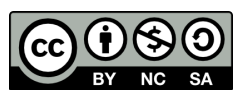
CDM

Model Checking and Automaticity

Klaus Sutner

Carnegie Mellon University

Spring 2024



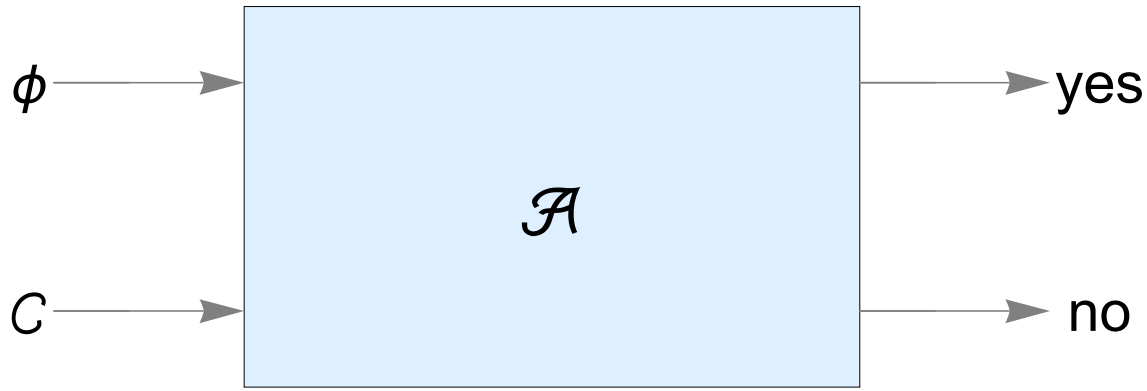
Model checking is CS terminology and fairly recent, in standard mathematical logic one usually speaks about the Suppose $\mathcal{C}$ is some first-order structure of signature Σ. The first-order theory[†] or complete diagram of $\mathcal{C}$ is defined to be

$$\mathrm{Th}(\mathcal{C}) = \{\, \varphi \text{ sentence} \mid \mathcal{C} \models \varphi \,\}$$

Here φ is any sentence (formula without free variables) of signature Σ. So the theory is nothing but the view of $\mathcal{C}$ through the lens of first-order logic, everything we can express about $\mathcal{C}$ in this framework.

$\mathrm{Th}(\mathcal{C})$ can easily be coded as natural numbers or strings, so a FO theory might well be decidable.

[†] Not great terminology, since "theory" is also used in connection with axioms and derivations.



The big problem: the formula is a finite, discrete structure and can be input for an algorithm. But the structure $\mathcal{C}$ lives in set theory la-la-land, so how are we going to turn it into input?

For finite structures $\mathcal{C}$, there is a straightforward solution: we can write down lookup tables for all the functions and relations. To check validity of a sentence over a finite structure, we can essentially just implement Tarski's definition of truth (recursion over the buildup of the formula).

Testing truth of the matrix of a formula, given bindings for all the quantified variables, really comes down to repeated table lookup. To deal with quantifiers we use loops ranging over the finite universe. So the algorithm is certainly primitive recursive.

While finite structures are amenable in principle, efficiency is a major issue. A closer look shows that PSPACE is already enough to handle finite model checking (we can easily set up all the necessary loops in polynomial space). Somewhat reassuring, but cold comfort in the end.

In fact, even when $A = \{0, 1\}$ and we only have the standard Boolean operations, validity checking for FOL formula is PSPACE-complete (this problem is essentially QBF: quantified Boolean formulae).

Another problem is that a finite structure may be very large so that maintaining lookup table is not possible in practice. One may have to resort to *succinct presentations* that complicate matters greatly at the complexity level (just think about an Intel chip).

One major step towards infinite structures is to consider computable structures where the carrier set is $\mathbb{N}$ (or a subset thereof) and all the functions and relations are computable.

The representation of $\mathcal{C}$ would then be a collection of computable functions, each given by a program. This setup rules out uncountable structures and a great many countable ones, too.

A restriction to computable structures may seem entirely reasonable; after all, if we cannot, say, determine whether a particular relation holds of two elements, how could we possibly come up with a model checking algorithm?

The prime example for a computable structure is the natural numbers with the usual operations of arithmetic:

$$\mathfrak{N} = \langle \mathbb{N}; +, \cdot, 0, 1, < \rangle$$

Incidentally, this is the only computable model of the Dedekind-Peano axioms: none of the non-standard models provided by the compactness theorem can be computable.

Theorem

The theory of $\mathfrak{N}$ is undecidable.

Horribly, terribly, hopelessly undecidable, in fact. Even single quantifiers wreak havoc.

So is this all this truth checking business just a pipedream?

No, we just need to simplify matters more. Presburger arithmetic uses the language $\mathcal{L}(+, -, 0, 1; <)$ of signature $(2, 2, 0, 0; 2)$, informally this is arithmetic without multiplication.

Theorem (M. Presburger, 1929)

Presburger arithmetic is decidable.

We'll give a complete proof based on finite state machines. Unsurprisingly, Presburger's algorithm is triple exponential: even for quantifier-free formulae the problem is $\mathbb{NP}$-hard.

Interestingly, replacing integers by rationals in full arithmetic does not help:

Theorem (J. Robinson, 1948)

The theory of the rationals with addition and multiplication is undecidable.

Somewhat counterintuitively, replacing the rationals by the reals (an uncountable structure, no less) makes things easier: now the theory is decidable by a famous result by Tarski.

Theorem (A. Tarski, 1948)

The theory of the reals with addition and multiplication is decidable.

But note: the structure here is fixed once and for all, we don't have to worry about how it could be somehow transmogrified into suitable input.

As a consequence, basic geometry is decidable. This is interesting e.g. for robotics.

The theorem is proved by a very interesting technique that provides a direct decision algorithm: quantifier elimination, meaning that a quantified formula is transformed into an equivalent one without the quantifier:

$$\exists\, x\, \varphi(x) \quad \rightsquigarrow \quad \widehat{\varphi}$$

The transformation is easily computable.

Tarski's original method was highly inefficient, though (not bounded by a stack of exponentials). There are better methods now, but the complexity is provably doubly exponential.

Deciding truth over some structure is a central topic in theoretical CS. One big difference is that the "structures" in question here are a bit more general: classical FO structures, protocols, software, hardware. And usually one needs query languages beyond just FOL.

So it makes sense to have special terminology, one speaks of model checking. Here is the version limited to just first-order.

Problem:	**Model Checking**
Instance:	A FO structure $\mathcal{C}$ and a FO sentence φ.
Question:	Is φ valid over $\mathcal{C}$?

Clearly, it can be very interesting to solve the model checking problem even when the structure $\mathcal{C}$ is fixed. For fixed structures one often speaks of expression model checking.

Problem:	**Expression Model Checking (for $\mathcal{C}$)**
Instance:	A FO sentence φ.
Question:	Is φ valid over $\mathcal{C}$?

EMC for $\mathfrak{N}$ is undecidable, but for Presburger arithmetic there is a decision algorithm.

Here is another variant of model checking that may seem slightly less natural: the formula is fixed and the structure varies.

Problem:	**Data Model Checking**
Instance:	A FO structure $\mathcal{C}$.
Question:	Is φ valid over $\mathcal{C}$?

Usually the class of structures under consideration is fairly narrow: one can think of this as a classification problem. For example, one might want to know whether a cellular automaton is reversible (see below).

In CS one often speaks of data complexity in connection with this variant.

We will work with structures that are very easily computable, using only finite state machines to describe the carrier set, the functions and the relations.

