Wiring diagrams and state machines

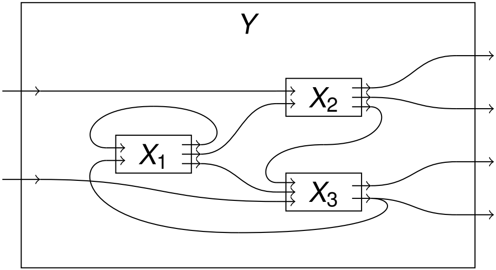
David I. Spivak

dspivak@math.mit.edu
Mathematics Department
Massachusetts Institute of Technology

Presented on 2014/02/19

My goal: a visual, formal language for processes

- I want to be able to draw pictures like this:

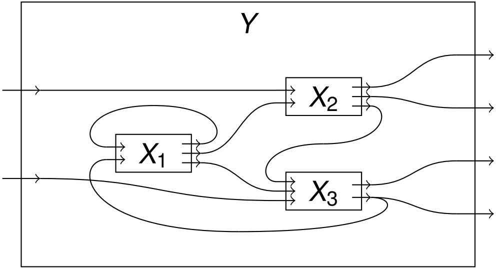


such that, if one fills in each box X_i with a machine, it results in a new machine for Y.

- And I want it all to work as expected.

What does all this mean?

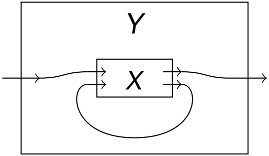
- But what *is* a picture like this?



- And what kind of machines have this fill-me-in property?
- And what expectations should we have about all this?

Plan of this talk

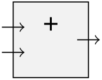
- I will show that wiring diagrams (WDs)



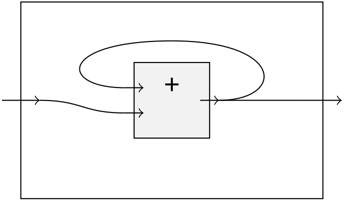
  form a symmetric monoidal category (or SMC), denoted **W**.
- I will show that there is an algebra $\mathcal{P}\colon \mathbf{W} \to \mathbf{Set}$ of machines.
- I will explain SMCs and their algebras as we go along.
- Time permitting, I'll talk about adding special symbols to the language.

First example: a running total

- Consider the machine

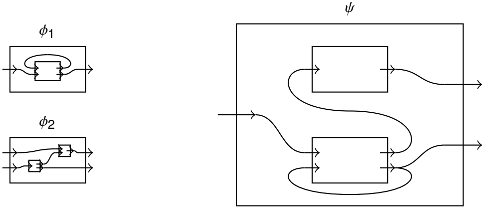


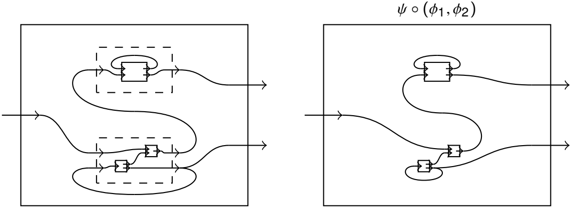
  which takes two integers and reports their sum.
- Installing it into the following wiring diagram



  constructs a new machine for the outer box.
  - The constructed machine reports a running total of its inputs.
  - It carries the previous sum on the internal wire as state.

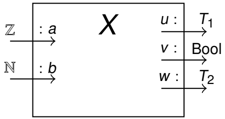
The picture of **W**





Wires and boxes

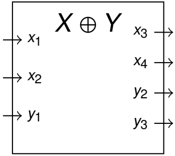
- Wires carry a defined set of values.
  - A *wire* $w \in \mathbf{Set}_*$ is a pointed set $w = (T, t_0)$, where $t_0 \in T$.
  - A *finite set of wires* is a pair (I, τ), where $I = \{i_1, \ldots, i_n\}$ is a finite set, and $\tau \colon I \to \mathbf{Set}_*$ is a function.
  - We write **TFS** ("typed finite sets") to denote the collection of (I, τ)'s.
- Boxes have input wires and output wires.
  - A *box* X consists of a pair $X := (\mathtt{inp}(X), \mathtt{out}(X))$
    - $\mathtt{inp}(X) \in$ **TFS** is called the *set of input wires to* X, and
    - $\mathtt{out}(X) \in$ **TFS** is called the *set of output wires to* X.
  - Another term for box might be *interface*.
- Example: Box $X = \left(\{a : \mathbb{Z},\ b : \mathbb{N}\}, \{u : T_1,\ v : Bool,\ w : T_2\}\right)$



Tensor product of boxes

- Given boxes $X = (\mathtt{inp}(X), \mathtt{out}(X))$ and $Y = (\mathtt{inp}(Y), \mathtt{out}(Y))$,

  we can stack them on top of each other and call that a box



- Define the *tensor product of* X *and* Y, denoted $X \oplus Y$, by

$$X \oplus Y := (\mathtt{inp}(X) + \mathtt{inp}(Y), \mathtt{out}(X) + \mathtt{out}(Y)).$$

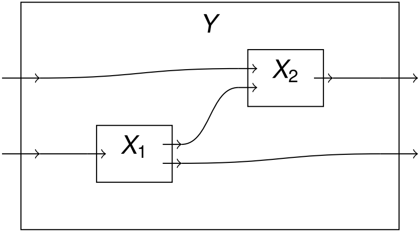
- We define the *inert box* to be $\square := (\emptyset, \emptyset)$. It is a $\oplus$-*unit*:

$$X \oplus \square \cong X \cong \square \oplus X.$$

Wiring diagrams, operad flavor: Many boxes inside

- Operads are many-inside, one-outside.
  - More precisely, morphisms in an operad have many domain objects.
  - For example $\phi\colon (X_1, X_2, \ldots, X_n) \longrightarrow Y$.
- These make for nicer, more intuitive pictures.
- If desired, one can restrict to the sub-operad of *loop-free WDs*.
  - Loop-free being a smaller syntax, it is more easily modeled.
  - For example, spreadsheets (incremental computation?).

