dag.Cassaigne()
A = a.alphabet
```

[117] :





```
[119]: # describe each domain as union of triangles
for d in am.domains_polyhedra():
    show(d)
```

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \bigcup_{j \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}^j \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 2 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \bigcup_{i \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}^i \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \bigcup_{j \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \bigcup_{j \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}^j \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$$\begin{aligned}
& \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \bigsqcup_{j \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 3 \\ 2 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3 \\
& \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \bigsqcup_{i \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}^i \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \bigsqcup_{j \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}^j \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3 \\
& \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \bigsqcup_{j \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}^j \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 2 \\ 1 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3 \\
& \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 2 & 2 \end{pmatrix} \mathbb{R}_+^3 \\
& \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 3 \\ 2 & 1 & 2 \end{pmatrix} \mathbb{R}_+^3
\end{aligned}$$

```
[120]: # compute density of each state
for d in am.invariant_density():
    show(d)
```

$$\sum_{j \in \mathbb{N}} \frac{8}{\left(((-1)^j + 2j + 5)x_0 - ((-1)^j - 2j - 7)x_1 + 4x_2 \right) \left(((-1)^j + 2j + 3)x_0 - ((-1)^j - 2j - 5)x_1 + 4x_2 \right) ((j + 2))x}$$

$$\sum_{j, i \in \mathbb{N}} - \frac{1}{\left((2((-1)^i + 1)((-1)^j + 2j + 11) - 2((-1)^i - 1)((-1)^j + 2j + 7) + ((-1)^i - 2i - 1)((-1)^j - 2j - 9) \right) x}$$

$$\sum_{j \in \mathbb{N}} \frac{16}{\left(((-1)^j + 2j + 7)x_0 + ((-1)^j + 2j + 11)x_1 - ((-1)^j - 2j - 9)x_2 \right) \left(((-1)^j + 2j + 3)x_0 + ((-1)^j + 2j + 7)x_1 - ((-1)^j - 2j - 9)x_2 \right)}$$