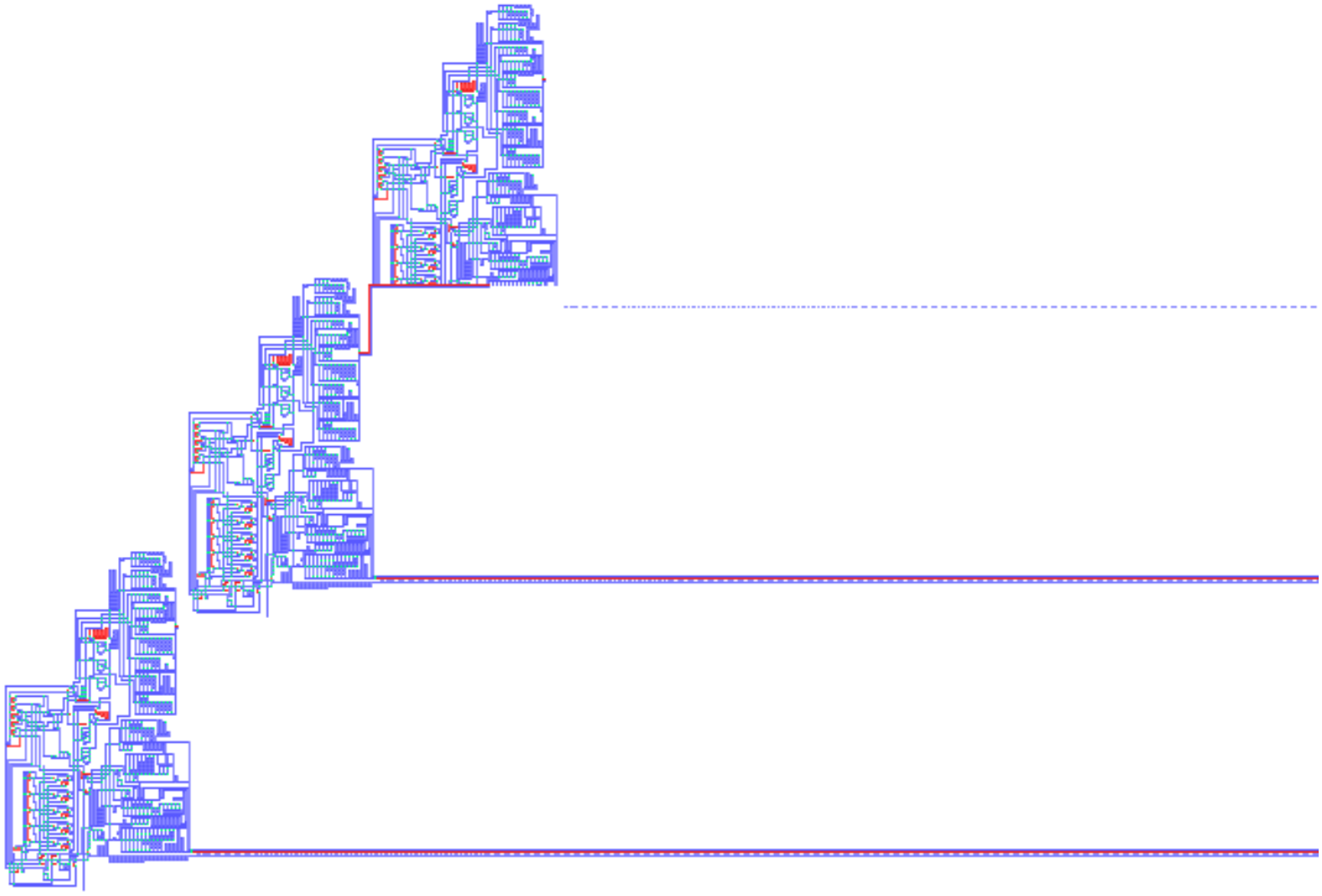
As concrete examples, we use

- Cellular Automata
  This is the fun-and-games part, plus it is really easy to see how basic decision procedures work in this setting.
- Presburger Arithmetic
  This is where real applications live, there are more technicalities here.

Cellular automata were invented by the universal 20th century genius John von Neumann. He was interested in the idea of settling Mars and realized that there is no realistic way to send huge amounts of equipment and large numbers of humans to another planet.

His solution: send a few machines and have them do all the preparatory work. For this to work, the machines have to be capable of self-replication: first, they make multiple copies of themselves, then they perform the actual work (like producing large amounts of oxygen).

Following a suggestion by Ulam, von Neumann came up with a solution based on a two-dimensional cellular automaton with 29 states. By comparison, Conway's infamous Game-of-Life CA has only 2 states.



Unfortunately, two-dimensional cellular automata are already too complicated. We will use one-dimensional CA as a cheap source of examples.

Definition (Elementary Cellular Automaton (ECA))

A ternary Boolean function $f : \mathbf{2}^3 \to \mathbf{2}$ is called an (elementary) local map or local rule. The corresponding global map or global rule $\widehat{f} : \mathbf{2}^{\mathbb{Z}} \longrightarrow \mathbf{2}^{\mathbb{Z}}$ is defined by

$$\widehat{f}(\boldsymbol{x})(i) = f(x_{i-1}, x_i, x_{i+1})$$

The elements of $\mathbf{2}^{\mathbb{Z}}$ are configurations.

We may also write G_f on occasion to avoid confusion.

Hence there are 256 ECA and they can by indexed naturally as $0 \leq e < 256$.

ECA are already amazingly and unexpectedly complicated. Things get even more interesting when we generalize a bit: We can allow more than 2 states, and we can look at neighbors the are further away. Local maps then look like

$$f : \Sigma^{2r+1} \to \Sigma$$

and the global maps are $\Sigma^{\mathbb{Z}} \longrightarrow \Sigma^{\mathbb{Z}}$ (r is the radius of the CA and $k = |\Sigma|$).

Note that there are $k^{k^{2r+1}}$ such general cellular automata. For $k = 3$ and $r = 2$ this produces

$$871896424859609582029110705858607716969640724047317500855252194379$$
$$909670937234399434755499068316831167910552256656 27 \approx 8.72 \times 10^{115}$$

For $k = 8$, $r = 2$ the number increases to 2.84×10^{29592} As far as search is concerned, these numbers might as well be infinite.

One can think of a CA as a discrete-time, discrete-space model of physics where the local rule encapsulates the kind of physics at work in the particular CA. Of course, for most local maps the physics makes little sense, but one can custom design rules that model, say, fluid dynamics. Given massively parallel hardware like the Connection Machine, this approach once held great hope for efficient physics simulations.

It should be intuitively clear that a 1-dim CA can simulate a Turing machine, so questions about the long-term evolution of a configuration are bound to be undecidable.

But questions that are more "local" might be manageable in general, and "simple" local rules might also be workable.

It bothers me that, according to the laws as we understand them today, it takes a computing machine an infinite number of logical operations to figure out what goes on in no matter how tiny a region of space, and no matter how tiny a region of time. How can all that be going on in that tiny space? Why should it take an infinite amount of logic to figure out what a tiny piece of space-time is going to do?

So I have often made the hypothesis that ultimately physics will not require a mathematical statement, that in the end the machinery will be revealed and the laws will turn out to be simple, like the checker board with all its apparent complexities.

R. Feynman, 1965

One can push this idea to the point of trying to develop a purely digital model of physics, a program carried out be Ed Fredkin[†].

Space has been chopped up into discrete cells, each cell requires only one a few bits of information to describe its current state. Time is also discrete, and this system evolves according to a simple local map that is uniform and synchronized throughout the space. There is no spooky action at a distance.

Admittedly, this makes much more sense for 2- and 3-dimensional cellular automata.

[†]Incidentally, Feynman did not agree with Fredkin's ideas in general.

With a view towards the physics angle, one thinks of locations $x \in \mathbb{Z}$ of a configuration as cells, with each cell carrying a particular state (which is chosen from a finite set of possible states.

In many wasy, a biinfinite one-dimensional grid is the most natural setting, but one can also consider one-way infinitude configurations so that the global map takes the form

$$\widehat{f} : \mathbf{2}^{\mathbb{N}} \longrightarrow \mathbf{2}^{\mathbb{N}}$$

Lastly, for actual computational simulations one needs to go one step further and work with finite configurations and global maps of the form

$$\widehat{f} : \mathbf{2}^{n} \longrightarrow \mathbf{2}^{n}$$

Note that there is a problem, though: some of the cells do not have two neighbors, as required by our definition.

There are two standard solutions to this. In particular for finite configurations we have

Cyclic Think of $\boldsymbol{x} \in \mathbf{2}^n$ as being cyclic, so x_0 is adjacent to x_{n-1}.

Fixed Apply the local map to the blocks of $0\,w\,0$.

The fixed boundary approach also works in the one-way infinite case.

Warning: One-way infinite and finite configurations are actually harder to deal with than the clean biinfinite ones. Trust me.