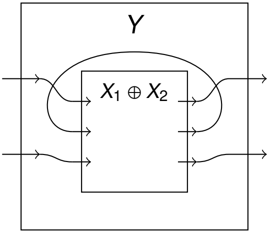
$\phi\colon (X_1, X_2) \longrightarrow Y$



Wiring diagrams, monoidal flavor: One box inside

- Monoidal categories are more like regular old categories.
  - Morphisms in a monoidal category have one domain object.
  - But there's a tensor operation that serves an operad-like purpose.
  - We can have $\phi\colon X_1 \oplus X_2 \oplus \cdots \oplus X_n \to Y$.
- Advantages to using monoidal categories:
  - The mathematics works out cleaner for wiring diagrams.
  - More people know about monoidal categories.
- Disadvantage: the pictures can be ugly and unintuitive.
  - Here's the monoidal version of the picture from the previous slide.

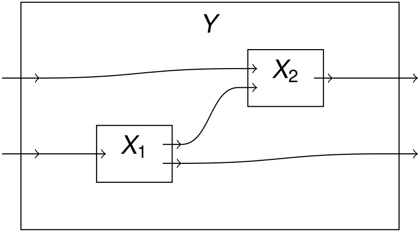
$$\phi\colon X_1 \oplus X_2 \longrightarrow Y$$



Today's compromise: monoidal math, operadic picture

- In our case (with loops allowed), these two notions are equivalent.
- So we'll go with the pretty option in both cases:
  - Pretty math: symmetric monoidal categories (SMCs)
  - Pretty pictures: operads.
- We'll write $\phi\colon X_1 \oplus X_2 \to Y$ and allow ourselves to draw the diagram below.

$$\phi\colon X_1 \oplus X_2 \longrightarrow Y$$



Where are we now?

- We're on our way to defining a symmetric monoidal category **W**.
    - I'll tell you the definition of SMC's soon.
    - For now just bear with me.
- An object $X \in \mathrm{Ob}(\mathbf{W})$ is called a *box*.
    - Recall a box is a pair $X = (\mathtt{inp}(X), \mathtt{out}(X))$ of typed finite sets.
    - The coincidence of the term "object" with OOP is not bad.
    - We are trying to formalize encapsulation.
- Boxes can be tensored together by stacking them.

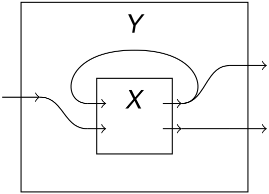
$$X \oplus Y = \Big(\ \mathtt{inp}(X) + \mathtt{inp}(Y), \mathtt{out}(X) + \mathtt{out}(Y)\ \Big)$$

- Morphisms in **W** are wiring diagrams.
    - I showed pictures of the monoidal version and the operadic version.
    - Hopefully these pictures make intuitive sense.
    - But I haven't told you what WDs are *mathematically*.

Thinking about wiring diagrams

- Let $X = (\mathtt{inp}(X), \mathtt{out}(X))$ and $Y = (\mathtt{inp}(Y), \mathtt{out}(Y))$ be boxes.
- What is a wiring diagram?

$\phi\colon X \to Y$



- Think of ϕ as an economy, in which every demand needs a supply.
  - The inputs of X are supplied either by inputs of Y or by internal wires.
  - Both the internal wires and the outputs of Y are sourced by X-outputs.
  - A wiring diagram expresses these relationships in terms of functions.

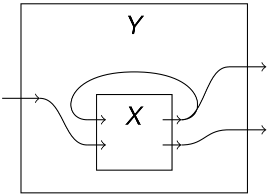
Mathematical formulation of wiring diagrams

Definition

Let $X = (\mathtt{inp}(X), \mathtt{out}(X))$ and $Y = (\mathtt{inp}(Y), \mathtt{out}(Y))$ be boxes. A *wiring diagram* $\phi\colon X \to Y$ consists of:

- a typed finite set $\mathtt{int}(\phi)$, called the set of *internal wires*,
- a typed function $\phi^{in}\colon \mathtt{inp}(X) \longrightarrow \mathtt{int}(\phi) + \mathtt{inp}(Y)$, and
- a typed function $\phi^{out}\colon \mathtt{int}(\phi) + \mathtt{out}(Y) \longrightarrow \mathtt{out}(X)$.

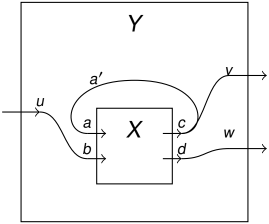
$\phi\colon X \to Y$



Example of a wiring diagram $(\mathtt{int}(\phi), \phi^{in}, \phi^{out})$

- Let X be the box with $\mathtt{inp}(X) = \{a, b\}$ and $\mathtt{out}(X) = \{c, d\}$.
- Let Y be the box with $\mathtt{inp}(Y) = \{u\}$ and $\mathtt{out}(Y) = \{v, w\}$.
- Here's a WD with internal wires $\mathtt{int}(\phi) = \{a'\}$:

$\phi\colon X \to Y$



- Here's the function $\phi^{in}\colon \mathtt{inp}(X) \longrightarrow \mathtt{inp}(Y) + \mathtt{int}(\phi)$:

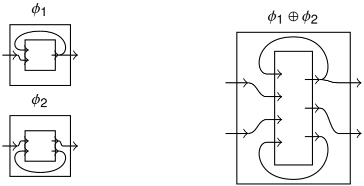
$$b \mapsto u \qquad \text{and} \qquad a \mapsto a'.$$

- Here's the function $\phi^{out}\colon \mathtt{int}(\phi) + \mathtt{out}(Y) \longrightarrow \mathtt{out}(X)$:

$$a' \mapsto c, \qquad v \mapsto c, \qquad \text{and} \qquad w \mapsto d.$$

Tensor product of wiring diagrams

- Suppose given two wiring diagrams, $\phi_1 \colon X_1 \to Y_1$ and $\phi_2 \colon X_2 \to Y_2$.
  - Say $\phi_1 = (\mathtt{int}(\phi_1), \phi_1^{in}, \phi_1^{out})$ and $\phi_2 = (\mathtt{int}(\phi_2), \phi_2^{in}, \phi_2^{out})$
- To tensor morphisms, we stack them.