$$\sum_{j \in \mathbb{N}} \frac{8}{\left(((-1)^j + 2j + 7)x_0 - ((-1)^j - 2j - 9)x_1 + 4x_2 \right) \left(((-1)^j + 2j + 5)x_0 - ((-1)^j - 2j - 7)x_1 + 4x_2 \right) ((j + 2))x}$$

$$\sum_{j, i \in \mathbb{N}} \frac{1}{\left((2((-1)^i + 1)((-1)^j + 2j + 11) - 2((-1)^i - 1)((-1)^j + 2j + 7) + ((-1)^i - 2i - 1)((-1)^j - 2j - 9) \right) x_0}$$

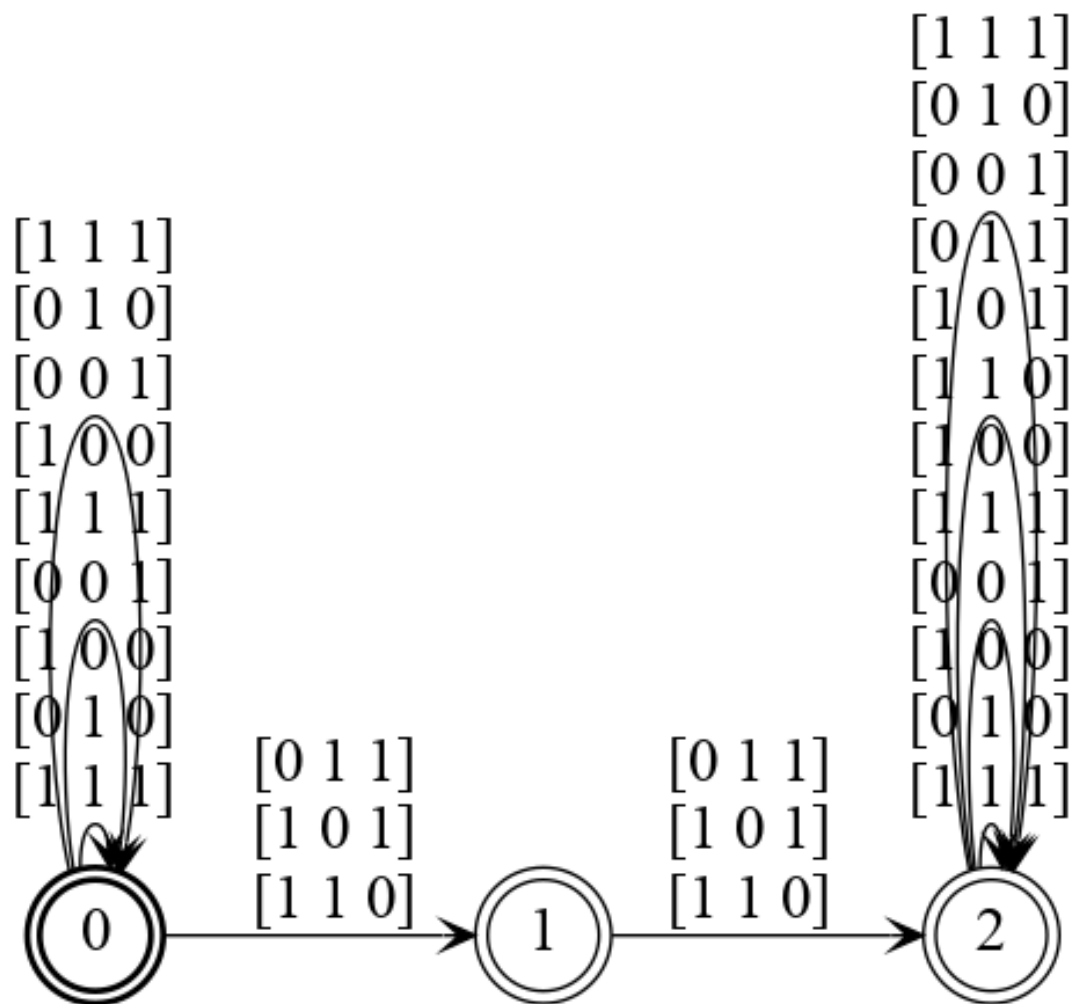
$$\sum_{j \in \mathbb{N}} - \frac{16}{\left(\left((-1)^j + 2j + 7 \right) x_0 + \left((-1)^j + 2j + 11 \right) x_1 - \left((-1)^j - 2j - 9 \right) x_2 \right) \left(\left((-1)^j - 2j - 5 \right) x_0 + \left((-1)^j - 2j - 9 \right) x_1 + \left((-1)^j - 2j - 13 \right) x_2 \right)}$$

$$\frac{1}{(x_0 + 3x_1 + 2x_2)(x_0 + 2x_1 + 2x_2)(x_1 + x_2)}$$

$$\frac{1}{(x_0 + 3x_1 + 2x_2)(x_0 + 2x_1 + 2x_2)(x_0 + 2x_1 + x_2)}$$

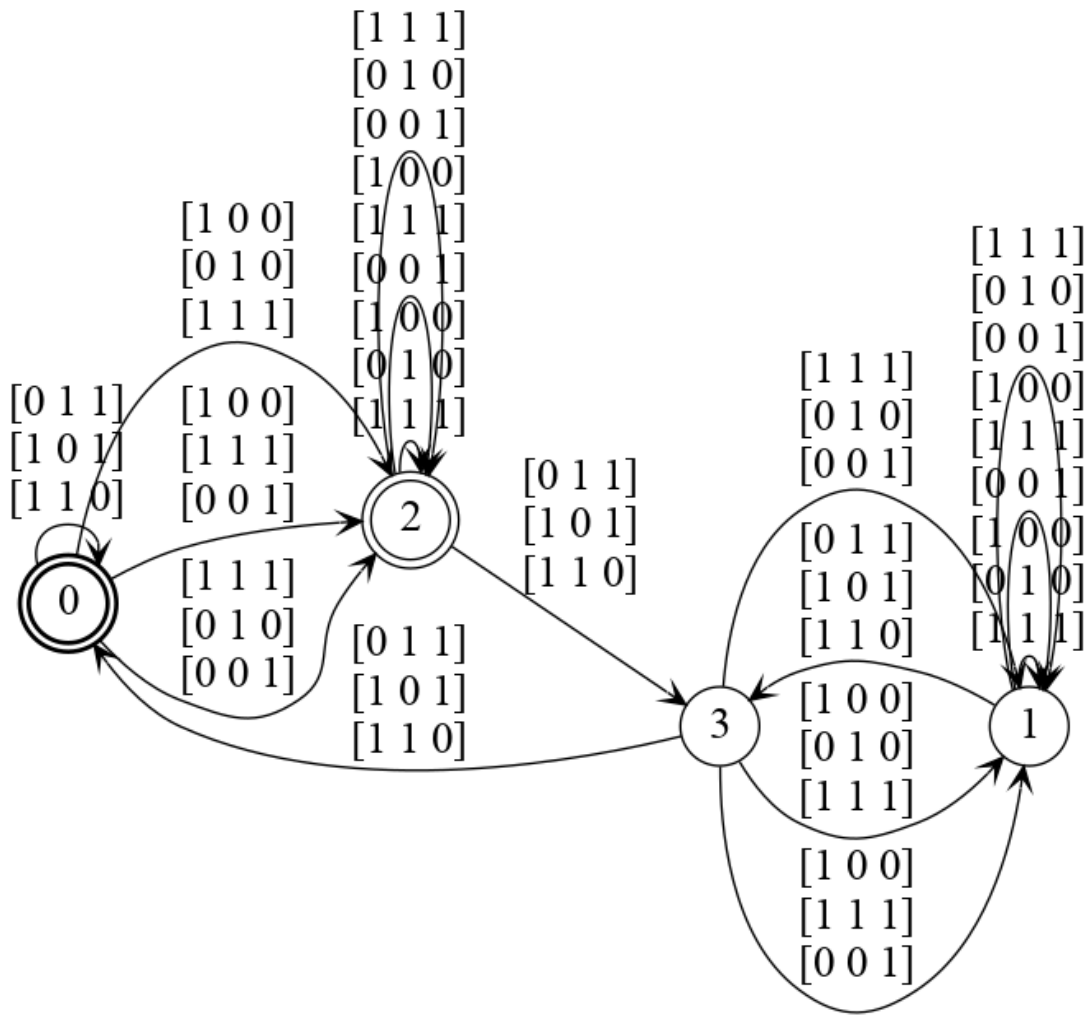
```
[124]: # describe accumulation points from the Reverse algo
a = dag.Reverse()
A = a.alphabet
a = □
↳ DetAutomaton([(0,A[0],0),(0,A[1],0),(0,A[3],0),(0,A[2],1),(1,A[2],2)]+[(2,m,2)□
↳ for m in A], i=0, avoidDiGraph=True)
a.plot()
```

[124]:

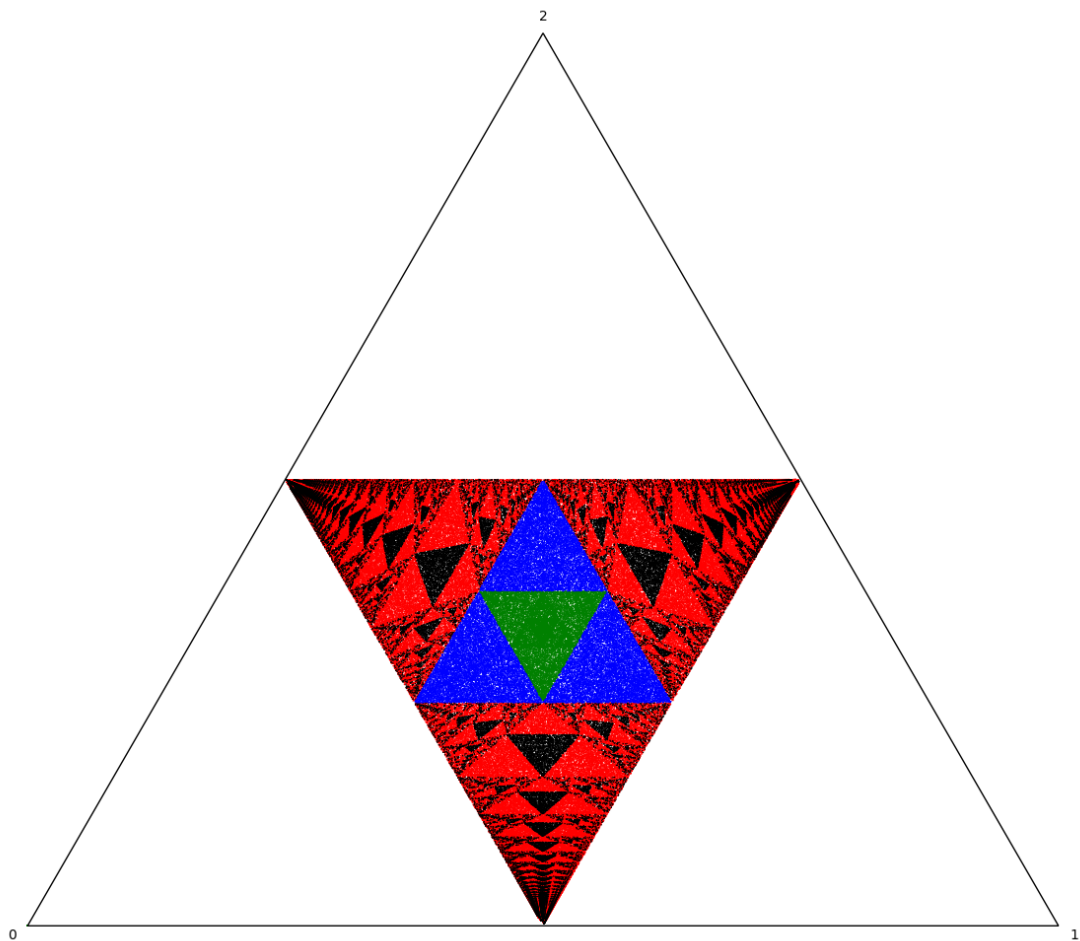