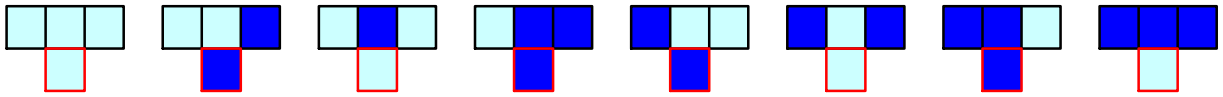
Exercise

Explain what cyclic boundary conditions have to do with biinfinite configurations.

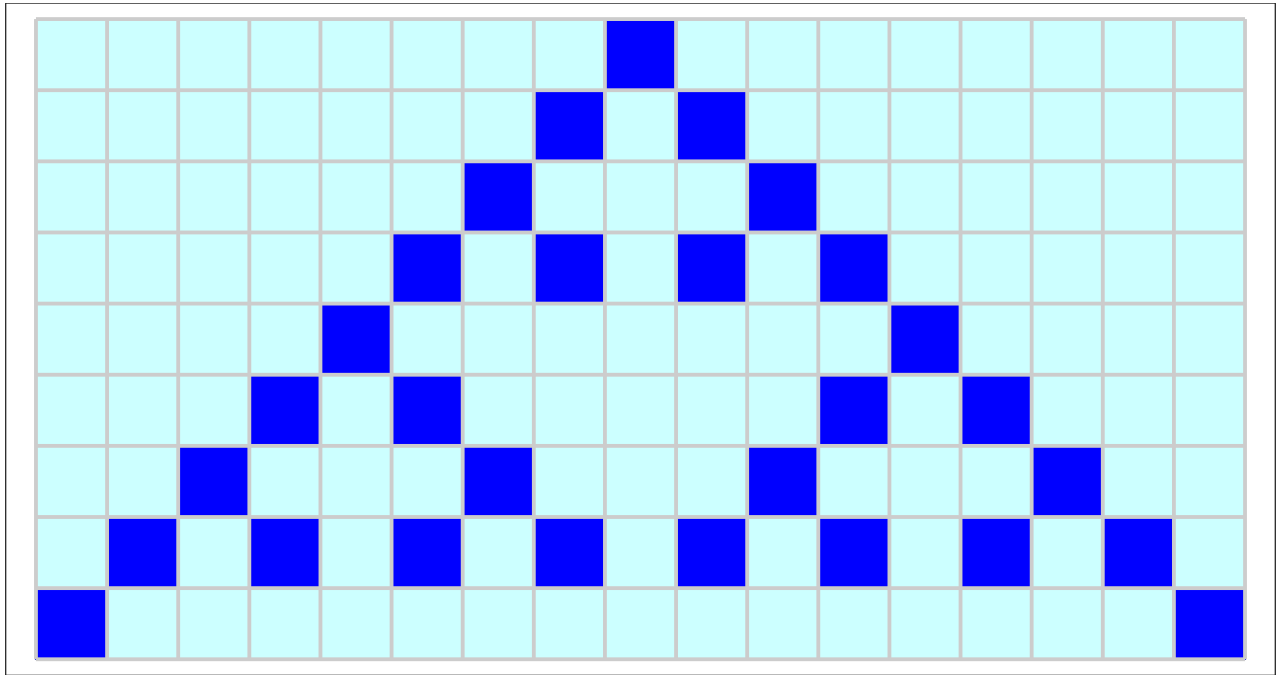
Cellular automata have been fashionable for a while (in particular since Conway's Game-of-Life), but they do work well in our situation:

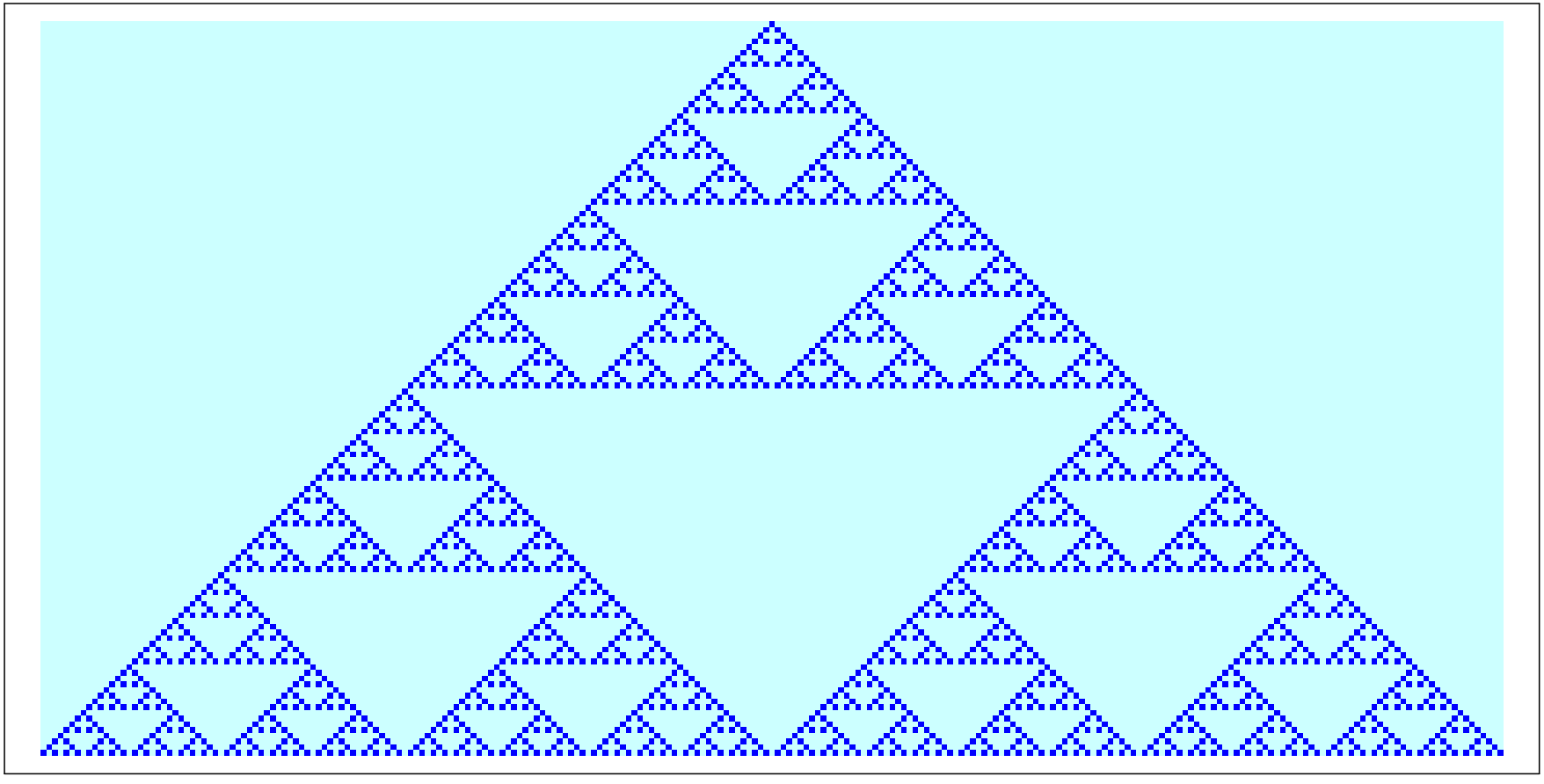
- There is a fairly well developed theory (stemming from the venerable area of *symbolic dynamics*).
- They produce beautiful pictures. More generally, geometry can help greatly in understanding their basic properties.
- There are only 256 ECA, so one can conveniently explore them all without too much effort.
- Lastly, and unexpectedly, even our exceedingly simple ECA display amazingly complex behavior, against all intuition.

Here is local map for ECA 90, $f(x, y, z) = x \oplus z$.



And here are the first few steps when iterating $\widehat{f}$ on the configuration $\ldots 0001000 \ldots$.





