Flipping Pebbles

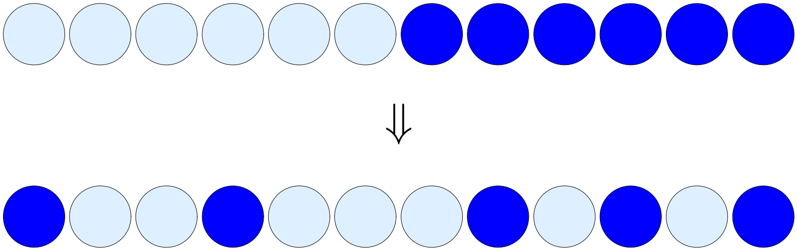
Klaus Sutner

Carnegie Mellon University

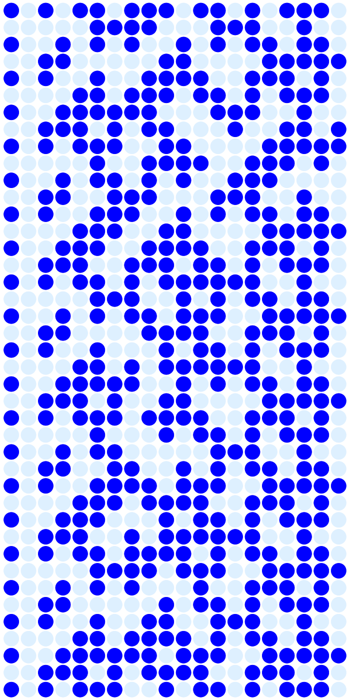
1 A Pebble Game

- The ToC Angle

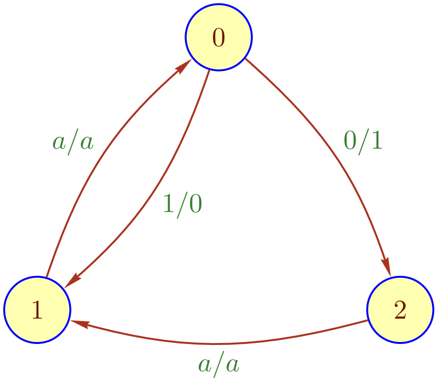
Flipping Pebbles



Given a row of tokens, white on one side, blue on the other. Flip the first token. If it was white, skip the next two tokens; otherwise, skip just one token. Keep flipping till you fall off the end.



Getting Serious



Our pebble-flipping operation is really a word map defined by a 3-state automaton $\mathcal{A}$.

The Word Maps

We can choose an initial state arbitrarily and write $\underline{0}$, $\underline{1}$ and $\underline{2}$ for the corresponding maps $\mathbf{2}^\star \to \mathbf{2}^\star$. The pebble flipping operation corresponds to $\underline{0}$ (map $\underline{i}$ copies i bits, then runs $\underline{0}$).

We can write the maps in terms of simultaneous recursion on words, $a \in \mathbf{2}$, $x \in \mathbf{2}^\star$:

$$\underline{0}(0x) = 1\,\underline{2}(x)$$

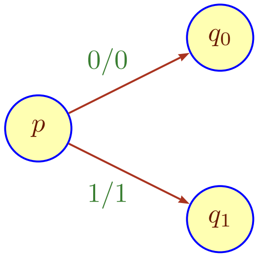
$$\underline{0}(1x) = 0\,\underline{1}(x)$$

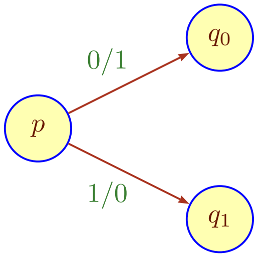
$$\underline{1}(ax) = a\,\underline{0}(x)$$

$$\underline{2}(ax) = a\,\underline{1}(x)$$

Invertible Mealy Machines

There are only two types of states: copy and toggle.


$$\begin{tikzcd} & q_0 \\ p \arrow[ur, "0/0"] \arrow[dr, "1/1"'] & \\ & q_1 \end{tikzcd}$$


$$\begin{tikzcd} & q_0 \\ p \arrow[ur, "0/1"] \arrow[dr, "1/0"'] & \\ & q_1 \end{tikzcd}$$

As a consequence, there is no loss of information when applying the maps $\underline{k}$, they are all length-preserving bijections on $\mathbf{2}^\star$.

