The Algebra and Geometry of Networks

R.F.C. Walters

http://rfcwalters.blogspot.com

in collaboration with

D. Maglia, R. Rosebrugh, N. Sabadini and F. Schiavio

Dipartimento di Scienze e Alta Tecnologia
Università dell' Insubria
via Carloni, 78, Como, Italy

–

Talk presented at ICTCS Conference
Varese

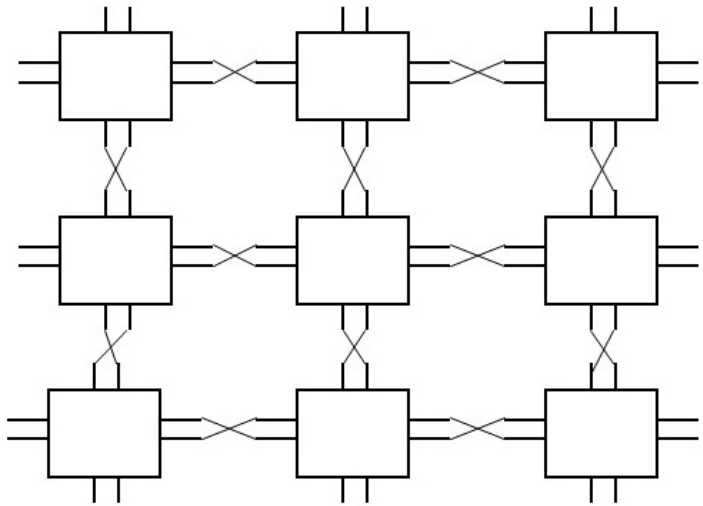
19th September 2012

Introduction

We will describe two kinds of networks of interconnected components:

- Networks of components with state;
- Tangled circuits - networks in which the tangling of the connecting "wires" is represented.

In both cases we will describe the networks geometrically and compositionally (i.e. algebraically, and by closely related algebras - monoidal categories with extra structure).



Part I: Networks with state

We formalize the notion of closed network by the following definition: A monoidal graph **M** consists of two sets M_0 (vertices or wires) and M_1 (edges or components) and two functions $domain : M_1 \to M_0^*$ and $codomain : M_1 \to M_0^*$ where M_0^* is the free monoid on M_0.

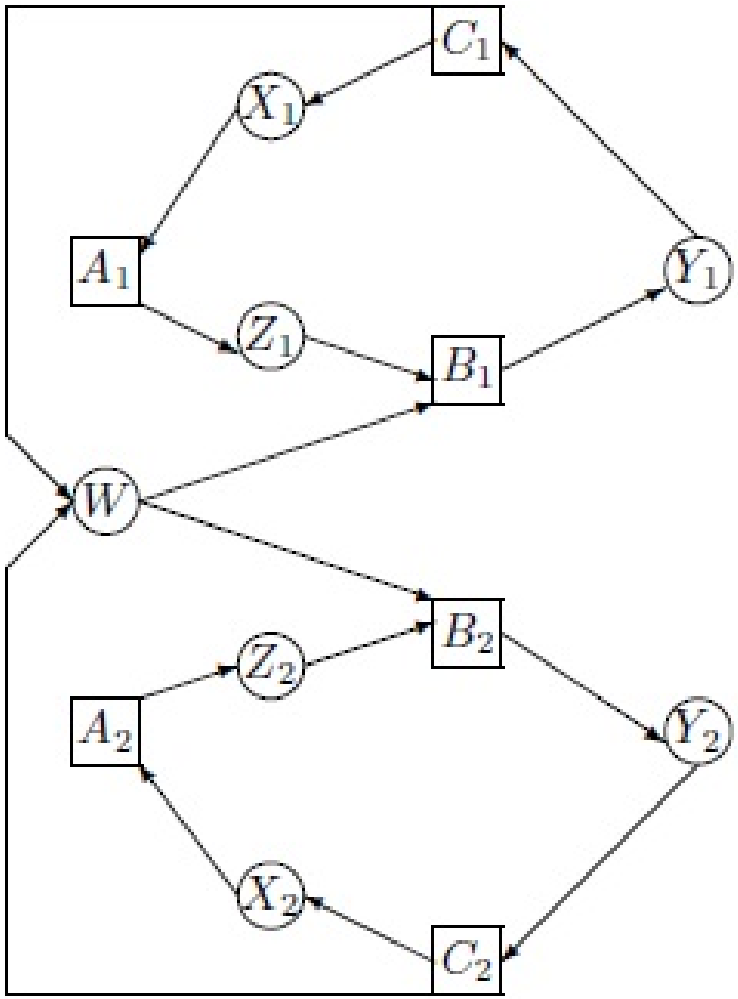
We can represent a monoidal graph by a list of arrows (the elements of M_1) with the tail of the arrow being it domain, and the point of the arrow its codomain. An example with $M_1=\{A_1,B_1,C_1,A_2,B_2,C_2\}$ and $M_0=\{X_1,Y_1,Z_1,X_2,Y_2,Z_2,W\}$ is:

$$
\begin{aligned}
A_1 &: X_1 \to Z_1 \\
B_1 &: Z_1 W \to Y_1 \\
C_1 &: Y_1 \to W X_1 \\
A_2 &: X_2 \to Z_2 \\
B_2 &: W Z_2 \to Y_2 \\
C_2 &: Y_2 \to X_2 W
\end{aligned}
$$

Networks: the geometry

However there is a different geometric representation which explains why we regard a closed network as a monoidal graph. The elements of M_1 are represented as components, the elements of M_0 are represented as connectors or wires (or places), and the domain and codomain functions describe how the ports are joined to the connectors.

Networks: The network of a mutual exclusion protocol

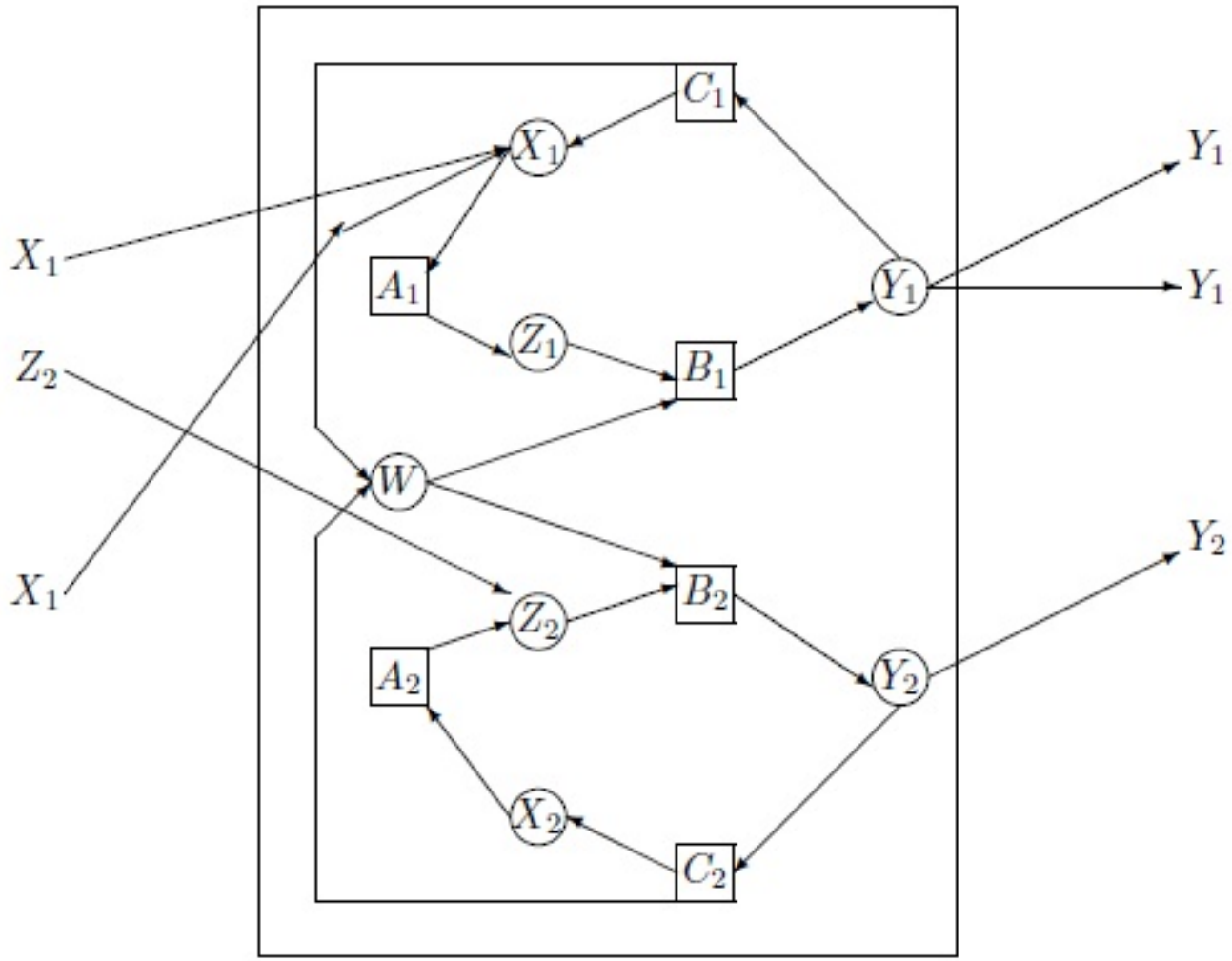


$$A_1 : X_1 \to Z_1$$
$$B_1 : Z_1 W \to Y_1$$
$$C_1 : Y_1 \to W X_1$$
$$A_2 : X_2 \to Z_2$$
$$B_2 : W Z_2 \to Y_2$$
$$C_2 : Y_2 \to X_2 W$$

Networks: the algebra

To describe such networks compositionally we need a notion of open network. An open network is a monoidal graph together with left and right interfaces. Before giving the formal definition I sketch an example. Notice it has the same form as a single component.

Networks: an open network



Networks

An open network has a simple standard categorical description. It is a cospan of monoidal graphs.

The notion of cospan was introduced by *Jean Benabou* in 1967 (*Introduction to Bicategories*, Springer Lecture Notes in Mathematics 47, pages 1-77).

Cospans form an algebra with properties first described by *Carboni and Walters* in 1987 (A. Carboni, R.F.C. Walters, *Cartesian Bicategories I*, J. Pure Applied Algebra, 49, 11–32, 1987).

The algebra was first applied to the parallel composition of automata in 1997 in *P. Katis, N. Sabadini, R.F.C. Walters*, *Span(Graph): an algebra of transition systems*, Proceedings AMAST '97, SLNCS 1349 (1997) 322-336.

In the same volume, Proceedings AMAST '97, *Katis, Sabadini and Walters* gave a compositional theory of P/T nets.

Cospans were used for graph rewriting in F. Gadducci, R. Heckel, M. Llabres, *A bi-categorical axiomatisation of concurrent graph rewriting*, ENTCS, 29, 1999.

More recently (2012) Montanari and collaborators have studied compositional theories of Petri Nets using the *Tile Model*, and ideas of Sobocinski who was influenced by Katis, Sabadini and Walters' 1997 papers.

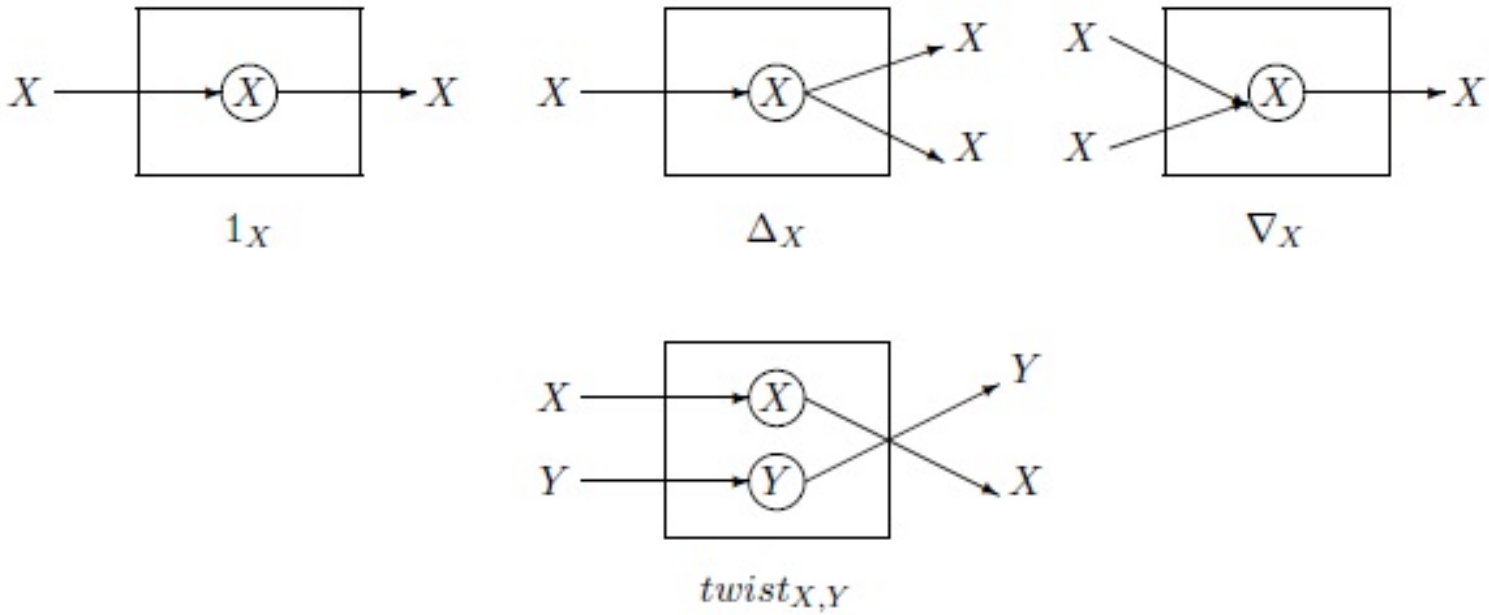
Networks: the algebra

The algebraic structure that cospans (open networks) admit is that they form the arrows of a symmetric monoidal category in which every object has a commutative separable algebra structure.
A commutative separable algebra in a symmetric monoidal category consists of an object X and four arrows $\nabla : X \otimes X \to X$, $\Delta : X \to X \otimes X$, $n : I \to X$ and $e : X \to I$ making (X, ∇, e) a commutative monoid, (X, Δ, n) a cocommutative comonoid and satisfying the equations

$$(1_X \otimes \nabla)(\Delta \otimes 1_X) = \Delta\nabla = (\nabla \otimes 1_X)(1_X \otimes \Delta) : X \otimes X \to X \otimes X$$
$$\nabla\Delta = 1.$$

Rosebrugh, Sabadini, Walters: Generic commutative separable algebras and cospans of graphs, TAC 2005, show that it is actually a free such algebra.

Networks: the constants of the algebra of networks



Composition of cospans is a pushout; that is, disjoint union with quotienting of places.
Tensor of cospans is disjoint union.
The idea is that starting with components, using these constants and operations we can build up any network.

Networks with state: adding state to components

We did not mention states and transitions of components so far.
How do we describe networks with state? As follows: we associate alphabets (of signals) to the places, and automata to the components. The transitions of the automata are labelled with the signals occuring on the ports when that transition occurs.
A cospan of monoidal graphs with this extra information is what we use to model open networks with state.

Networks with state: adding state to components

Again this has a standard categorical description: open networks with state are arrows in the monoidal category

$$Cospan(MonoidalGraphs/Span(Graph)),$$

a category with the same structure as open networks.
The process of forming the global statespace of a network with state is a structure-preserving functor

$$Cospan(MonoidalGraphs/Span(Graph)) \longrightarrow Span(Graph),$$

which happens to be obtained by calculating a limit in *Graph*; that is, a kind of product of automata.

Rosebrugh, Sabadini, Walters: Calculating colimits compositionally, Ugo Montanari Volume, LNCS 5065, 2008

Networks with state

Consider the functor

$$limit : Cospan(MonoidalGraphs/Span(Graph)) \longrightarrow Span(Graph).$$

An arrow in the domain may be thought of as either (i) algebraically, an expression in components with state (the compositional view of a system) or (ii) geometrically, a network (cospan of monoidal graphs) labelled in graphs. The functor *limit* yields the global states and transitions of the system.
The two points of view of the domain yield two methods of calculating the global states and transitions of a system: (i) evaluating the expression in the codomain, or(ii) by evaluating the expression in the domain and taking a limit.
The second method is much more efficient (low-order polynomial) than the first (exponential) in finding single paths in the system, and Filippo Schiavio has written a program for doing that, with a graphical output. We present an example of such a calculation for a simple mutual exclusion protocol with 2^{16} global states.

Networks with state: the Spantools application - Filippo Schiavio

The application accepts in input the definition of components (labelled automata), networks of components, described as algebraic expressions, and commands to generate output diagrams representing structure (static) and behavior (dynamic) of given expressions.
The program is entirely developed in Java by using two external applications for input/output management, respectively Antlr3 and Graphviz. The first one is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions. The second one is a graph generation tool, good for drawing networks and state/transition diagrams.
The application is available at https://sourceforge.net/projects/spantool/.

Networks with state: mutual exclusion components

```
c = span(1,1){
  1->1 : 0/0
  1->2 : 1/0
  2->2 : 1/0
  2->3 : 0/1
  3->3 : 0/1
  3->1 : 0/0
  init -> 2 : 1/0
}

in_1 = span(2,1){
        init -> 1 : 0,n/0
        1 -> 1 : 0,n/0
        1 -> 1 : 0,s/0
        1 -> 2 : 1,n/0
        1 -> 2 : 1,s/0
        2 -> 2 : 1,n/0
        2 -> 2 : 1,s/0
        2 -> 3 : 0,f/c
        3 -> 3 : 0,f/c
        3 -> 1 : 0,n/0
        3 -> 1 : 0,s/0
}

in_2 = span(2,1){
        init -> 1 : n,0/0
        1 -> 1 : n,0/0
        1 -> 1 : f,0/0
        1 -> 2 : n,1/0
        1 -> 2 : f,1/0
        2 -> 2 : n,1/0
        2 -> 2 : f,1/0
        2 -> 3 : s,0/c
        3 -> 3 : s,0/c
        3 -> 1 : n,0/0
        3 -> 1 : f,0/0
}

out_1 = span(1,2){
        init -> 2 : 0/1,n
        1 -> 1 : 0/0,n
        1 -> 1 : 0/0,s
        1 -> 2 : c/0,f
        2 -> 2 : c/0,f
        2 -> 3 : 0/1,n
        3 -> 3 : 0/1,n
        3 -> 3 : 0/1,s
        3 -> 1 : 0/0,n
        3 -> 1 : 0/0,s
}

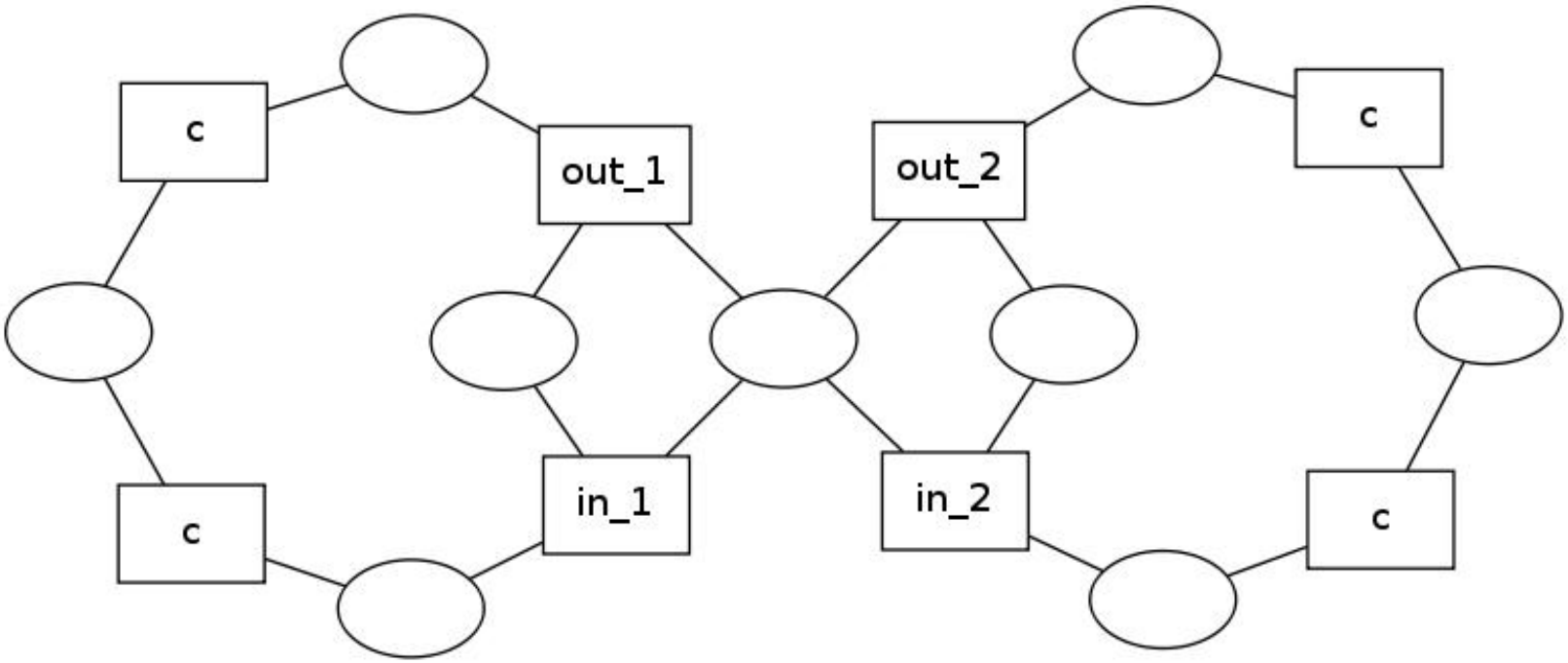
out_2 = span(1,2){
        init -> 2 : 0/n,1
        1 -> 1 : 0/n,0
        1 -> 1 : 0/f,0
        1 -> 2 : c/s,0
        2 -> 2 : c/s,0
        2 -> 3 : 0/n,1
        3 -> 3 : 0/n,1
        3 -> 3 : 0/f,1
        3 -> 1 : 0/n,0
        3 -> 1 : 0/s,0
}
```

Networks with state: mutual exclusion network

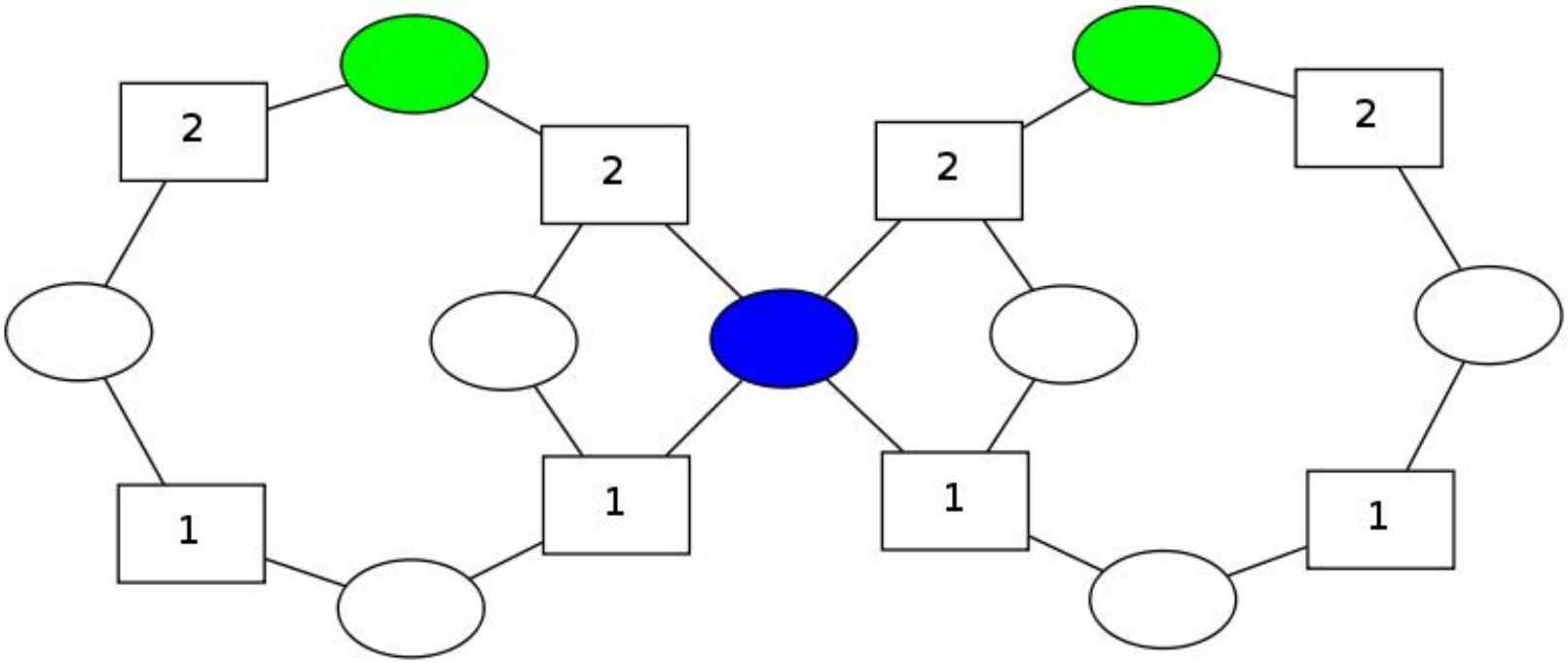
```
expr_feed = (UNIT * UNIT * UNIT * UNIT)
              ; (IDENTITY * in_1;out_1 * CODIAGONAL;DIAGONAL * in_2;out_2 * IDENTITY)
              ; (IDENTITY * c * COUNIT * COUNIT * c * IDENTITY)
              ; (IDENTITY * c * c * IDENTITY) ; (COUNIT * COUNIT)
```

```
ProceedToNextStateDrawingGraph(
        expr_feed,
        [init,init,init,init,init,init,1,1],
        100,
        out_petri/execution,
        out_petri/execution,
        {
        transitions_colors -> {0 -> white, 1-> green,
                               h -> blue, f -> white, s -> white, c -> red }
        spans_shapes -> {in_1 -> box, in_2 -> box, out_1 -> box, out_2 -> box, c->box}
        }
)
```

