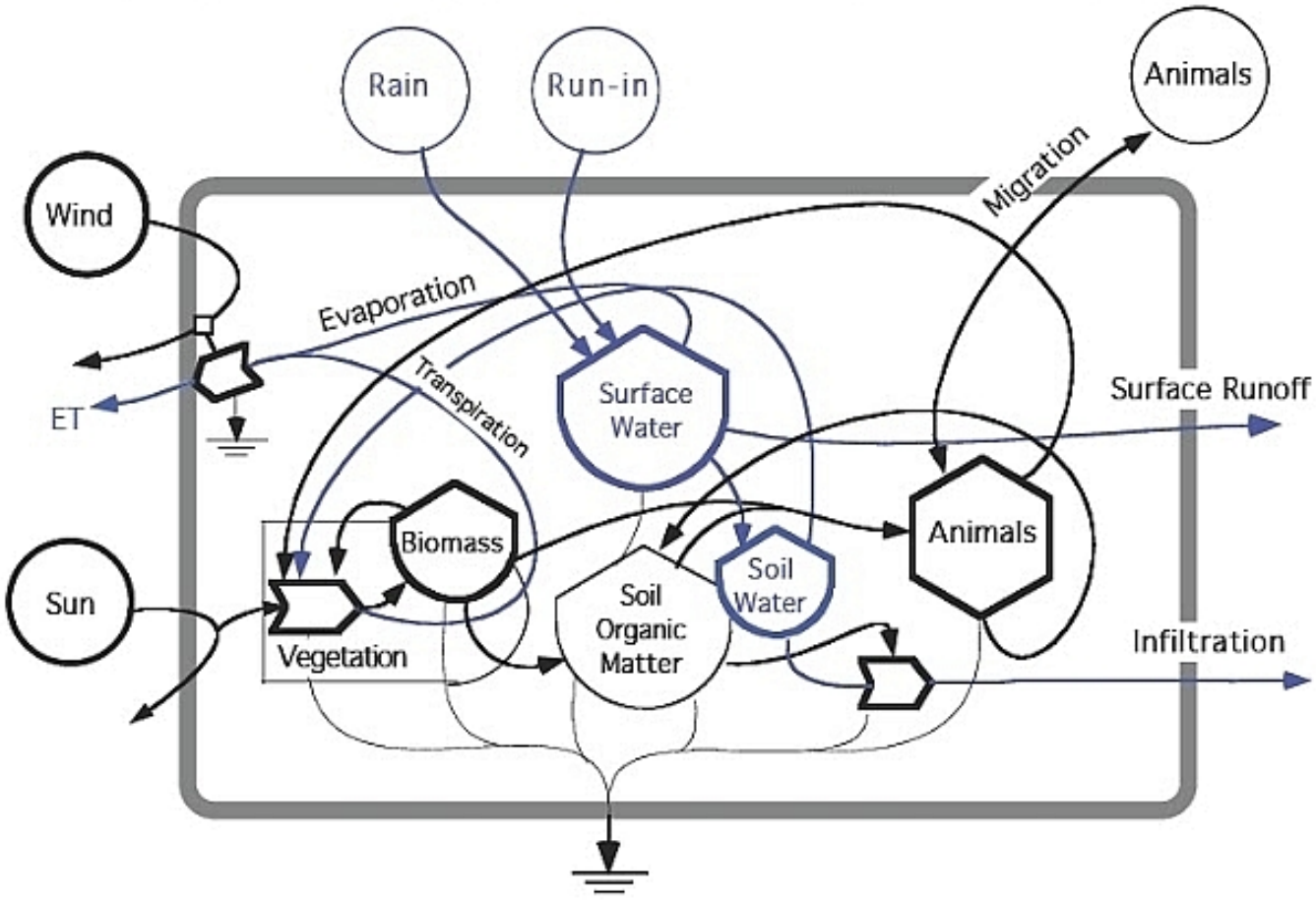
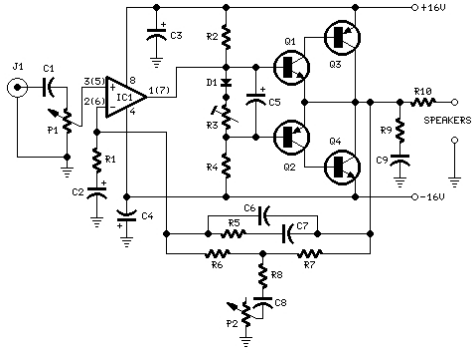
John Baez

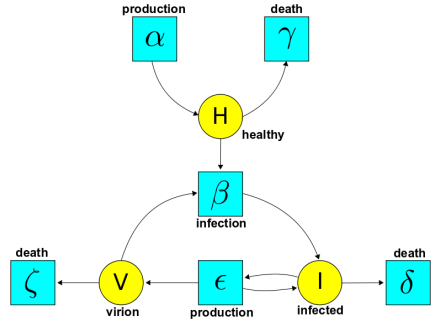
NETWORK THEORY

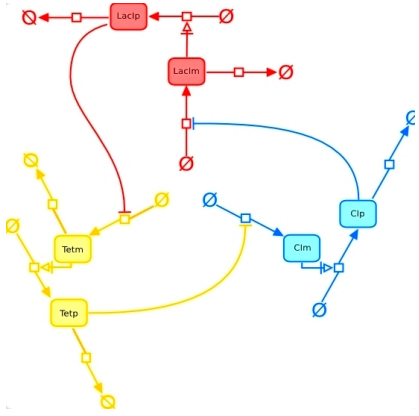


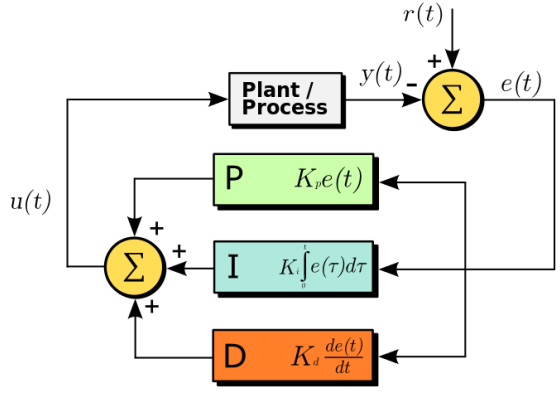
For slides go to http://tinyurl.com/networks-broadcom