Suppose we have an initial configuration $X \in \mathbf{2}^n$. It is natural to ask whether one can easily compute the configuration at time t

$$Y = \widehat{f}^t(X)$$

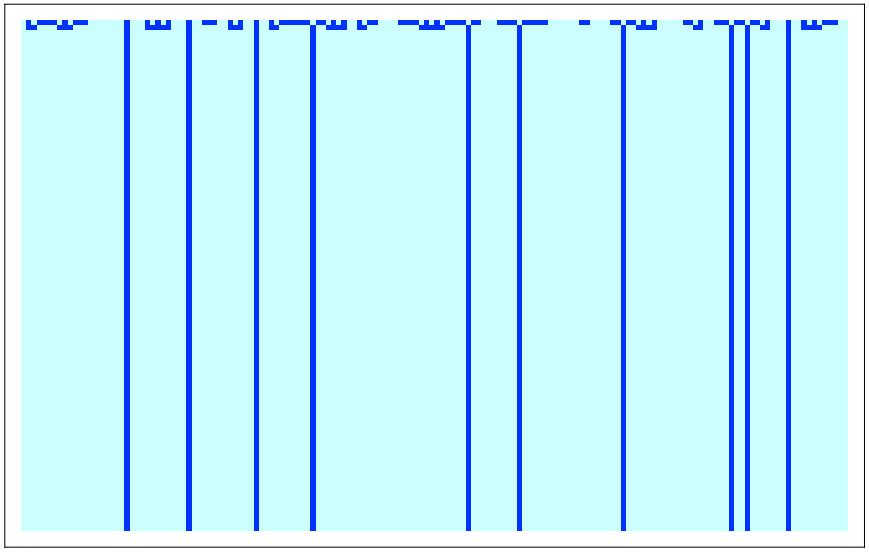
Of course, we can just iterate $\widehat{f}$ on X and be done with it.

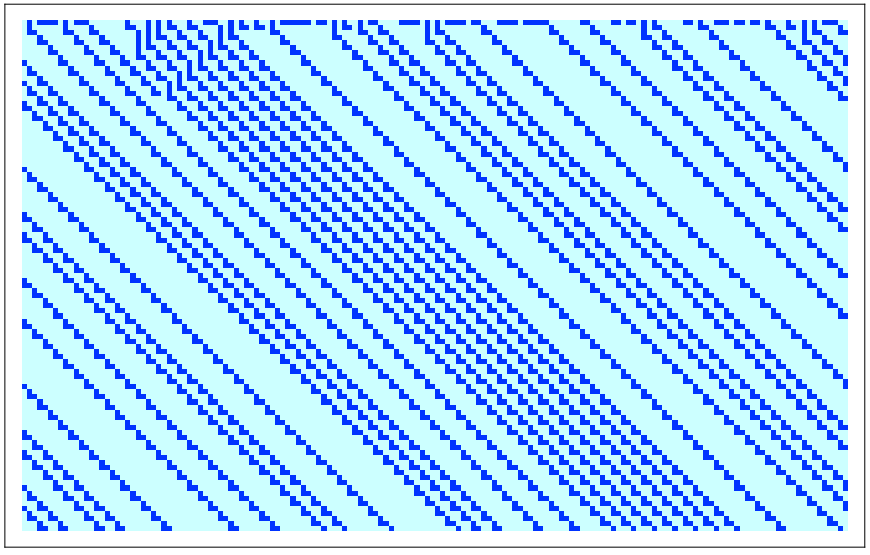
The real question is this: can we compute Y without computing all t configurations in the orbit. For example, can we get away with $O(\log t)$ configurations?

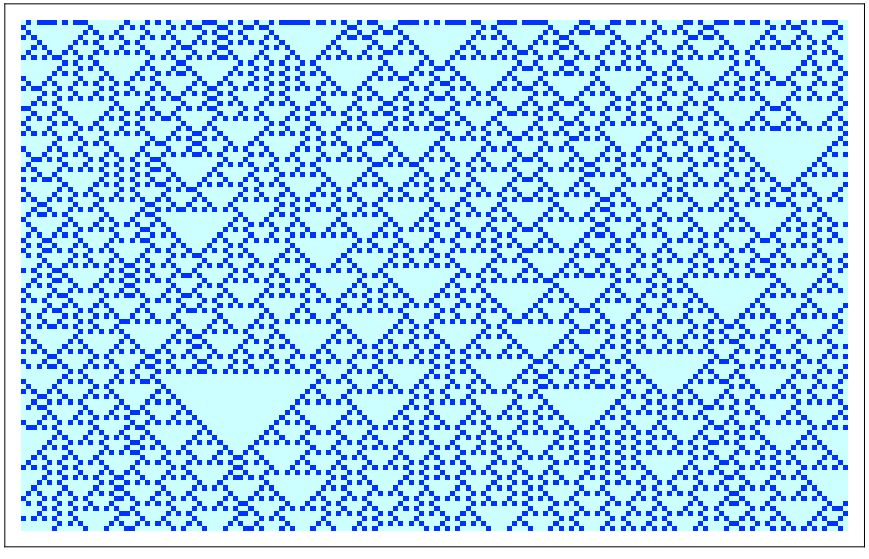
Exercise

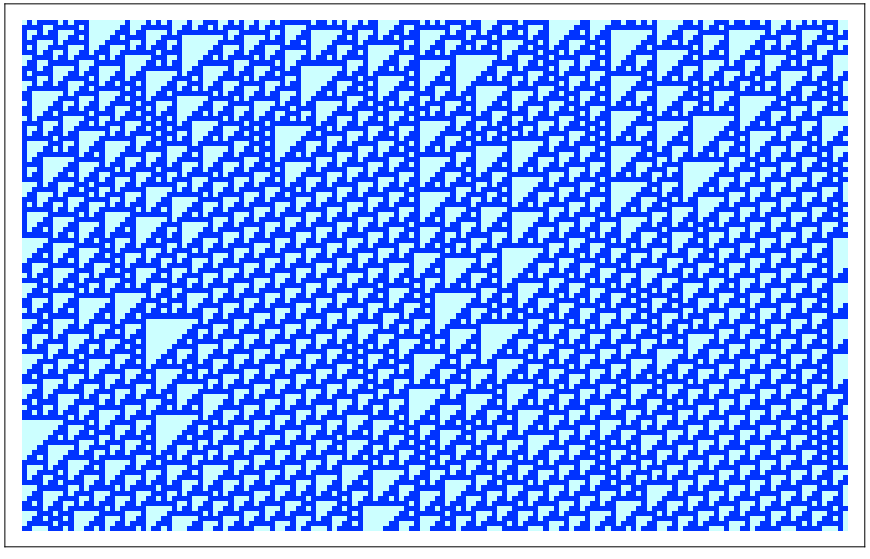
Consider ECA 90, local map $f(x, y, z) = x \oplus z$.
Show that a $\log t$ shortcut exists for this ECA.