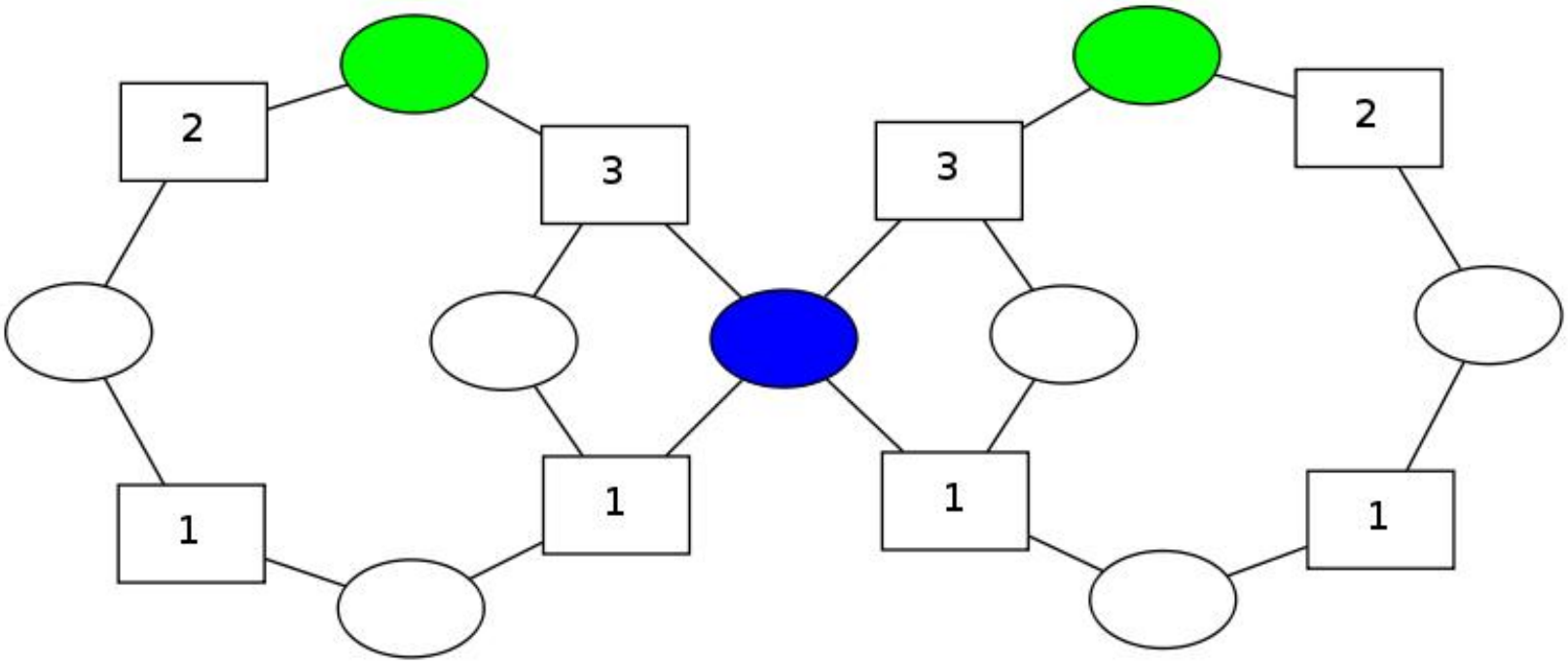
Networks with state: mutex execution



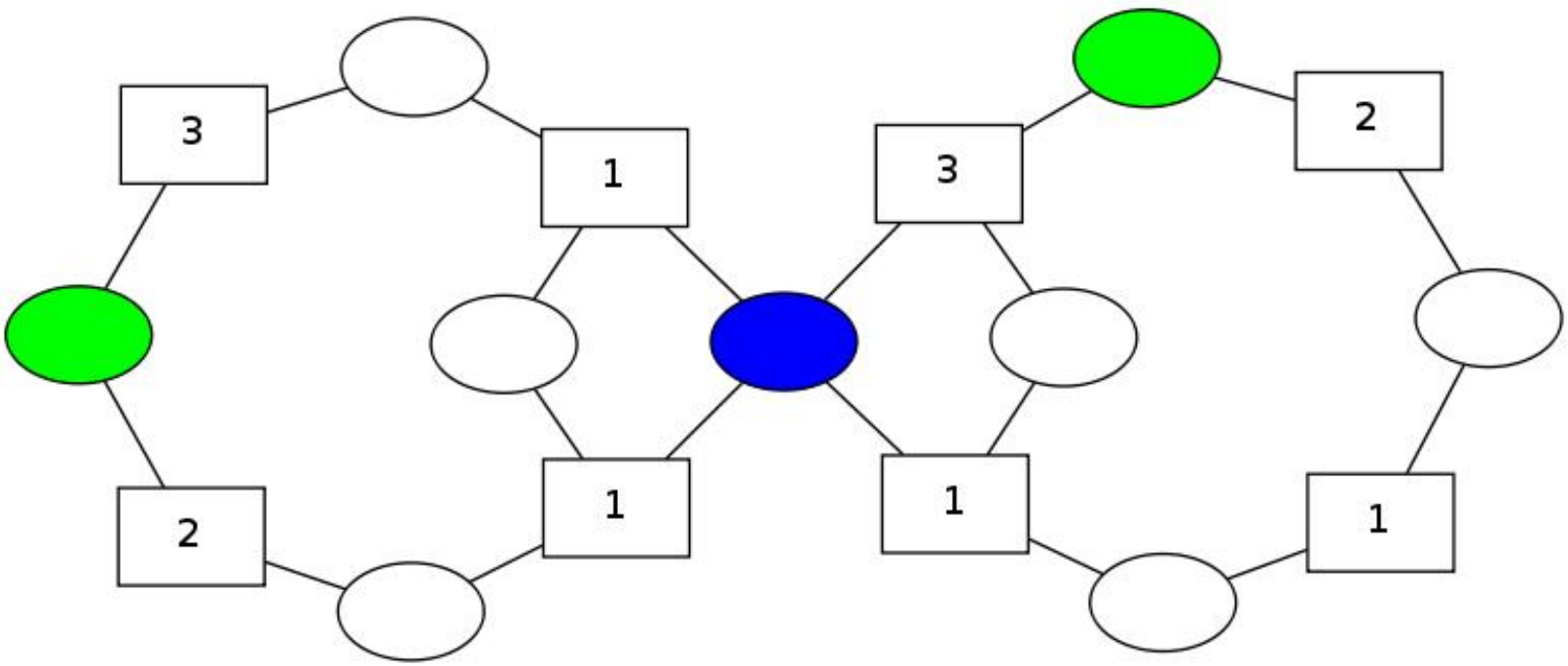
Networks with state: mutex execution



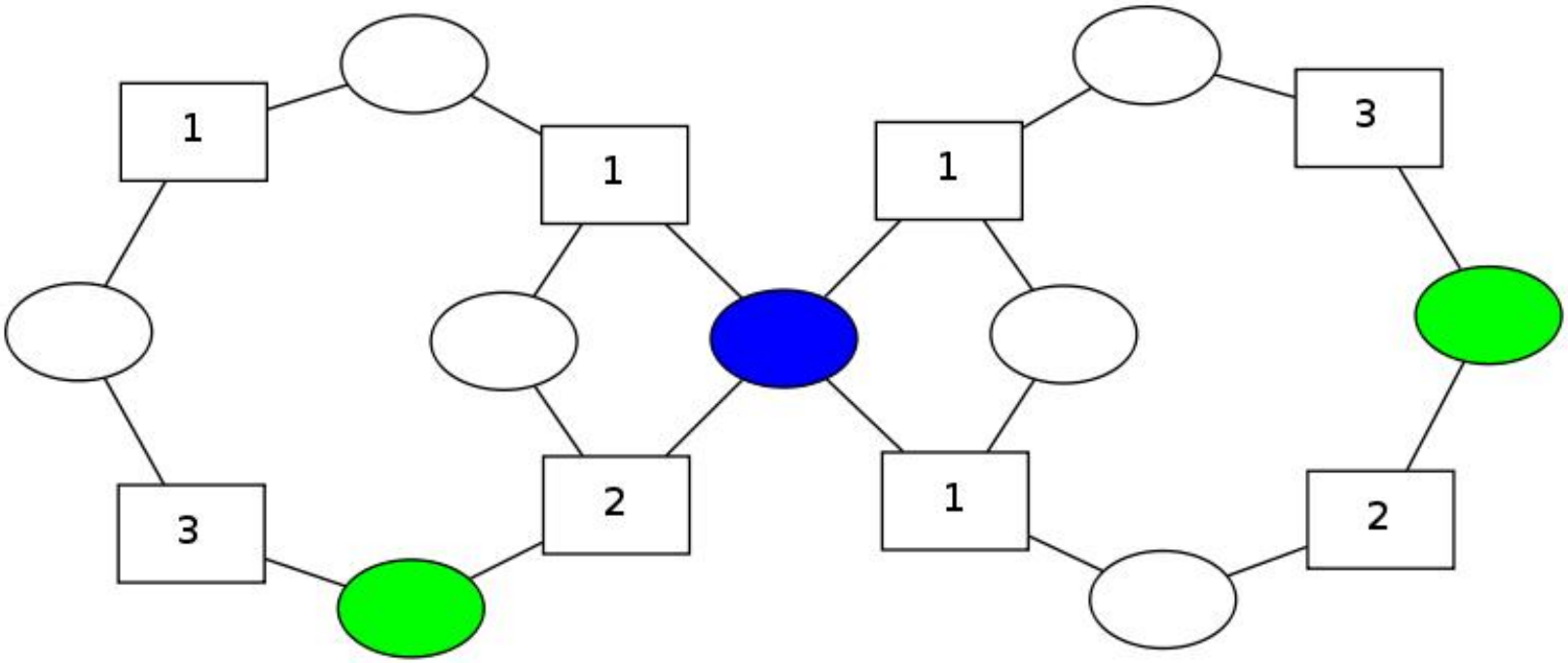
Networks with state: mutex execution



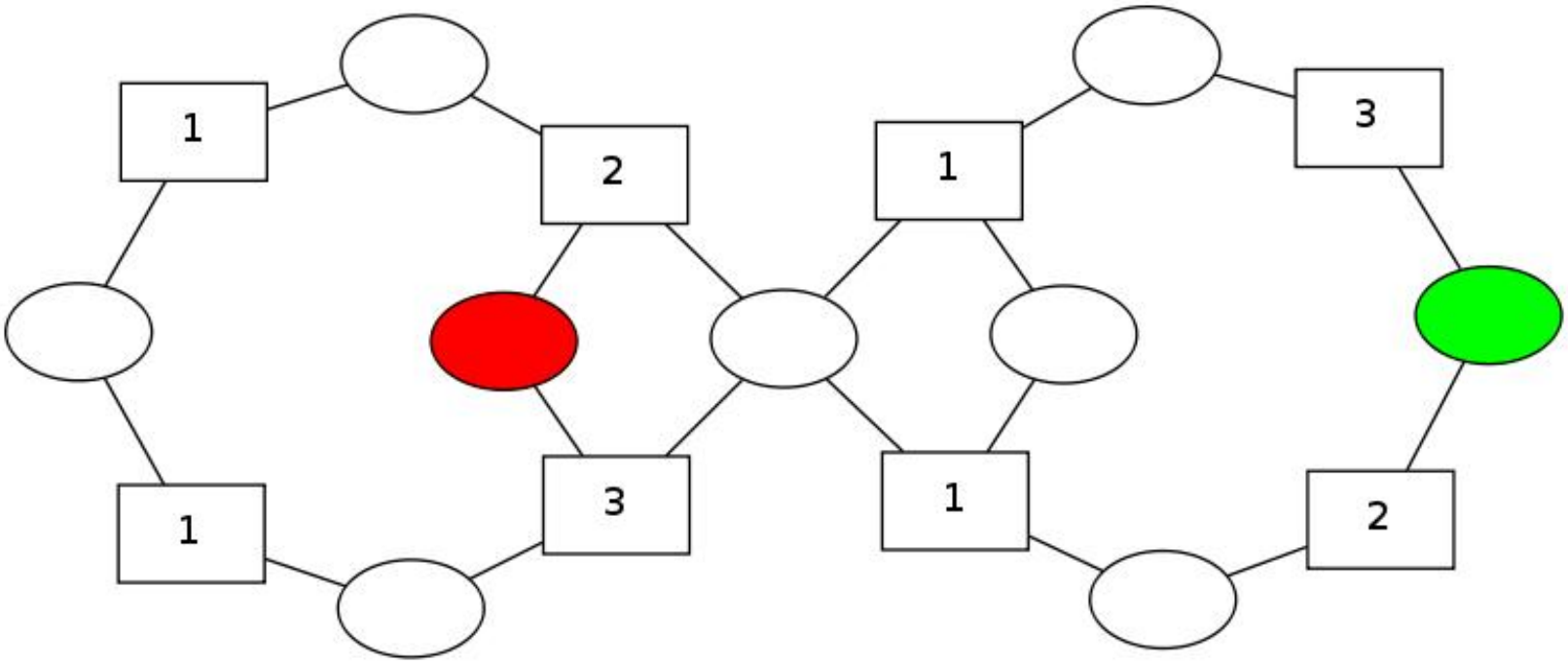
Networks with state: mutex execution



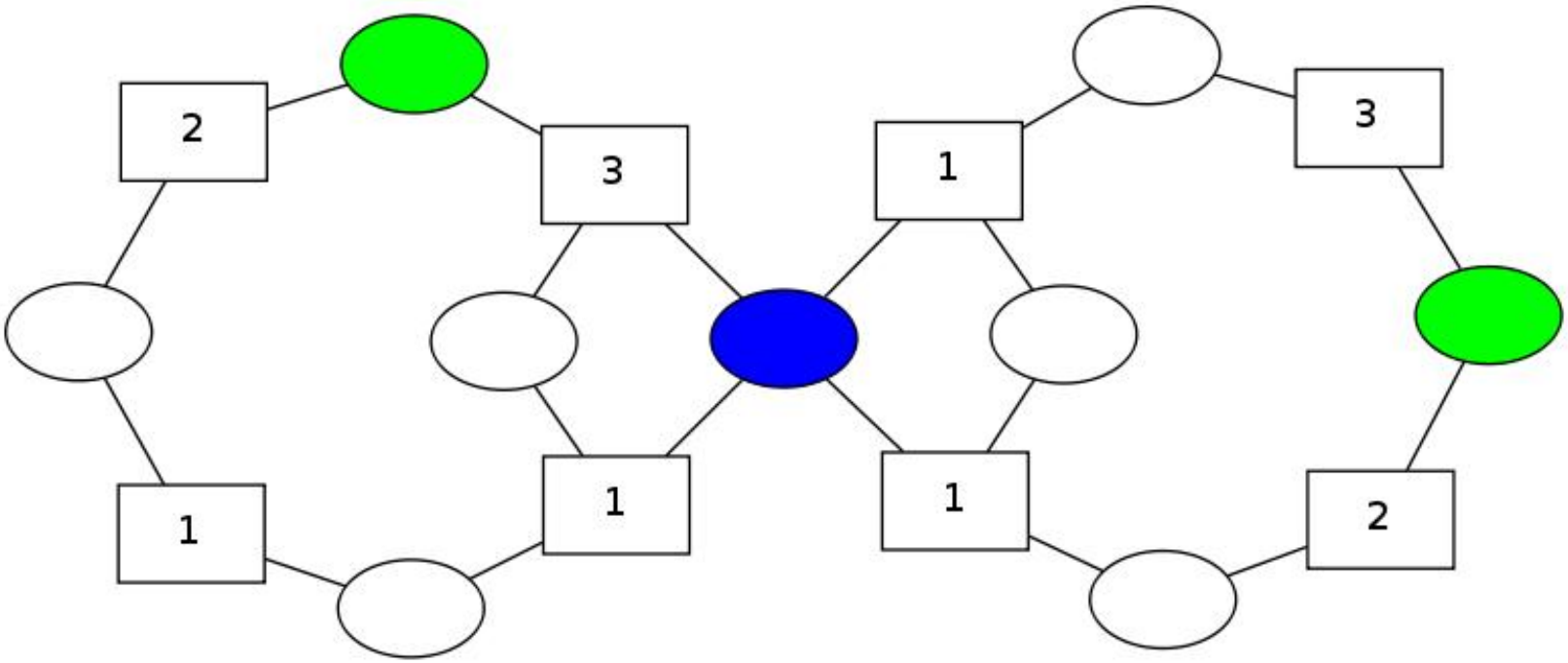
Networks with state: mutex execution



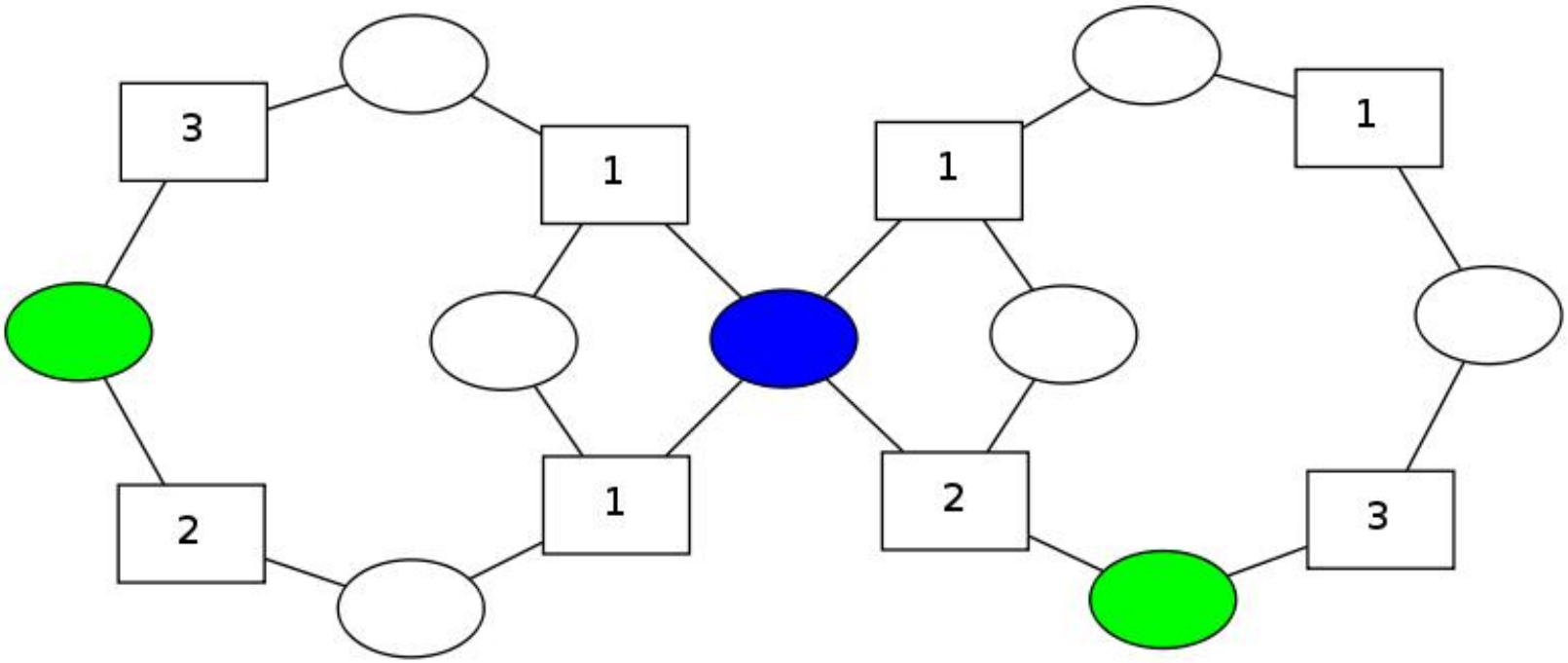
Networks with state: mutex execution



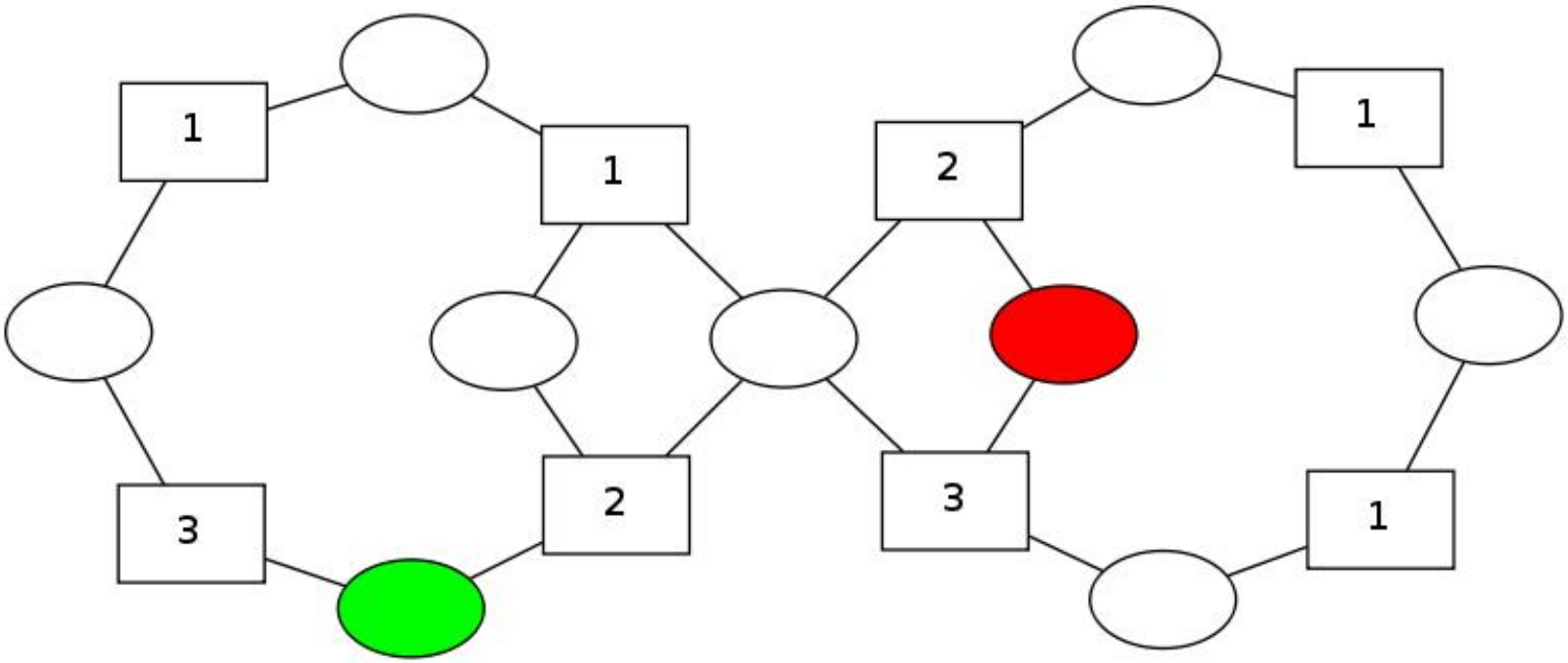
Networks with state: mutex execution



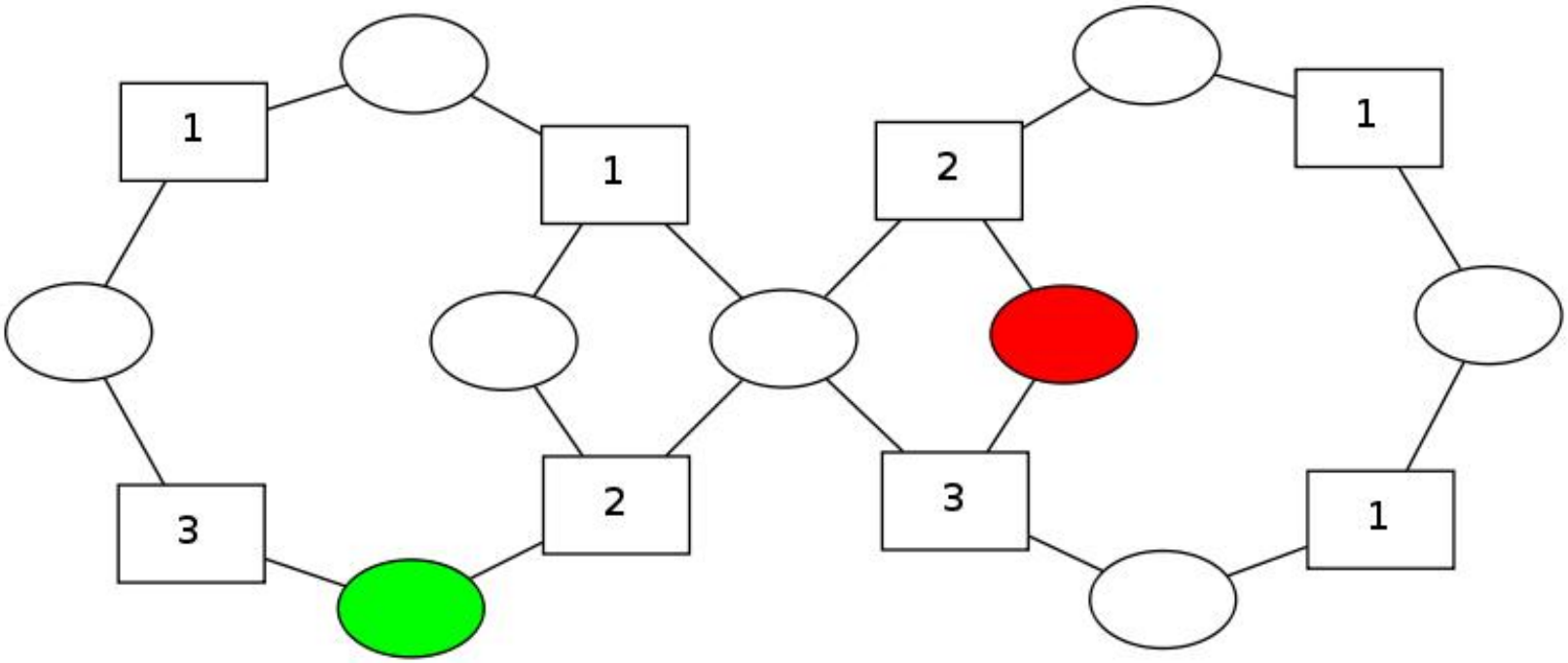
Networks with state: mutex execution



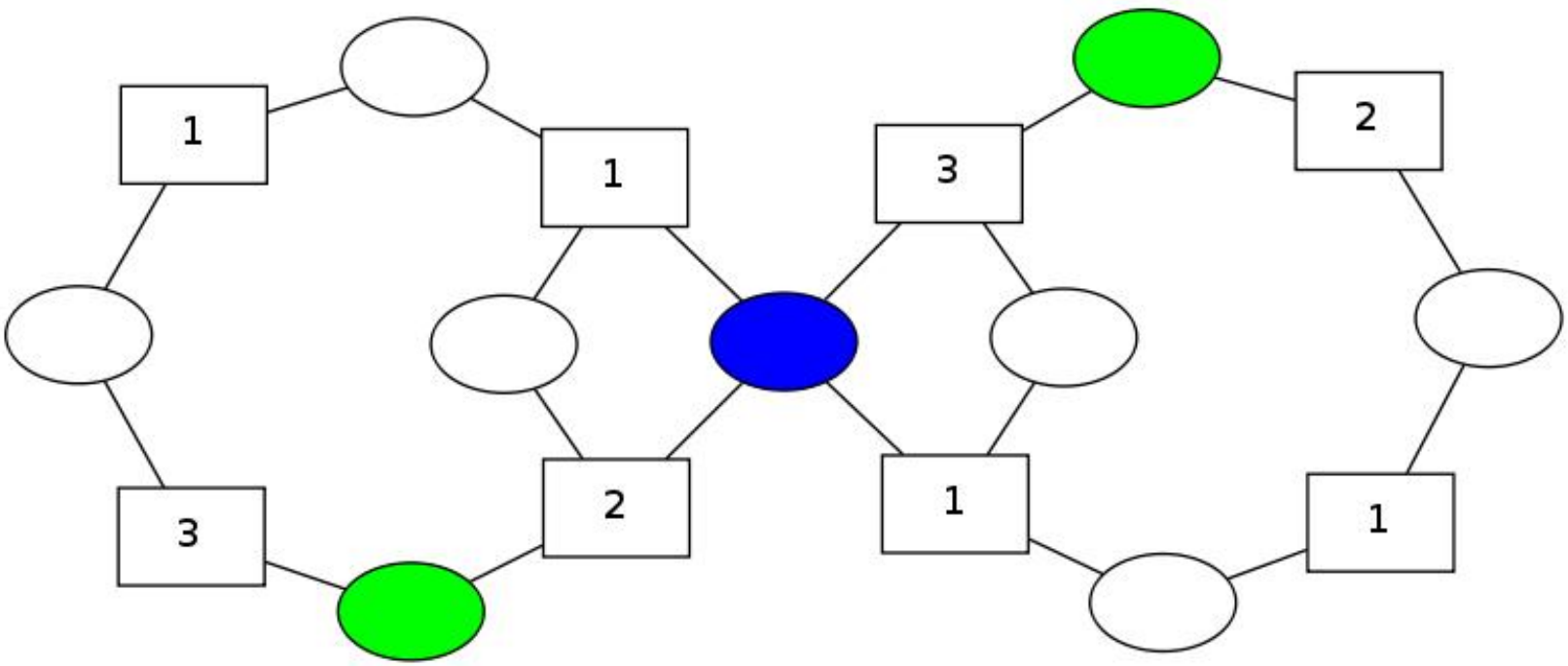
Networks with state: mutex execution