Recall: DFAs

These machines are very similar to DFAs, but slightly different.

- There is no initial state (computation can start at any state).
- There are no final states (computation just goes on and on).
- The edge labels are not in $\mathbf{2}$ but in $\mathbf{2} \times \mathbf{2}$ (it's a transducer, not an acceptor).

Sequential Maps

Our maps are special in the sense that there is no look-ahead:

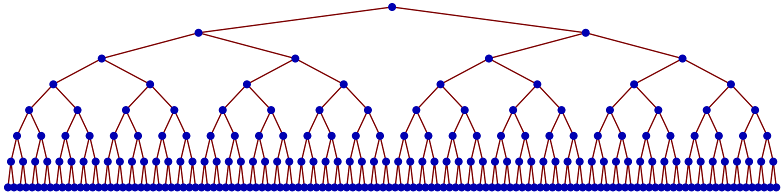
$$f(uv) = f(u)\, g(v)$$

where $f, g \in \{\underline{0}, \underline{1}, \underline{2}\}$.

Key Idea:

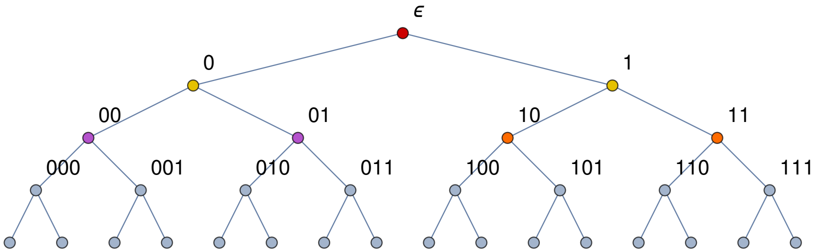
Think of such maps as automorphisms of the infinite binary tree.

Tree Automorphisms



We are interested in any permutation that preserves the tree structure (root to root, children to children).

As Word-Maps



Suppose we want to interchange the two orange nodes and their subtrees (everything else stays). Sequential map

$$f(0\,x) = 0\,x$$
$$f(1\,a\,x) = 1\,\overline{a}\,x$$

Getting Really Serious

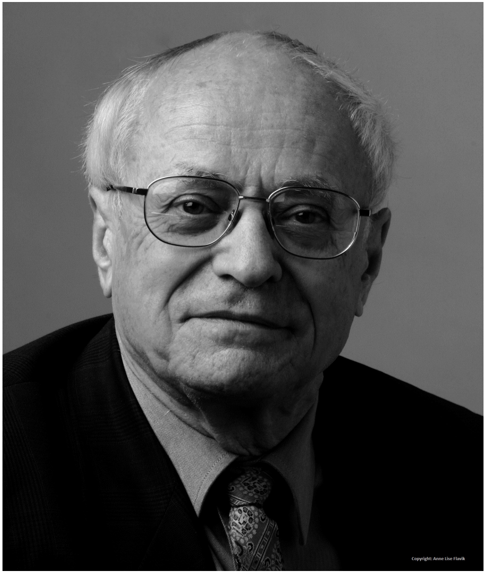
The collection of all automorphisms of $\mathbf{2}^\star$ forms a huge uncountable group

$$\mathsf{Aut}(\mathbf{2}^\star)$$

Group theorists are very interested in particular subgroups $G \subseteq \mathsf{Aut}(\mathbf{2}^\star)$.

This is idea is about 40 years old.