Some ECA Behaviors









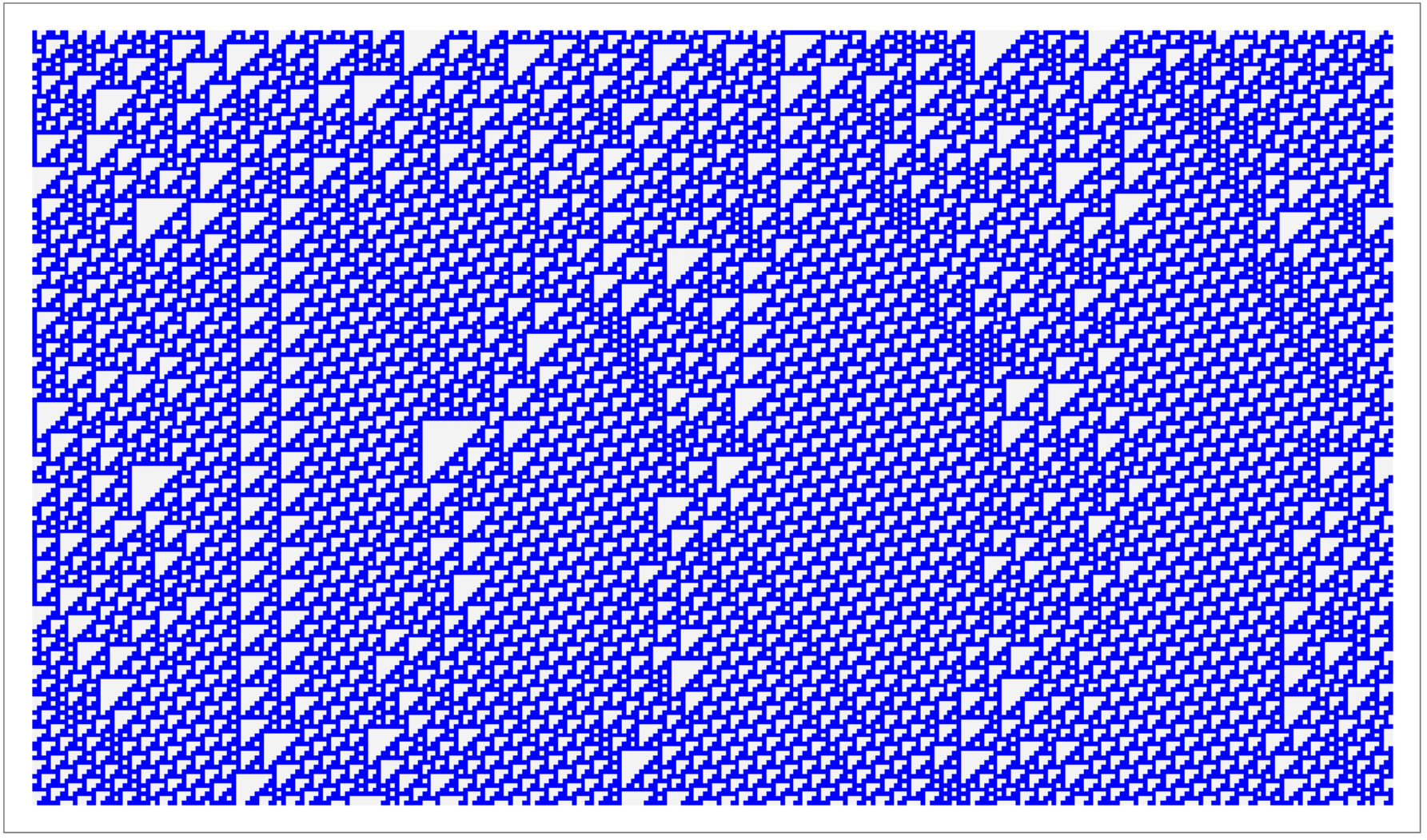
Somewhat surprisingly, elementary cellular automata seem to display roughly 4 types of behavior.

- All configurations become stable after a while.
- All configurations become periodic after a while.
- Orbits are chaotic and seemingly random.
- Orbits produce complex persistent structures.

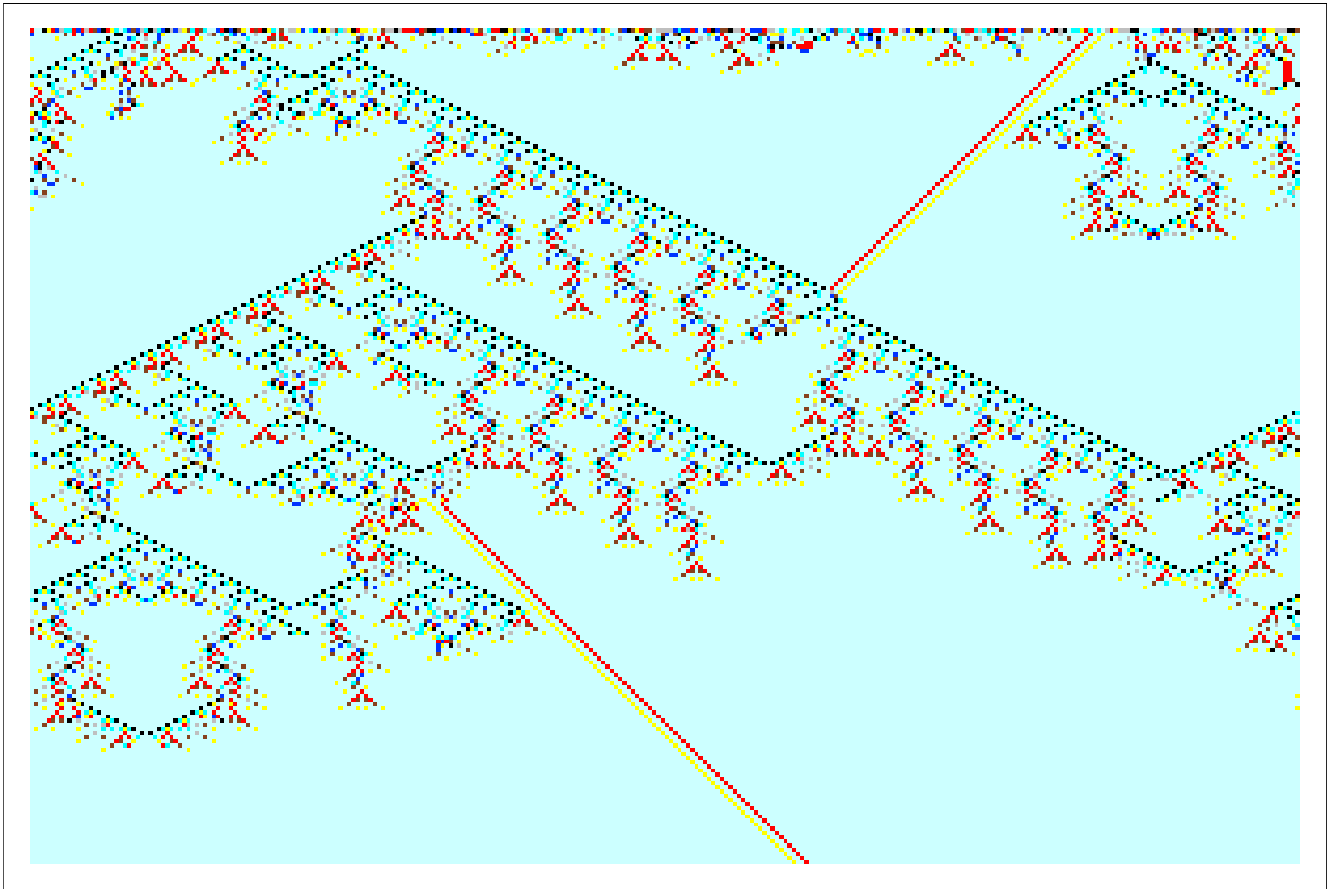
These are called Wolfram classes.

The ECA on the bottom right is ECA 110, and has been proven to be computationally universal (in a certain technical sense).

Note, though, that Wolfram's classification is entirely heuristic, it seems hopeless to formalize it in terms of computability theory: one can construct cellular automata that mix up the four classes as shown by Baldwin and Shelah.



A Wild CA ($k = 8$, $r = 2$)



The definition of a global map may seem a bit ad hoc, but they are actually quite natural.

Theorem (Curtis-Hedlund-Lyndon 1969)

A map $\Sigma^{\mathbb{Z}} \longrightarrow \Sigma^{\mathbb{Z}}$ is a global map iff it is continuous and shift invariant.

Shift invariant means that the map commutes with $\sigma : \Sigma^{\mathbb{Z}} \to \Sigma^{\mathbb{Z}}$ defined by $\sigma(X)(i) = X(i+1)$. In other words, we want to get rid of the coordinate system imposed by $\mathbb{Z}$.

There has been a huge amount of research on cellular automata since the CHL theorem.

We can think of an elementary cellular automaton as a FO structure of the form

$$\mathcal{C} = \langle \mathbf{2}^{\mathbb{Z}}; f \rangle$$

where $f : \mathbf{2}^{\mathbb{Z}} \to \mathbf{2}^{\mathbb{Z}}$ is the global map. So we are dealing with an uncountable space (a zero-dimensional compact Hausdorff space), but f is completely defined by the corresponding local map, a bit-vector of length 8.

We would like to understand the properties of $\mathcal{C}$.

Amazingly, if we limit ourselves to propositions in first-order logic, then we can check them automatically. Model checking works just fine in this case.

The Truth: FOL here is not terribly strong, e.g., we cannot say anything about orbits in general.

One of the basic questions about dynamical systems is reversibility: is the map f injective?

Theorem (KS 1991)

There is a quadratic time algorithm to test reversibility.
Essentially the same algorithm also tests surjectivity and openness.