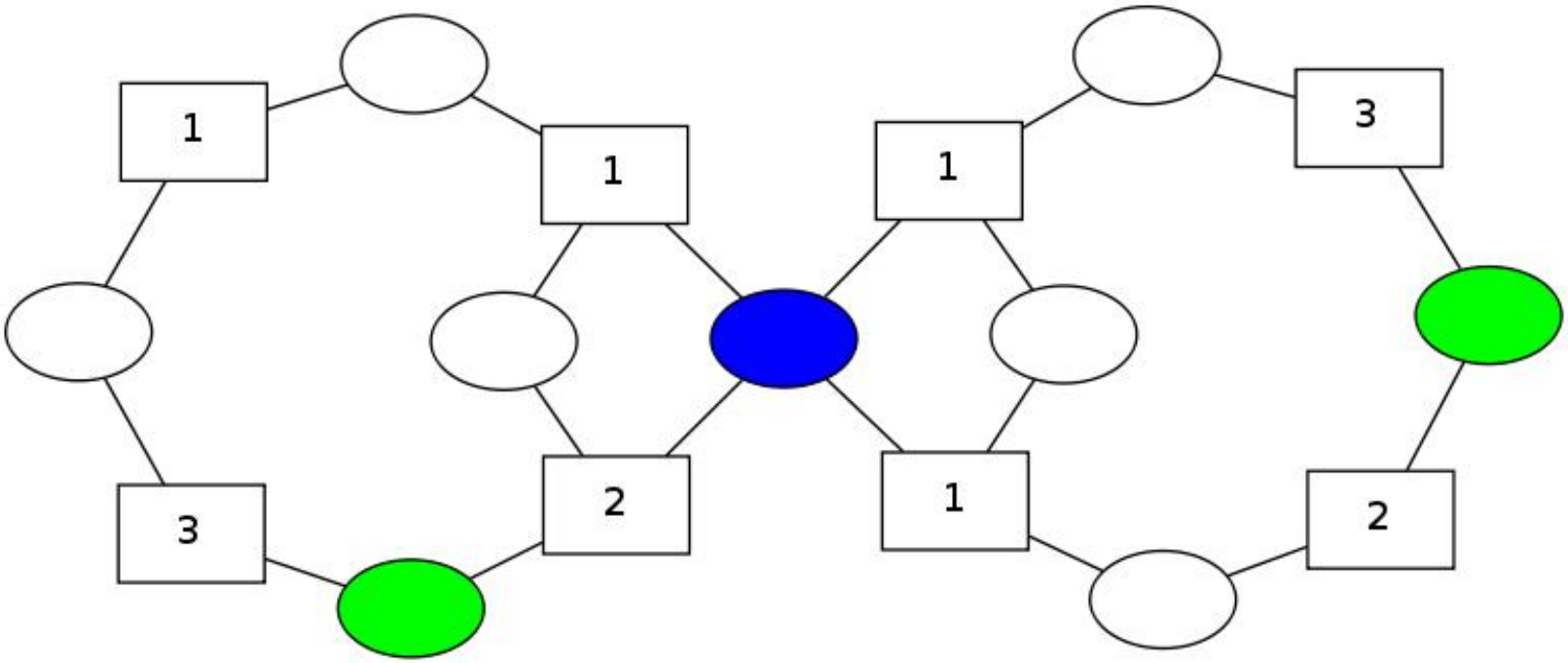
Networks with state: mutex execution



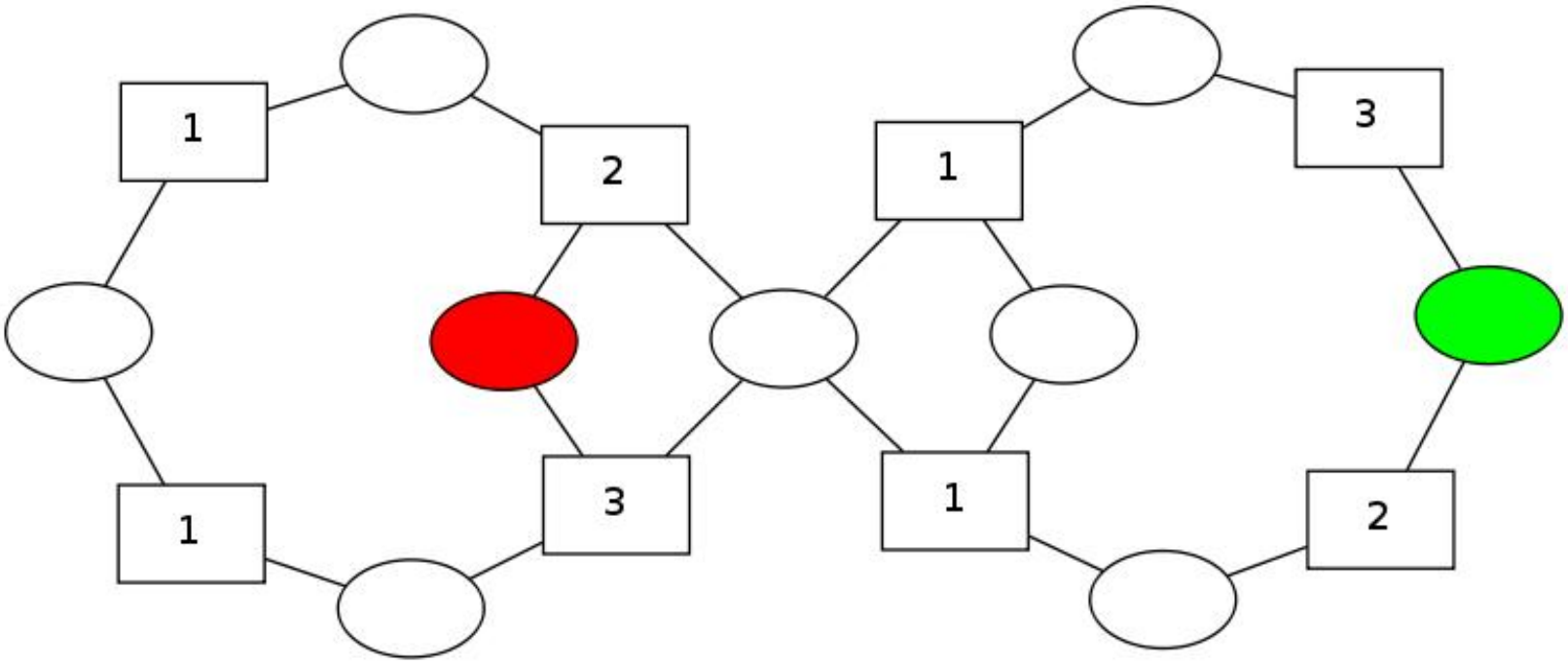
Networks with state: mutex execution



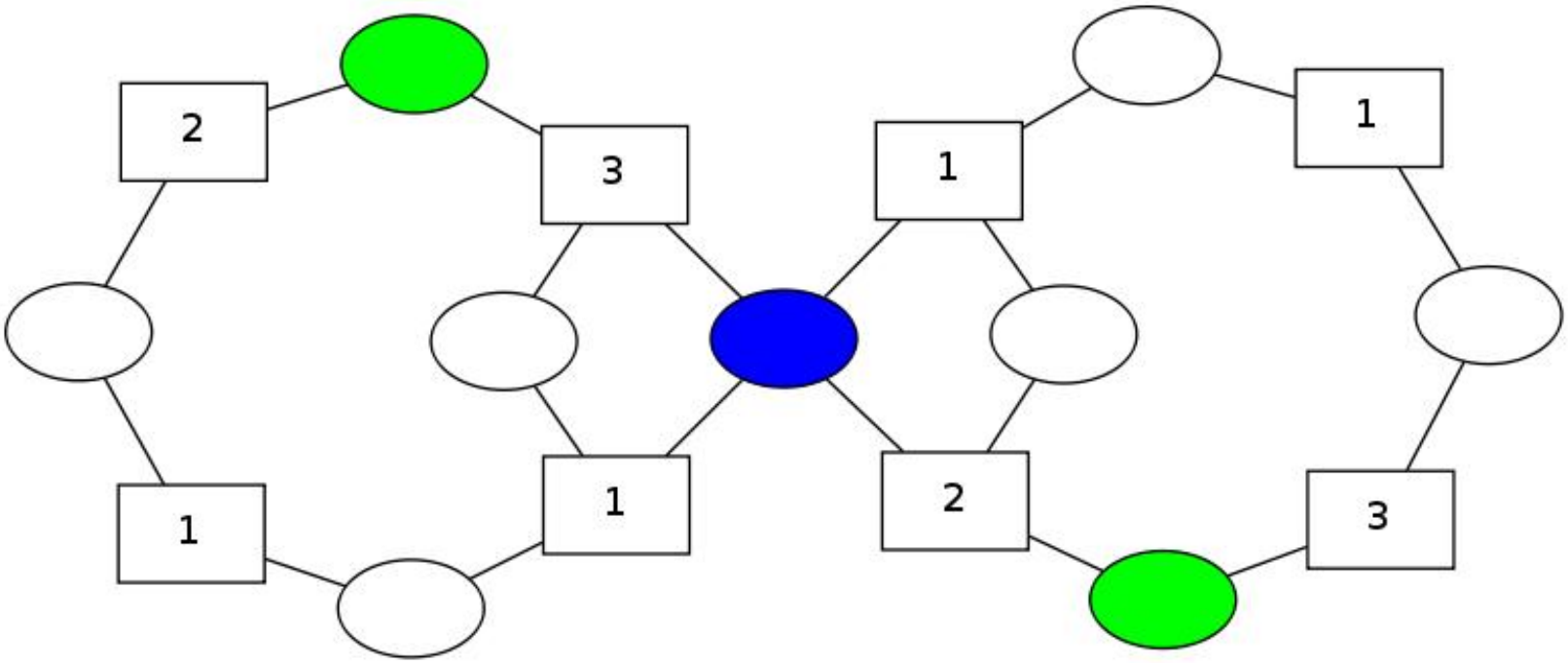
Networks with state: mutex execution



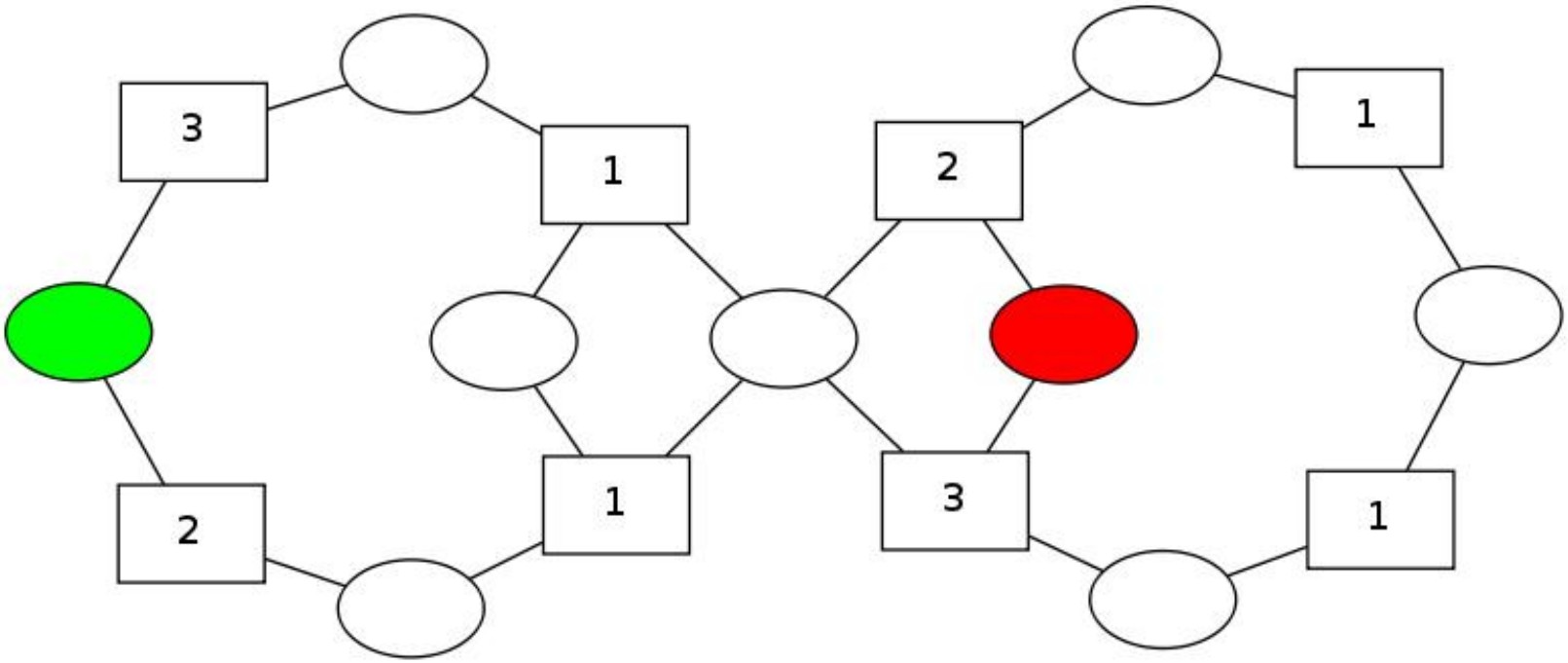
Networks with state: mutex execution



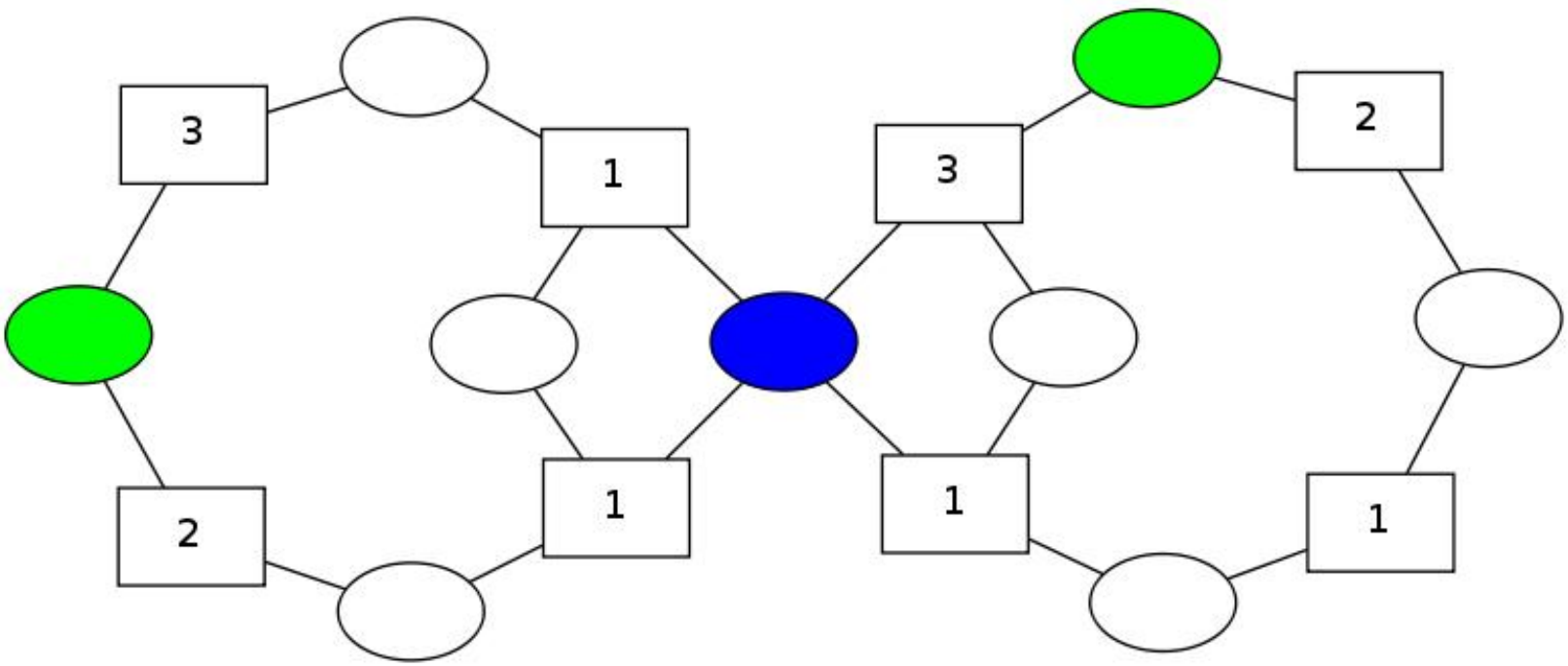
Networks with state: mutex execution



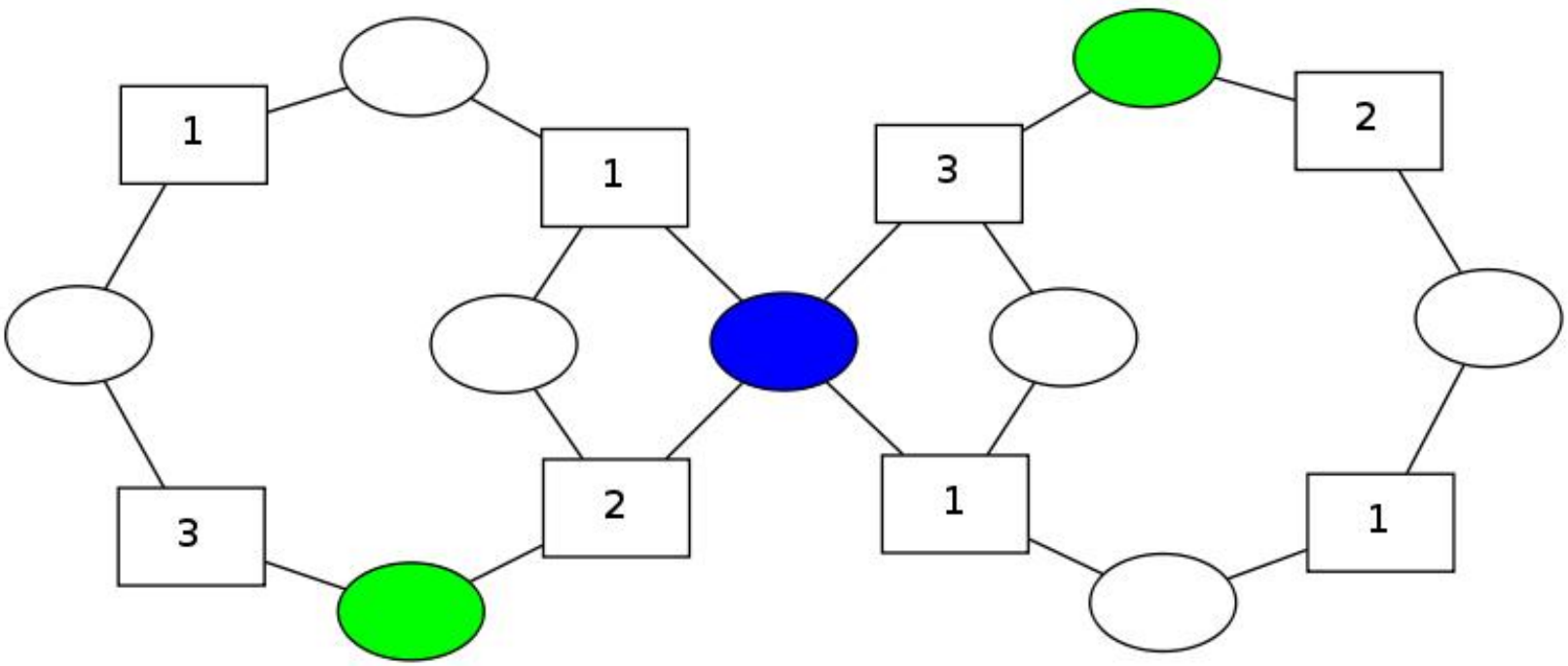
Networks with state: mutex execution



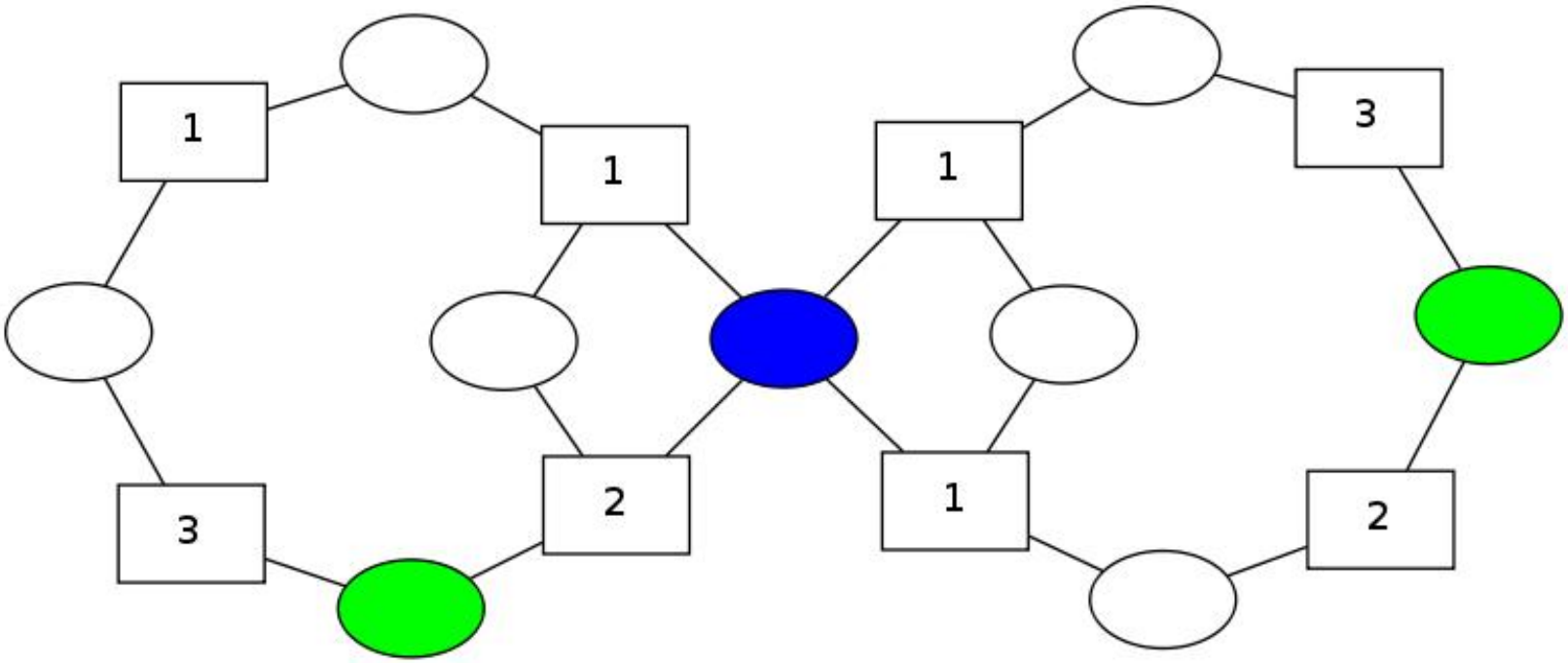
Networks with state: mutex execution



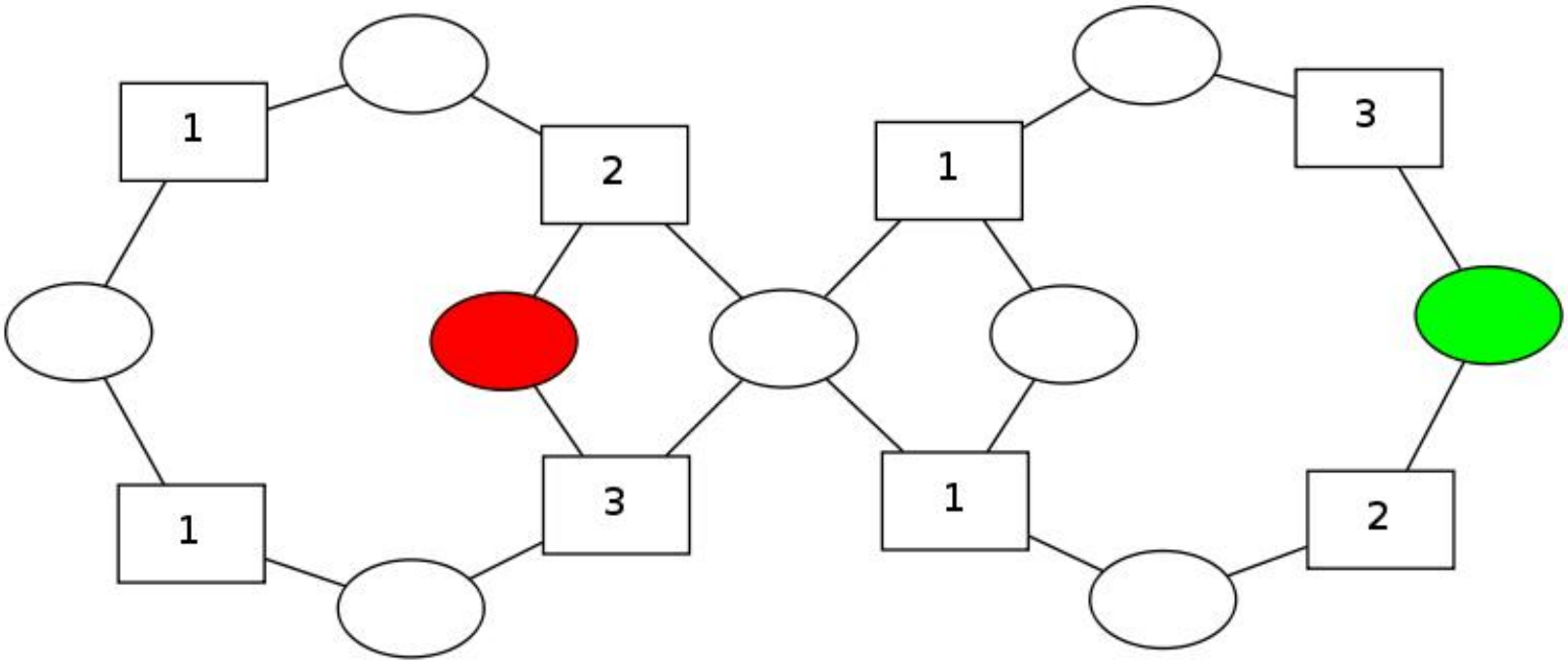
Networks with state: mutex execution



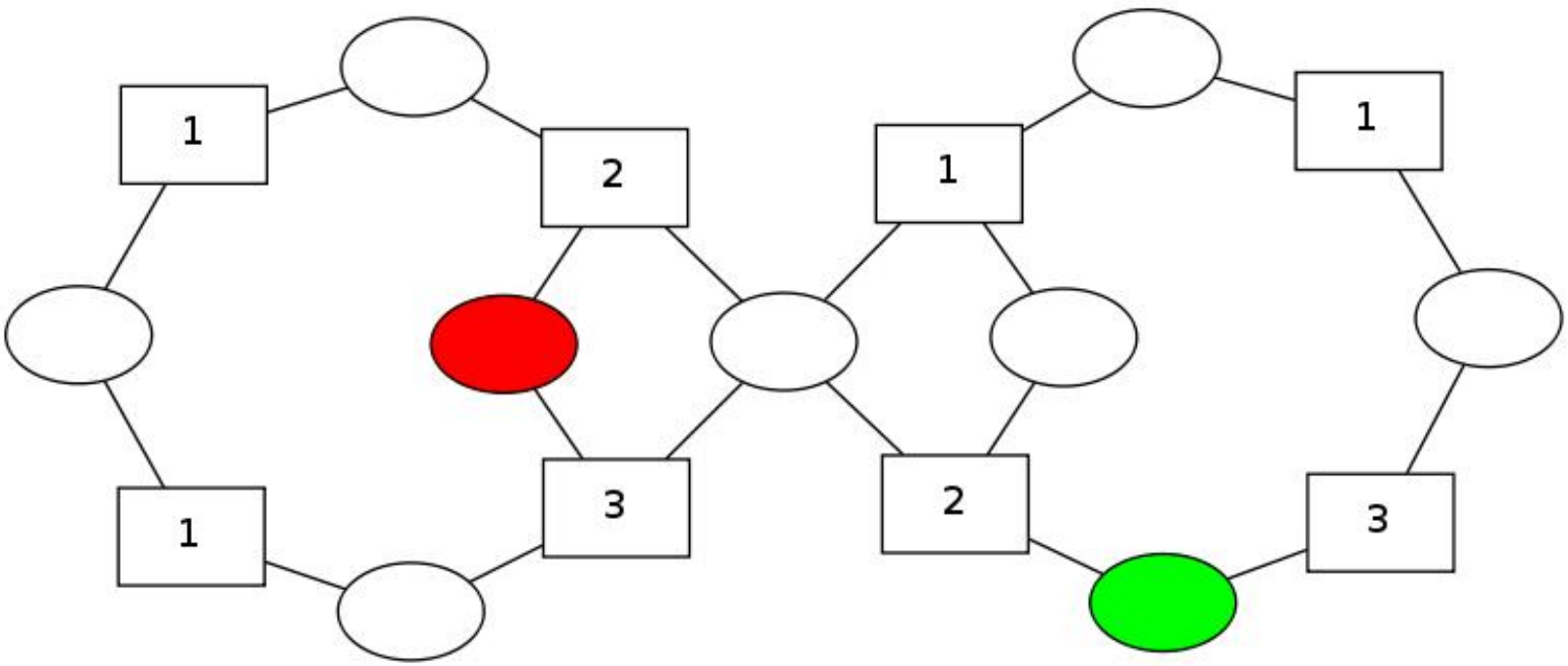
Networks with state: mutex execution