Jean-Pierre Serre



Fields Medal, Abel Prize, Steele Prize, Wolf Prize

Member Bourbaki

Arbres, Amalgames, SL_2 (1977)

A New Idea: Automata

In general, the subgroups $G \subseteq \mathsf{Aut}(\mathbf{2}^\star)$ are hopelessly complicated.

But how about the subgroups that can be defined in terms of a finite state machine?

We take the group generated by the all the maps $\underline{p}$ where p ranges over the state set:

$$G = \{\underline{0}, \underline{1}, \underline{2}, \underline{0}^{-1}, \underline{0}^2, \underline{0}\,\underline{1}, \underline{0}^{-1}\underline{2}, \ldots\}$$

Milnor's Question

Suppose your group has generators $\Sigma = \{g_1, \ldots, g_k\}$.

Words over Σ correspond to group elements.

Get a growth function

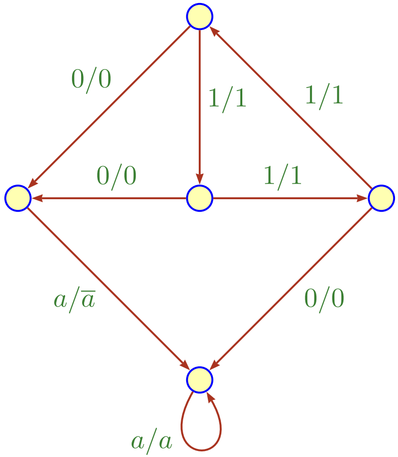
$$\gamma(n) = \#\ \text{group elm. obtained from}\ \Sigma^n$$

It is easy to find examples where γ is

- polynomial
- exponential

Is there anything in between?

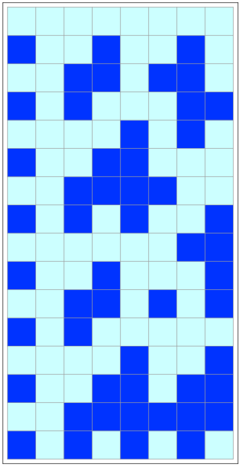
Grigorchuk Group



■ A Pebble Game

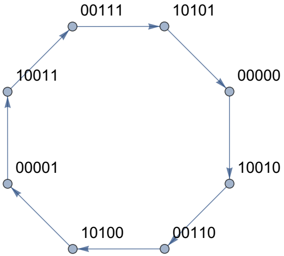
2 The ToC Angle

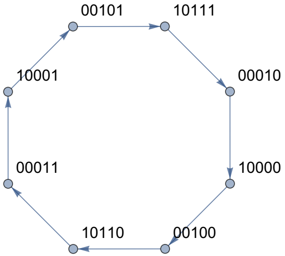
Orbits

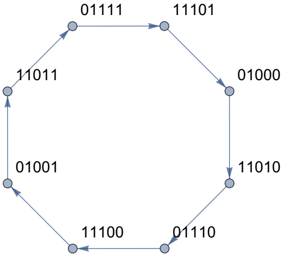


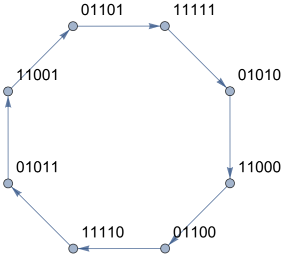
Applying a map $\underline{p}$ over and over to a given word, we get a cycle, the orbit of the word.

