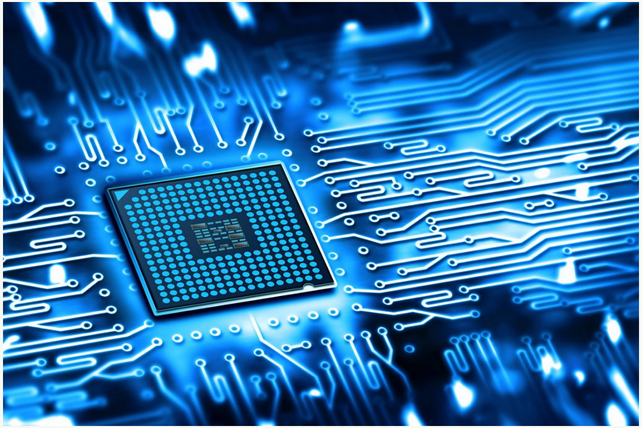
A Fully Compositional Theory of Digital Circuits

George Kaye
University of Birmingham

15 February 2024 – PPLV Research Seminar

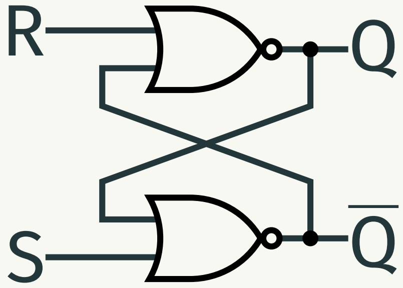
What are we going to be talking about?

Digital circuits!



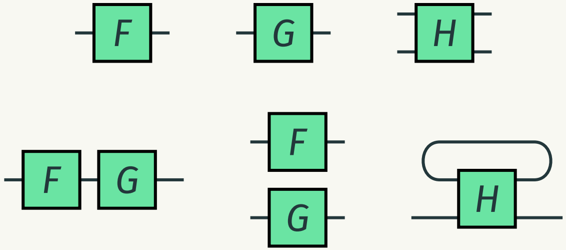
What are we going to be talking about?

Digital circuits!



What are we going to be talking about?

We want a compositional theory of digital circuits.



Using string diagrams removes much of the bureacracy

(also they look pretty)

The story so far

How did we get here?

The story so far

2003

The story so far

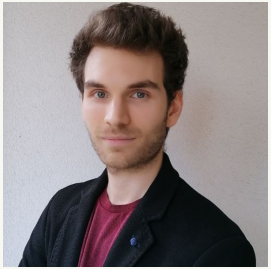
Yves Lafont

‘Towards an algebraic theory of Boolean circuits’

The story so far

2016

The story so far



Dan Ghica, Achim Jung, Aliaume Lopez

‘Diagrammatic semantics for digital circuits’

The story so far

‘Wow, this guy seems pretty groovy’

OCaml noises

The story so far

2019

The story so far

‘No’

‘Okay’

‘Do you know category theory’

‘Do you want to do circuits stuff’

The story so far

2021

The story so far

+ =

+ =

The story so far

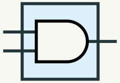
David Sprunger

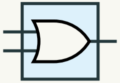
(now at Indiana State University)

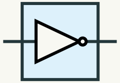
Let’s get dangerous

Combinational circuit components

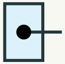
gates

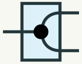

AND gate

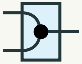

OR gate

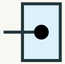

NOT gate

(co)monoid structure

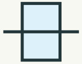

disconnected

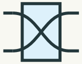

fork


join


stub

categorical structure


identity


symmetry

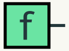
Light circuits

only contain gates and structure.

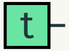
(actually, we do it more generally than this, but let's keep it simple)

Sequential circuit components

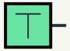
Values



false

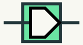


true

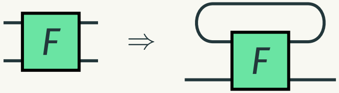


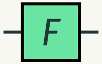
short circuit

Delay



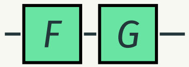
Feedback

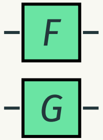


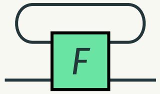
Dark circuits  may contain delay or feedback.

Building circuits

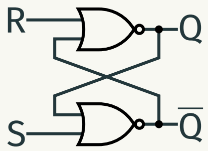
Circuits are morphisms in a freely generated symmetric traced monoidal category (STMC).