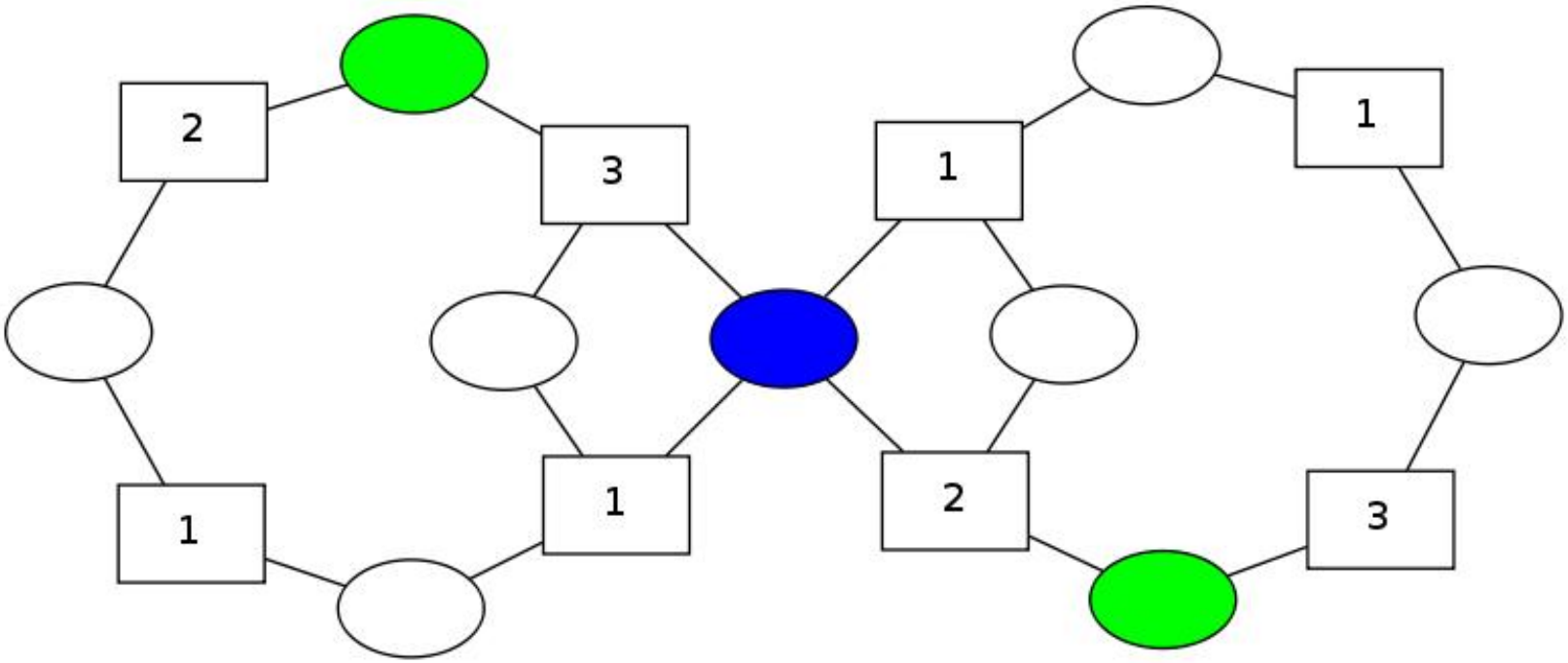
Networks with state: mutex execution



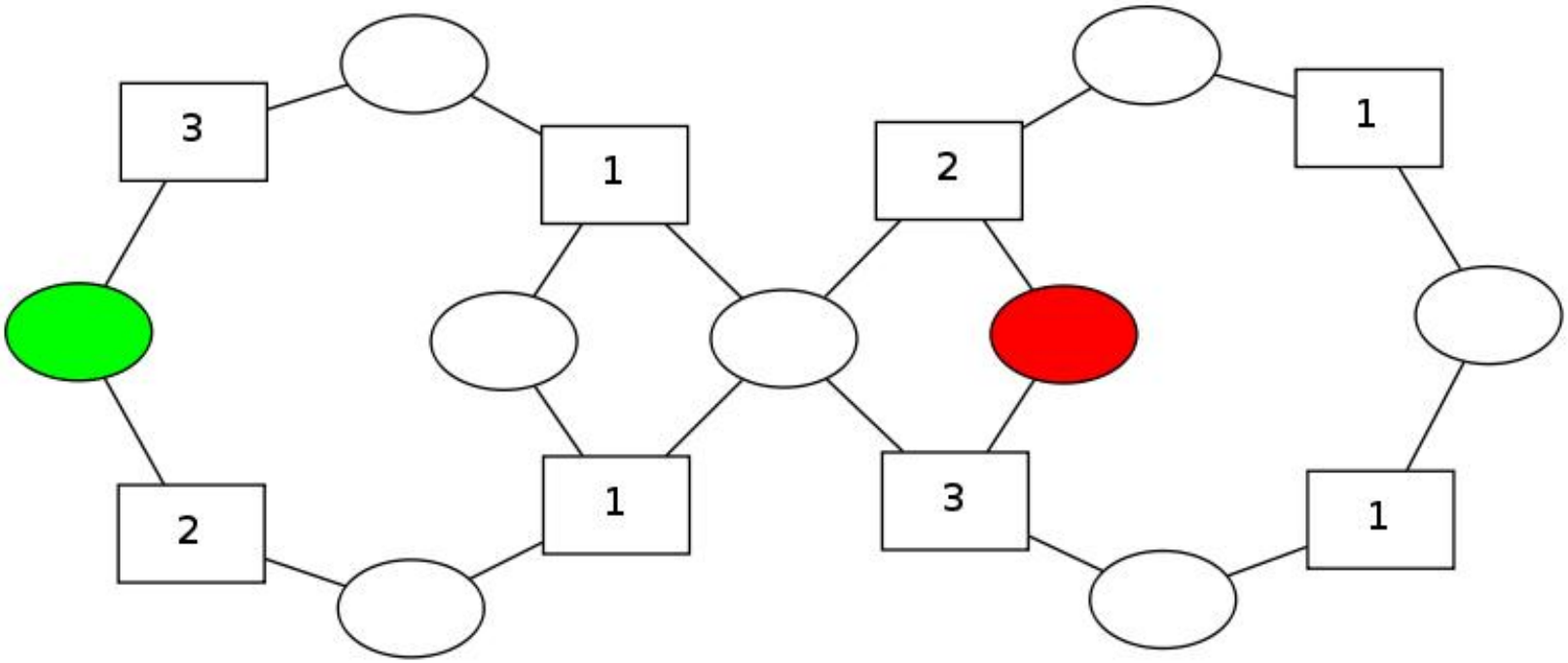
Networks with state: mutex execution



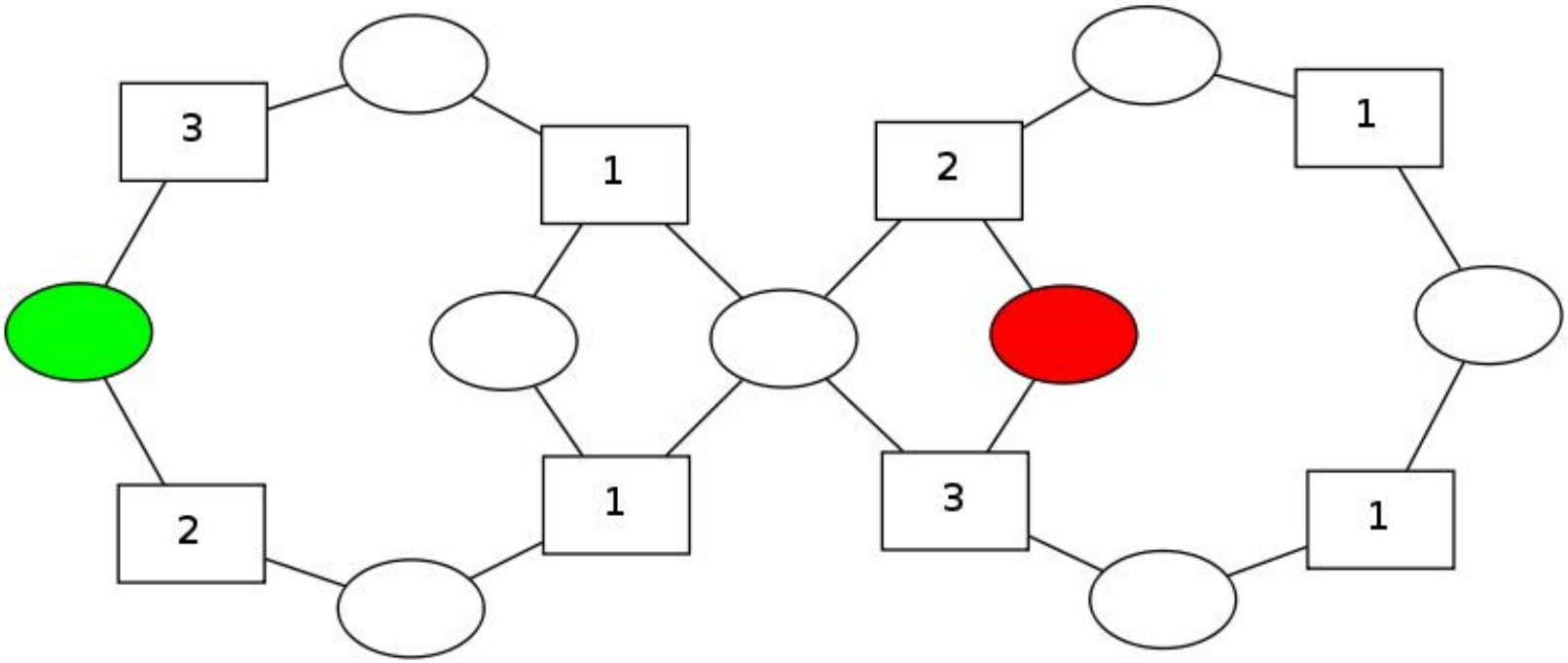
Networks with state: mutex execution



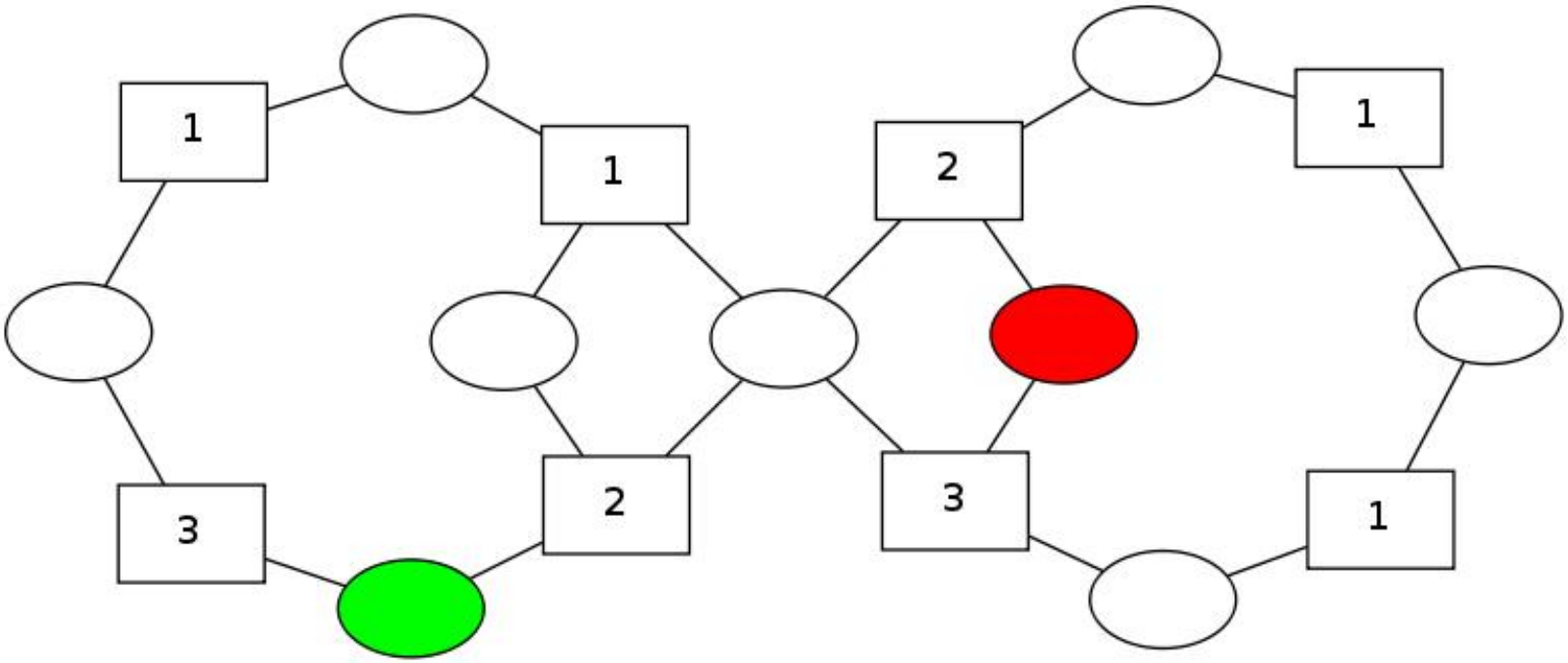
Networks with state: mutex execution



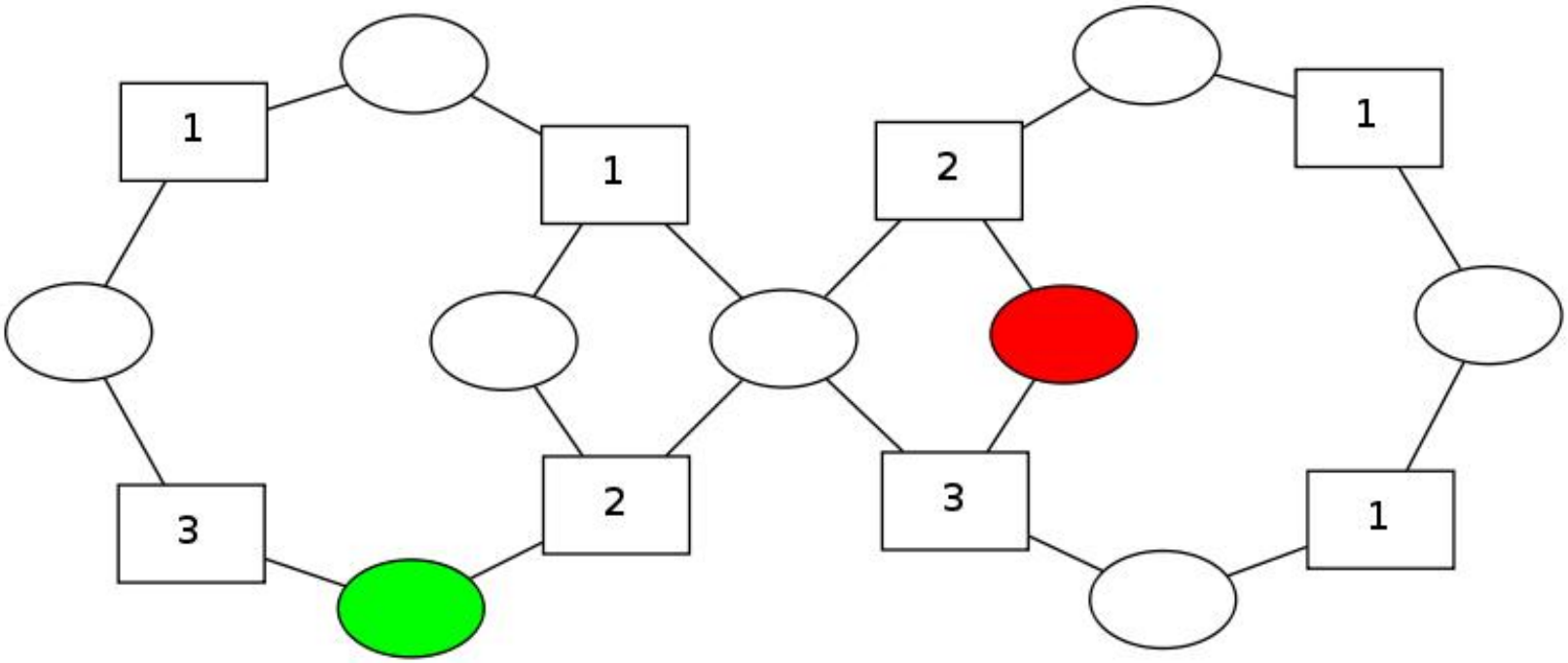
Networks with state: mutex execution



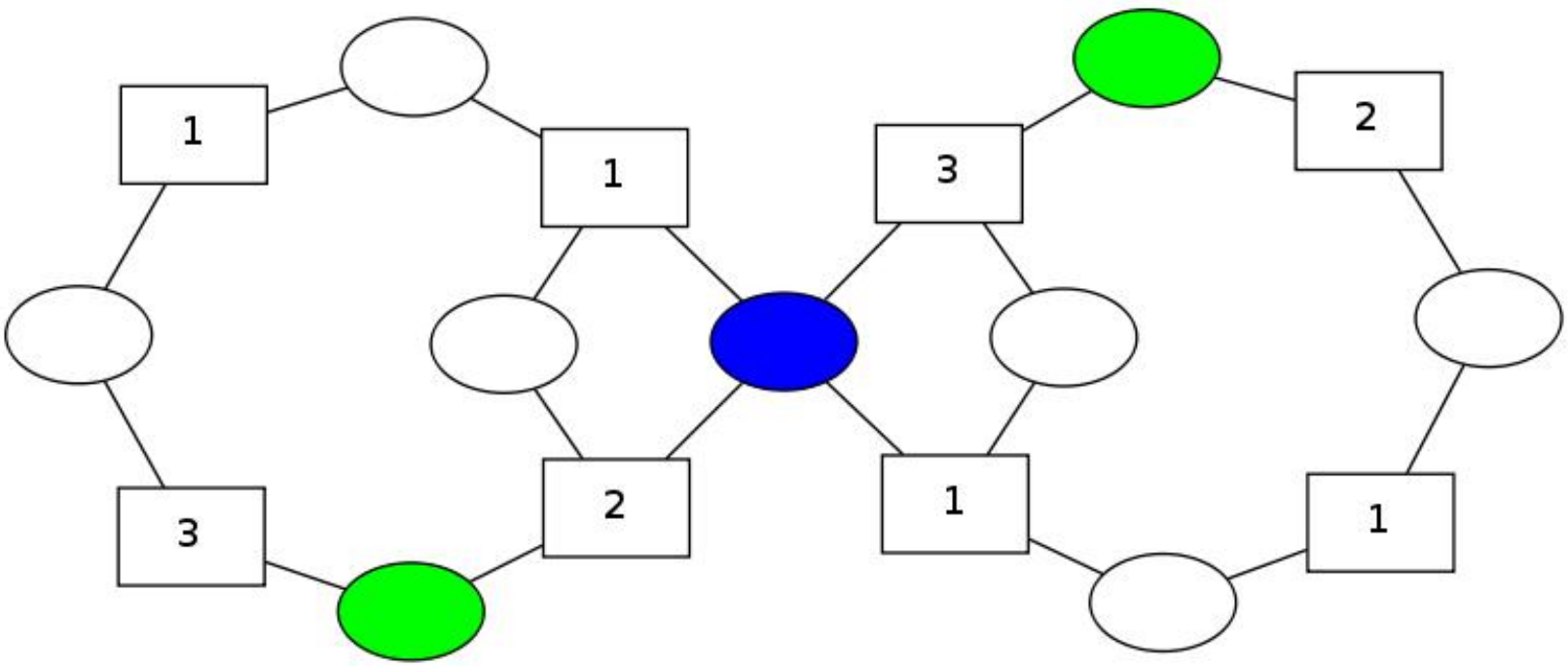
Networks with state: mutex execution



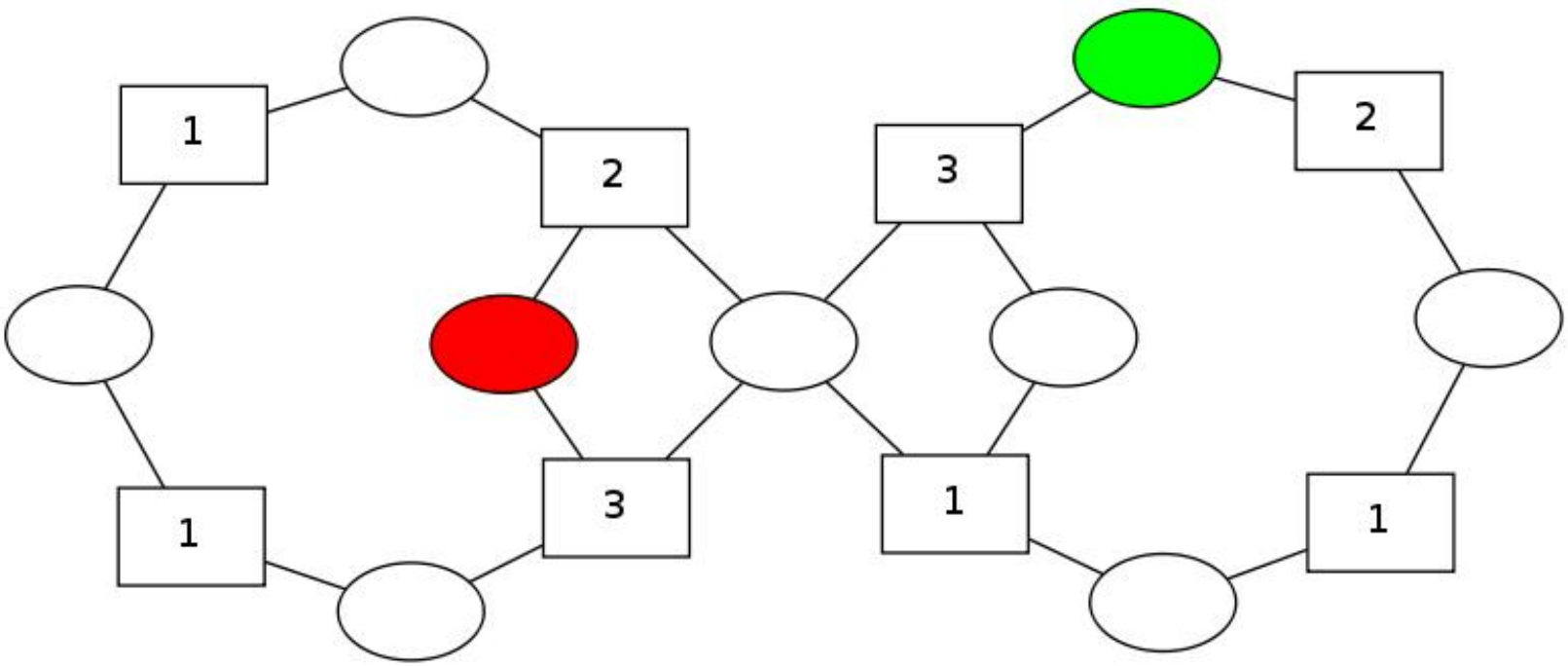
Networks with state: mutex execution



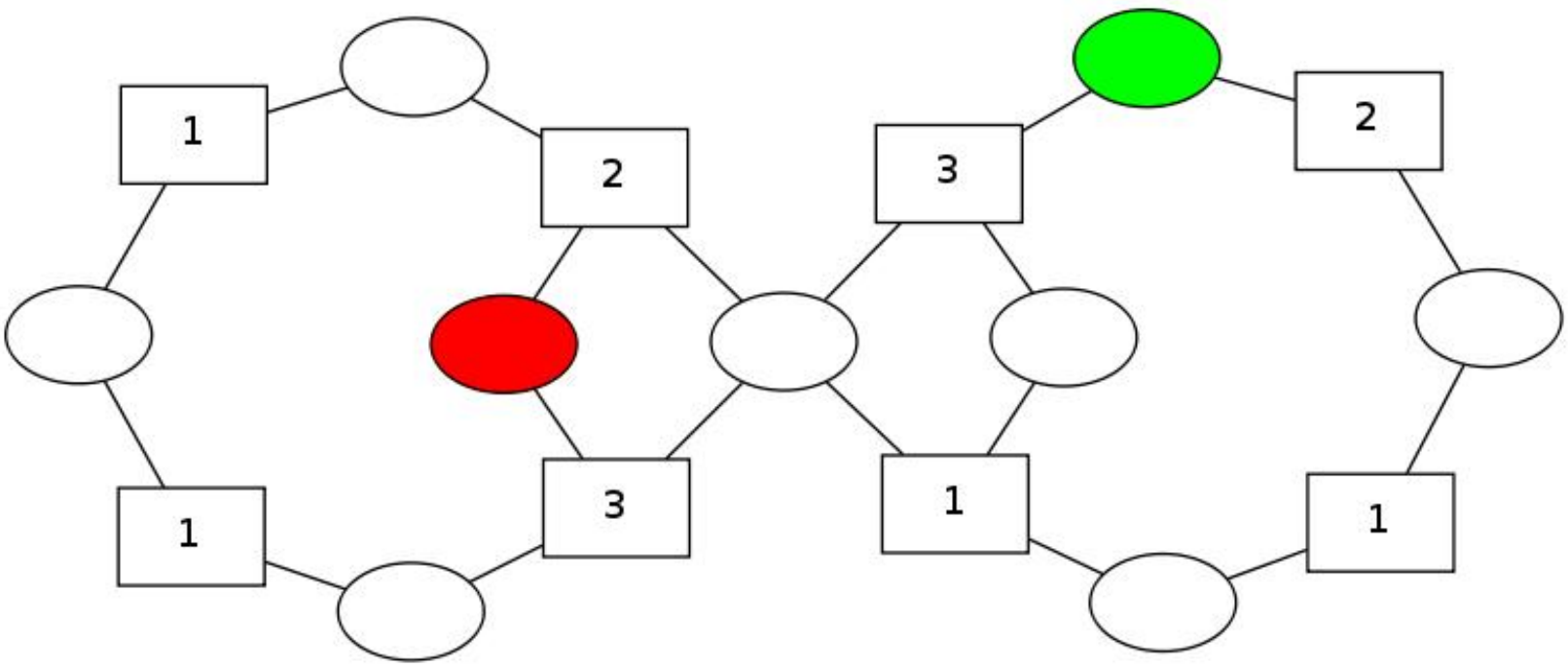
Networks with state: mutex execution



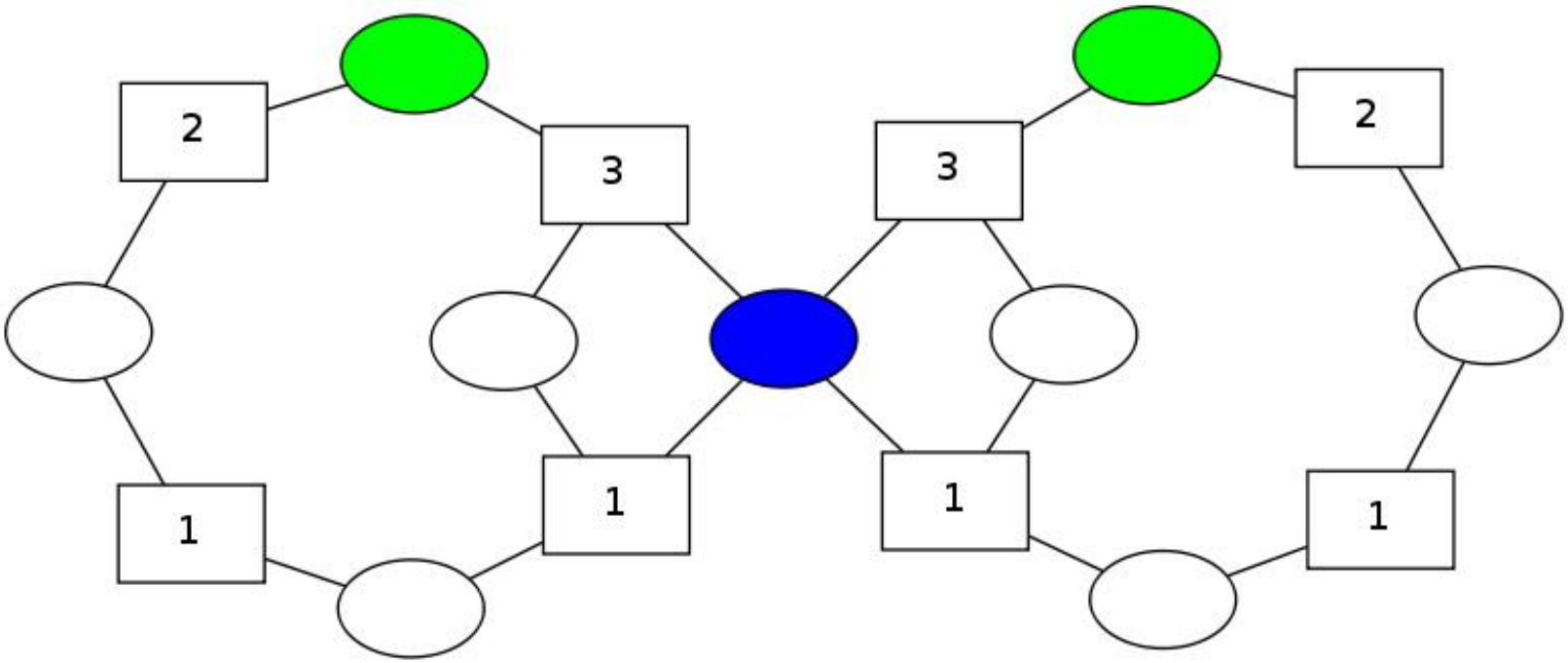
Networks with state: mutex execution