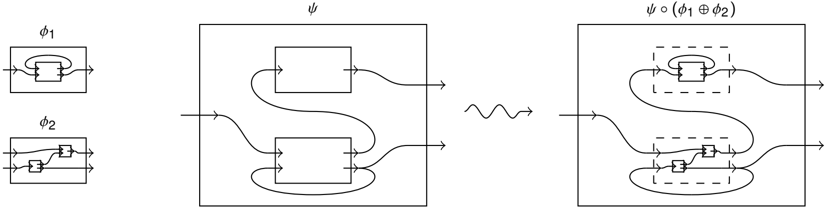


- As with boxes, tensor is achieved by summation across the board:

$$\mathtt{int}(\phi_1 \oplus \phi_2) = \mathtt{int}(\phi_1) + \mathtt{int}(\phi_2),$$
$$(\phi_1 \oplus \phi_2)^{in} = \phi_1^{in} + \phi_2^{in},$$
$$(\phi_1 \oplus \phi_2)^{out} = \phi_1^{out} + \phi_2^{out},$$

Composing wiring diagrams

- We want to be able to plug wiring diagrams into wiring diagrams.



- Quiz: what are the internal wires of $\psi \circ (\phi_1 \oplus \phi_2)$?

Composing wiring diagrams, $X \xrightarrow{\phi} Y \xrightarrow{\psi} Z$

- Recall that each wiring diagram, say ϕ, consists of
  - a typed finite set of internal wires $\mathtt{int}(\phi)$,
  - a typed function $\phi^{in}\colon \mathtt{inp}(X) \to \mathtt{int}(\phi) + \mathtt{inp}(Y)$, and
  - a typed function $\phi^{out}\colon \mathtt{int}(\phi) + \mathtt{out}(Y) \to \mathtt{out}(X)$.
- The internal wires of $\psi \circ \phi$ are $\mathtt{int}(\psi \circ \phi) := \mathtt{int}(\phi) + \mathtt{int}(\psi)$.
- The function $(\psi \circ \phi)^{in}\colon \mathtt{inp}(X) \to \mathtt{int}(\psi \circ \phi) + \mathtt{inp}(Z)$ is given by

$$\mathtt{inp}(X) \xrightarrow{\phi^{in}} \mathtt{int}(\phi) + \mathtt{inp}(Y) \xrightarrow{\mathtt{int}(\phi)+\psi^{in}} \mathtt{int}(\phi) + \mathtt{int}(\psi) + \mathtt{inp}(Z).$$

- The function $(\psi \circ \phi)^{out}\colon \mathtt{int}(\psi \circ \phi) + \mathtt{out}(Z) \to \mathtt{out}(X)$ is given by

$$\mathtt{int}(\phi) + \mathtt{int}(\psi) + \mathtt{out}(Z) \xrightarrow{\mathtt{int}(\phi)+\psi^{out}} \mathtt{int}(\phi) + \mathtt{out}(Y) \xrightarrow{\phi^{out}} \mathtt{out}(X).$$

W is a symmetric monoidal category

- Let's recap what we know about **W**.
- First of all, **W** is a category:
  - We defined an object of **W** to be a box (a pair of typed finite sets).
  - We defined a morphism $\phi\colon X \to Y$ in **W** to be a wiring diagram,

$$(\mathtt{int}(\phi), \phi^{in}, \phi^{out}).$$

  - On the last slide we showed the composition formula for $\psi \circ \phi$.
  - The identity (having $\mathtt{int}(\mathrm{id}_X) = \emptyset$) is straightforward.
  - Proving the associativity law is straightforward too.
  - So we indeed have a category.
- Add a tensor product to that, and we have an SMC.
- The tensor product needs to satisfy some laws:
  - For example, we need $X \oplus Y \cong Y \oplus X$.
  - Another example: $\square \oplus X \cong X \cong X \oplus \square$.
  - But these are all straightforward, because we're just working with finite sets and their sums.

What is a symmetric monoidal category

- A symmetric monoidal category consists of
  - a category $\mathcal{M}$,
  - a functor $\otimes\colon \mathcal{M}\times\mathcal{M}\to\mathcal{M}$, called the *tensor*,
  - an object $I\in\mathrm{Ob}(\mathcal{M})$ called the *unit*,
  - as well as various *coherence isomorphisms* and commutative diagrams that ensure that everything works as expected, e.g.
    - $X\otimes I\cong X\cong IX$,
    - $(X\otimes Y)\otimes Z\cong X\otimes(Y\otimes Z)$, etc.
- Your favorite: **Type** with Cartesian $\times$, and unit type 1.
- Another: **Set** with disjoint union $+$, and unit set $\emptyset$.
- Another: $\mathbf{Vect}_{\mathbb{R}}$ with tensor product $\otimes$, and unit vector space $\mathbb{R}$.
- Another: **W** with stacking tensor $\oplus$, and inert box $\square$.

Quick aside: how is $\otimes$ different than $\times$?

- Some people want to know how $\otimes$ is different than Cartesian product.
- Note that $(\mathbf{Set}, \times, 1)$ is an SMC, so we must be saying SMCs are more general, i.e. that $\times$ is more constrained than arbitrary $\otimes$.
  - The additional constraint on $\times$ is that you can project,

$$A \longleftarrow A \times B \longrightarrow B.$$

  - Note that $(\mathbf{Set}, +, 0)$ is an SMC, but there is no canonical map $A + B \to B$.