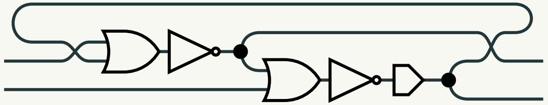






Need an example?





We need some meaning

What is the meaning?

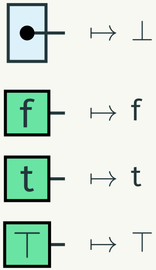
We need some meaning

Denotational semantics

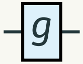
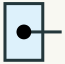
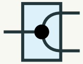
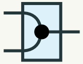
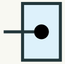
Interpreting the values

Values are interpreted in a lattice:

$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[dr, dash] & \\
t \arrow[dr, dash] & & f \arrow[dl, dash] \\
 & \bot &
\end{tikzcd}$$



Let’s make everything a function

	monotone functions	$\overline{g} : \mathbf{V}^m \to \mathbf{V}$
	initialise	$() \mapsto (\bot)$
	copy	$x \mapsto (x, x)$
	join in the lattice	$(x, y) \mapsto x \sqcup y$
	discard	$x \mapsto ()$

Feedback is interpreted as the least fixed point.

Functions are not enough

How do we model delay?

Streams!

Streams

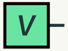
A stream $\mathbf{V}^\omega$ is an infinite sequence of values.

$$v_0 :: v_1 :: v_2 :: v_3 :: v_4 :: v_5 :: v_6 :: v_7 :: \cdots$$

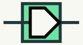
A stream function $\mathbf{V}^\omega \to \mathbf{V}^\omega$ consumes and produces streams.

$$f(v_0 :: v_1 :: v_2 :: v_3 :: v_4 :: \cdots) = w_0 :: w_1 :: w_2 :: w_3 :: w_4 :: \cdots$$

Interpreting the sequential components



$$() := v :: \bot :: \bot :: \bot :: \cdots$$



$$(v_0 :: v_1 :: v_2 :: \cdots) := \bot :: v_0 :: v_1 :: v_2 :: \cdots$$

Maybe there are too many streams

Does every circuit correspond to a stream function $(\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$?

No.

(but this is to be expected!)

Restricting the stream functions

Circuits are causal.

They can only depend what they've seen so far.

Circuits are monotone.

They are constructed from monotone functions.

Is that all? Not quite... (but we'll get there)

Some operations on stream functions

Given a causal stream function $f : (\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$ and an element $a \in \mathbf{V}^m$...

initial output $\quad f[a] \in \mathbf{V}^n$

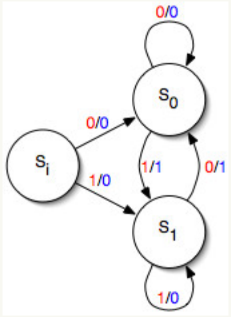
‘the first thing f produces given a’

stream derivative $\quad f_a \in (\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$

‘how f behaves after seeing a first’

Hold on, these look familiar...

An old friend



Mealy machines!

Stream functions are the *states* in a Mealy machine.

Circuits have finitely many behaviours

Circuits have a finite number of components.

So there are finite number of states in the Mealy machine.

So the outputs of streams given some input must be periodic.

(There are finitely many stream derivatives).

These are the streams we're looking for

Theorem

A stream function is the interpretation of a sequential circuit if and only if it is ***causal***, ***monotone*** *and has* ***finitely many stream derivatives***.

Sound and complete denotational semantics

Doing something useful

Suppose we have two circuits
with the same denotation

$$[\![-\boxed{F}-]\!] = [\![-\boxed{G}-]\!]$$

What does this tell us about the
structure of these circuits?

Reducing it down

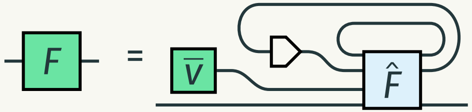
Operational semantics

Reducing it down

We want to find a set of reductions for digital circuits

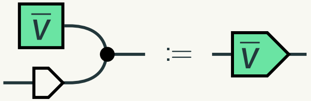
We want to reduce circuits to their outputs syntactically in a step-by-step manner

Going global

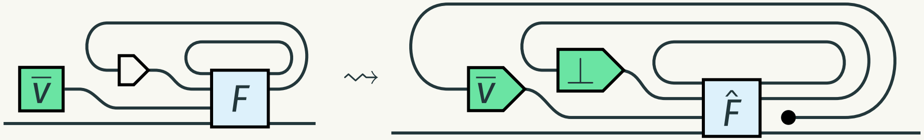


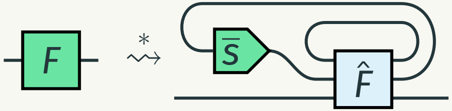
by moving boxes and wires around

Going global



‘Register’