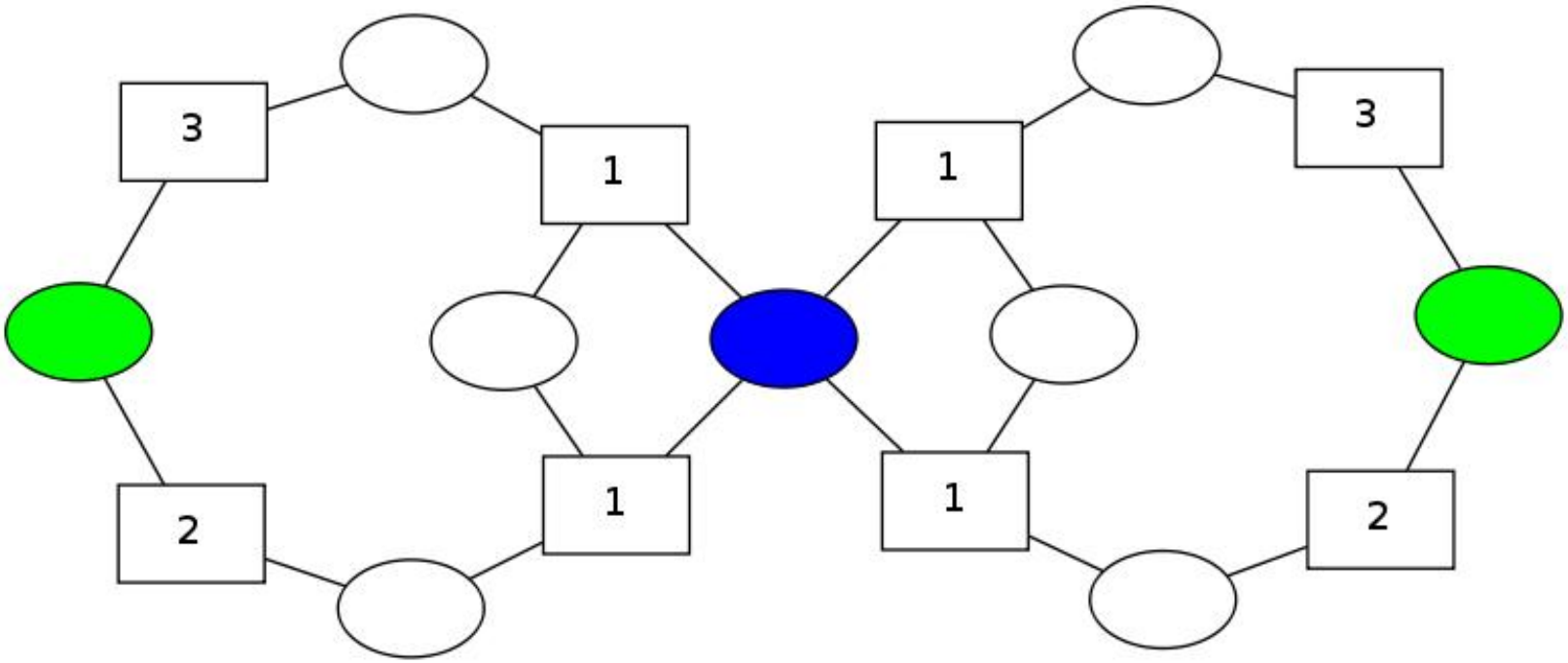
Networks with state: mutex execution



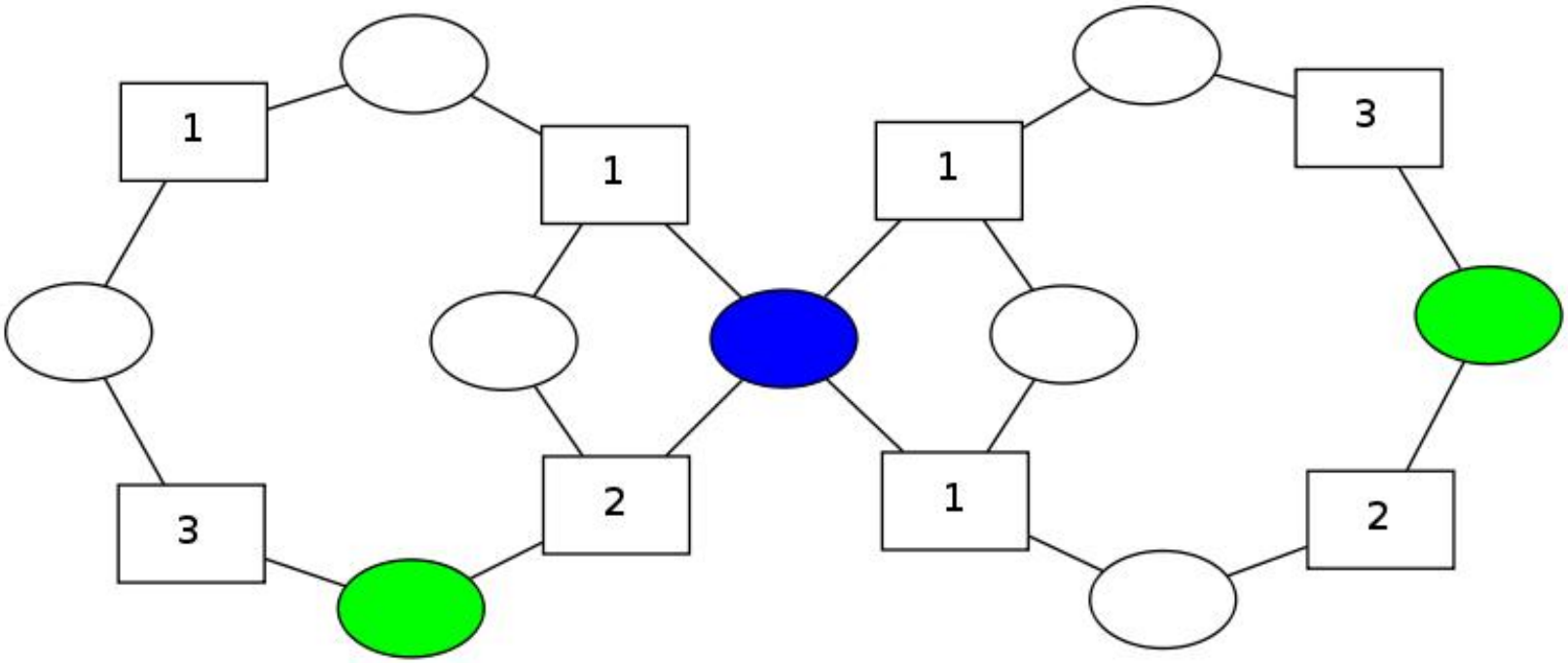
Networks with state: mutex execution



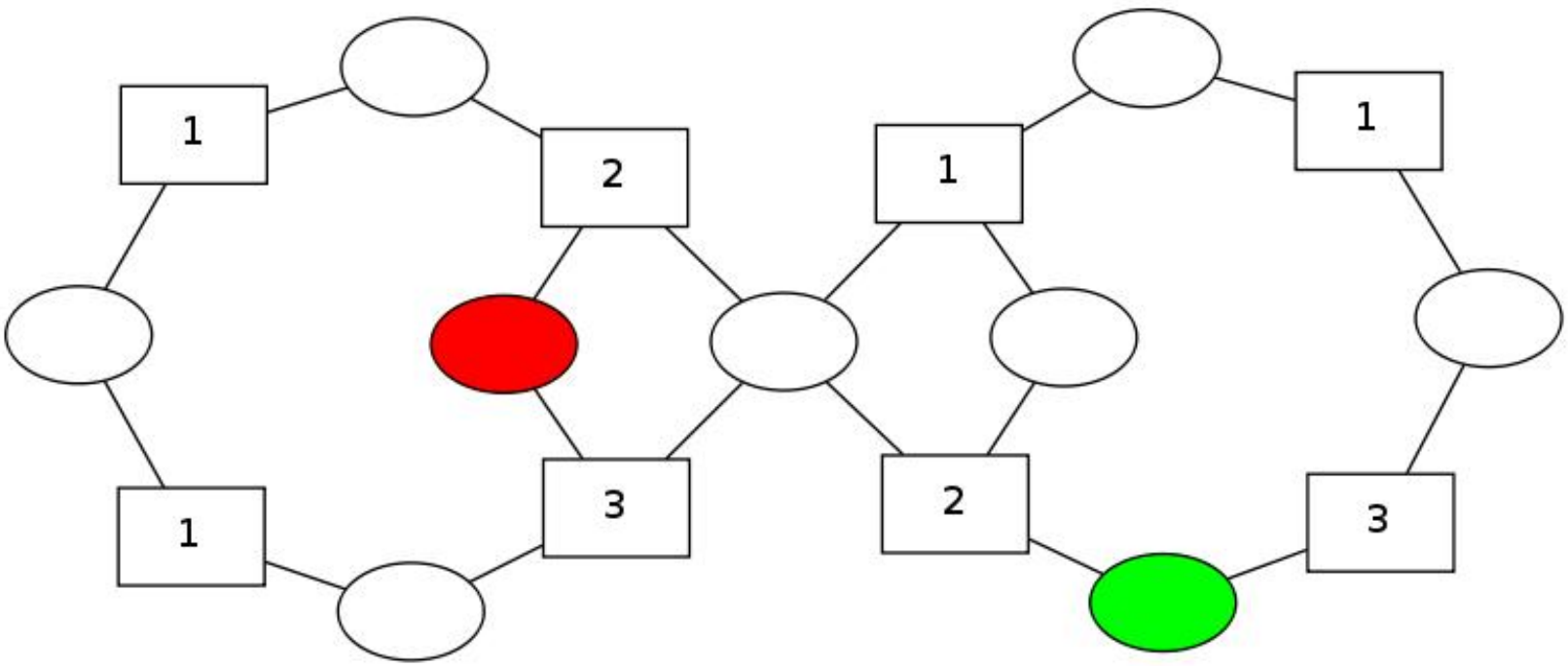
Networks with state: mutex execution



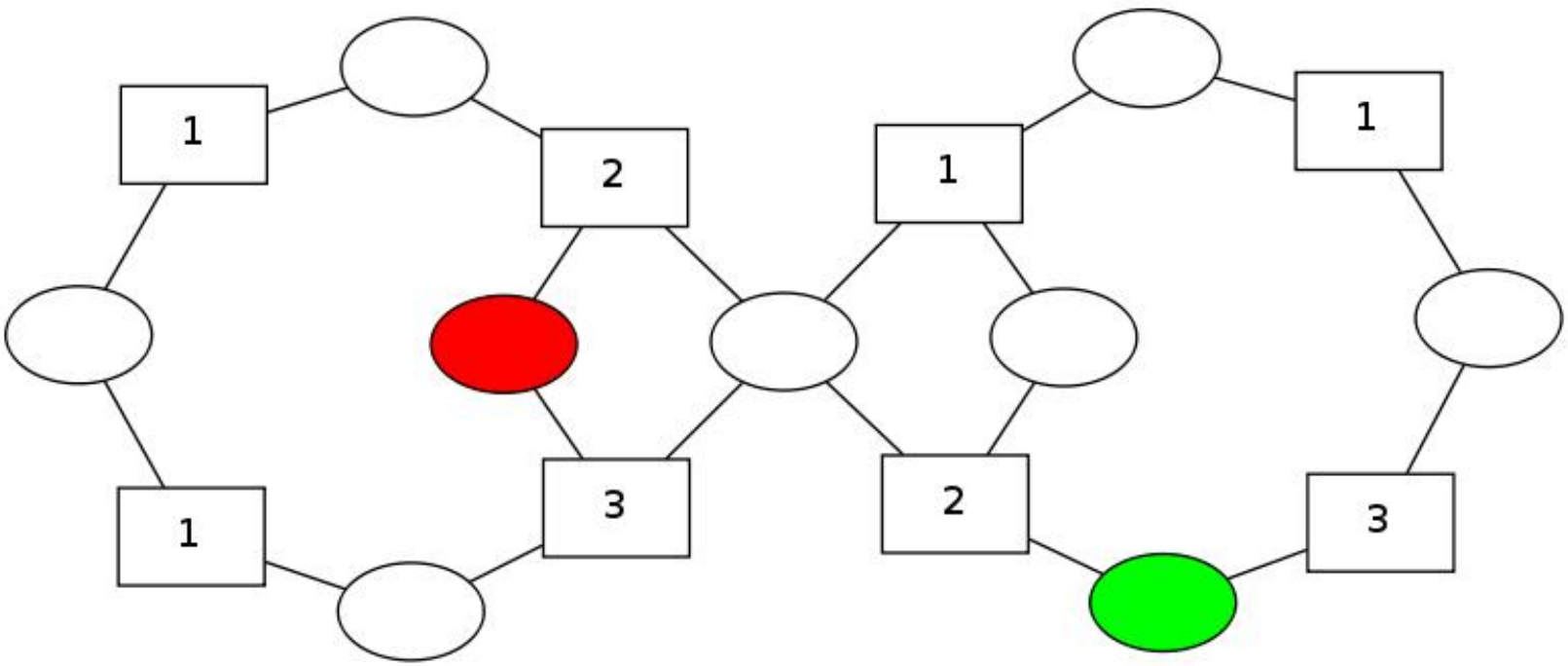
Networks with state: mutex execution



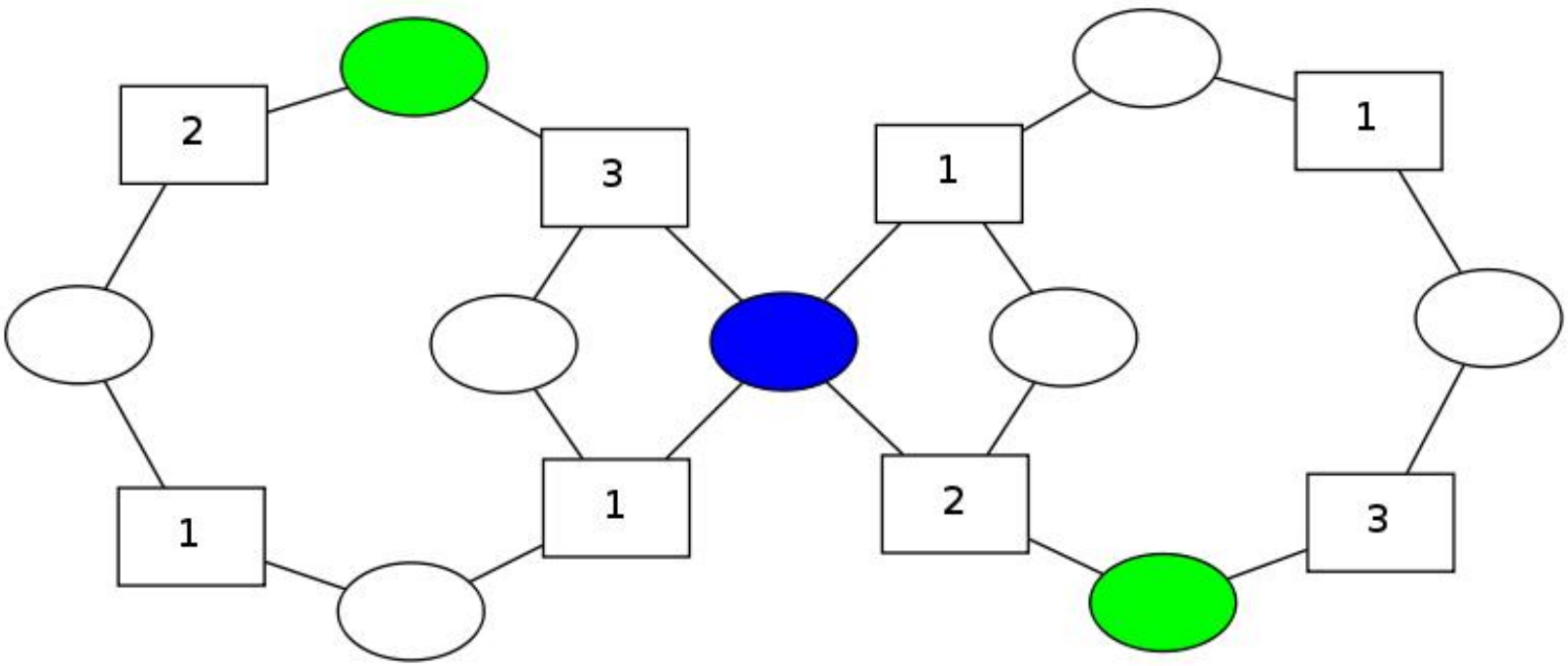
Networks with state: mutex execution



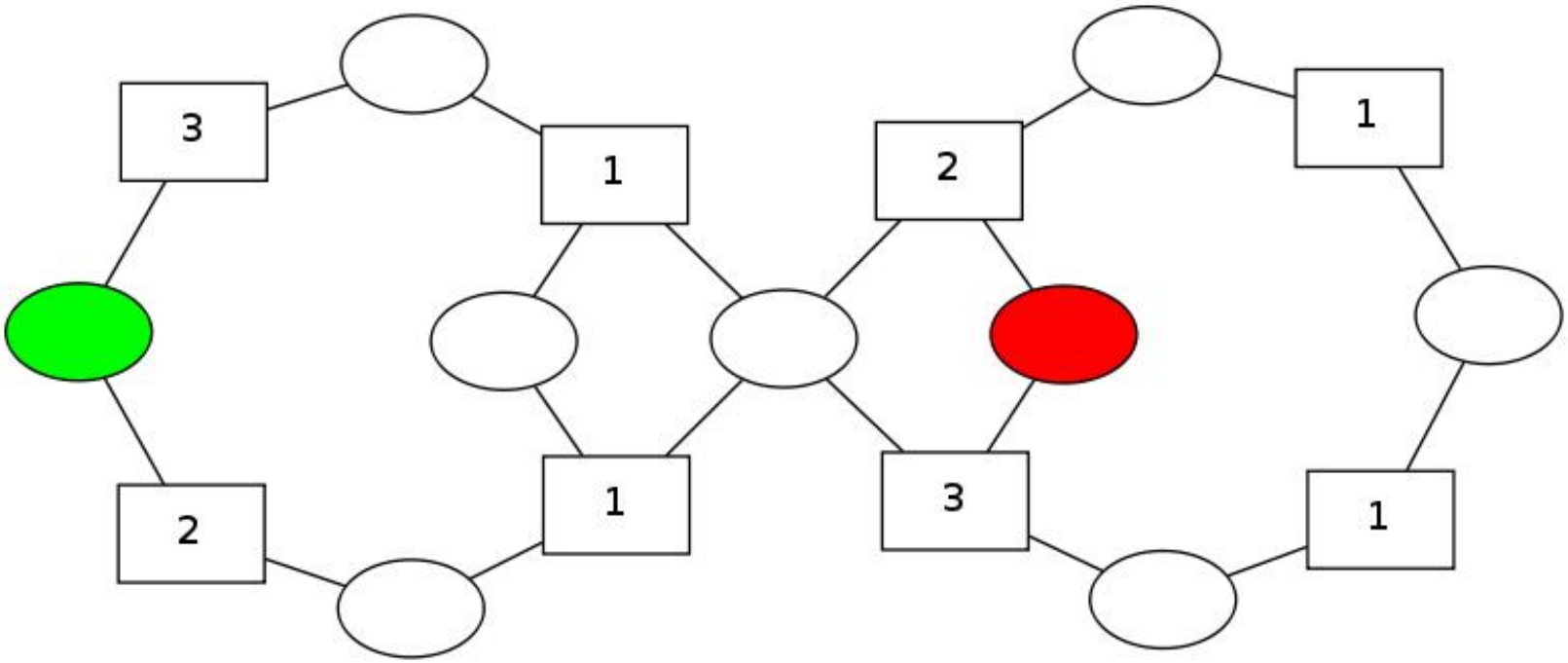
Networks with state: mutex execution



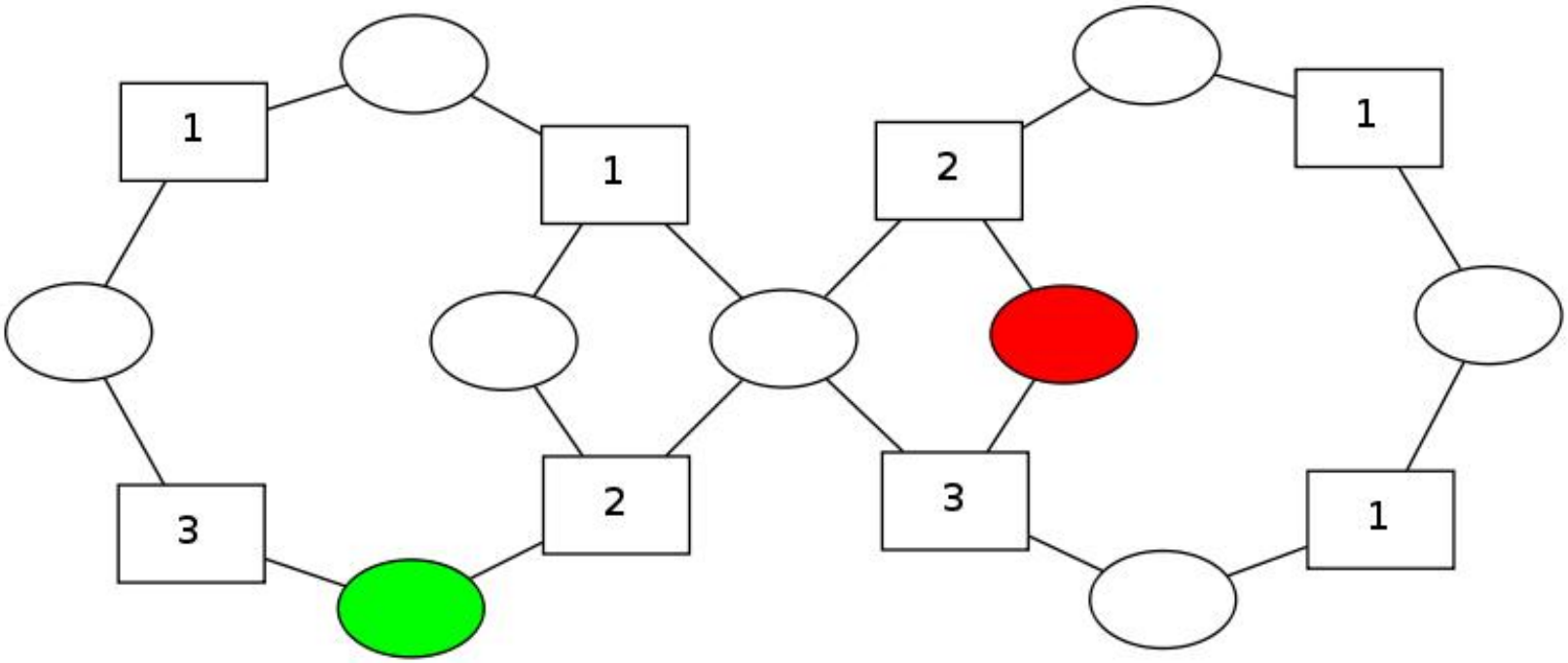
Networks with state



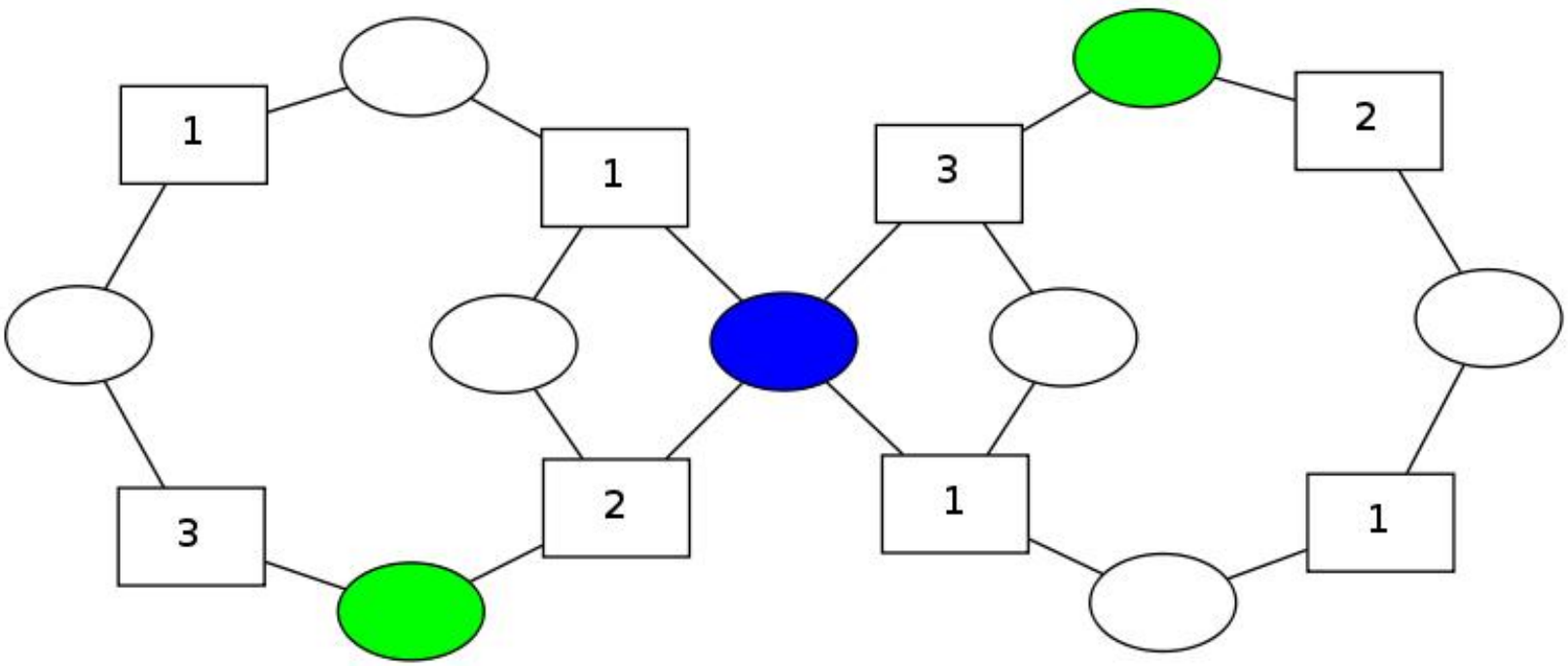
Networks with state



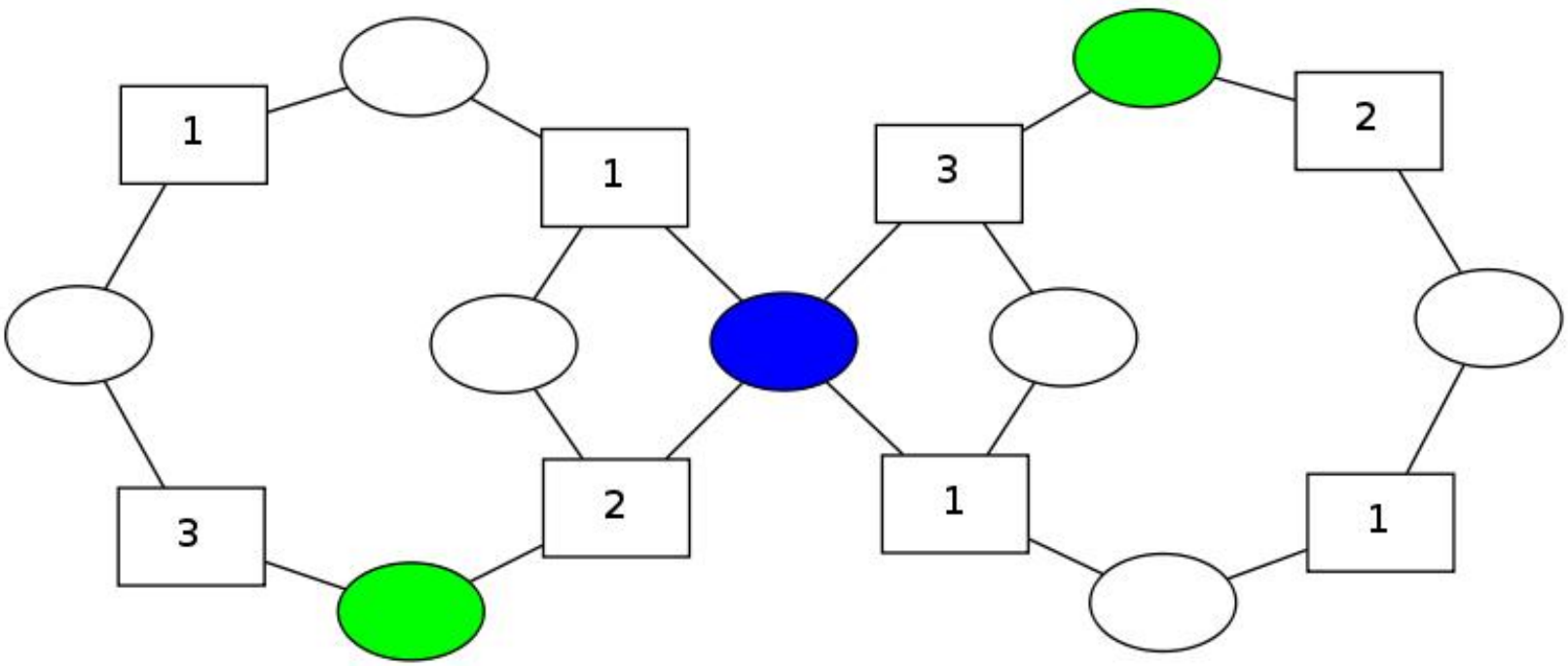
Networks with state



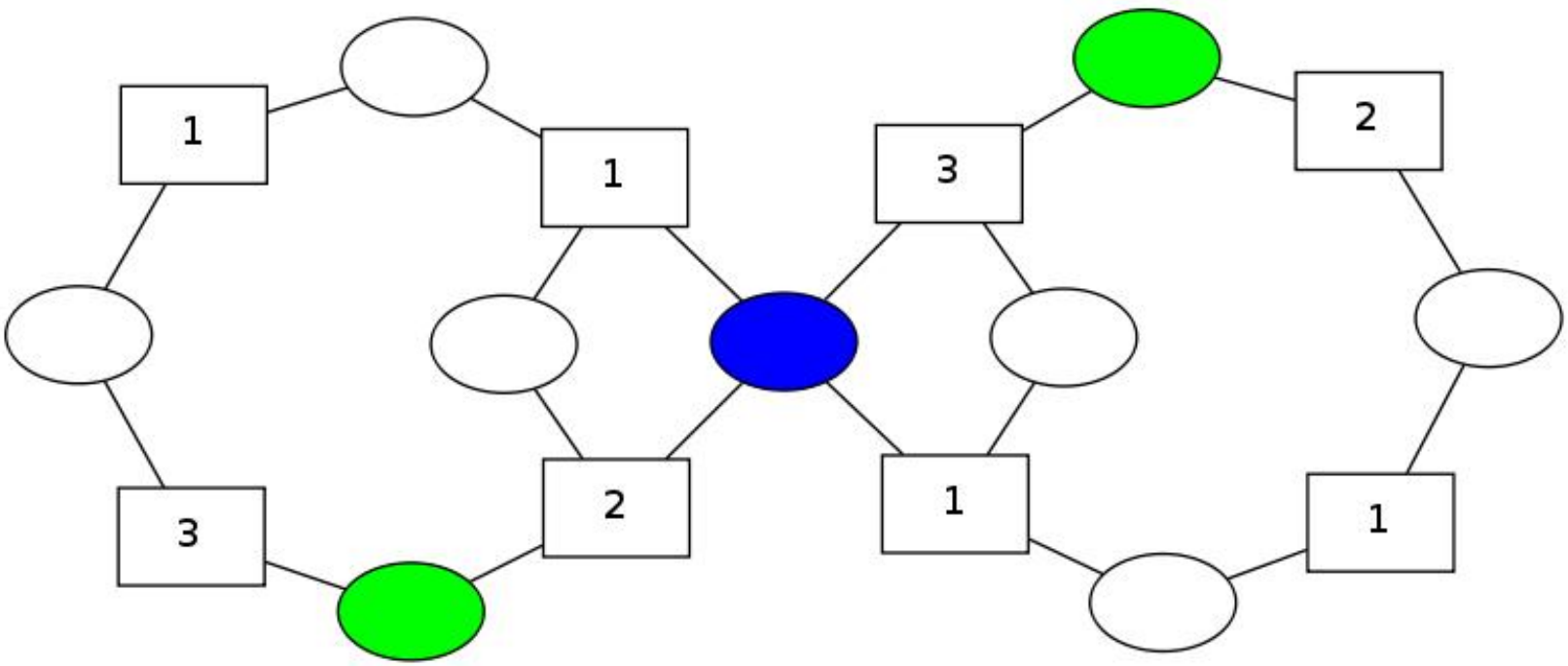