In many areas of science and engineering, people use *diagrams of networks*, with boxes connected by wires:





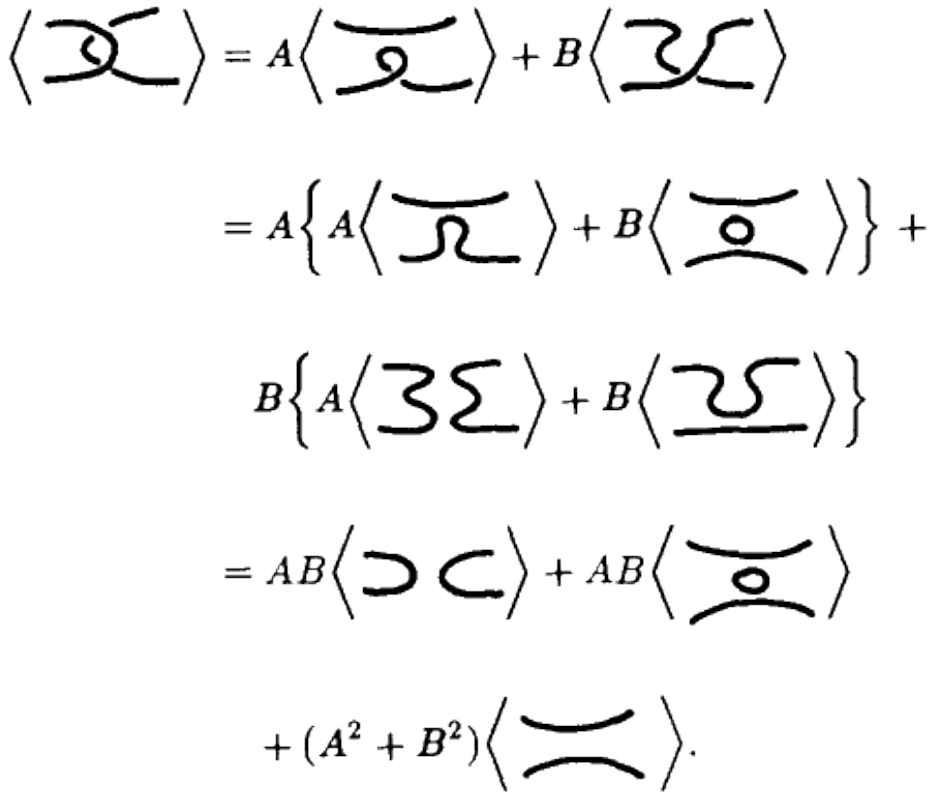




We need a good mathematical theory of these.

Categories must be part of the solution. This became clear in the 1980s, at the interface of knot theory and quantum physics:

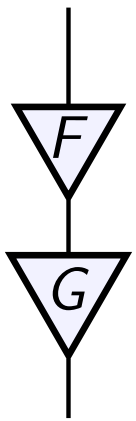
Proof. (a)



Part (b) is left for the reader.

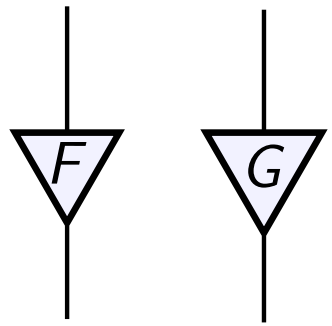
Categories are great for describing processes. A process with input x and output y is a *morphism* $F : x \to y$, and we can draw it like this:

We can do one process after another if the output of the first equals the input of the second:



Here we are composing morphisms $F : x \to y$ and $G : y \to z$ to get a morphism $GF : x \to z$.

In a **monoidal** category, we can also do processes 'in parallel':



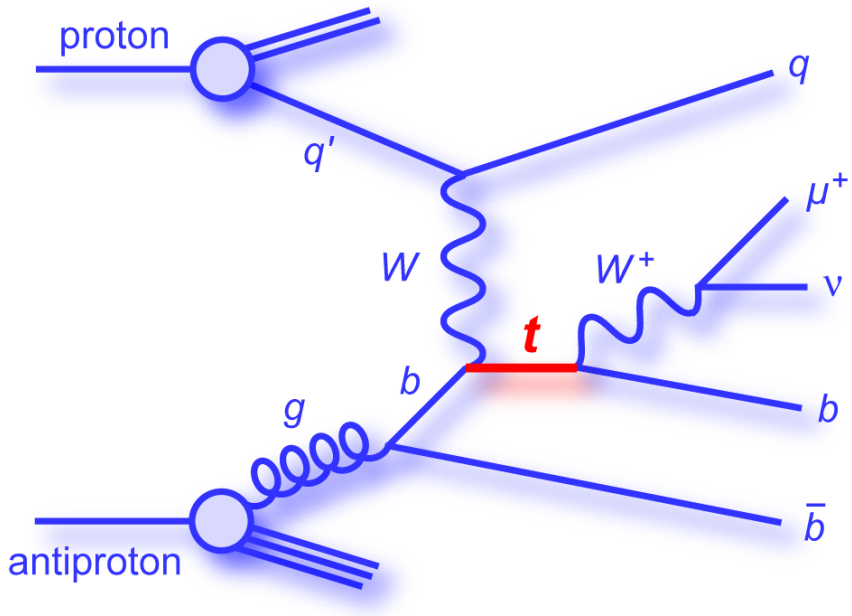
Here we are **tensoring** $F : x \to y$ and $G : x' \to y'$ to get a morphism $F \otimes G : x \otimes x' \to y \otimes y'$.

Composition and tensoring must obey some laws, which all look obvious when drawn as diagrams.

In a **symmetric monoidal category** we also have morphisms $B_{x,y} : x \otimes y \to y \otimes x$ called **braidings**:

These let us draw diagrams where wires cross. They are required to obey some obvious-looking laws, such as the one above.

In quantum field theory, 'Feynman diagrams' describe interactions between elementary particles:



In the 1990s it became clear that any Feynman diagram theory is a symmetric monoidal category.

But why should quantum field theorists have all the fun?

In fact, engineering is full of diagrams that are worthy of mathematical study. That's what I've been looking at lately.

Each style of diagram with quantities flowing along 'wires' corresponds to some symmetric monoidal category $\mathbf{C}$.