Length 5, $\underline{0}$

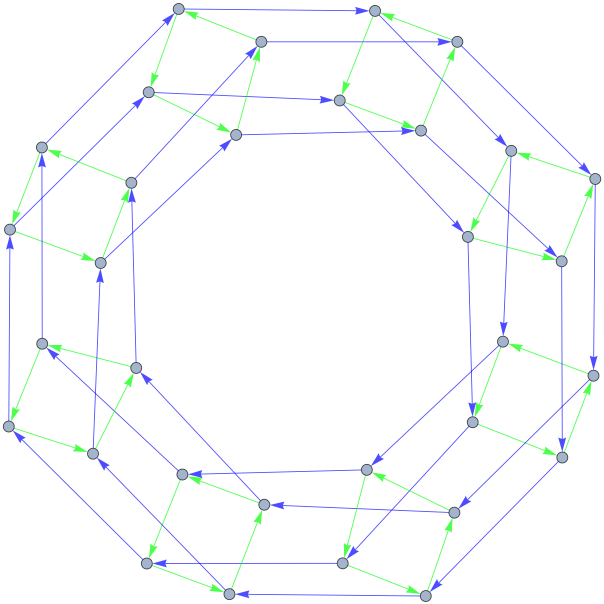








Length 5, $\underline{0}$, $\underline{1}$



Obvious Questions

- How long are these orbits?
- How many orbits are there of a given length?

Of course, we want the answer for all of $\mathbf{2}^k$ in terms of k.

For our pebble game, the answer is pretty nice.

Computational Attack

	1	2	4	8	16	32
0	1					
1		1				
2		2				
3			2			
4			4			
5				4		
6				8		
7					8	
8					16	
9						16
10						32

$\rightarrow$: orbit length, $\downarrow$: word length, missing entries are 0

Counting Orbits

Conjecture

For words of length k, there are $2^{\lfloor k/2 \rfloor}$ orbits.

The length of each orbit is $2^{\lceil k/2 \rceil}$.

Of course, an actual proof requires some kind of idea as to why this should be true.

Exercise

Show that every orbit must have length 2^ℓ for some ℓ.

Orbit Problem

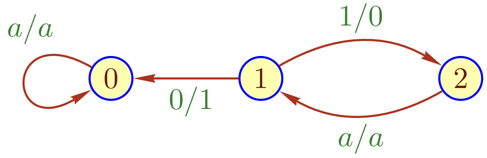
Suppose we have two words x and y of length k.

How hard is it to test if x and y lie on the same orbit?

Suppose x and y lie on the same orbit.

How hard is to compute a timestamp t such that $\underline{0}^t(x) = y$?

Digression: An Easy Case



For this automaton, one can give a simple description of all the maps $\underline{p}$.

As a consequence, the orbit problem is almost trivial.

Non-Piazza Poll

Suppose you have $x, y \in \mathbf{2}^{1,000,000}$.

How long does it take to check whether y is in the orbit of x under our pebble flipping map $\underline{0}$?

- one second
- one minute
- one hour
- one year
- longer than the lifespan of the universe
- requires a quantum computer with 1,000,000 qbits
- it's clearly undecidable

Some Surprising Facts

To simplify matters slightly, let us ignore the inverse maps $\underline{p}^{-1}$ for the time being.

Techically, let $\mathcal{S}$ be the semigroup generated by our three basic maps $\underline{0}$, $\underline{1}$ and $\underline{2}$. Thus $\mathcal{S}$ consists of all compositions that generically look like

$$f = \underline{0}\,\underline{0}\,\underline{2}\,\underline{1}\,\underline{0}\,\underline{1}\,\underline{1}\,\underline{2}$$

A priori, it is completely unclear what maps like f look like, or how they interact.

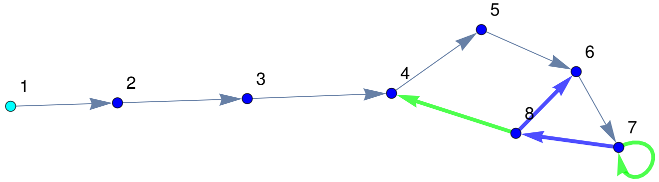
Recall: Product Automata

We saw a theorem in class that says that whenever f and g are rational, so is their composition $f \circ g$.

Moreover, a transducer for $f \circ g$ can be constructed from the machines for f and g (product construction).