Networks with state



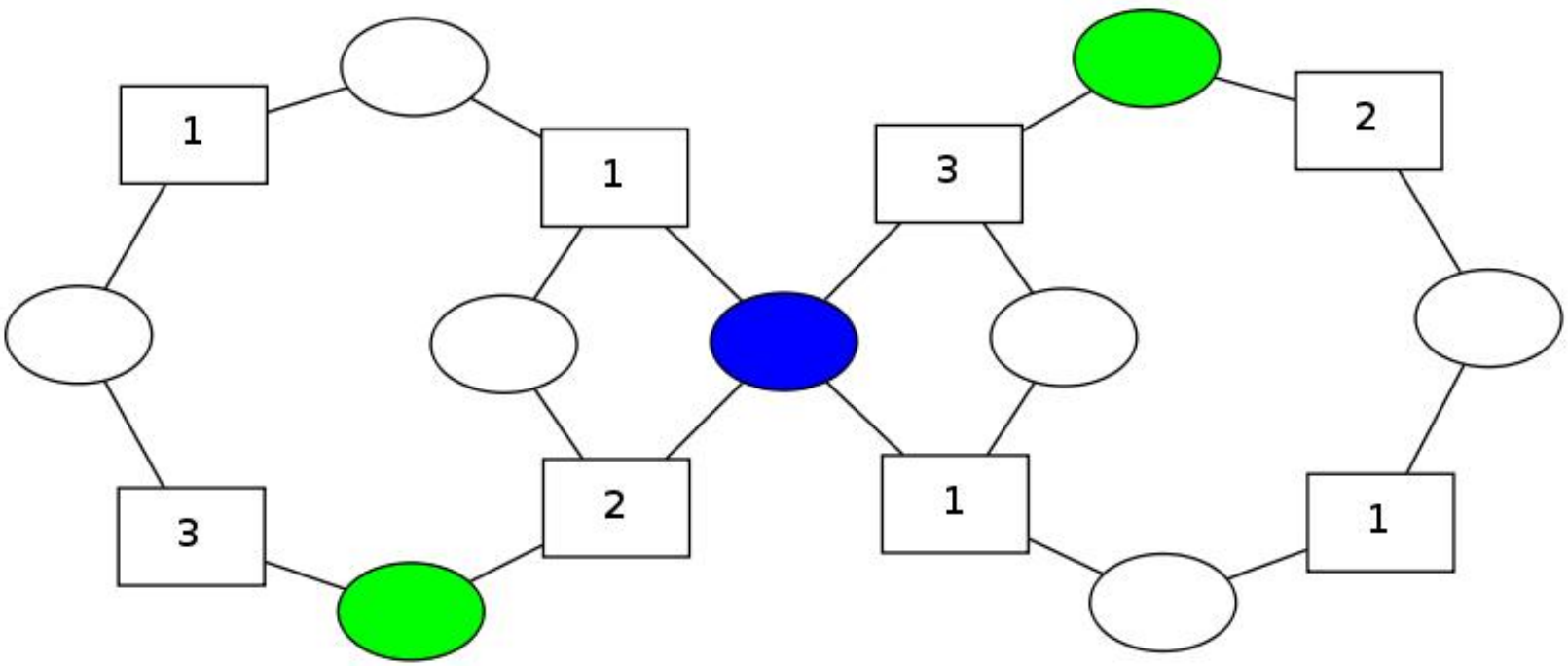
Networks with state



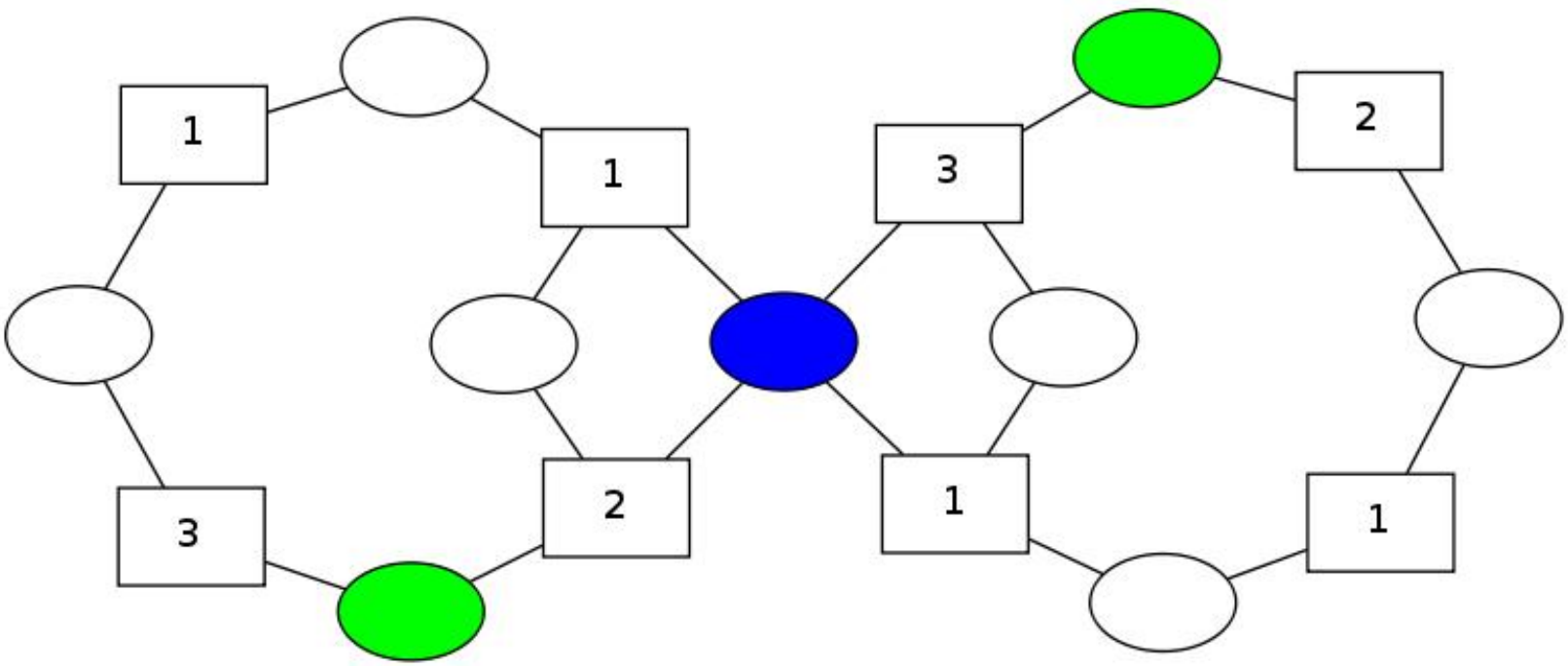
Networks with state



Networks with state



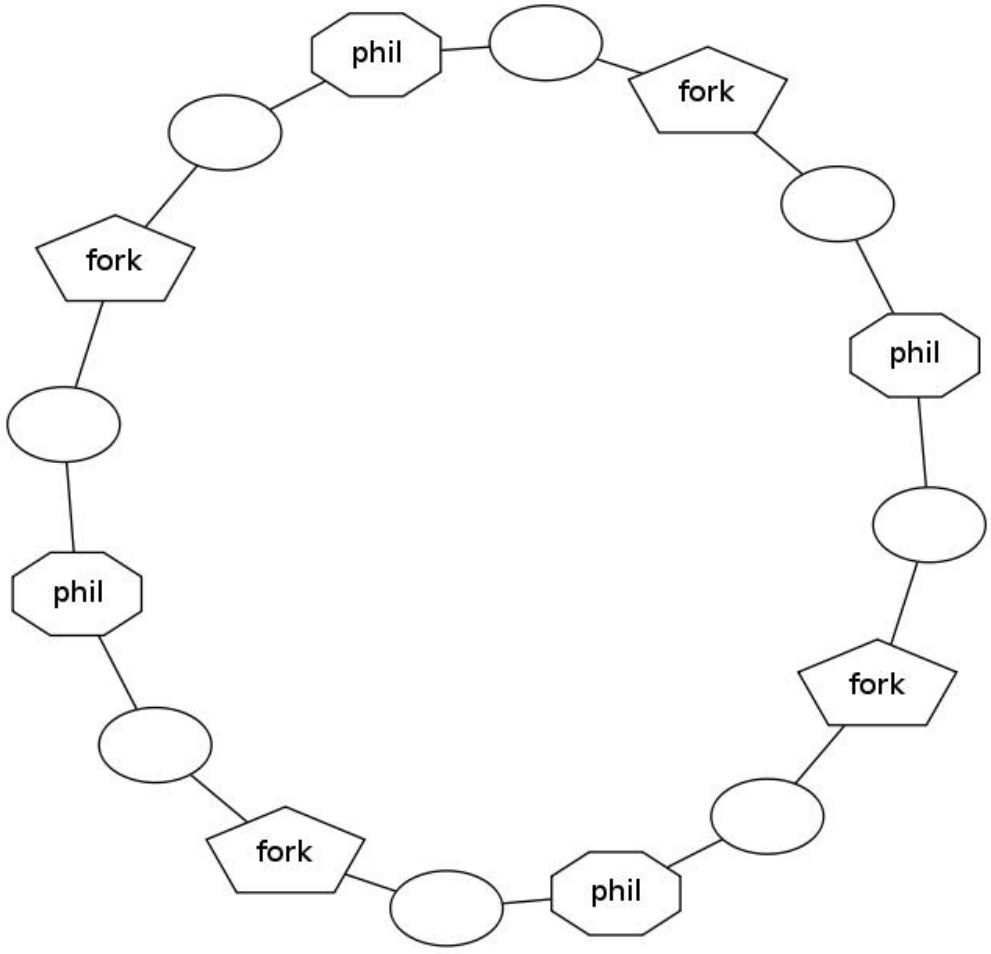
Networks with state



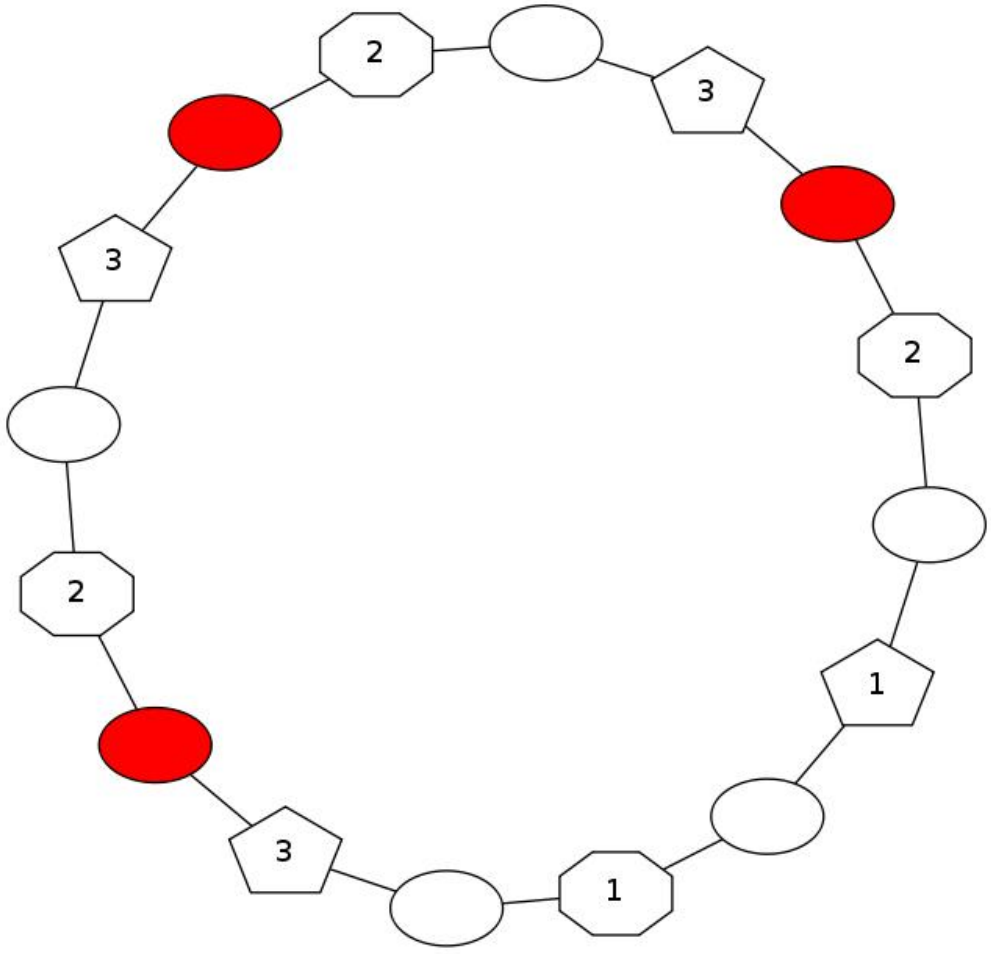
Networks with state: geometry plus components with states and transitions

This last example might give the impression that Networks with state are just Petri nets. Hence we give an example where the separation between the geometry and the component automata is clearer. In the Dining Philosopher Problem the geometry is a circle of Philosophers and Forks but the Philosophers and Forks are communicating components with states and transitions.