Each way of translating between styles of diagrams should then correspond to some symmetric monoidal functor $\Phi\colon \mathbf{C} \to \mathbf{D}$. A 'functor' is a map between categories, sending morphisms of the first to morphisms of the second, preserving composition:

$$\Phi(GF) = \Phi(G)\,\Phi(F)$$

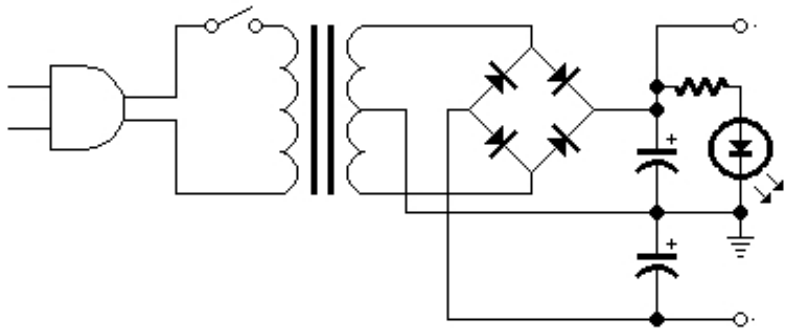
A 'monoidal functor' also preserves tensoring:

$$\Phi(F \otimes G) = \Phi(F) \otimes \Phi(G)$$

and the braiding.

So, instead of just studying one diagram language, we can study many, and how they're related.

Circuit diagrams let us specify relations between voltages and currents on a network of wires:



However, circuit diagrams are good for much more, thanks to these analogies:

	displacement q	flow $\dot{q}$	momentum p	effort $\dot{p}$
Electronics	charge	current	flux linkage	voltage
Mechanics (translation)	position	velocity	momentum	force
Mechanics (rotation)	angle	angular velocity	angular momentum	torque
Hydraulics	volume	flow	pressure momentum	pressure
Thermodynamics	entropy	entropy flow	temperature momentum	temperature
Chemistry	moles	molar flow	chemical momentum	chemical potential

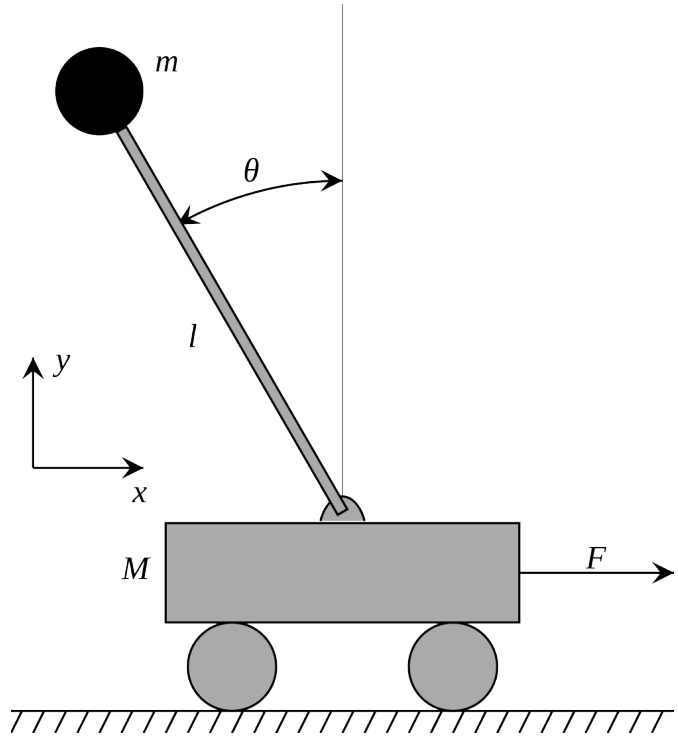
Each analogy between subjects is really a symmetric monoidal functor!

But these particular functors consist merely of renaming quantities: they're important but not mathematically deep.

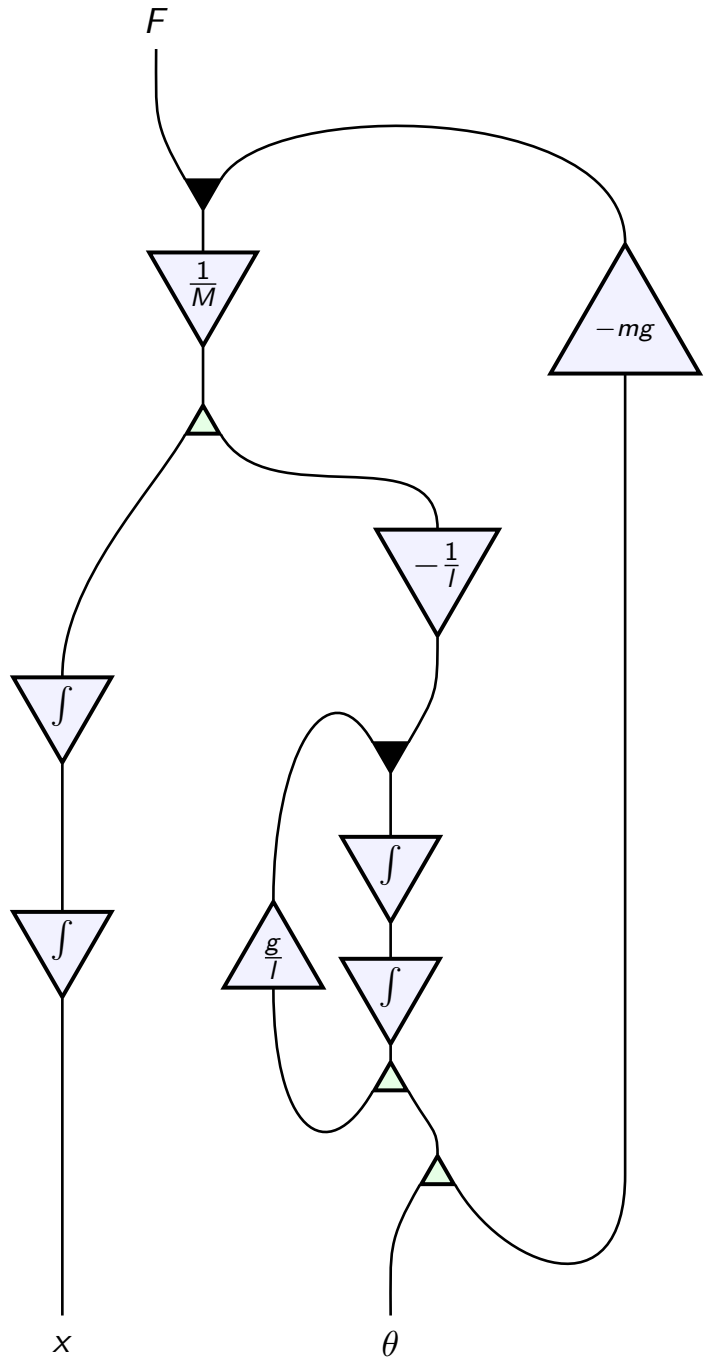
More interesting is the symmetric monoidal functor from circuit diagrams to 'signal-flow graphs'.