It is known that for the global maps of cellular automata, reversible implies open implies surjective (but not the other way around). This is obvious from the algorithm that checks for increasingly restrictive properties of a certain graph associated with the CA.

Reversibility of a cellular automaton can be expressed in first-order:

$$\forall\, x, y \left(f(x) = f(y) \Rightarrow x = y \right)$$

For technical reasons, it is better to think of the function f as a binary relation $\rightarrowtriangle$. Then the formula looks like so:

$$\forall\, x, y, z \left(x \rightarrowtriangle z \wedge y \rightarrowtriangle z \Rightarrow x = y \right)$$

So, if we can model check structures $\mathcal{C} = \langle \mathbf{2}^{\mathbb{Z}}; \rightarrowtriangle \rangle$, we can decide reversibility.

The algorithm in the paper was lovingly handcrafted. Sadly, it turns out that it can be extracted with very little effort from the workings of the model checking method we will discuss.

It is tempting to avoid infinite configurations and focus on finite ECA:

$$\mathcal{C}_n = \langle \mathbf{2}^n; \rightarrowtriangle \rangle$$

We can think of $\mathcal{C}_n$ as directed graph and will also refer to these structures as the phase space of the ECA (for size n configurations).

To specify a phase space, we only need 8 bits for the local map, plust $\log n$ bits for the size of the configurations.

As one might suspect, the properties of $\mathcal{C}_n$ can depend quite strongly on n, there typically is not a single uniform answer that works for all n.

Spectra

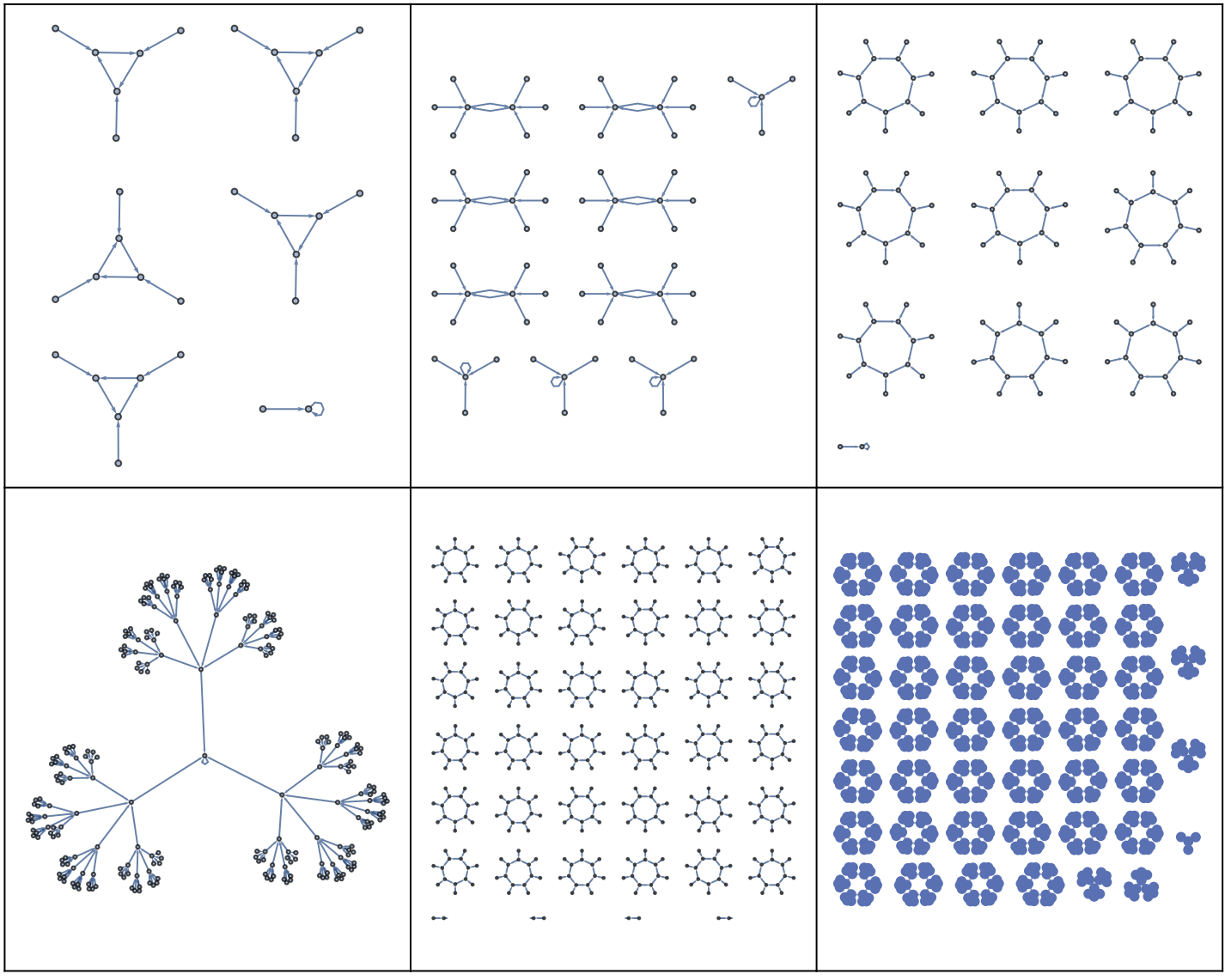
It is natural to ask for which values of the parameter n a certain proposition holds (this is data model checking, in essence):

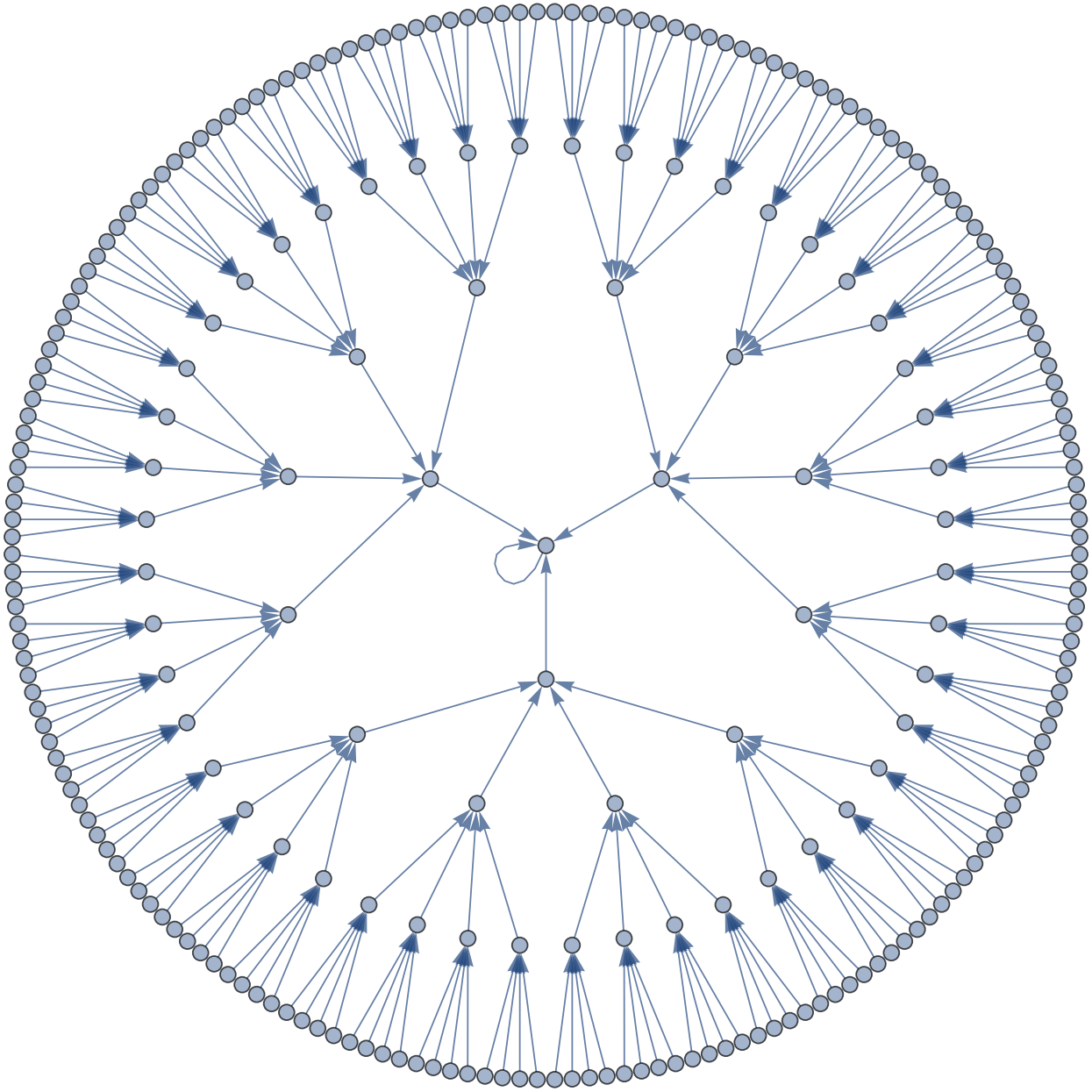
$$\operatorname{spec} \varphi = \{\, n \in \mathbb{N} \mid \mathcal{C}_n \models \varphi \,\}$$

This set of "good" grid sizes is called the spectrum of φ.