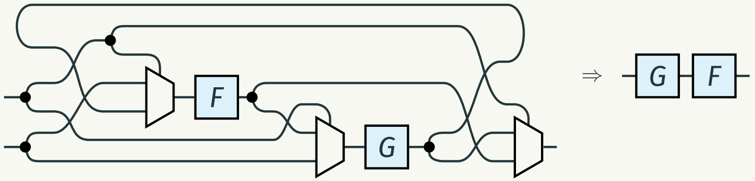


The sticking point

What are we going to do about the non-delay-guarded trace?

In industry, feedback is usually delay-guarded.

But this rules out some clever circuits!



(And also it would be cheating)

Getting rid of non-delay-guarded feedback

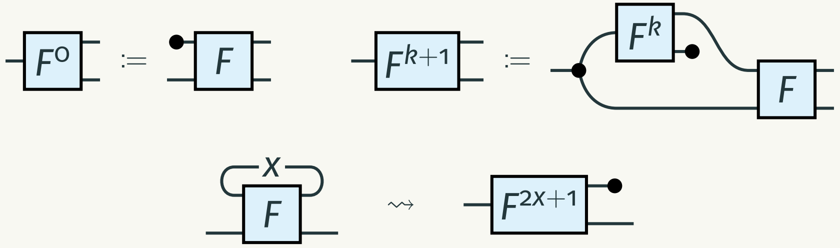
$\mathbf{V}$ is a finite lattice...

The functions are monotone...

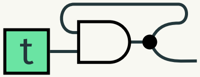
We can compute the least fixed point

in finite iterations!

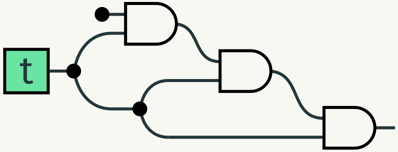
Getting rid of non-delay-guarded feedback



Getting rid of non-delay-guarded feedback

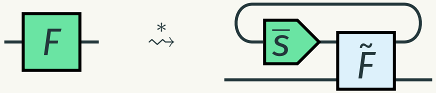


$\rightsquigarrow$



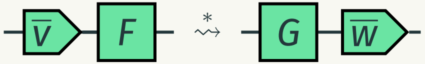
Here’s Mealy

For any circuit



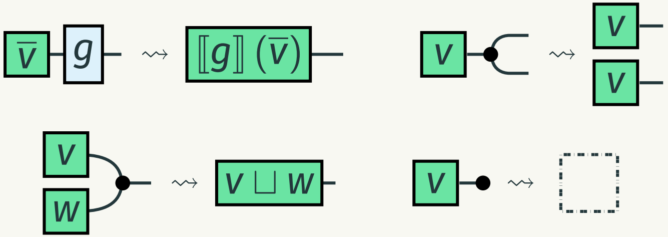
What is the goal

We want to compute the outputs of circuits given some inputs



How does a circuit process a value?

Reducing values



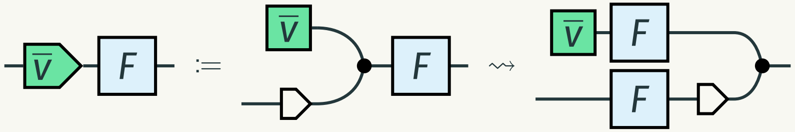
”

Lemma

For every $-\boxed{F}-$ *there exists* $\boxed{\overline{w}}-$ *s.t.* $\boxed{\overline{v}}-\boxed{F}- \rightsquigarrow \boxed{\overline{w}}-$.