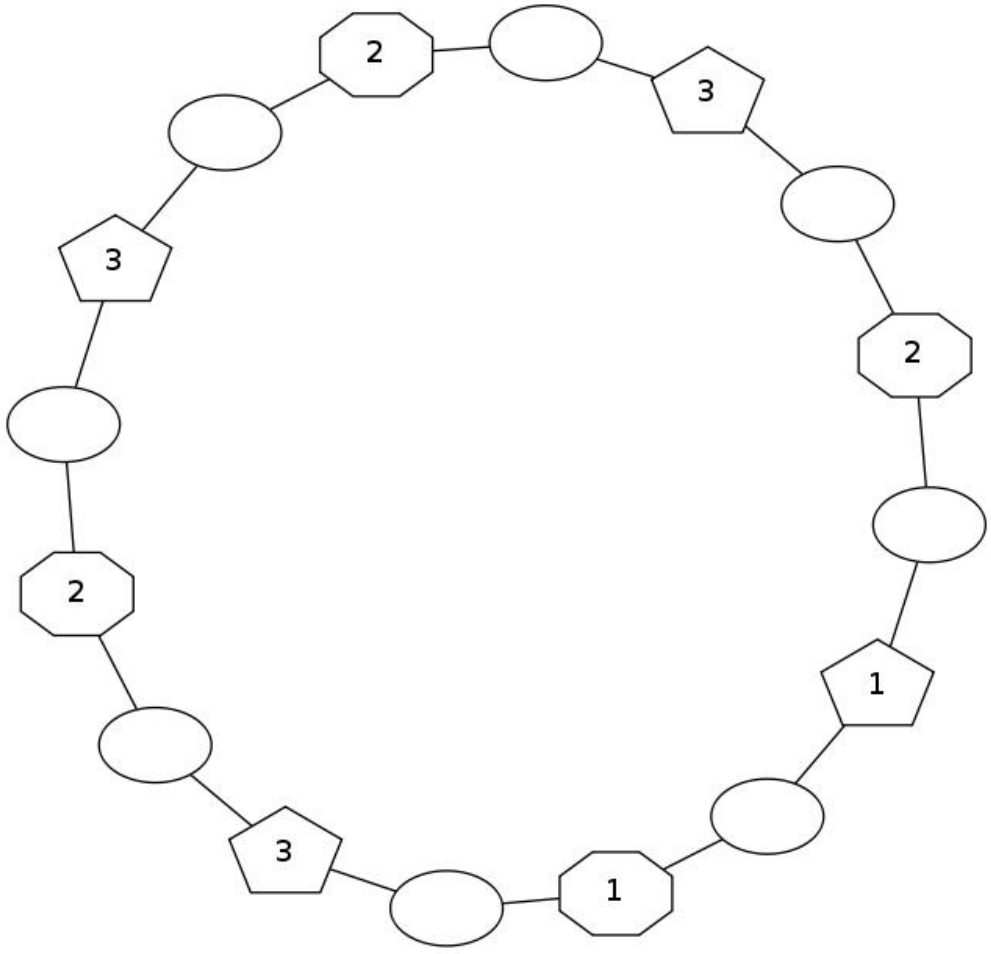
Networks with state: Dining Philosophers



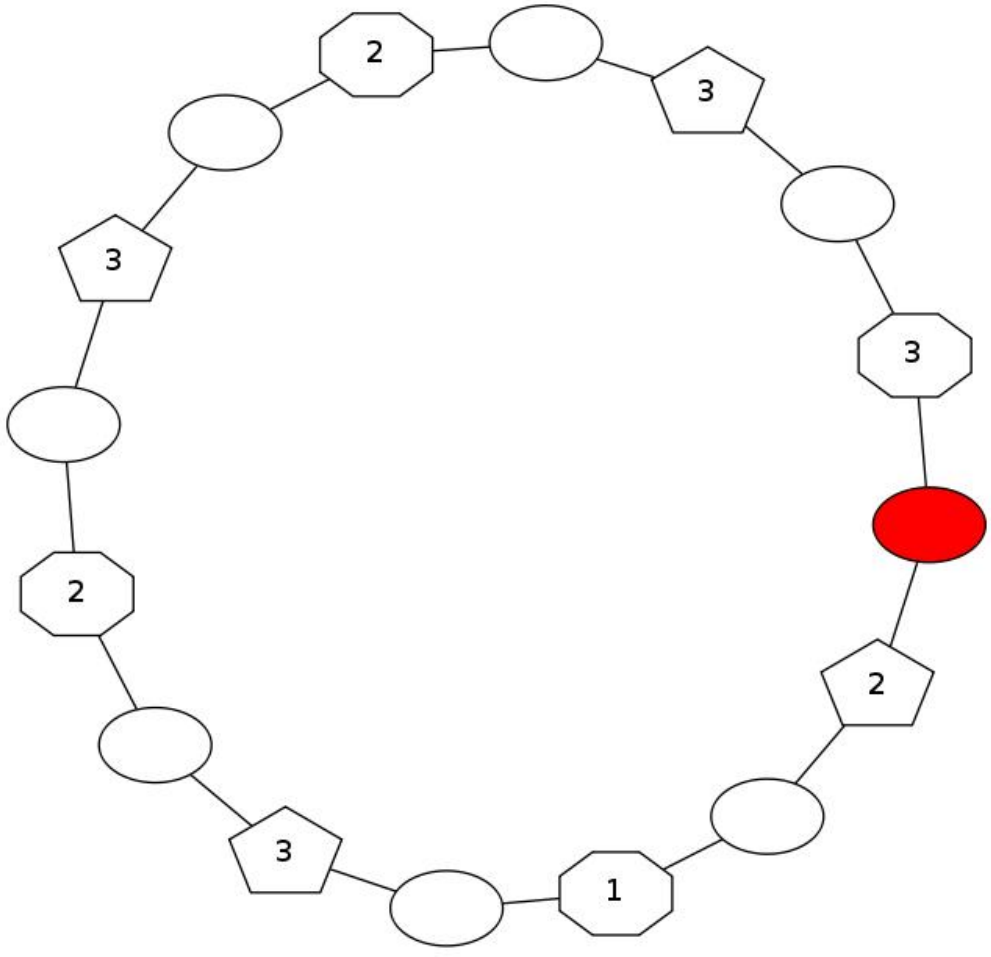
Networks with state: Dining Philosophers



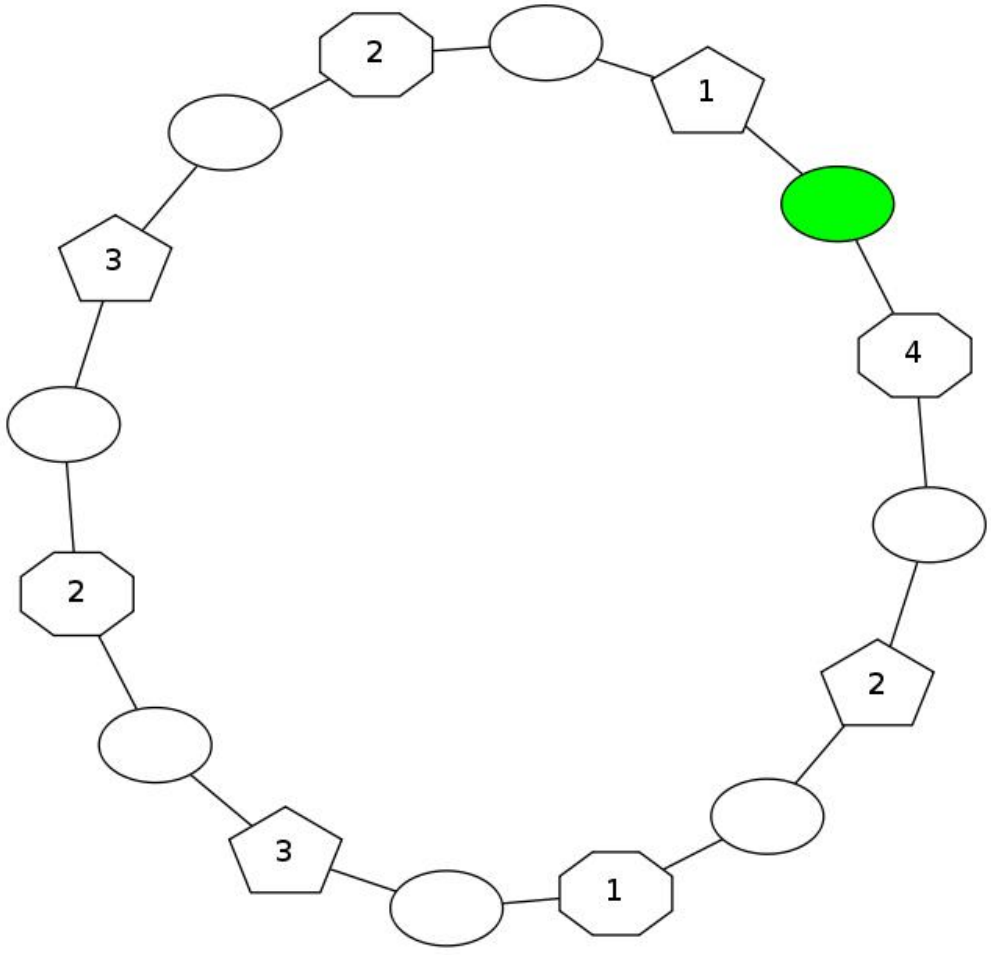
Networks with state: Dining Philosophers



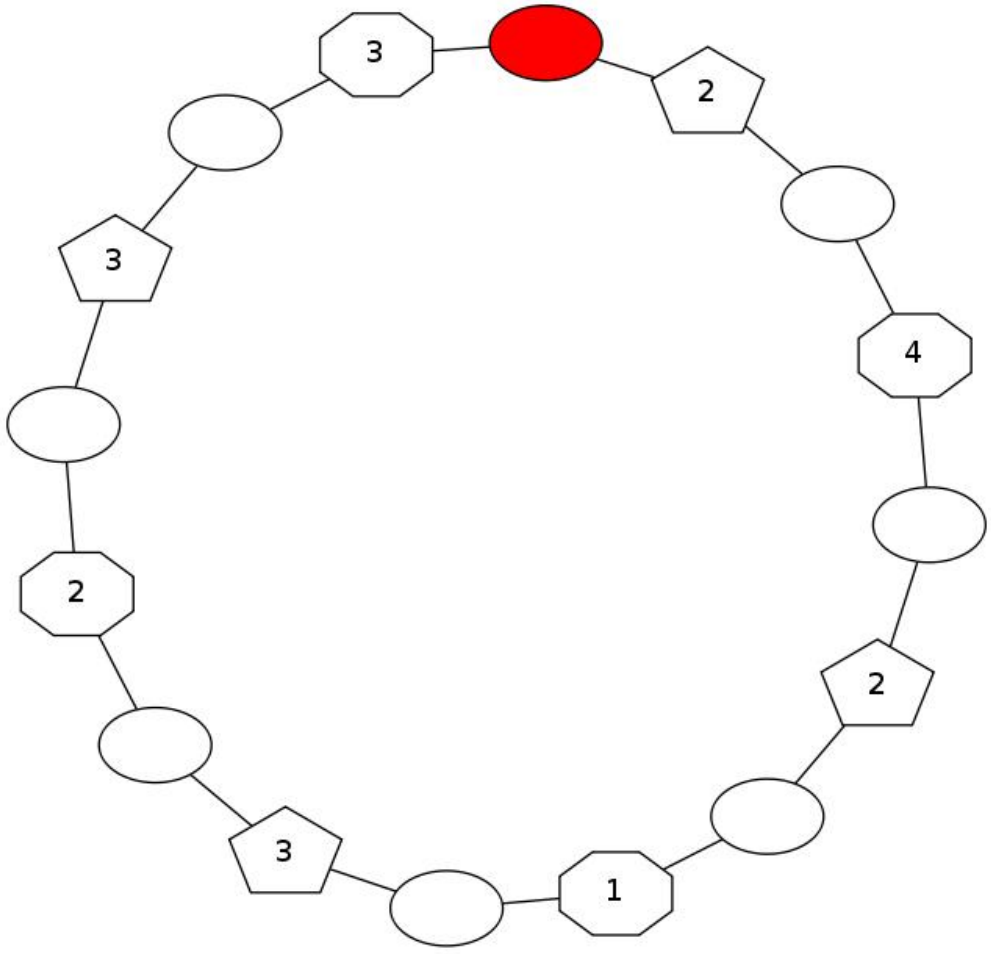
Networks with state: Dining Philosophers



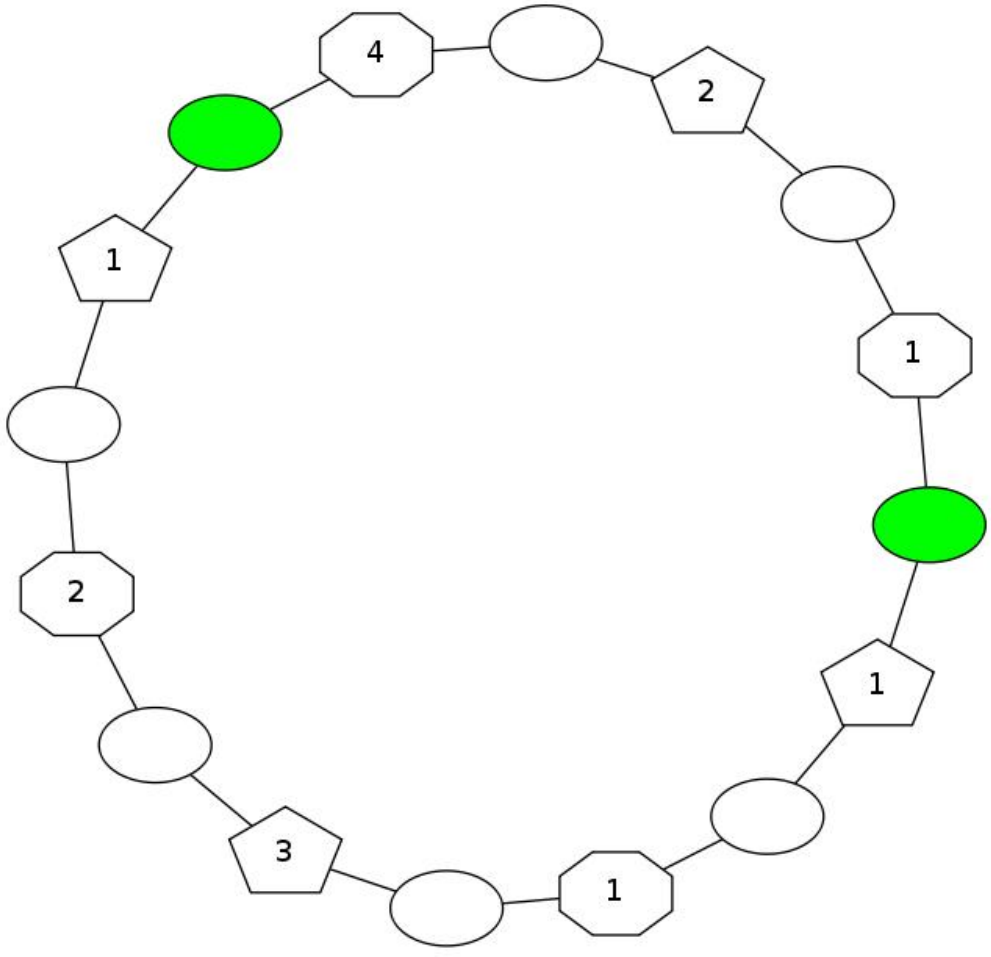
Networks with state: Dining Philosophers



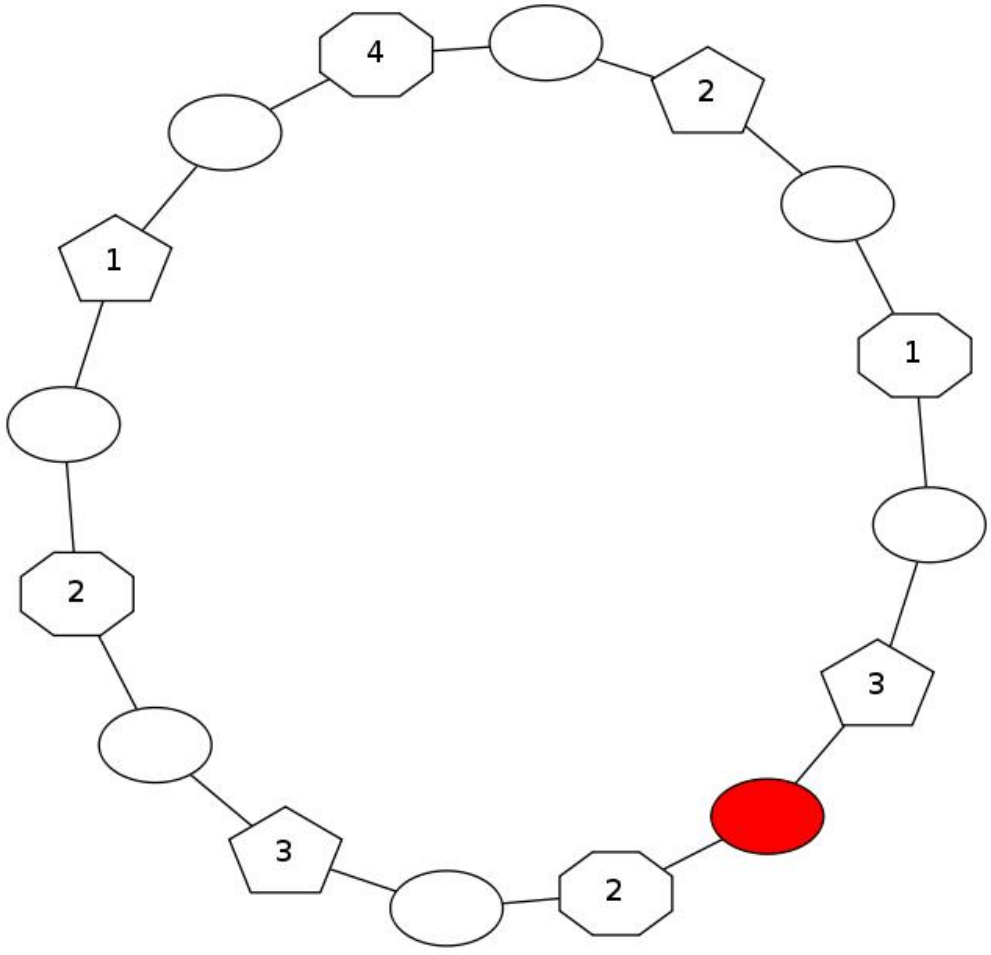
Networks with state: Dining Philosophers



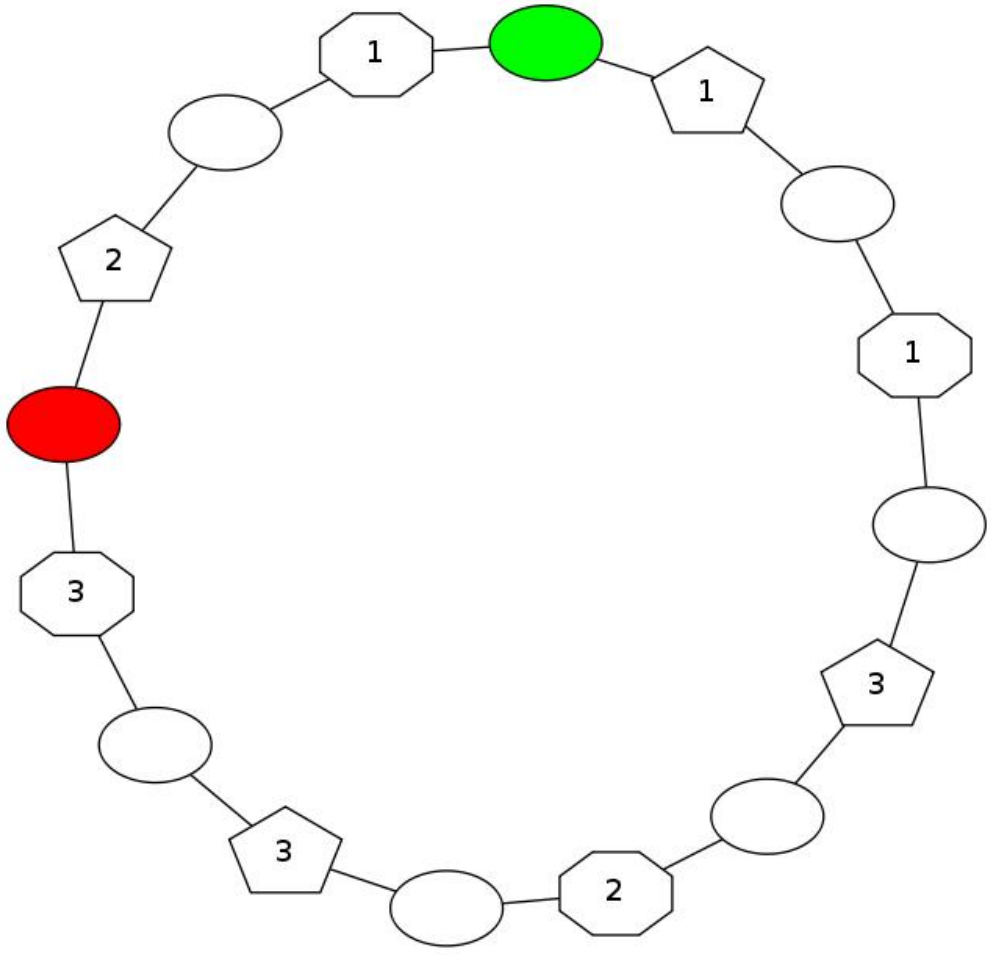
Networks with state: Dining Philosophers



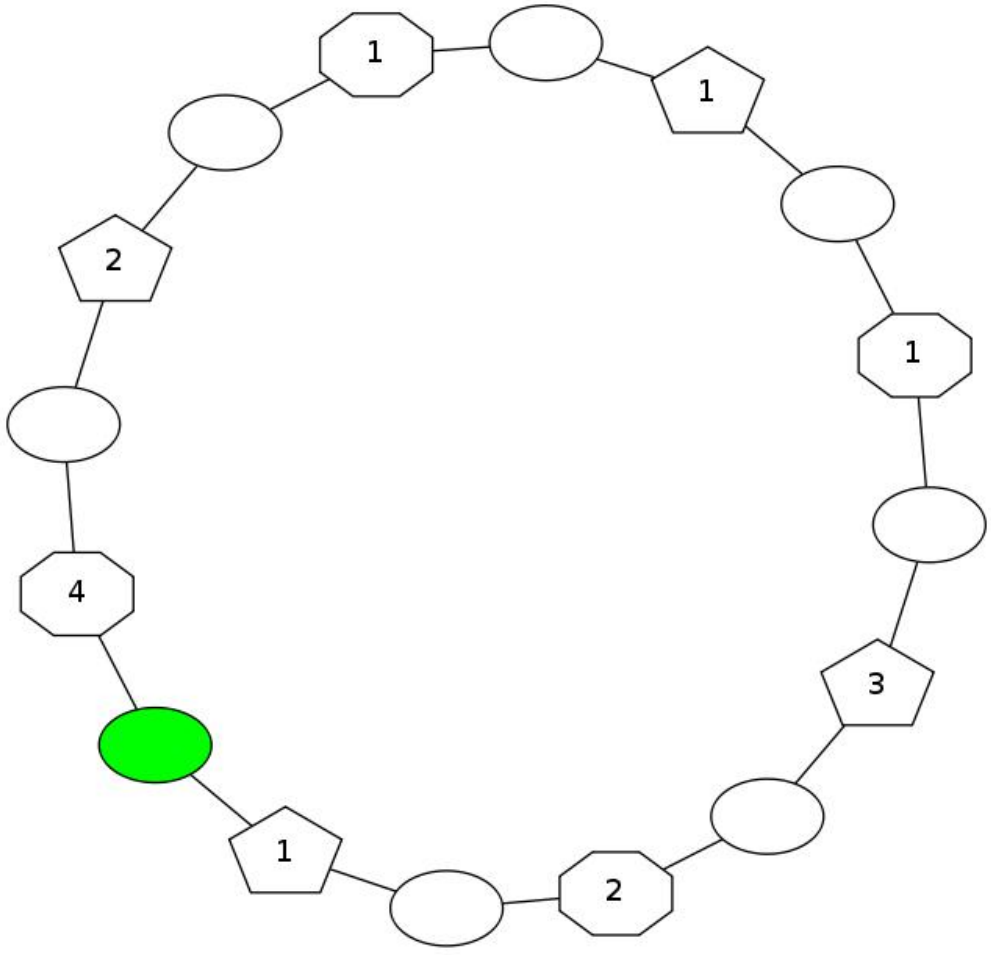
Networks with state: Dining Philosophers



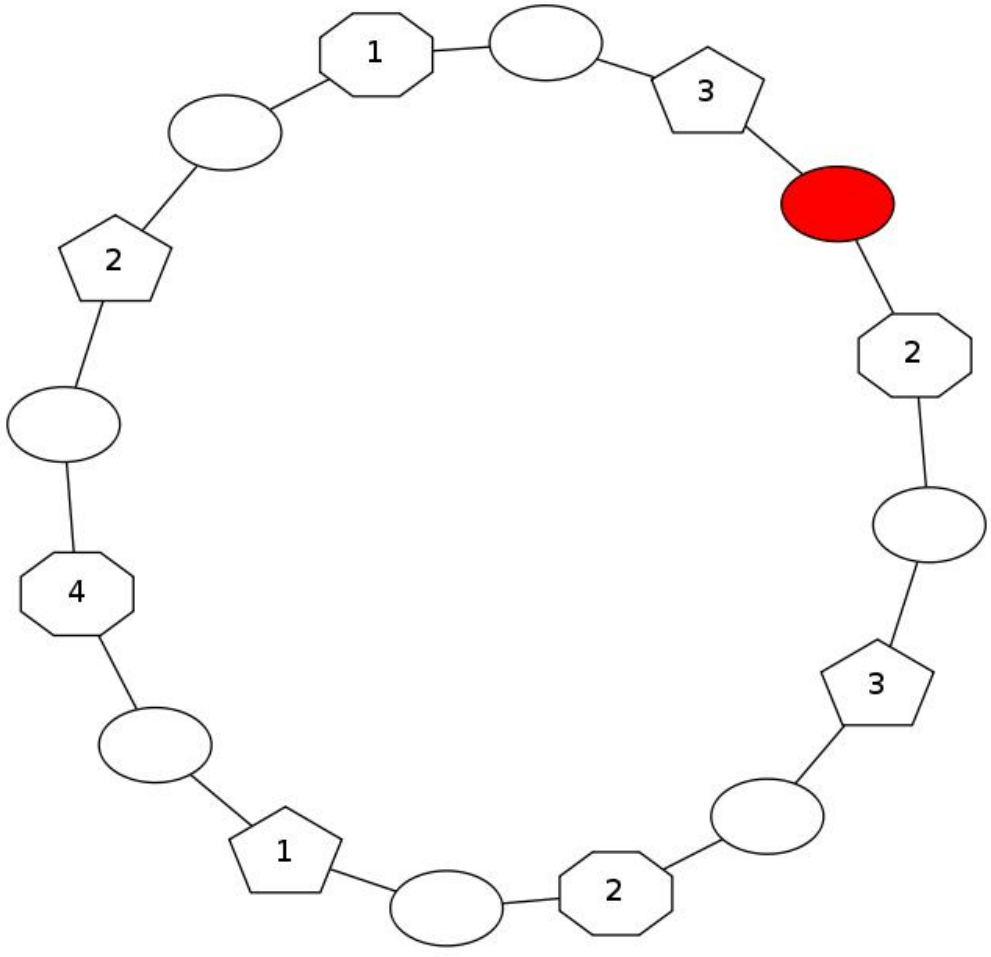
Networks with state: Dining Philosophers