```
[125]: am = a.mirror().determinize().minimize()
       am.plot()
```

[125]:



```
[126]: am.plot_domains(colors=["green","red","black","blue"], N2=200).show(figsize=15)
```



```
[127]: for d in am.domains_polyhedra():
        show(d)
```

$$\begin{pmatrix} 2 & 3 & 3 \\ 3 & 2 & 3 \\ 3 & 3 & 2 \end{pmatrix} \mathbb{R}_+^3$$

$\bigsqcup_{m \in L} m \mathbb{R}_+^3$, where L is recognized by a DetAutomaton with 5 states and an alphabet of 4 letters

$\bigsqcup_{m \in L} m \mathbb{R}_+^3$, where L is recognized by a DetAutomaton with 5 states and an alphabet of 4 letters

$$\begin{pmatrix} 2 & 3 & 3 \\ 1 & 3 & 2 \\ 1 & 2 & 3 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 3 & 3 & 2 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 3 & 1 & 2 \\ 3 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix} \mathbb{R}_+^3$$

```
[128]: for d in am.invariant_density():
        show(d)
```

$$\frac{8}{(3x_0 + 3x_1 + 2x_2)(3x_0 + 2x_1 + 3x_2)(2x_0 + 3x_1 + 3x_2)}$$

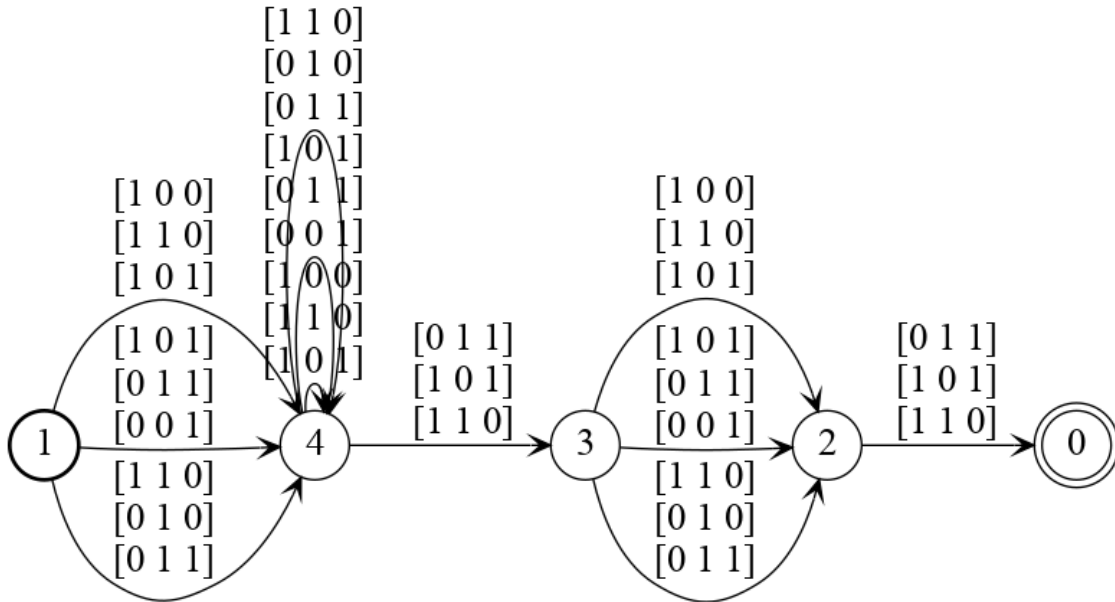
$\sum_{m \in L} \frac{|\det(m)|}{\prod m x}$, where L is recognized by a DetAutomaton with 5 states and an alphabet of 4 letters

$\sum_{m \in L} \frac{|\det(m)|}{\prod m x}$, where L is recognized by a DetAutomaton with 5 states and an alphabet of 4 letters

$$\frac{4}{(3x_0 + 3x_1 + 2x_2)(3x_0 + 2x_1 + 3x_2)(2x_0 + x_1 + x_2)} + \frac{4}{(3x_0 + 3x_1 + 2x_2)(2x_0 + 3x_1 + 3x_2)(x_0 + 2x_1 + x_2)} + \dots$$