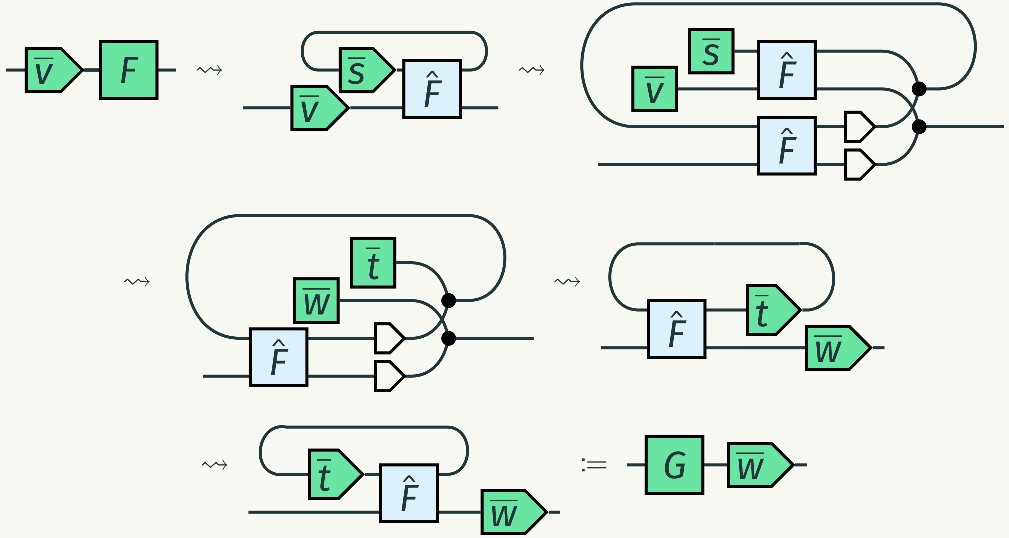
Catching the jet stream

What about delays?



'Streaming'

Catching the jet stream



Observe this

When are two circuits observationally equivalent?

Circuits have finitely many states...

Definition

Two circuits with at most c delay components are observationally equivalent if the reduction procedure creates the same outputs for all inputs of length $|\mathbf{V}|^c + 1$.

Observe this

Theorem

Two circuits are observationally equivalent if and only if they are denotationally equivalent.

Sound and complete operational semantics

Still takes a while

This is a superexponential upper bound for testing circuit equivalence

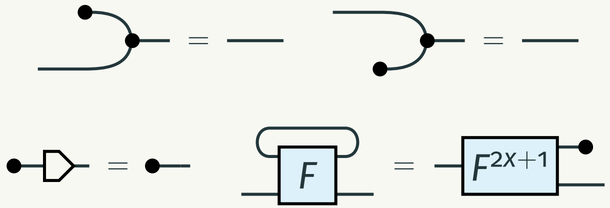
Can we do better?

Efficiency is everything

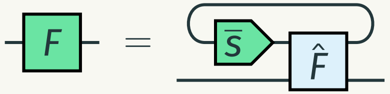
Algebraic semantics

Mealy is so back

First things first...



By these equations,



It’s completely normal

We want a way to use equations to translate a circuit into another circuit with the same behaviour

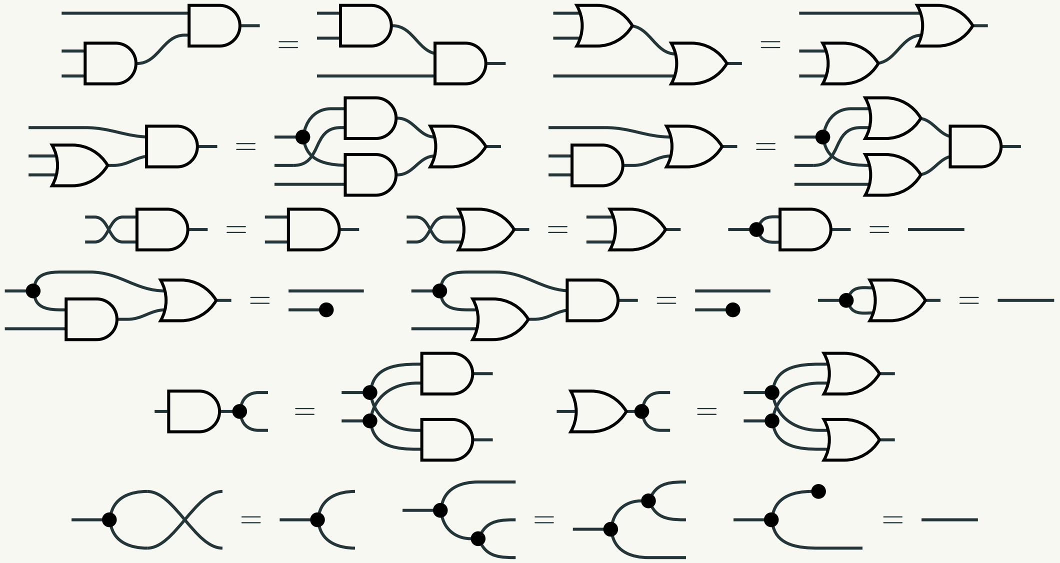
It's completely normal

Say we have a procedure $|-|$ for establishing a canonical circuit for a function $f : \mathbf{V}^m \to \mathbf{V}^n$

A circuit is normalised if it is in the image of $|-|$

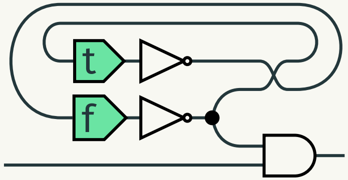
What equations are needed to normalise any circuit?