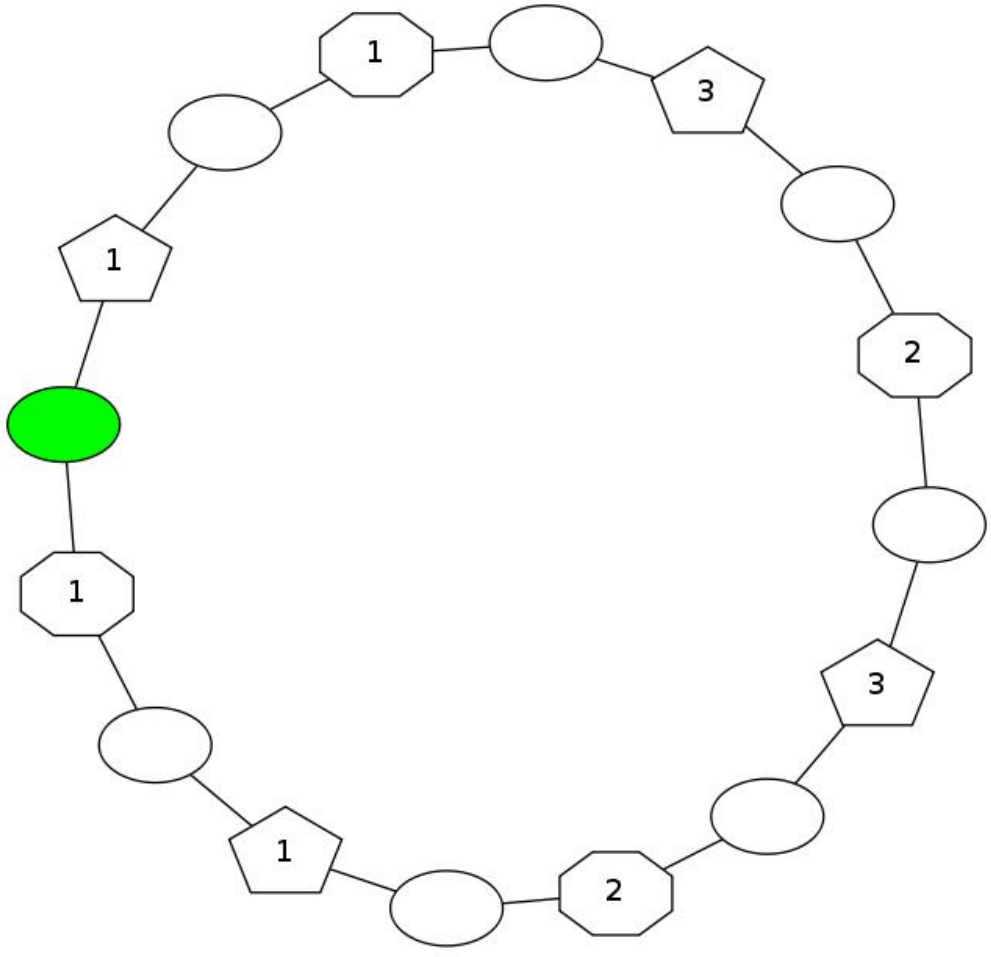
Networks with state: Dining Philosophers



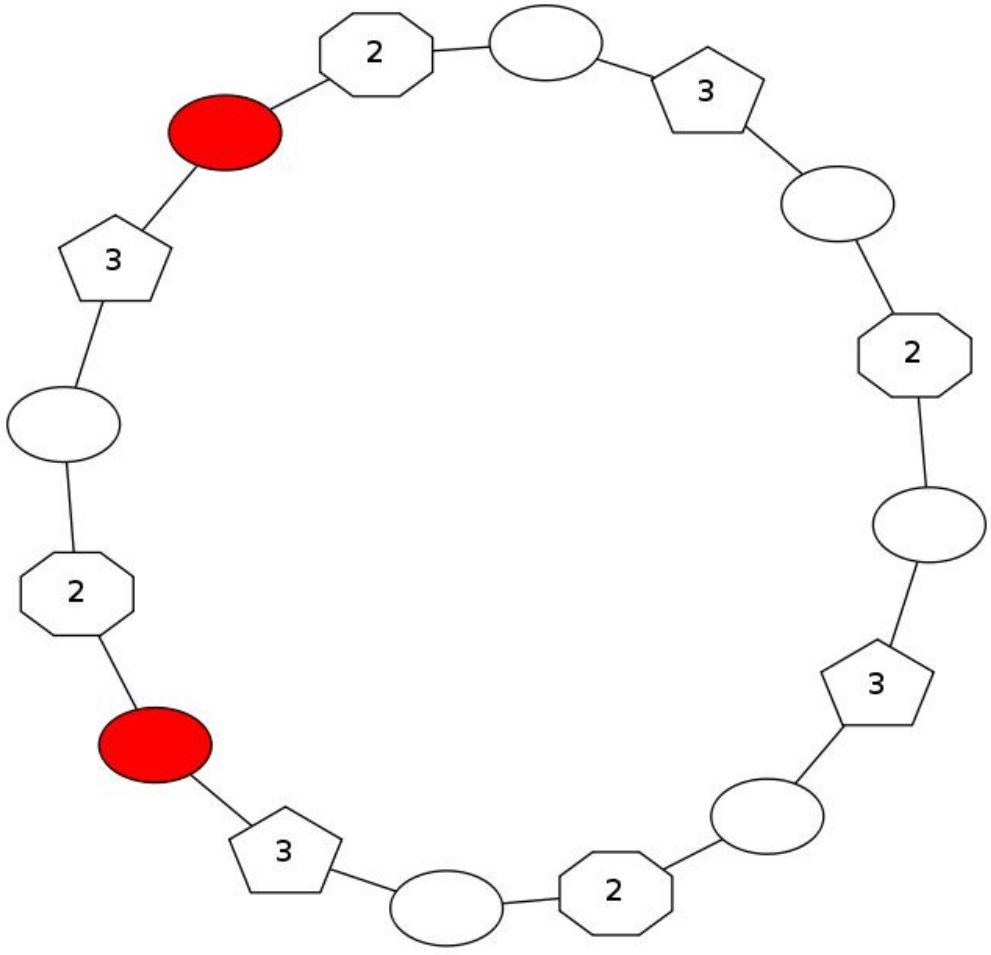
Networks with state: Dining Philosophers



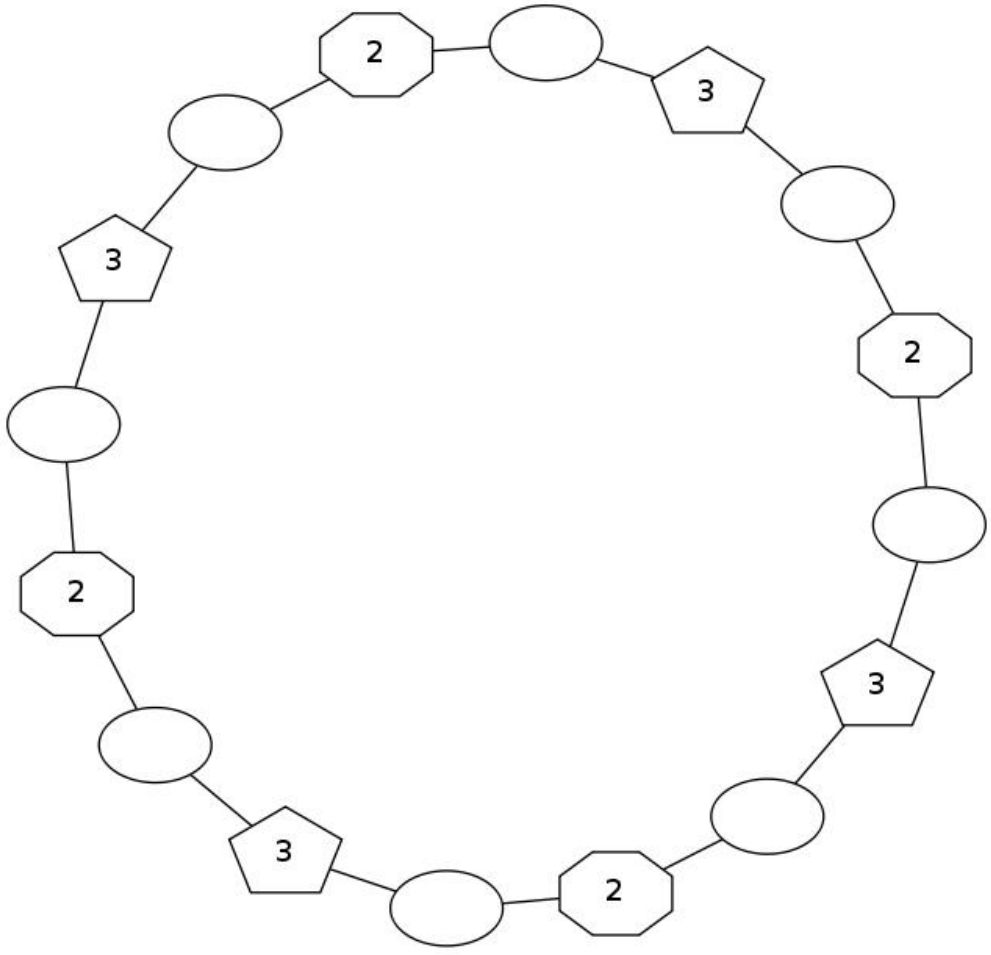
Networks with state: Dining Philosophers



Networks with state: Dining Philosophers



Networks with state: Dining Philosophers



PART II: Tangled circuits

Rosebrugh, Sabadini, Walters, Tangled circuits, arXiv:1110.0715 (2011)

A commutative Frobenius algebra in a braided monoidal category consists of an object G and four arrows $\nabla : G \otimes G \to G$, $\Delta : G \to G \otimes G$, $n : I \to G$ and $e : G \to I$ making (G, ∇, e) a monoid, (G, Δ, n) a comonoid and satisfying the equations

$$(1_G \otimes \nabla)(\Delta \otimes 1_G) = \Delta\nabla = (\nabla \otimes 1_G)(1_G \otimes \Delta) : G \otimes G \to G \otimes G$$

$$\nabla\tau = \nabla : G \otimes G \to G$$

$$\tau\Delta = \Delta : G \to G \otimes G$$

where τ is the braiding.
Given a monoidal graph M the free braided strict monoidal category in which the objects of M are equipped with commutative Frobenius algebra structures is called $\mathbf{TCircD}_M$. Its arrows are called tangled circuit diagrams, or more briefly circuit diagrams.

Tangled circuits

If M is the monoidal graph $\{R : G \otimes G \to G \otimes G\}$ then the following are distinct tangled circuits $G \otimes G \to G \otimes G$:

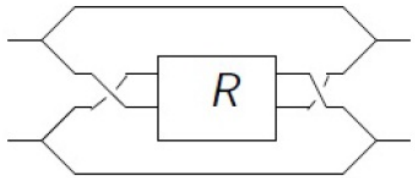


Figure : $(\nabla \otimes \nabla)(1 \otimes \tau \otimes 1)(1 \otimes R \otimes 1)(1 \otimes \tau \otimes 1)(\Delta \otimes \Delta)$

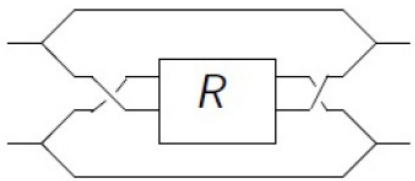
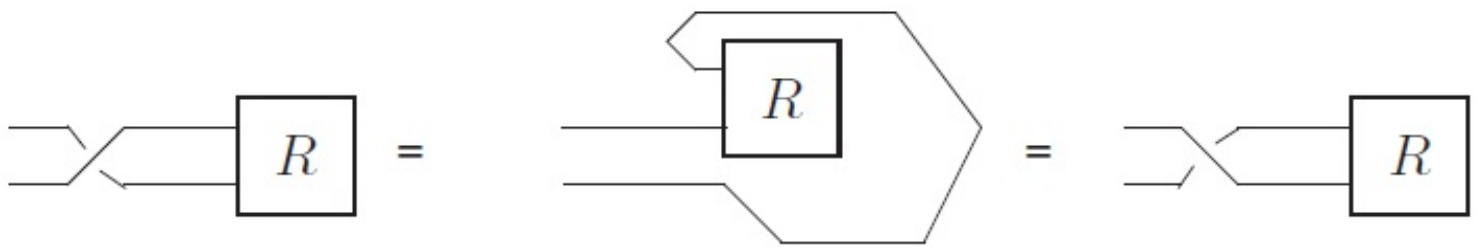


Figure : $(\nabla \otimes \nabla)(1 \otimes \tau^{-1} \otimes 1)(1 \otimes R \otimes 1)(1 \otimes \tau \otimes 1)(\Delta \otimes \Delta)$

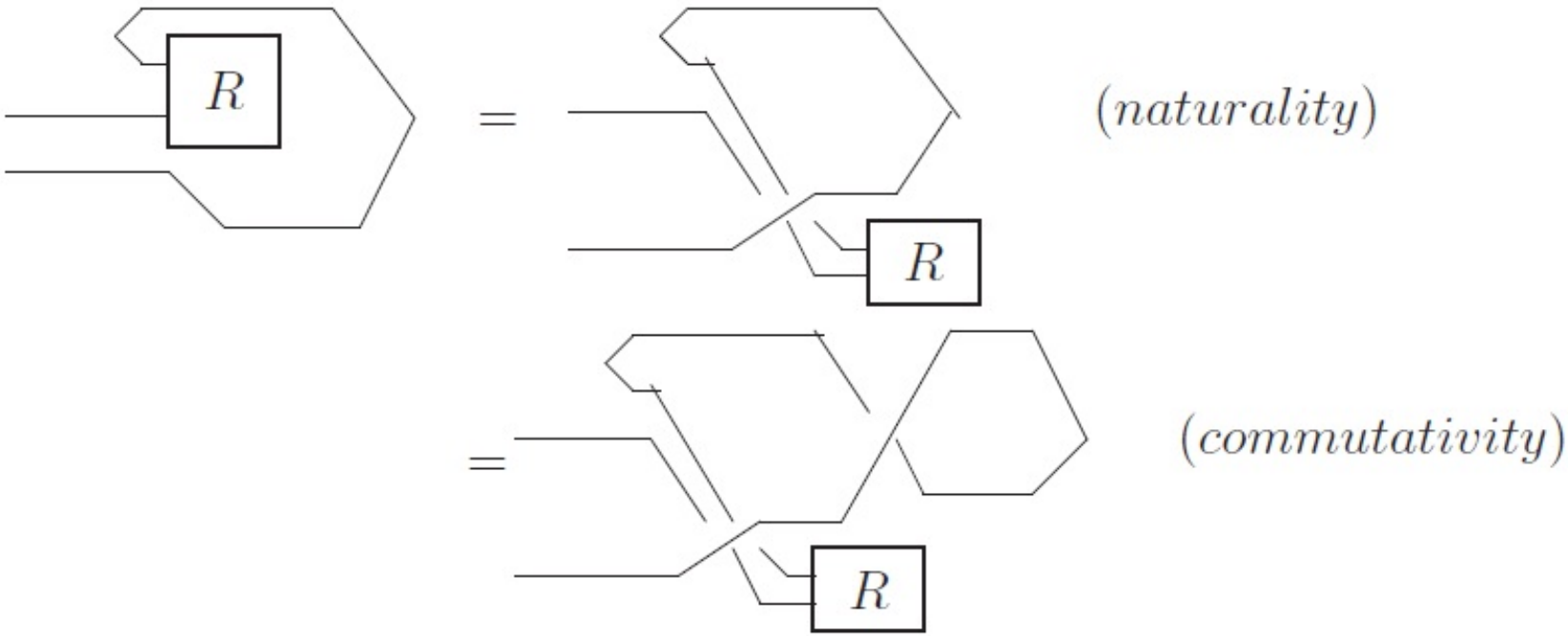
Equality of tangled circuits

We claim that the following three circuits are equal. It clearly suffices to verify the first equation.

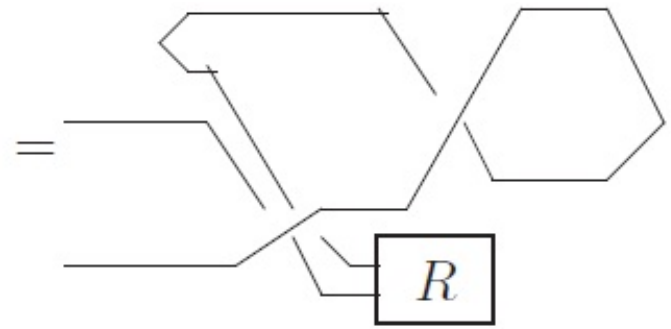


Equality of tangled circuits

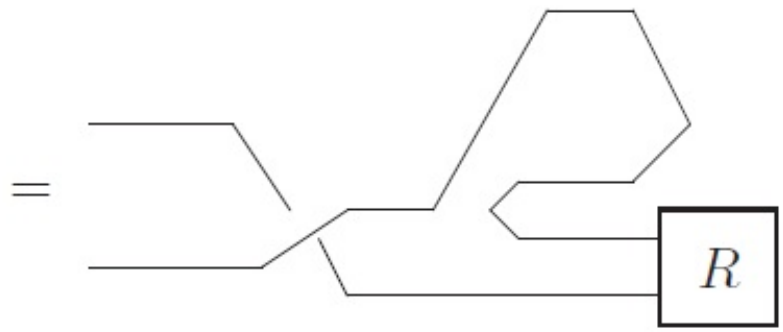
Proof of equality:



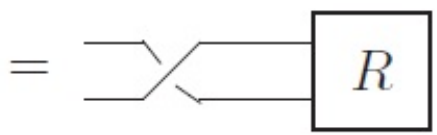
Equality of tangled circuits



(*commutativity*)



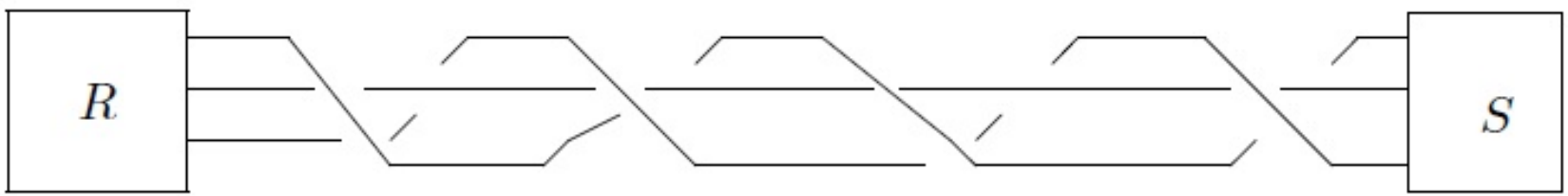
(*naturality*)



(*duality*)

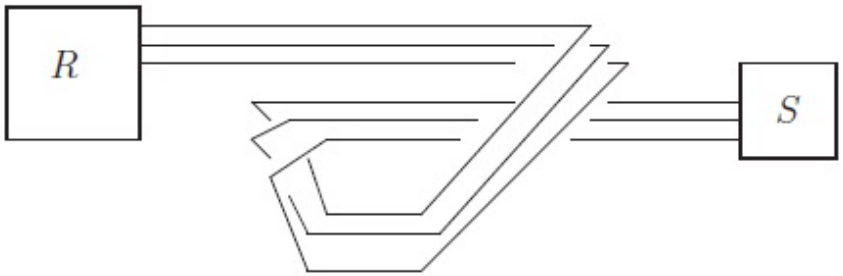
Dirac Belt Trick

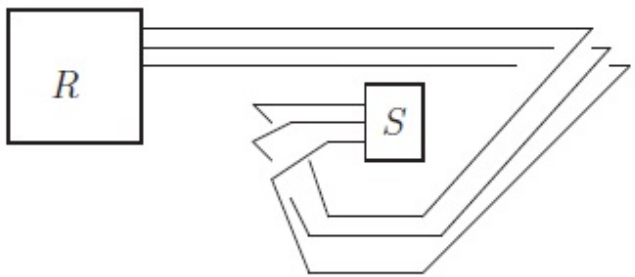
Two whole twists can be unwound without rotating the ends - this is called Dirac's belt trick.

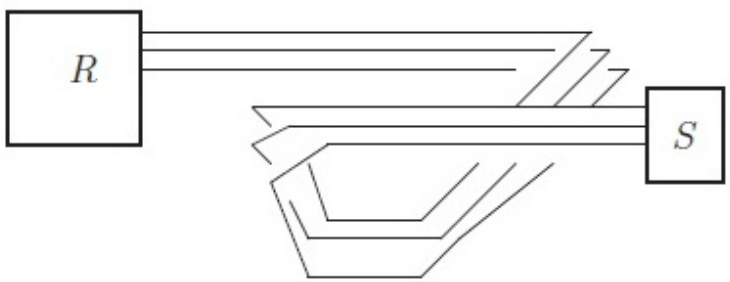


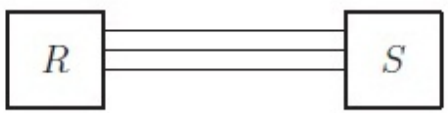
Dirac Belt Trick

Rough indication of steps of proof - the two complete twists are equal to









Proving tangled circuits distinct

To prove two circuits distinct it suffices to find a braided monoidal category with a commutative Frobenius algebra in which the two circuits are distinct. Given any group G there is an interesting such category which may be thought of as tangled relations $\mathbf{TRel}_G$.

Proving tangled circuits distinct

Let G be a group. The objects of $\mathbf{TRel}_G$ are the formal powers of G, and the arrows from G^m to G^n are relations R from the set G^m to the set G^n satisfying:

1) if $(x_1, ..., x_m)R(y_1, ...y_n)$ then also for all g in G $(g^{-1}x_1g, ..., g^{-1}x_mg)R(g^{-1}y_1g, ..., g^{-1}y_mg)$,
2) if $(x_1, ..., x_m)R(y_1, ...y_n)$ then $x_1...x_m(y_1...y_n)^{-1} \in Z(G)$ (the center of G).

Composition and identities are defined to be composition and identity of relations.
It is straightforward to verify that $\mathbf{TRel}_G$ is a category.

Proving tangled circuits distinct

We introduce some useful notation. Write $x = (x_1, ..., x_m)$, $y = (y_1, ..., y_n)$, and so on. Write $\overline{x} = x_1 x_2 ... x_m$ and for g, h in G, as $g^h = hgh^{-1}$. For g in G write $x^g = (x_1^g, x_2^g, ..., x_m^g)$. Thus, $(\overline{x})^g = \overline{x^g}$, and of course for any x, y in $G^m \times G^n$, $x^g y^g = (xy)^g$ where we write xy for $(x_1, ..., x_m, y_1, ..., y_n)$.
Then $\mathbf{TRel}_G$ is a braided strict monoidal category with tensor defined on objects by $G^m \otimes G^n = G^{m+n}$ and on arrows by product of relations. The twist

$$\tau_{m,n} : G^m \otimes G^n \to G^n \otimes G^m$$

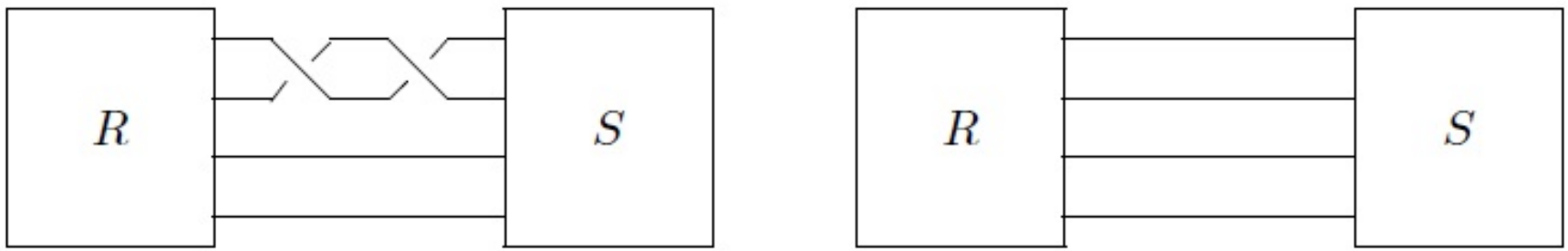
is the functional relation

$$(x, y) \sim (y^{\overline{x}}, x)$$

The arrow $\nabla : G \otimes G \to G$ is the functional relation of multiplication of the group.

Proving tangled circuits distinct

The following two circuits can be shown to be distinct by looking in $TRel_{S_3}$:



In fact with four or more strings and n twists on the first two strings as above we obtain always distinct tangled circuits. With two or three strings there are only finitely many tangled circuits of this type (blocked braids) (Davide Maglia).

Summary

Networks of components have a compositional description in terms of the algebra of symmetric or braided monoidal categories in which each object has a commutative Frobenius algebra structure compatible with the tensor product. They also have a geometric description [12] - the free such algebra (in the symmetric case) is the category of cospans of monoidal graphs, arrows of which have a pictorial representation; in the braided case the geometry is more complicated, capturing not only the connection between components but also their entanglement.
These results are in the line introduced by Penrose [1], and Joyal and Street [3], and were obtained by Sabadini and Walters with collaborators Katis and Rosebrugh in earlier work, especially [6,8,9,10,12,13], beginning with the work on relations with Carboni [2] in 1987. The work has numerous antecedents - we mention just S. Eilenberg, S.L. Bloom, Z. Esik, Gh. Stefanescu. The algebra has connections with quantum field theory [11]).

The present work presents two developments. The first is a tool for composing cospans of graphs and calculating executions of nets of parallel automata; the second is some initial work in classifying tangled circuits.

Bibliography

- [1] R. Penrose, *Applications of negative dimensional tensors*, in Combinatorial mathematics and its applications, Academic Press (1971).
- [2] A. Carboni and R.F.C. Walters, *Cartesian bicategories I*, Journal of Pure and Applied Algebra, 49 (1987) 11-32.
- [3] A. Joyal, R.H. Street, *The geometry of tensor calculus I*, Adv. Math. 88 (1991) 55-112.
- [4] A. Joyal, R.H. Street, *Braided monoidal categories*, Adv. Math. 102 (1993) 20-87.
- [5] P. Katis, N. Sabadini, R.F.C. Walters, *Representing P/T nets in Span(Graph)*, Proceedings AMAST '97, SLNCS 1349 (1997) 307-321.

Bibliography

- [6] P. Katis, N. Sabadini, R.F.C. Walters, *Span(Graph): an algebra of transition systems*, Proceedings AMAST '97, SLNCS 1349 (1997) 322-336.
- [7] F. Gadducci, R. Heckel, M. Llabres, A bi-categorical axiomatisation of concurrent. graph rewriting, ENTCS, 29, 1999.
- [8] P. Katis, N. Sabadini, R.F.C. Walters, *A formalisation of the IWIM Model*, in: Proc. COORDINATION 2000,(Eds.) Porto A., Roman G.-C., LNCS 1906, Springer Verlag, (2000) 267-283.
- [9] P. Katis, N. Sabadini, R.F.C. Walters, *On the algebra of systems with feedback and boundary*, Rendiconti del Circolo Matematico di Palermo Serie II, Suppl. 63 (2000), 123-156.
- [10] P. Katis, N. Sabadini, R.F.C. Walters, *Compositional minimization in Span(Graph): Some examples*, Electronic Notes in Theoretical Computer Science 104C,(2004) 181-197.

Bibliography

- [11] J. Kock, *Frobenius algebras and 2D Topological Quantum Field Theories*, Cambridge University Press (2004).
- [12] R. Rosebrugh, N. Sabadini, R.F.C. Walters, *Generic commutative separable algebras and cospans of graphs*, Theory and Applications of Categories, 15, (2005) 264-277.
- [13] R. Rosebrugh, N. Sabadini, R.F.C. Walters, *Calculating colimits compositionally*, Montanari Festschrift, LNCS 5065,(2008) 581592
- [14] L. de Francesco Albasini, N. Sabadini and R.F.C. Walters, *An algebra of automata that includes both classical and quantum entities*, Electr. Notes Theor. Comput. Sc. 270 (2011) 263-270.
- [15] R. Rosebrugh, N. Sabadini, R.F.C. Walters, *Tangled Circuits*, arXiv:1110.0715 (2011).
- [16] Jennifer Lantair, Pawel Sobocinski, *WiCcA: LTS generation tool for wire calculus*, Proceeding CALCO'11 Proceedings of the 4th international conference on Algebra and coalgebra in computer science, (2011) 407-412.