So... what to plug into these boxes?

- We have this syntax of boxes; what are we going to do with it?
  - We can fill these boxes with any kind of thing we want....
  - As long as we understand stacking and wiring.
- A **W**-algebra is a lax monoidal functor

$$F\colon \mathbf{W} \to \mathbf{Set}.$$

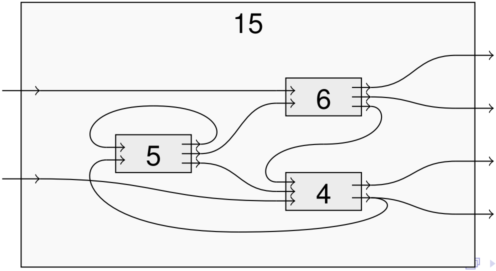
- To choose a **W**-algebra F is to choose semantics for the box syntax.

What is a lax monoidal functor $F\colon \mathbf{W} \to \mathbf{Set}$?

- Suppose we want to choose semantics F for this box syntax.
- We get to choose what we allow ourselves to put into the boxes.
  - For a box $X \in \mathrm{Ob}(\mathbf{W})$ we get to choose a set $F(X)$.
  - Once we've done so, we'll call $f \in F(X)$ an *F-fill for box X*.
- We get to say how to stack F-fills.
  - Given boxes X, Y and F-fills $f \in F(X)$ and $g \in F(Y)$,
  - we need to give an F-fill for their tensor, $\sigma(f, g) \in F(X \oplus Y)$.
- We get to say how a wiring diagram $\phi\colon X \to Y$ sends fills for X to fills for Y.
- Once we do that, we will have specified:
  - a function $\mathrm{Ob}(F)\colon \mathrm{Ob}(\mathbf{W}) \to \mathrm{Ob}(\mathbf{Set})$,
  - a function $\sigma_{X,Y}\colon F(X) \times F(Y) \to F(X \oplus Y)$, and
  - a function $\mathrm{Hom}_F\colon \mathrm{Hom}_{\mathbf{W}}(X, Y) \to \mathrm{Hom}_{\mathbf{Set}}(F(X), F(Y))$.
- For our choices to constitute a **W**-algebra, various laws must hold.

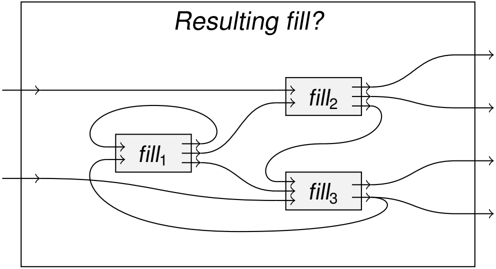
Some stupid W-algebras

- Let $\mathcal{M} = (M, \star, e)$ be any commutative monoid.
  - For example the natural numbers, with addition, $(\mathbb{N}, +, 0)$,
  - or the integers, with multiplication, $(\mathbb{Z}, *, 1)$,
  - or the subsets of some set, with union, $(\mathbb{P}(\{0, 1, \ldots, 9\}), \cup, \emptyset)$.
- Then there is an algebra $F\colon \mathbf{W} \to \mathbf{Set}$ that assigns
  - $F(X) := M$,
  - $\sigma := \star\colon M \times M \to M$, and
  - $\mathrm{Hom}_F(\phi) := \mathrm{id}_M$.
- For example, with $M = (\mathbb{N}, +, 0)$, we have



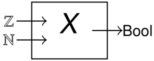
More interesting algebras **W** → **Set**

- The previous algebras didn't take advantage of the wiring structure.
- We will focus on machines, taking input-streams to output-streams.
- Variations include:
  - asking the machines to be continuous or differentiable.
  - continuous-time machines, etc.
- In each case, just say what to put into boxes and how stacking and wiring are to work.

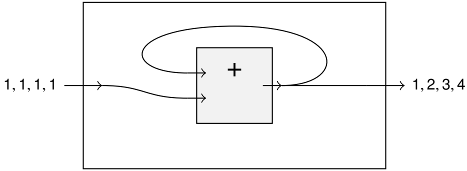


A questionable algebra

- One idea might be to put into each box the set of *functions* of the specified type.
  - That is, suppose X is the box below.
  - Define $\mathcal{F}(X) = \text{Hom}(\mathbb{Z} \times \mathbb{N}, T1 \times Bool \times T2)$, the set of functions.



- But then how do wiring diagrams operate on functions?
- Recall the running total.
  - It is made out of a pure function, but the result is not functional.
  - The same input in two successive moments returns different outputs.



State machines

Definition

Let A and B be sets. An (A, B)*-machine* consists of

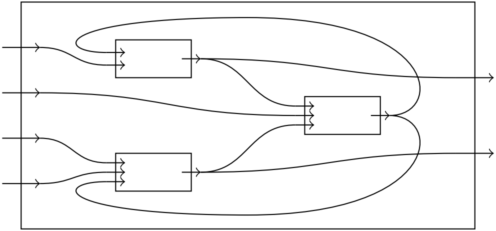
1. a set S, called the *state-set*,
2. a function $f \colon S \times A \to S \times B$, called the *state-update function*.

An (A, B)-machine is called *initialized* if we have chosen

3. an element $s_0 \in S$, called the *initial state*.

We call a machine (S, f) *simple* if its state-set has one element, $|S| = 1$.

Motivation for state machines



- My motivation: how does the brain work?
  - The architecture of the brain is of neurons with dendrites (inputs) and axons (outputs)
  - How does this architecture form a mind, i.e. something that can think?
  - What about learning, habituation, sensitization?
- The machine model may also have applications to functional reactive programming, etc, because it was designed with computation in mind.