```
[129]: # automaton describing the density of state 1
        am.invariant_density()[1].a.plot()
```

[129]:



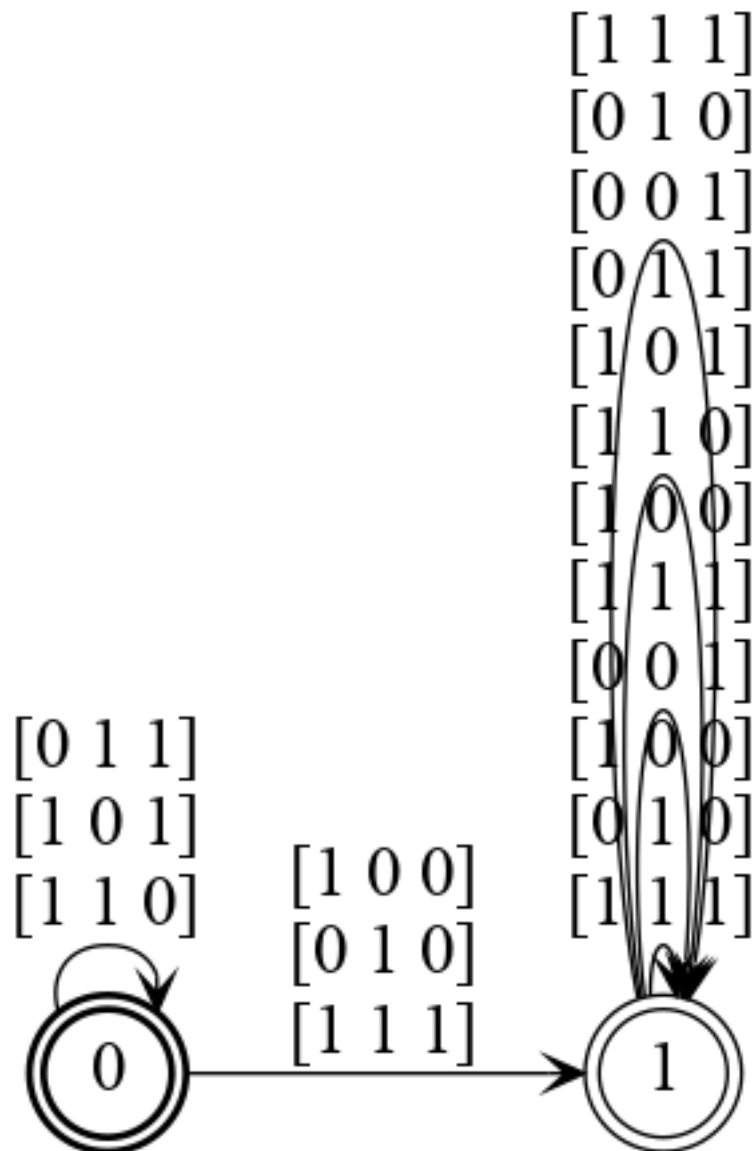
```
[130]: # describe accumulation points from the Reverse algo
        from badic import *
        a = dag.Reverse()
```

```

A = a.alphabet
a = DetAutomaton([(0,A[2],0),(0,A[0],1)]+[(1,m,1) for m in A], i=0,
↳avoidDiGraph=True)
a.plot()