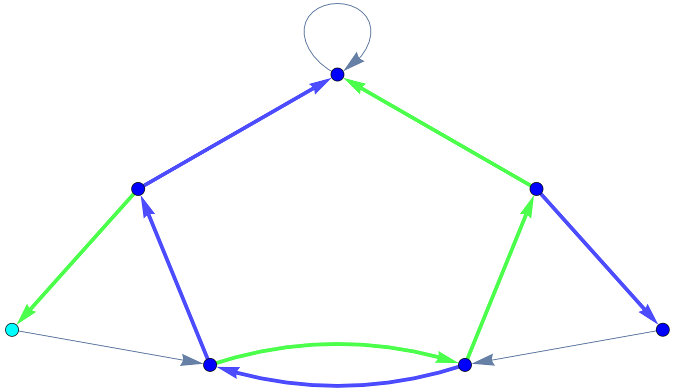
$f = \underline{0}^8$

Here is the product automaton for $f = \underline{0}^8$:



gray: a/a, green $0/1$, blue $1/0$

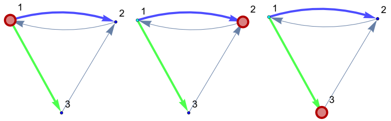
$f = \underline{0}^2\underline{1}\,\underline{2}^2$



Computability

So $\mathcal{S}$ is a computable structure: there are finite data structures to represent the elements of $\mathcal{S}$, and we can manipulated them algorithmically to deal with composition.

$\underline{0}$, $\underline{1}$, $\underline{2}$



$f \circ g$

$\mathcal{A} \otimes \mathcal{B}$

That's nice, but we want to understand its structure. Just because we have an algorithm does not mean we understand what is going on.

Richard Hamming

The purpose of computation is insight, not numbers.

Commutativity

Lemma

$\mathcal{S}$ is commutative.

There are two proofs for this:

- induction on the length of an input word, or
- building the corresponding product automata and checking for isomorphism.

The second approach has one huge advantage: it is easily automated.

Simplify

So every map in $\mathcal{S}$ can be written as

$$f = \underline{0}^a \, \underline{1}^b \, \underline{2}^c$$

where $a, b, c \in \mathbb{N}$.

Even more concisely, we can write each of these maps as $(a, b, c) \in \mathbb{N}^3$.

It's a Group

But how about the inverses that are apparently missing from $\mathcal{S}$?

We don't need to add them, they pop up automatically.

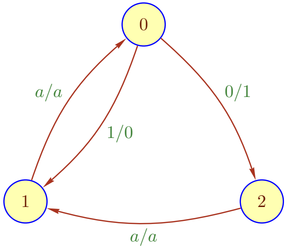
This can be seen from the unexpected identity

$$\underline{0}^2\,\underline{1}^2\,\underline{2} = I$$

So for example $\underline{2}^{-1} = \underline{0}^2\,\underline{1}^2$.

Digression: Markov Chain

We can think of the transition matrix of $\mathcal{A}$ as a stochastic matrix:



$$B = \begin{pmatrix} 0 & 1 & 0 \\ 1/2 & 0 & 1 \\ 1/2 & 0 & 0 \end{pmatrix}$$

This matrix B has eigenvalue 1 with eigenvector $(2,2,1)$ as in $\underline{0}^2\ \underline{1}^2\ \underline{2}^1$.

Pinning Down $\mathcal{S}$

So $\mathcal{S}$ is a commutative group, and there are only two generators: we can express $\underline{2}$ in terms of $\underline{0}$ and $\underline{1}$.

One might suspect that

$$\mathcal{S} \cong \mathbb{Z}^2$$

but to show this we need to make sure there are no other identities lying around.