Control theorists use signal-flow graphs to describe processes where signals flow through a system and interact.

For example, an upside-down pendulum on a cart:



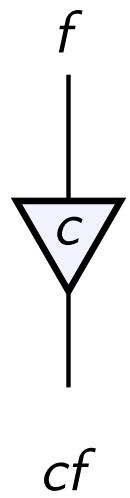
has the following signal-flow graph...


F
$\frac{1}{M}$
$-mg$
$-\frac{1}{l}$
$\int$
$\int$
$\int$
$\int$
$\frac{g}{l}$
x
θ

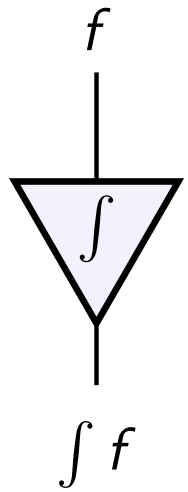
The idea is that each 'wire' in a signal-flow graph carries a **signal**, a smooth real-valued function of time:

$$f : \mathbb{R} \to \mathbb{R}$$

We can multiply a signal by a constant and get a new signal:



We can integrate a signal:



We can use Laplace transforms to write signals as linear combinations of exponentials:

$$f(t) = e^{-st} \quad \text{for some } s > 0$$

Then we can define

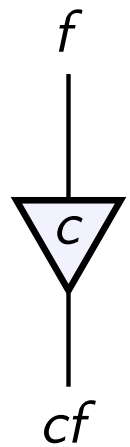
$$(\int f)(t) = \frac{e^{-st}}{s}$$

This lets us think of integration as a special case of scalar multiplication! We extend our 'scalars' from $\mathbb{R}$ to

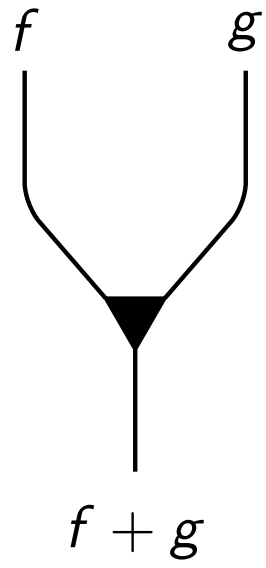
$$k = \left\{ \frac{a_n s^n + \cdots + a_1 s + a_0}{b_m s^m + \cdots + b_1 s + b_0} : a_i, b_i \in \mathbb{R} \right\}$$

Signal-flow graphs are morphisms in a symmetric monoidal category **SigFlow**. We build general signal-flow graphs by composing and tensoring certain building blocks, called **generators**:

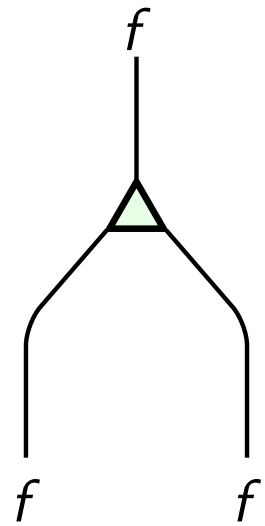
1. For each $c \in k$ we can multiply signals by c:



2. We can add signals:



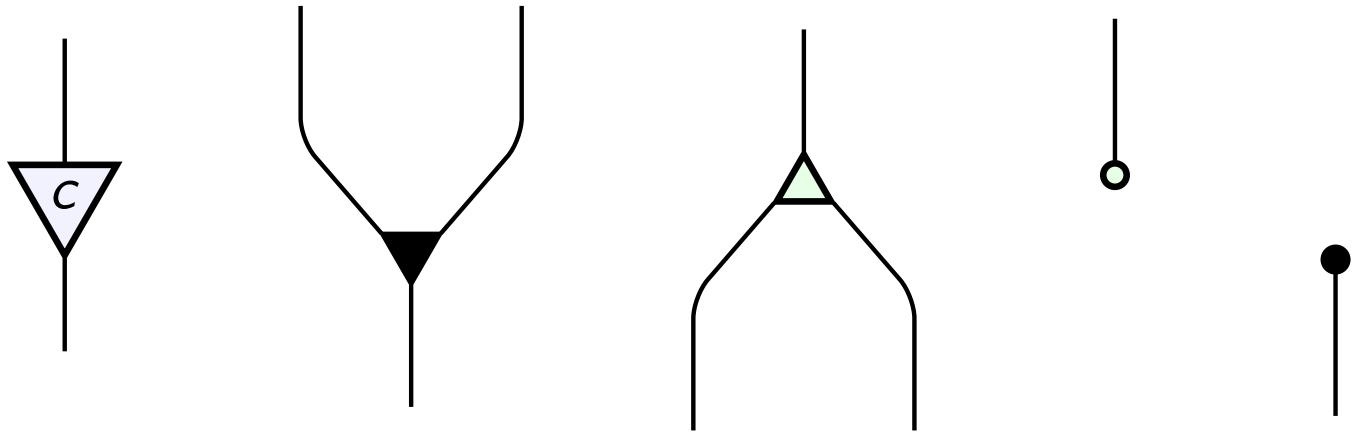
3. We can **duplicate** a signal:



4. We can **delete** a signal:

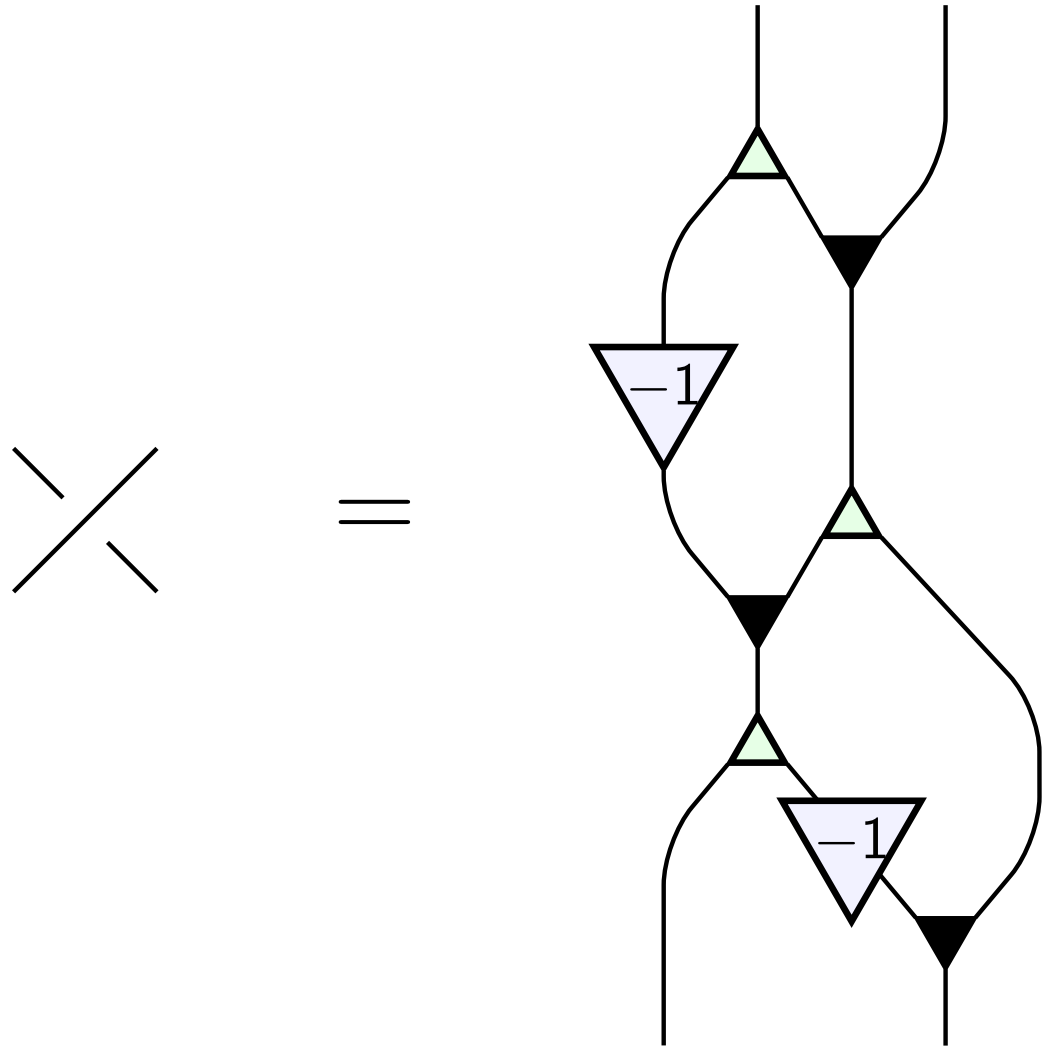
5. We have the zero signal:

From these generators:



together with composition, tensoring and the braiding, we can build signal-flow graphs that describe *any* linear map $F : k^m \to k^n$.

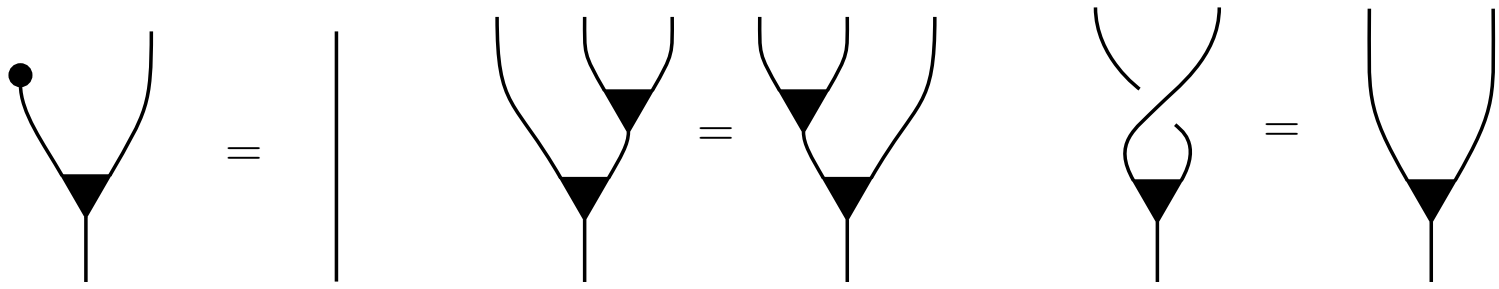
However, the generators obey some unexpected relations:



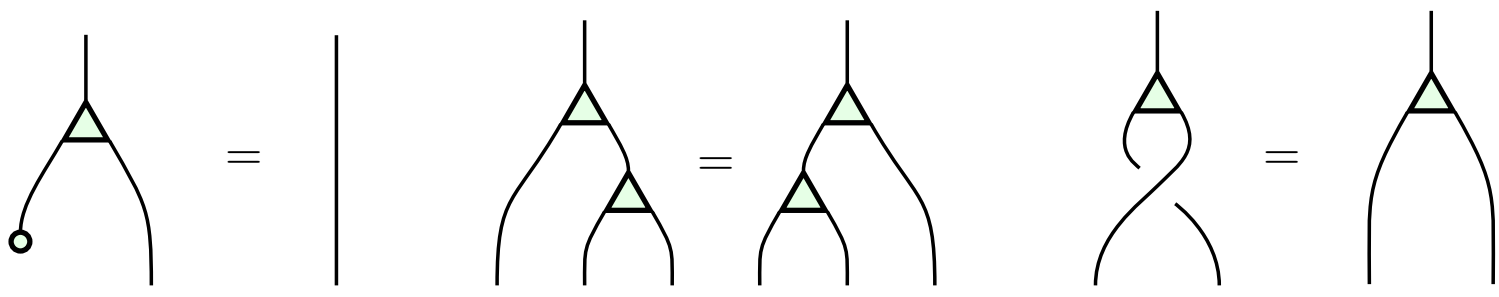
Lucikly, in our paper Categories in Control, Jason Erbele and I found a finite list of relations that imply all the rest.