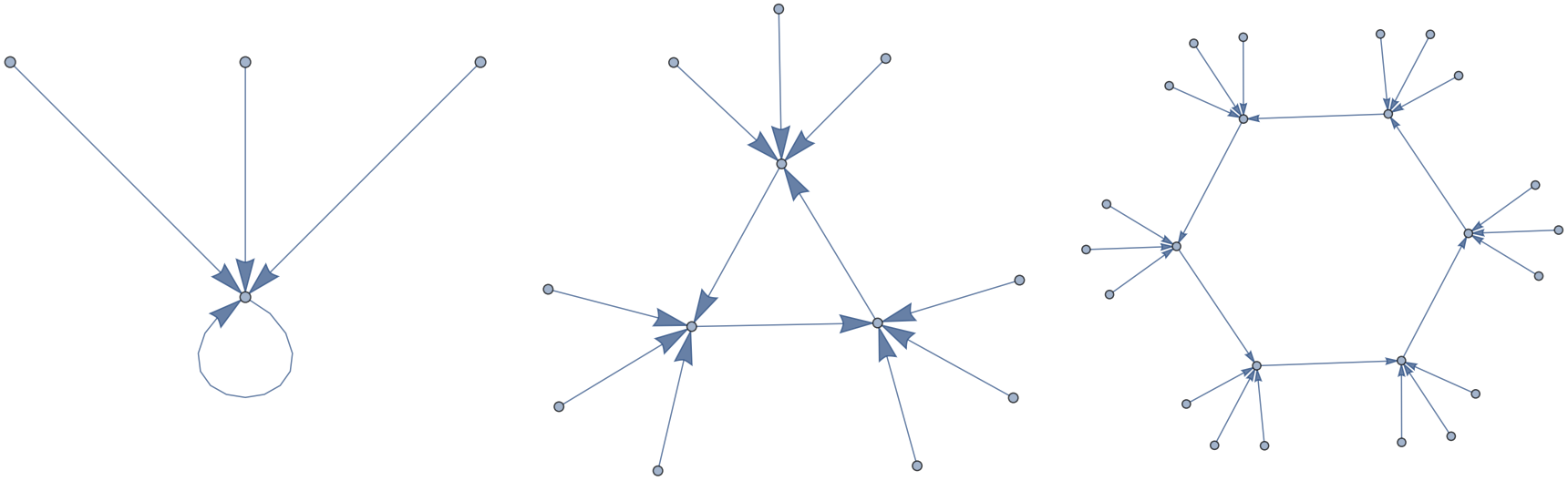
For example, we might want to know for which n a fixed local rule produces a reversible global rule on $\mathbf{2}^n$, say, under cyclic boundary conditions.

We'll see how to compute the spectrum of a first-order formula.