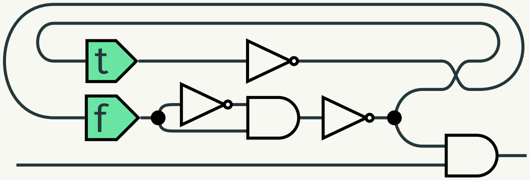
It's completely normal



It’s completely normal

Is this enough?



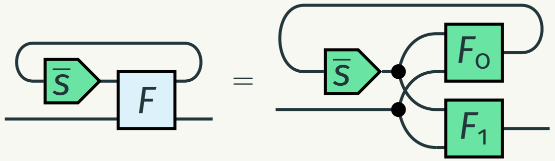


The cores may not have the same semantics!

Idea: encode circuits so their state words have length equal to the number of states

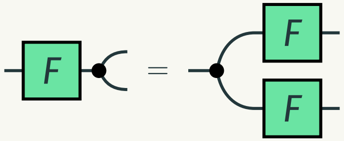
State your intention

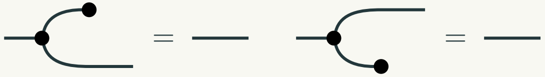
Need to know the number of states: isolate the state transition and output

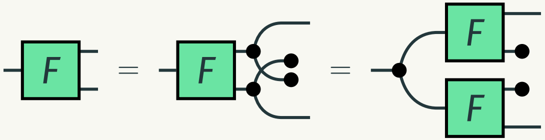


Cartesian doubt

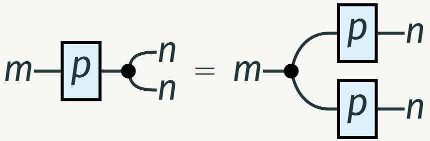
Need equations to make the fork natural and unital

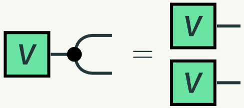


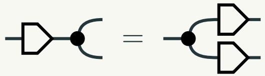


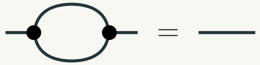


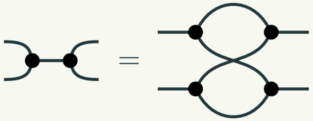
Cartesian doubt





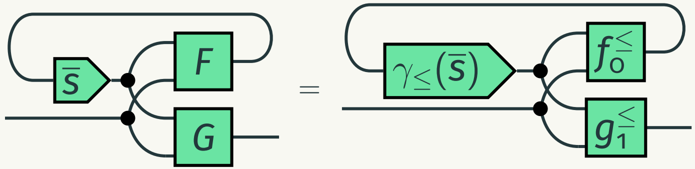






Encoding

Now add an encoding equation



(I’m hiding some of the internal machinations here)

It is complete

By the completeness of the denotational semantics, each stream function has a corresponding encoded circuit...

Theorem

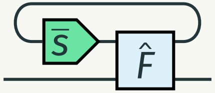
Two circuits are equal by the equations if and only if they are denotationally equal.

Sound and complete algebraic semantics

Making it combinatorial

Graph rewriting

Making it combinatorial

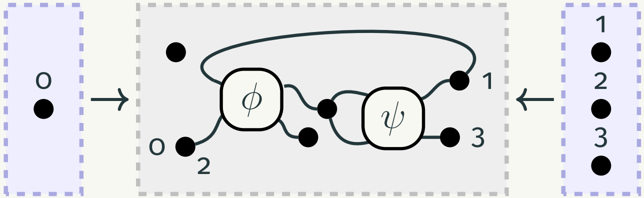


$\Rightarrow$

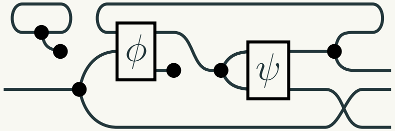
It is hard for computers to work with string diagrams...

...but computers love graphs!

A hyper kind of graph



There are correspondences between certain classes of hypergraphs and circuit string diagrams.