Aside: Initialized machines act on lists

- Let (S, s_0, f) be an initialized (A, B)-machine, where $s_0 \in S$.
- For convenience, swap the outputs of the state-update function:

$$f \colon S \times A \longrightarrow B \times S.$$

- For $n \in \mathbb{N}$, we define $f_n \colon A^n \to B^n \times S$, as follows:
  - define $f_0 = s_0$, the initial state, and
  - define $f_{n+1} \colon A^{n+1} \longrightarrow B^{n+1} \times S$ to be the composite

$$A^n \times A \xrightarrow{f_n \times A} B^n \times S \times A \xrightarrow{B^n \times f} B^n \times B \times S$$

- Project each $f_n \colon A^n \to B^n \times S$ and then sum the results to obtain

$$\mathrm{LP}(S, s_0, f) \colon \mathrm{List}(A) \longrightarrow \mathrm{List}(B),$$

called the *list machine associated to* (S, s_0, f).

Fill box X with the set of $\overline{X}$-machines

- Quick aside on dependent products: notation and contravariance.
  - Given a typed finite set (I, τ) we denote the dependent product by

$$\overline{(I, \tau)} := \prod_{i \in I} \tau(i).$$

  - This is contravariant: given a typed function $p \colon (I, \tau) \to (I', \tau')$ we get

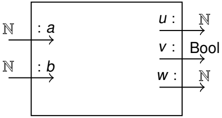
$$\overline{p} \colon \overline{(I', \tau')} \to \overline{(I, \tau)}.$$

- Recall that a box $X = (\mathtt{inp}(X), \mathtt{out}(X))$ is a pair of typed finite sets.
  - For example, if $\mathtt{inp}(X) = \{a : \mathbb{Z}, b : Bool\}$, then $\overline{\mathtt{inp}(X)} = \mathbb{Z} \times Bool$.
  - Define $\overline{X} := (\overline{\mathtt{inp}(X)}, \overline{\mathtt{out}(X)})$.
- So an $\overline{X}$-machine includes a state-set S and a state-update function

$$f \colon S \times \overline{\mathtt{inp}(X)} \longrightarrow S \times \overline{\mathtt{out}(X)}.$$

$\mathcal{P}$: **W** → **Set** on objects

- On boxes $X \in \mathrm{Ob}(\mathbf{W})$, define $\mathcal{P}(X)$ to be the set of $\overline{X}$-machines,
- For example, let $X = (\{a : \mathbb{N}, b : \mathbb{N}\}, \{u : \mathbb{N}, v : Bool, w : \mathbb{N}\})$,



- Choosing an initialized $\overline{X}$-machine means:
    - choosing a state set S, an initial state $s_0 \in S$, and a function,

$$f\colon S \times (\mathbb{N} \times \mathbb{N}) \to S \times (\mathbb{N} \times Bool \times \mathbb{N}).$$

- For example, let's choose $S = \mathbb{N} \times \mathbb{N}$, with $s_0 = (0, 0)$, and

$$f((s_1, s_2), a, b) = ((s_1 + a, s_2 + b),\ (s_1, s_1 \overset{?}{=} s_2, s_2)).$$

- This returns running totals of a and b, as well as whether they're equal.

Stacking machines

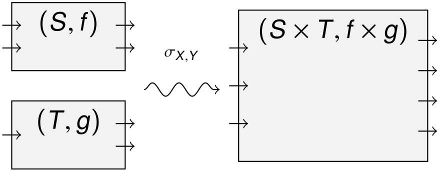
- Recall an $\overline{X}$-machine consists of a set S and a function

$$f\colon S \times \overline{\mathtt{inp}(X)} \to S \times \overline{\mathtt{out}(X)}.$$

- For any two boxes $X, Y \in \mathrm{Ob}(\mathbf{W})$, we need a stacking function

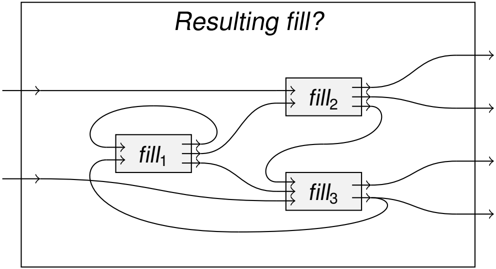
$$\sigma_{X,Y}\colon \mathcal{P}(X) \times \mathcal{P}(Y) \to \mathcal{P}(X \oplus Y).$$

  - Given an $\overline{X}$-machine (S, f) and a $\overline{Y}$-machine (T, g), we need a $\overline{X \oplus Y}$-machine.
  - We use $\sigma_{X,Y}((S, f), (T, g)) := (S \times T, f \times g)$.



Wiring machines together

- We've decided how $\mathcal{P}\colon \mathbf{W} \to \mathbf{Set}$ works on boxes $X \in \mathrm{Ob}(\mathbf{W})$;
- We've decided how $\mathcal{P}$ works with stacking.
- Now we need to decide how $\mathcal{P}$ works with wiring diagrams.



- Afterwards we need to check that the composition formula holds.

$\mathcal{P}(\phi)\colon \mathcal{P}(X) \longrightarrow \mathcal{P}(Y)$

- We begin with boxes X and Y, and a wiring diagram $\phi\colon X \to Y$.
- Recall that each wiring diagram, say ϕ, consists of
  - a typed finite set of internal wires $\mathtt{int}(\phi)$,
  - a typed function $\phi^{in}\colon \mathtt{inp}(X) \to \mathtt{int}(\phi) + \mathtt{inp}(Y)$, and
  - a typed function $\phi^{out}\colon \mathtt{int}(\phi) + \mathtt{out}(Y) \to \mathtt{out}(X)$.
- Recall the contravariance of dependent products, e.g.

$$\overline{\phi^{in}}\colon \overline{\mathtt{int}(\phi)} \times \overline{\mathtt{inp}(Y)} \longrightarrow \overline{\mathtt{inp}(X)}.$$

- Suppose given an $\overline{X}$-machine $(S, f) \in \mathcal{P}(X)$, where

$$f\colon S \times \overline{\mathtt{inp}(X)} \longrightarrow S \times \overline{\mathtt{out}(X)}.$$