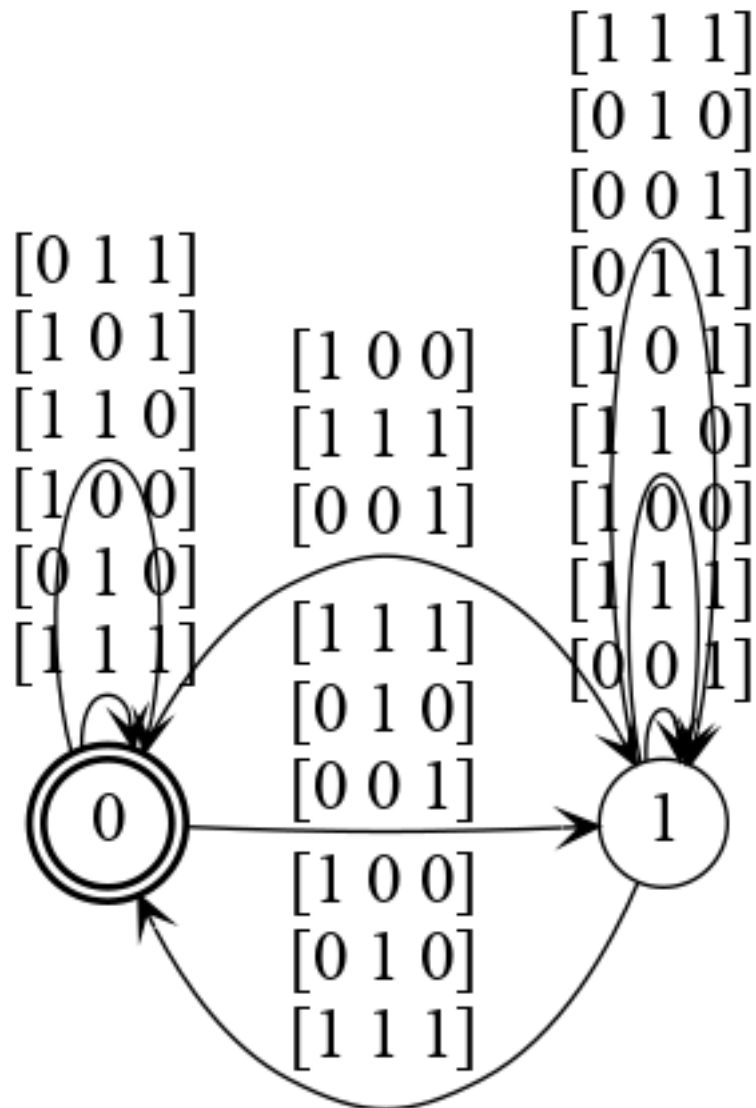
```

[130]:



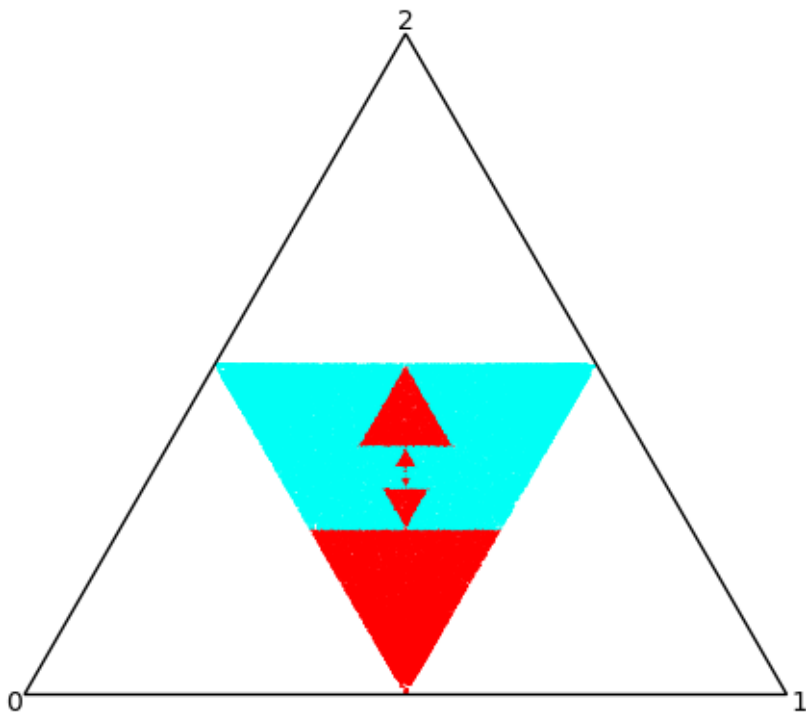
```
[131]: am = a.mirror().determinize().minimize()
am.plot()
```

[131]:



```
[181]: am.plot_domains()
```

[181]:



```
[132]: # describe domains as infinite unions of triangles
for d in am.domains_polyhedra():
    show(d)
```

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}^j \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \mathbb{R}_+^3$$

$$\bigcup_{j \in \mathbb{N}} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}^j \left(\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{pmatrix} \mathbb{R}_+^3 \cup \begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 1 \end{pmatrix} \mathbb{R}_+^3 \right)$$