Knuth to the Rescue

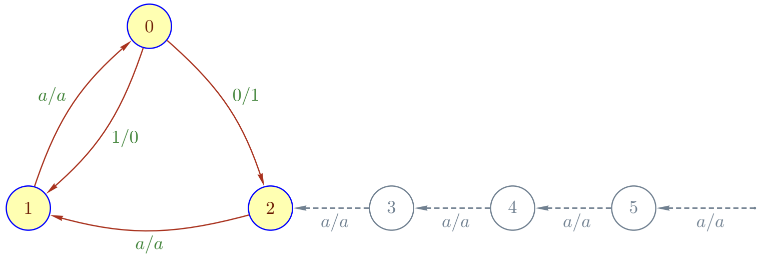
We need to rule out things like $\underline{0}^{42}\underline{1}^{17} = I$.

But how? We can't just try out all possible identities.

Knuth came up with the following brilliant idea:

Let's make the transducer infinite.

Knuth's Infinite Machine



As it turns out, adding copy states $\underline{3}$, $\underline{4}$ ... does not change $\mathcal{S}$: we get no new permutations.

Why Bother?

Because now we have more identities:

$$\underline{k}^2\;\underline{k+1}^2\;\underline{k+2} = I$$

These hold for all $k \geq 0$.

More Identities

$$\underline{k}^2 = \underline{k+2}\;\underline{k+3}$$

This follows directly from the last slide.

Knuth Normal Form

The new identities give us a rewrite system for any function in $\mathcal{S}$.

Theorem: Every element f of $\mathcal{S}$ has a unique flat normal form

$$f = \underline{k_1}\,\underline{k_2}\,\ldots\underline{k_n}$$

where $k_1 < k_2 < \ldots < k_n$.

KNF for $\underline{0}^{20}\underline{1}^{20}$

0	1	2	3	4	5	6	7	8	9
20	20								
-20	-20	-10							
		10	10	5					
			-10	-10	-5				
				6	6	3			
						-2	-2	-1	
							2	2	1
0	0	0	0	1	1	1	0	1	1

$$\text{KNF}(\underline{0}^{20}\underline{1}^{20}) = \underline{4}\,\underline{5}\,\underline{6}\,\underline{8}\,\underline{9}$$

Free Abelian

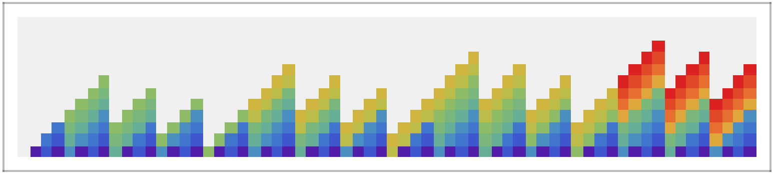
It follows from KNF that we already have all identities in $\mathcal{S}$.

Hence we have

$$\mathcal{S} = \{\, \underline{0}^a \, \underline{1}^b \mid a, b \in \mathbb{Z} \,\} \cong \mathbb{Z}^2$$

And, there is a fractal.