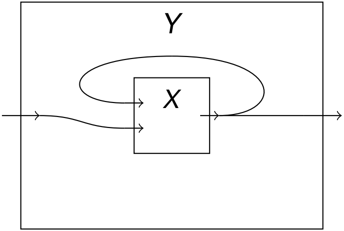
- We need to define a $\overline{Y}$-machine $(T, g) = \mathcal{P}(\phi)(S, f) \in \mathcal{P}(Y)$.
  - For the new state-set, use the product, $T := S \times \overline{\mathtt{int}(\phi)}$.
  - For the new state-update function, use the composite,

$$S \times \overline{\mathtt{int}(\phi)} \times \overline{\mathtt{inp}(Y)} \xrightarrow{S\times\overline{\phi^{in}}} S \times \overline{\mathtt{inp}(X)} \xrightarrow{f} S \times \overline{\mathtt{out}(X)} \xrightarrow{S\times\overline{\phi^{out}}} S \times \overline{\mathtt{int}(\phi)} \times \overline{\mathtt{out}(Y)}.$$

Example wiring diagram $\phi\colon X \to Y$

- Let all wires carry the pointed type $(\mathbb{N}, 0)$.
- Note that there is one internal wire, so $\overline{\mathtt{int}(\phi)} = \mathbb{N}$.

$\phi\colon X \to Y$



- Consider the $\overline{X}$-machine $(\{*\}, +)$, where $+\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is sum.
- Then $\mathcal{P}(\phi)(\{*\}, +) = (\mathbb{N}, f)$ has state-update function given by

$$f(s, y) = (s + y, s + y)$$

- As a list machine, it reports the running total as advertised, $(y_1, \ldots, y_n) \mapsto \sum_{i=1}^{n} y_i$.

Checking $\mathcal{P}$ on the composition $X \xrightarrow{\phi} Y \xrightarrow{\psi} Z$

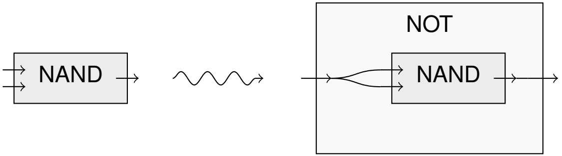
- We have defined $\mathcal{P}\colon \mathbf{W} \to \mathbf{Set}$ on objects, morphisms, and stacking.
- We must check that it works well with composition.
- The computation is very straightforward:

$$\begin{tikzcd}
S \times \overline{\mathtt{int}(\phi)} \times \overline{\mathtt{int}(\psi)} \times \overline{\mathtt{inp}(Z)} \arrow[r, "S\times\overline{\mathtt{int}(\phi)}\times\overline{\psi^{in}}"] \arrow[d, "\mathcal{P}(\psi\circ\phi)(f)"'] & S \times \overline{\mathtt{int}(\phi)} \times \overline{\mathtt{inp}(Y)} \arrow[r, "S\times\overline{\phi^{in}}"] \arrow[d, "\mathcal{P}(\phi)(f)"] & S \times \overline{\mathtt{inp}(X)} \arrow[d, "f"] \\
S \times \overline{\mathtt{int}(\phi)} \times \overline{\mathtt{int}(\psi)} \times \overline{\mathtt{out}(Z)} & S \times \overline{\mathtt{int}(\phi)} \times \overline{\mathtt{out}(Y)} \arrow[l, "S\times\overline{\mathtt{int}(\phi)}\times\overline{\psi^{out}}"'] & S \times \overline{\mathtt{out}(X)} \arrow[l, "S\times\overline{\phi^{out}}"']
\end{tikzcd}$$

- I show you this not because it's hard, but because it's easy.
  - We worked hard to make this as simple as possible.
  - Our goal was to have something people would want to use!

The subalgebra generated by NANDs?

- Each transistor on a chip acts as a NAND gate, a simple machine.



  - From here we can get NOT gates, then AND gates, and all logic gates.
  - Then *n*-bit adders, multiplication circuits, etc.
- Consider the box $T := (\{a, b : Bool\}, \{c : Bool\}) \in \mathrm{Ob}(\mathbf{W})$.
  - Begin with the *free algebra on* T, denoted $Fr(T) \colon \mathbf{W} \to \mathbf{Set}$.
  - It is the algebra that sends X to $\sum_{n \in \mathbb{N}} \mathrm{Hom}_{\mathbf{W}}(T^{\oplus n}, X)$.
  - Now, there's a unique map $Fr(T) \to \mathcal{P}$, sending $T \mapsto$ NAND.
  - Its image defines *the algebra of machines generated by NAND*.
- Question: How does it compare to the computable functions?

Morphisms of machines

- Let A and B be sets.
- Suppose we have two (A, B)-machines, (S, f) and (T, g).
- A *morphism of machines from* (S, f) *to* (T, g) consists of:
    - a function $\rho\colon S \to T$,
    - such that the following diagram commutes:

$$\begin{tikzcd} S \times A \arrow[r, "f"] \arrow[d, "\rho\times A"'] & S \times B \arrow[d, "\rho\times B"] \\ T \times A \arrow[r, "g"'] & T \times B \end{tikzcd}$$

- If we're working with initialized machines, we require $\rho(s_0) = t_0$.
- We want brains/manufacturers to reduce the complexity of their problem.

Connected machines act the same on lists

- Let (S, s_0, f) be an initialized (A, B)-machine.
  - Recall: for each $n \in \mathbb{N}$, it induces a function $A^n \to B^n$, and
  - their sum is a function $\mathrm{LP}(S, s_0, f)\colon \mathrm{List}(A) \to \mathrm{List}(B)$.
- Suppose given a morphism $\rho\colon (S, s_0, f) \to (T, t_0, g)$ of machines.
- In this case it is easy to show that $\mathrm{LP}(S, s_0, f) = \mathrm{LP}(T, t_0, g)$.
- So if two machines are connected, they act the same on lists.
  - We write $(S, s_0, f) \sim (T, t_0, g)$ if they are connected by a zigzag.
  - (Aside: zigzags are chains like this, $P_0 \leftarrow P_1 \to P_2 \leftarrow \cdots \to P_n$.)
  - The relation $\sim$ is an equivalence relation on $\overline{X}$-machines.