```
[133]: # compute density of both states
for d in am.invariant_density():
    show(d)
```

$$\sum_{j \in \mathbb{N}} \frac{54 \cdot 2^j}{\left((4 \cdot 2^j - (-1)^j)x_0 + 2(2 \cdot 2^j + (-1)^j)x_1 + (4 \cdot 2^j - (-1)^j)x_2 \right) \left(2(2 \cdot 2^j + (-1)^j)x_0 + (4 \cdot 2^j - (-1)^j)x_1 + (4 \cdot 2^j - (-1)^j)x_2 \right)}$$

$$\left(\sum_{j \in \mathbb{N}} \frac{54 \cdot 2^j}{((4 \cdot 2^j - (-1)^j)x_0 + (4 \cdot 2^j - (-1)^j)x_1 + 2(2 \cdot 2^j + (-1)^j)x_2)((4 \cdot 2^j - (-1)^j)x_0 + 2(2 \cdot 2^j + (-1)^j)x_1 +$$

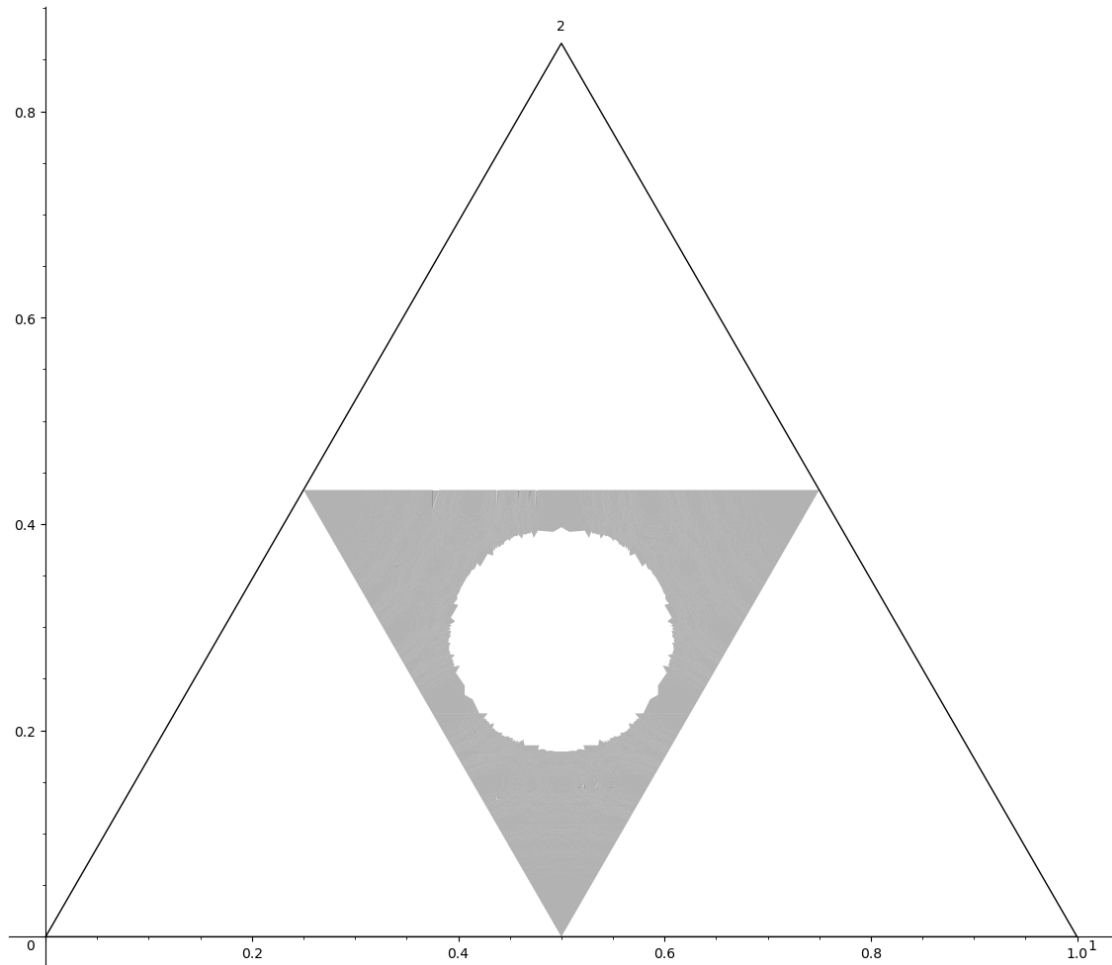
17 Domains approximating a given shape

```
[226]: # construct an automaton approximating a disk, using the reverse algorithm
from badic import *
a = dag.Reverse()

from badic.automata import proj_to_pt
center = proj_to_pt((1,1,1))

def in_disk (m, r):
    for c in m.columns():
        p = proj_to_pt(c) - center
        if norm(p) >= r:
            return False
    return True