Apart from those built into the definition of a symmetric monoidal category, they are these:

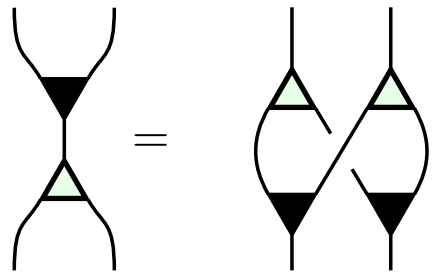
(1)–(3)

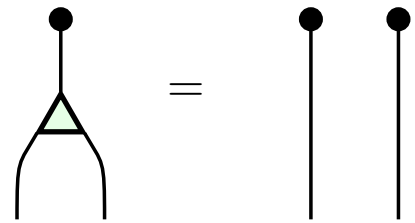


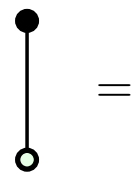
(4)–(6)

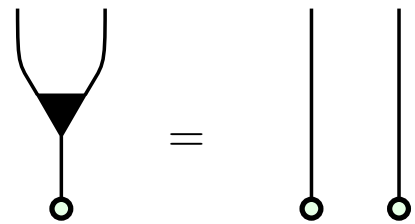


(7)–(10)

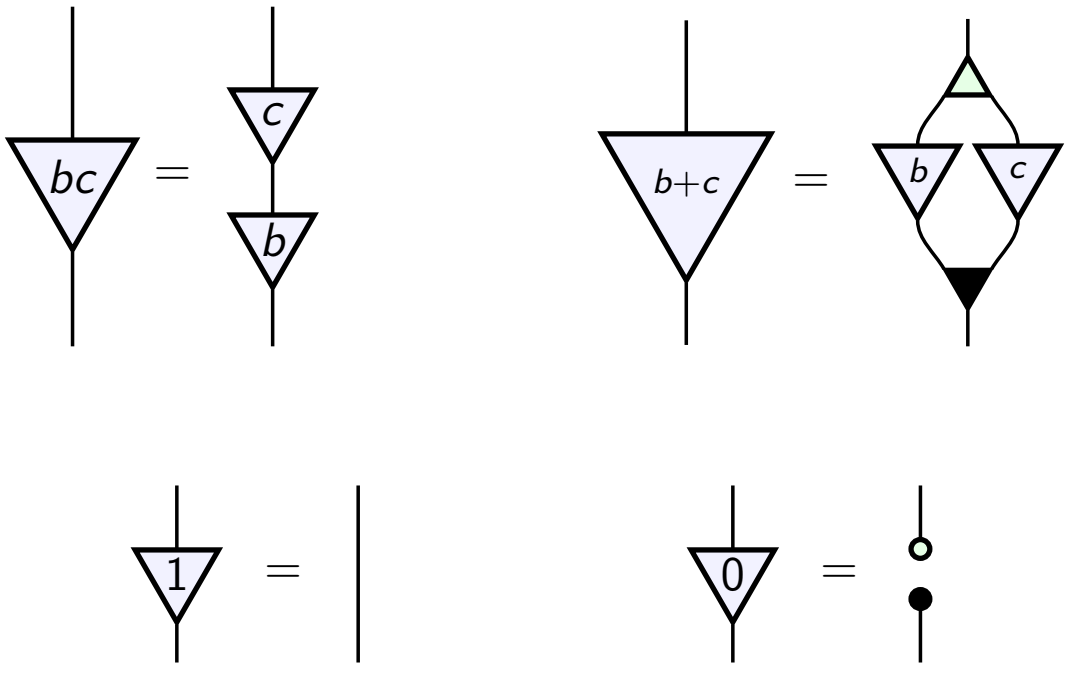




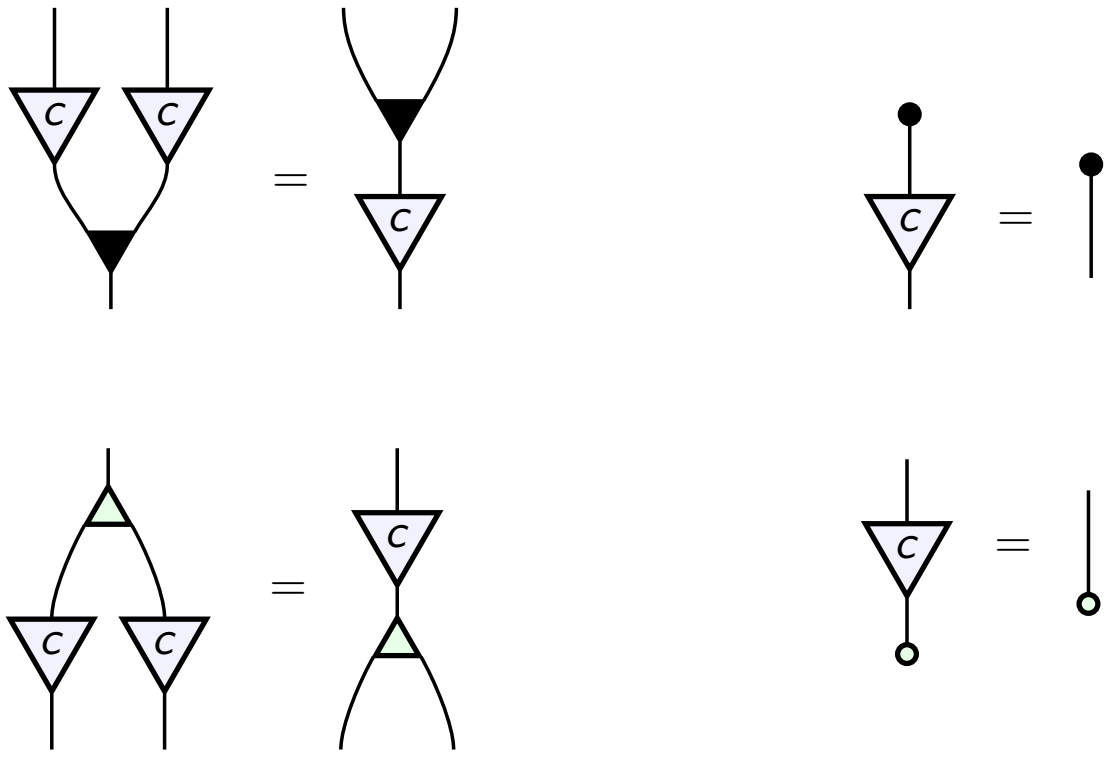




(11)–(14)