List(A) can always serve as state-set

- Let (S, s_0, f) be an initialized (A, B)-machine.
  - For each $n \in \mathbb{N}$, it induces a function $f_n \colon A^n \to S \times B^n$.
  - For convenience, we give names to its first and last projections,

$$\sigma_n \colon A^n \to S \qquad \text{and} \qquad \omega_{n+1} \colon A^{n+1} \to B.$$

- We'll find an equivalent machine with state-set List(A).
  - Let $T = \text{List}(A)$ and let $t_0 = [\,]$ be the empty list.
  - We need a state-update function $\widehat{f} \colon T \times A \longrightarrow T \times B$.
  - It's sufficient to provide $\widehat{f_n} \colon A^n \times A \longrightarrow A^{n+1} \times B$ for every $n \in \mathbb{N}$.
  - Use the top row in the diagram below.

$$\begin{tikzcd}
A^n \times A \arrow[r, equal] \arrow[d, "\sigma_n \times A"'] & A^{n+1} \arrow[r, "(A^{n+1}, \omega_{n+1})"] & A^{n+1} \times B \arrow[d, "\sigma_{n+1} \times B"] \\
S \times A \arrow[rr, "f"'] & & S \times B
\end{tikzcd}$$

  - The rest of the diagram shows the morphism $(T, t_0, \widehat{f}) \to (S, s_0, f)$.

State reduction

- For any (A,B)-machine (S, s_0, f) we found a morphism

$$\rho\colon (\mathrm{List}(A), [\,], \widehat{f}) \longrightarrow (S, s_0, f).$$

  - In fact ρ is unique.
  - The image of ρ is some (S', s_0, f) having a subset of states $S' \subseteq S$.
  - S' is the set of *reachable states*, those that obtain on some list of input.
- We can also quotient by an equivalence relation on states.
  - Declare two states equivalent if they act the same on any input list.
  - We have $\mathrm{LP}(S, -, f)\colon S \longrightarrow \mathrm{List}(B)^{\mathrm{List}(A)}$.
  - Let $\widetilde{S}$ be its image, so we have $q\colon S \twoheadrightarrow \widetilde{S} \subseteq \mathrm{List}(B)^{\mathrm{List}(A)}$.
  - So $\widetilde{S}$ is the quotient of S by the equivalence relation.
  - It is easy to show that $\widetilde{S}$ is the state-set for an equivalent machine.

$$\begin{tikzcd}
S \times A \arrow[r, "f"] \arrow[d, "q\times A"'] & S \times B \arrow[d, "q\times B"] \\
\widetilde{S} \times A \arrow[r, "\widetilde{f}"'] & \widetilde{S} \times B
\end{tikzcd}$$

Algorithmic state reduction

- Given an (A, B)-machine, we want the smallest equivalent one.
  - If (S, s_0, f) is such that every state is reachable, use $(\widetilde{S}, s_0, \widetilde{f})$.
  - In this case, and if A and S are finite, Hopcroft's algorithm finds the smallest equivalent machine $(\widetilde{S}, s_0, \widetilde{f})$ in $O(|S||A|log|S|)$ time.
  - If some states are not reachable, use $\left(\widetilde{\mathrm{List}(A)}, [\,], \widetilde{\widetilde{f}}\right)$.
- Call this the *minimal reduction* of (S, s_0, f).
- It is a normal form for machines.

State reduction and wiring diagrams

- Back to the main theme, we had $\mathcal{P}\colon \mathbf{W} \to \mathbf{Set}$.
- But in fact it can be extended to a monoidal functor $\mathcal{P}\colon \mathbf{W} \to \mathbf{Cat}$.
  - For each $X \in \mathrm{Ob}(\mathbf{W})$ we now have a category $\mathcal{P}(X)$ of machines.
  - For stacking boxes, there's a functor $\mathcal{P}(X) \times \mathcal{P}(Y) \to \mathcal{P}(X \oplus Y)$.
  - For each WD $\phi\colon X \to Y$ there's a functor $\mathcal{P}(\phi)\colon \mathcal{P}(X) \to \mathcal{P}(Y)$.
  - And these all work together as required.
- This means that reducing commutes with wiring.
  - Given a morphism $\phi\colon X \to Y$ and a machine $P \in \mathcal{P}(X)$,
  - you can reduce $P \twoheadrightarrow P'$ then apply $\mathcal{P}(\phi)$,
  - and the result is a reduction, $\mathcal{P}(\phi)(P) \twoheadrightarrow \mathcal{P}(\phi)(P')$.

Invariants for state machines?

- We have a functor $\mathcal{P}\colon \mathbf{W} \to \mathbf{Cat}$.
- If X is an object, every morphism in $\mathcal{P}(X)$ acts like an equivalence.
  - That is, its domain and codomain machines treat lists the same way.
- An invariant of machines should respect this kind of equivalence.
  - Let $\pi_0\colon \mathbf{Cat} \to \mathbf{Set}$ be the "connected components" functor.
  - We want to understand the functor $\pi_0\mathcal{P}\colon \mathbf{W} \to \mathbf{Set}$.
- Can you find: a functor I and a natural transformation q:

$$\begin{tikzcd} \mathbf{W} \arrow[r, bend left, "\pi_0\mathcal{P}"] \arrow[r, bend right, "I"'] \arrow[r, phantom, "q\Downarrow"] & \mathbf{Set} \end{tikzcd}$$

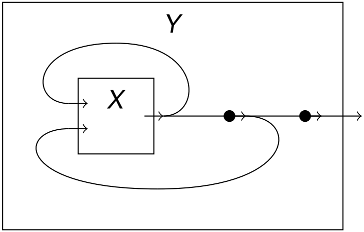
- (We want I to be non-trivial and q to be surjective.)