r0 = .11 # radius of the approximated disk
A = a.alphabet
lm = [((m*m2*m3*m4*m5*m6*m7).transpose()*A[2], [m7,m6,m5,m4,m3,m2,m]) for m in A
      ↪ A for m2 in A for m3 in A for m4 in A for m5 in A for m6 in A for m7 in A]
#lm = [((m*m2*m3*m4*m5*m6).transpose()*A[2], [m6,m5,m4,m3,m2,m]) for m in A for
      ↪ m2 in A for m3 in A for m4 in A for m5 in A for m6 in A]
#lm = [((m*m2*m3*m4*m5).transpose()*A[2], [m5,m4,m3,m2,m]) for m in A for m2 in A
      ↪ A for m3 in A for m4 in A for m5 in A]
#lm = [((m*m2*m3*m4).transpose()*A[2], [m4,m3,m2,m]) for m in A for m2 in A for
      ↪ m3 in A for m4 in A]
#lm = [((m*m2*m3).transpose()*A[2], [m3,m2,m]) for m in A for m2 in A for m3 in A
      ↪ A]
#lm = [((m*m2).transpose()*A[2], [m2,m]) for m in A for m2 in A]
lm = [(m,l) for m,l in lm if not in_disk(m, r0)]
#lm = [(A[0]*A[1]).transpose()*A[2], [A[1], A[0]]]
g = plot_tri(identity_matrix(3))
for m,l in lm:
    g += plot_tri(m, text=False, fill=True, alpha=.3)
g.show(figsize=15)
```



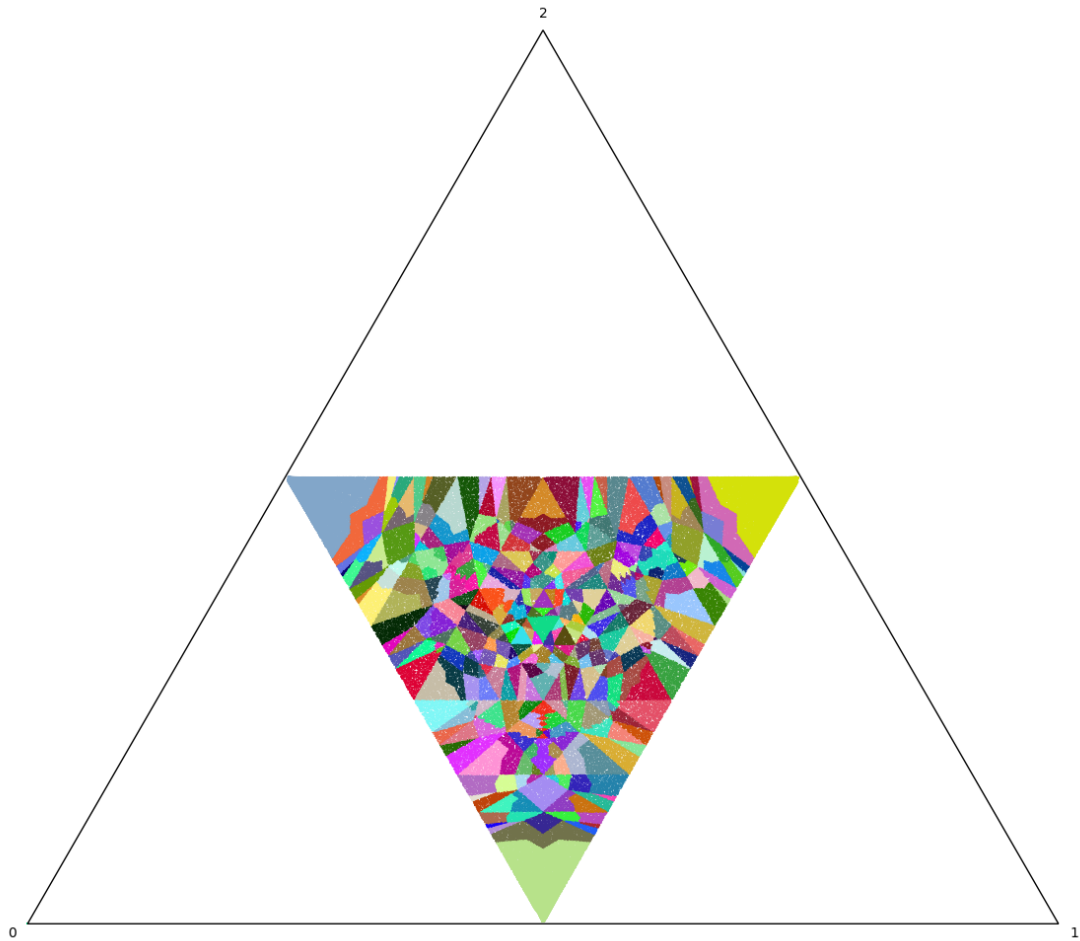
```
[227]: # construct the automaton
a = dag.Empty(A=A)
for m,l in lm:
    a = a.union(dag.Word(l, A=A).concat(dag.AnyWord(A=A)))
a #.plot()
```

[227]: DetAutomaton with 125 states and an alphabet of 4 letters

```
[228]: # compute the corresponding matrices graph (which is an extension of the
↪reverse algorithm)
am = a.mirror().determinize().minimize()
am.set_final_states(am.states)
am #.plot()
```

[228]: DetAutomaton with 1006 states and an alphabet of 4 letters

```
[346]: g = am.plot_domains(colors=[(random(), random(), random()) for _ in range(am.  
    ↪ n_states)], N2=200)  
g.show(figsize=15)
```



```
[ ]:
```