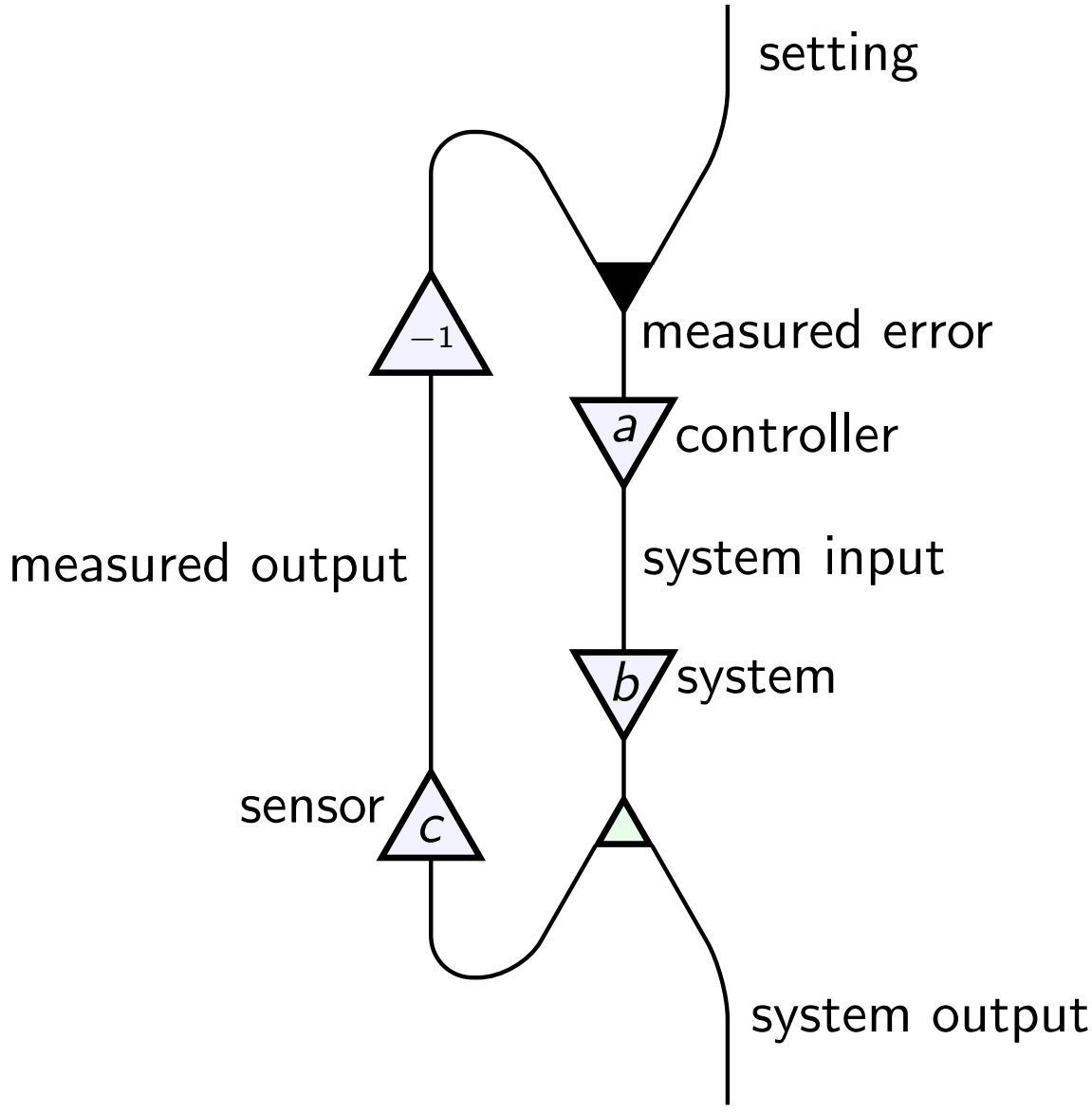


(15)–(18)



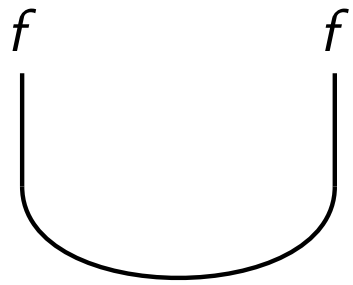
These are all the relations we need!

However, control theory also involves more general signal-flow graphs, which have 'feedback loops':

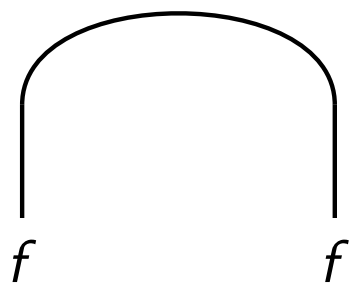


Feedback is the most important concept in control theory: letting the output of a system affect its input. For this we should let wires 'bend back'. So, we need two more generators:

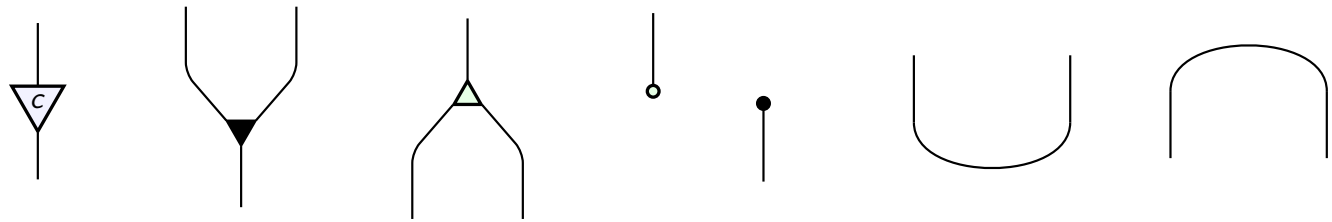
6. The **cup**:



7. The **cap**:



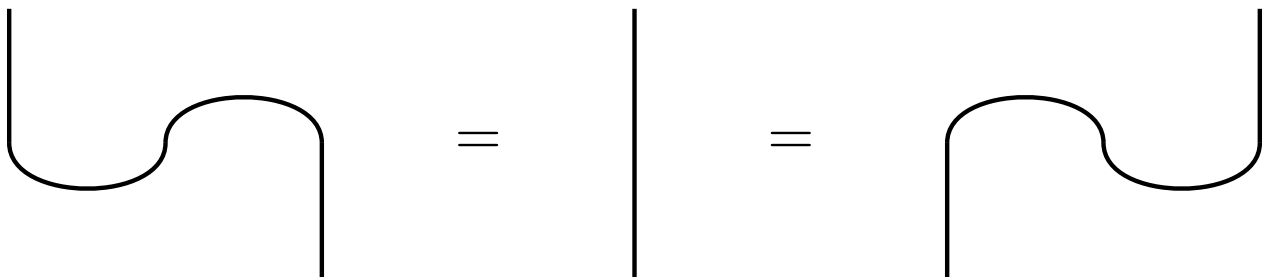
Every linear subspace of k^{m+n} can be described by a signal-flow graph built from these generators:



using composition, tensoring and the braiding.

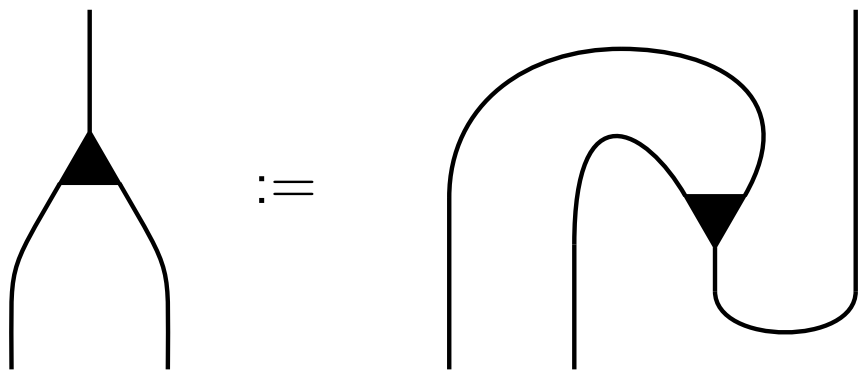
Thus, any set of linear time-translation-invariant ordinary differential equations relating inputs and outputs can be described by such a signal-flow graph.

The extra generators obey some extra relations. Jason Erbele and I worked out a finite set which imply all the rest. The most fundamental are the **zigzag relations**:



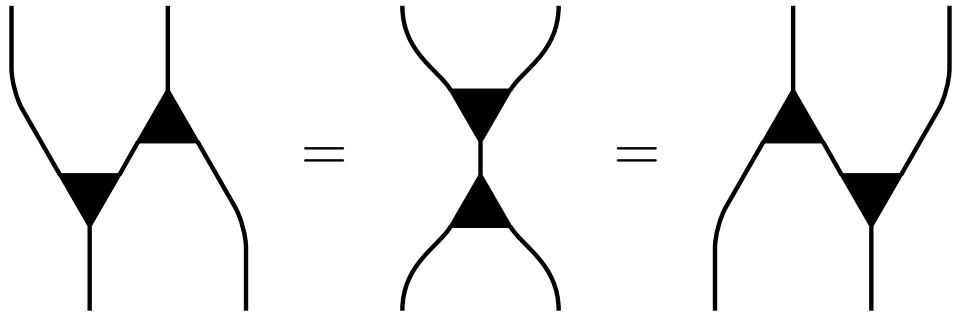
The cap and cup let us 'turn around' any morphism $F : x \to y$ and get its **adjoint** $F^\dagger : y \to x$.

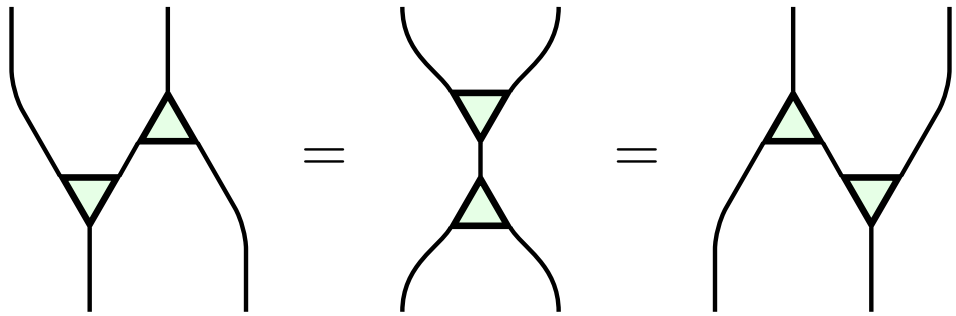
For example, turning around addition gives **coaddition**:



This forces the two outputs to sum to the input.

I won't list all the other relations, but the most interesting are the **Frobenius relations**:





In A Compositional Framework for Passive Linear Networks, Brendan Fong and I studied a symmetric monoidal category **Circ** whose morphisms are circuit diagrams made of passive linear components such as resistors, inductors and capacitors.

There is a symmetric monoidal functor

$$\Phi \colon \mathbf{Circ} \to \mathbf{SigFlow}$$

This says, for any passive linear circuit, how the voltages and currents on the input wires are related to those on the ouput wires. Note that

$$\Phi(GF) = \Phi(G)\,\Phi(F)$$

and

$$\Phi(F \otimes G) = \Phi(F) \otimes \Phi(G)$$

This functor fits into a larger network of functors translating between different diagram languages. So far I've studied these:

- circuit diagrams
- signal flow graphs
- bond graphs
- Markov chains
- chemical reaction networks

Markov chains can be seen as a generalization of passive linear electrical circuits. Chemical reaction networks can be seen as a nonlinear generalization of Markov chains! But the project of synthesizing and proving general results about diagram languages is just starting.