Using the correspondence

$$\begin{tikzcd}
\text{Circuit string diagram} \arrow[r] \arrow[d, equal] & \text{Hypergraph representation} \arrow[d, "\text{Graph rewrite}"] \\
\text{New circuit string diagram} & \text{Rewritten hypergraph} \arrow[l]
\end{tikzcd}$$

Implementing it all

The rewriting framework has been *implemented* in a hardware description language.

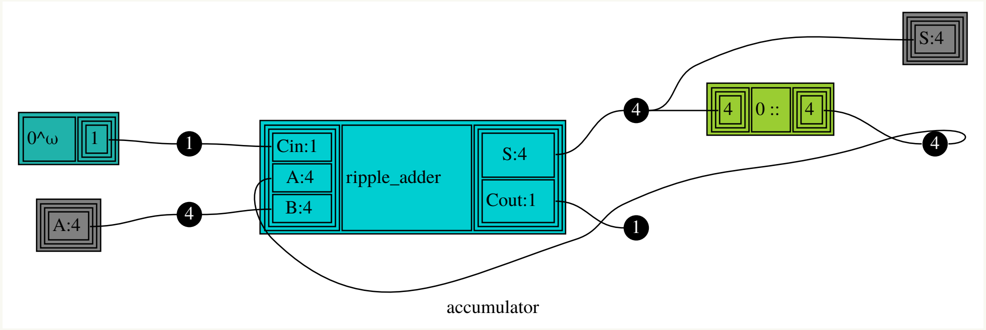
(the language is still in development)

(so I’ve been warned not to accidentally announce anything)

(also they changed everything so I can’t actually compile it at the moment)

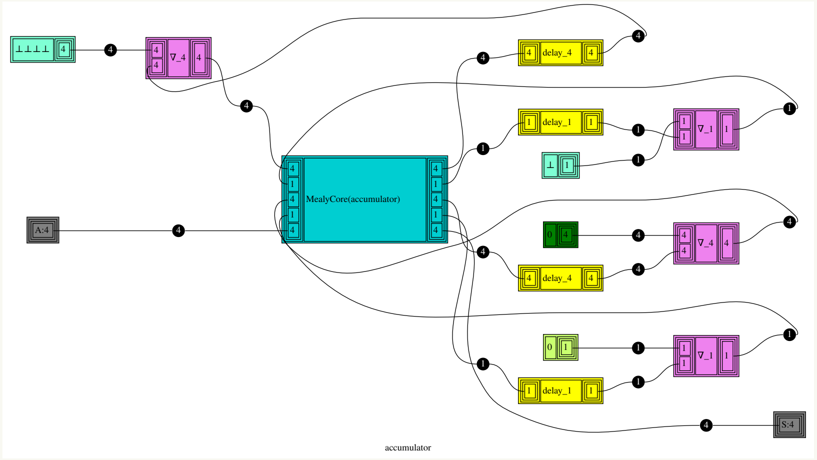
I can still show you something

Create a circuit...



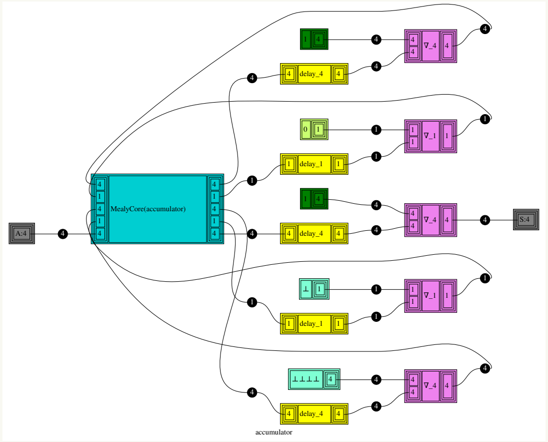
I can still show you something

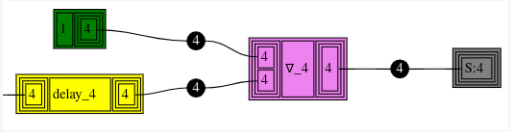
The evaluator converts it into Mealy form...



I can still show you something

...and then evaluates an input.





To the end

Three different semantics for sequential digital circuits

$$\begin{tikzcd}
 & \text{Denotational} \arrow[dl, leftrightarrow, "\cong"'] \arrow[dr, leftrightarrow, "\cong"] & \\
\text{Operational} \arrow[rr, leftrightarrow, "\cong"'] & & \text{Algebraic}
\end{tikzcd}$$

Can adapt for automatic reasoning using graph rewriting