How's the time?

Shall we change gears a bit, or skip to the end?

Wiring diagrams as a visual language

- One major feature of wiring diagrams is to engage the human visual system.
  - Operadic pictures are a visual language for building instructions.
  - The category **W** purely syntactic.
- We can build predefined functions into **W**.
  - For example, delay machines might be denoted by nodes .

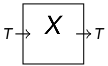


  - Or machines that bundle four wires into a bus might be denoted by 

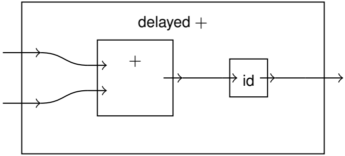
.

Aside: timing in a wiring diagram

- The formulas are written above; here we interpret them in terms of timing.
  - Wires move data instantaneously.
  - Each machine takes one “clock-cycle” to process data.
- Consider a box X with one input wire and one output wire, $\mathtt{inp}(X) = \{T\} = \mathtt{out}(X)$ of the same type, T.

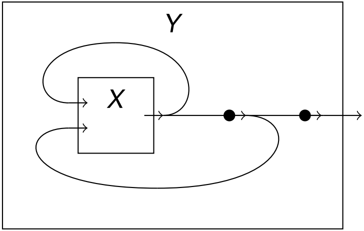


- We define the *delay machine of type T* to be the simple machine with state-update function $\mathrm{id}_T \colon T \to T$.



Baking in special machines

- What does it mean to bake the delay node $\multimap\!\!\bullet$, etc., into **W**?
- We want the following to count as a wiring diagram $\phi\colon X \to Y$.



- That is, we name special boxes for which we have chosen interpretations.
- What's the math?

The math for baking in special symbols, part 1

- We need to choose special symbols in **W** and machines for them.
  - Fix a SMC, $\mathcal{S}$, objects of which are called *special symbols*.
  - Fix a strong monoidal functor $\iota\colon \mathcal{S} \to \mathbf{W}$.
  - For each symbol $s \in \mathrm{Ob}(\mathcal{S})$ choose an element $m_s \in \mathcal{P}(\iota(s))$.

$$\begin{tikzcd} \mathcal{S} \arrow[r, "\iota"] \arrow[d, "!"'] \arrow[dr, phantom, "m \nearrow"] & \mathbf{W} \arrow[d, "\mathcal{P}"] \\ \bullet \arrow[r, "\{1\}"'] & \mathbf{Set} \end{tikzcd}$$

- Now define a new SMC, denoted $\mathbf{W}[\mathcal{S}]$ as follows:
  - It has the same objects as **W**, but morphisms are defined as:

$$\mathrm{Hom}_{\mathbf{W}[\mathcal{S}]}(X, Y) = \sum_{s \in \mathrm{Ob}(\mathcal{S})} \mathrm{Hom}_{\mathbf{W}}(X \oplus \iota(s), Y).$$

  - Given $X \oplus \iota(s) \to Y$ and $Y \oplus \iota(t) \to Z$,
  - we can compose to $X \oplus \iota(s \oplus t) \to Z$, because ι is strong.
  - Stacking $\oplus$ in $\mathbf{W}[\mathcal{S}]$ is also achieved by the strong-ness of ι.

The math for baking in special symbols, part 2

- We have constructed a SMC denoted $\mathbf{W}[\mathcal{S}]$ out of our setup,

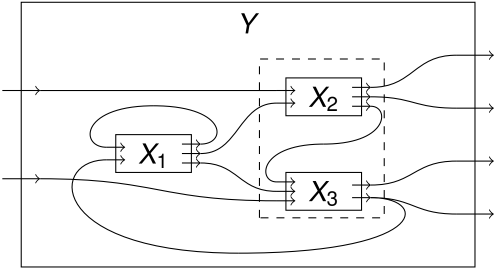
$$\begin{tikzcd} \mathcal{S} \arrow[r, "\iota"] \arrow[d, "!"'] \arrow[dr, phantom, "m\nearrow"] & \mathbf{W} \arrow[d, "\mathcal{P}"] \\ \bullet \arrow[r, "\{1\}"'] & \mathbf{Set} \end{tikzcd}$$

  - Note we haven't used m yet, we've only used ι up to now.
- We need an algebra $\mathcal{P}[\mathcal{S}] \colon \mathbf{W}[\mathcal{S}] \to \mathbf{Set}$.
  - Have it act the same on boxes as $\mathcal{P}$ does: $\mathcal{P}[\mathcal{S}](X) := \mathcal{P}(X)$.
  - A morphism $\phi \colon X \to Y$ in $\mathbf{W}[\mathcal{S}]$ is a morphism $\phi \colon X \oplus \iota(s) \to Y$ in $\mathbf{W}$.
  - We need to assign a function $\mathcal{P}[\mathcal{S}](\phi) \colon \mathcal{P}(X) \to \mathcal{P}(Y)$.
  - Use the following composite:

$$\mathcal{P}(X) \cong \mathcal{P}(X) \times \{1\} \xrightarrow{\mathcal{P}(X)\times m} \mathcal{P}(X) \times \mathcal{P}(\iota(s)) \xrightarrow{\cong} \mathcal{P}(X \oplus \iota(s)) \xrightarrow{\mathcal{P}(\phi)} \mathcal{P}(Y).$$

Summary

- We can draw pictures like this:



- Such a picture represents a morphism $\phi\colon X_1 \oplus X_2 \oplus X_3 \longrightarrow Y$ in a symmetric monoidal category called **W**.
- We can fill each interior box of ϕ with a machine, and thus derive a machine for the exterior box.
- We can abstract away the details of any part by enclosing it.
- The requisite formulas are straightforward and written out here in full.

Thanks!

Thanks for inviting me!