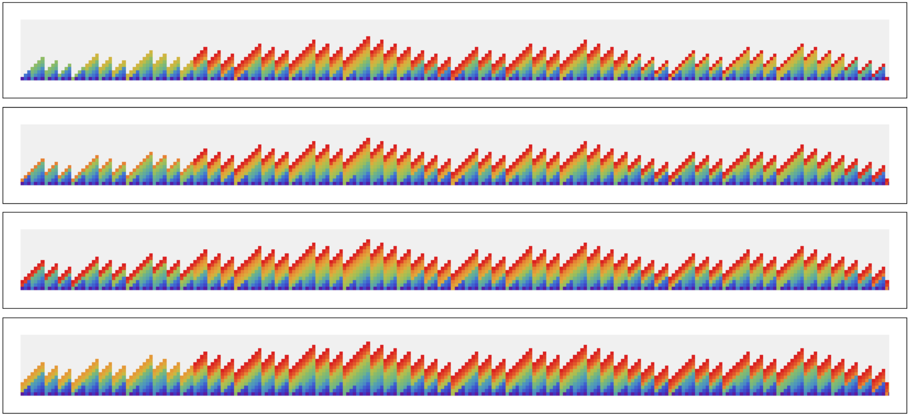
KNF for $\underline{0}^k$

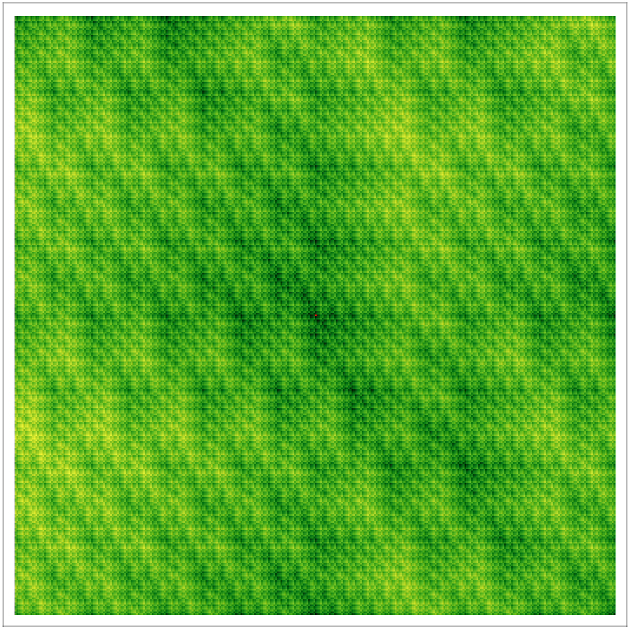


KNF of $\underline{0}^k$, $0 \leq k < 64$

column height: number of terms

colors: represent numerical value

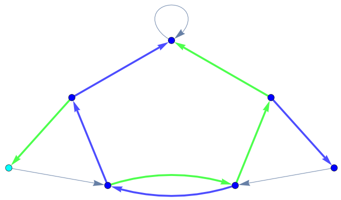




And Equivalence Testing?

The description of $\mathcal{S}$ as $\mathbb{Z}^2$ makes it much easier to understand its structure: with a bit of effort, everything can be expressed in terms of linear algebra.

There is no need to deal with complicated machines like



Even better: we have lots of algorithms at our disposal. In particular one can check whether x and y are on the same orbit in polynomial time.

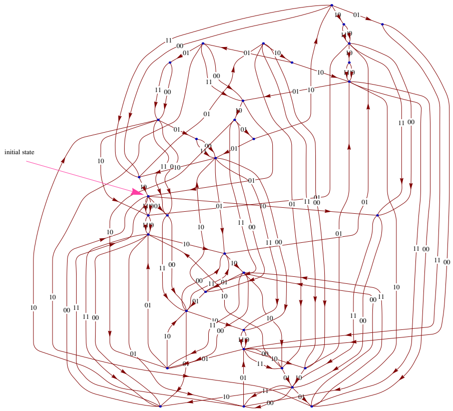
The Whopper

Theorem

There is a finite state machine that can check if x and y lie on the same orbit under $\underline{0}$.

Truth in advertising: I spent a lot of time trying to disprove this.

The Orbit Checker



This machine is minimal and has 34 states. Don't ask why.

And Timestamps?

With a little more effort one can show that timestamps can also be computed by a finite state machine:

There is a transducer that takes $x{:}y$ as input, and returns the least timestamp t as output, if it exists (otherwise the machine returns no output).

Similarly there is a transducer that takes as input x and returns as output the lexicographically least element of the orbit of x.

The Message

Combining classic machinery from math with computation produces lots of interesting results.

It is hard to imagine that these results could be obtained solely with traditional methods.

Computation provides a new lense on reality.

Computational Discrete Math

Classical topics in math and theory are often quite inaccessible, it may take years to get to the point where new contributions are possible.

By throwing computation into the mix, the landscape shifts.

Kevin Lewi	Stanford CS
Tsutomo Okano	UIUC math
Tim Becker	Chicago math