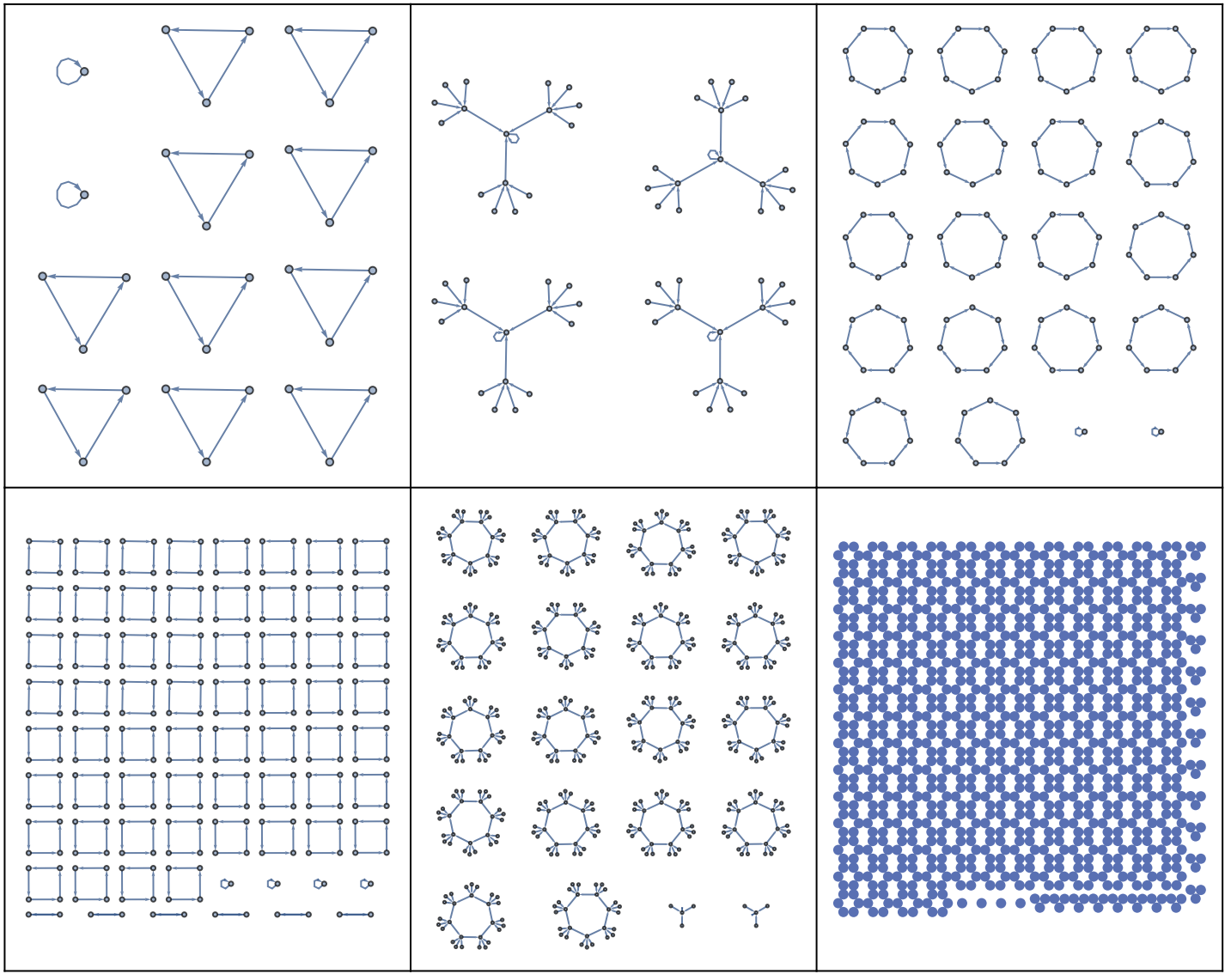


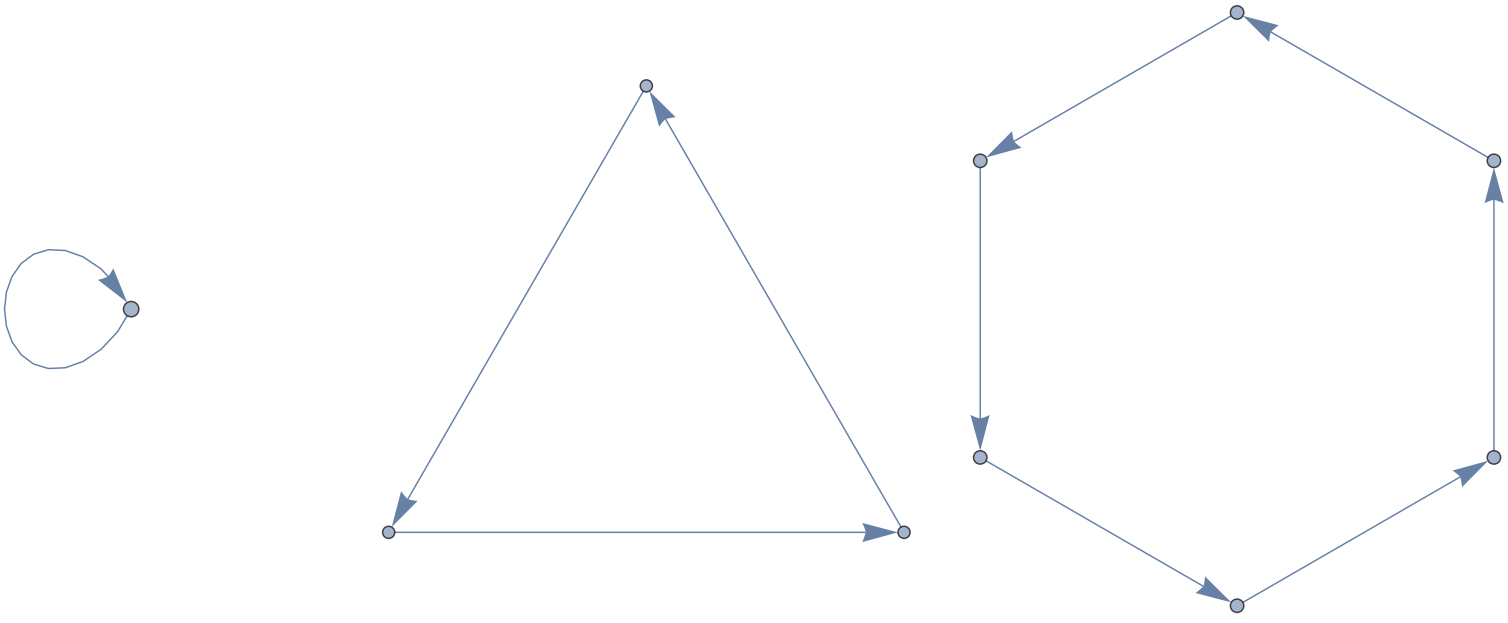


ECA 90, Phase Space $n = 10$ Isomorphs



Counts: 1, 5, 40.





Counts: 4, 20, 160.

Time to get real. For the general model checking problem we need, as part of the input, structures of the form

$$\mathcal{A} = \langle A; f_1, f_2, \ldots, R_1, R_2, \ldots \rangle$$

So we have to specify the carrier set as well as a collection of functions and relations on this set.

In set theory, this is a total non-issue; just write down the definitions of all these sets. Unfortunately, in the computational universe, this is usually quite meaningless.

Instead, we need a finite data structure that represents $\mathcal{A}$.

Recall our crazy idea from a while ago:

> How about structures that can be described by finite state machines?

The carrier set would be a regular set of words. This may sound awful, but it is not too bad: the words in the language are names for the elements. For example, we could use binary strings to describe natural numbers.

But for the functions and relations we need to do a bit of groundwork first. Recall: our finite state machine can deal with languages, but not with functions and relations.

Actually, we will get around functions by simply assuming there are no function symbols in our language. This is not a big restriction, we can always translate a function into its graph, a relation.

A relational structure is a FO structure of the form

$$\mathcal{C} = \langle A; R_1, R_2, \ldots, R_k \rangle$$

In other words, we simply do not allow any functions in our signature. No harm at all, we can always fake functions as relations:

$$x \rightarrowtriangle y \iff f(x) = y$$

Here $\rightarrowtriangle$ is just a binary relation with certain special properties (total and single-valued).

Note that this switch to relations changes our formulae a bit.

For example, consider the simple atomic formula $f(f(x)) = y$. Utterly standard notation, but it actually hides a quantifier:

$$\exists\, z\, (f(x) = z \wedge f(z) = y)$$

Just to be clear, the notation is perfectly good, but any decision algorithm has to cope with this invisible quantifier, one way or another.

In a purely relational structure everything is clearly visible, we have to write something like

$$\exists\, z\, (x \rightarrowtriangle z \wedge z \rightarrowtriangle y)$$

This can make life slightly easier for the decision algorithm.

So the structures we are interested in have the restricted form

$$\mathcal{C} = \langle A; R_1, R_2, \ldots, R_k \rangle$$

where

- $A \subseteq \Sigma^\star$ is a recognizable language, and
- $R_i \subseteq A^{\ell_i}$ is a recognizable relation on words of arity ℓ_i.

We already know how to handle the carrier set, but we do not have anything like a "recognizable relation" at this point.

You might object at this point that the spaces $\Sigma^{\mathbb{Z}}$ and $\Sigma^{\mathbb{N}}$ from our cellular automata example are not recognizable languages according to our (entirely reasonable) definitions.

Entirely true, but we will soon generalize finite state machines to languages of infinite strings. It will turn out that $\Sigma^{\mathbb{Z}}$ and $\Sigma^{\mathbb{N}}$ are trivially recognizable given the right definitions.

For the time being we will stick with finite strings, though.

> The author (along with many other people) has come recently to the conclusion that the functions computed by the various machines are more important–or at least more basic–than the sets accepted by these devices.

Dana Scott, *Some Definitional Suggestions for Automata Theory*, 1967

So the next project is to generalize recognizable languages to some reasonable class of recognizable relations, which are called rational relations.

We have two basic options to tackle this problem:

- Invent some kind of memoryless machine that takes k-tuples of words as input, rather than just single words.
- Exploit Kleene's algebraic characterization of regular languages and somehow lift it to "regular relations."

We'll start with the machine model and then develop the corresponding algebraic approach. As it turns out, they agree entirely (just like general models of computation agree).

A transduction is a relation of the form

$$\rho \subseteq \Sigma^\star \times \Gamma^\star$$

In other words, ρ is a binary relation on words (or, alternatively, a language of 2-track words).

It often very useful to think of such a relation as a map

$$\rho : \Sigma^\star \longrightarrow \mathfrak{P}(\Gamma^\star)$$

where $\rho(x) = \{\, y \mid \rho(x, y) \,\}$. We are given x as input, and want to compute y as output (but note that transductions are nondeterministic in general, they are not single-valued).

Our definition nicely generalizes to k-ary relations for $k > 2$. Instead of single words over an alphabet we have k-tuples of words, possibly over different alphabets:

$$u \in \Sigma_1^\star \times \Sigma_2^\star \times \ldots \times \Sigma_k^\star$$

To emphasize that we still have word-specific operations such as concatenation on these objects we will refer to them as k-track words or multi-words.

To display the component words we usually write

$$u = u_1{:}u_2{:}\ldots{:}u_k$$

The most important case is when $k = 2$, $u = x{:}y$. This is just a pair of words, but, as we will see, there is algebra hiding in the background, so it's better to have distinctive notation.

What would a finite state machine $\mathcal{A}$ describing a transduction look like? We will stick with the binary case for simplicity.

We have a collection $R \subseteq \Sigma^\star \times \Gamma^\star$ of 2-track words. and we need to build an acceptor for such a language. This is quite straightforward: we simply change the transition labels from ordinary words to 2-track words:

$$\tau \subseteq Q \times \Sigma^\star \times \Gamma^\star \times Q$$

Since we already have some experience with FSMs we can now try to copy over all the old definitions from the language case.

As for ordinary transition systems, define the trace or label of a run π

$$\pi = p_0 \overset{u_1:v_1}{\longrightarrow} p_1 \overset{u_2:v_2}{\longrightarrow} p_2 \ \ldots \ p_{n-1} \overset{u_n:v_n}{\longrightarrow} p_n$$

in the diagram as the product of the respective labels in the monoid:

$$\mathsf{lab}(\pi) = u{:}v = (u_1 u_2 \ldots u_n \,,\, v_1 v_2 \ldots v_n) \in \Sigma^\star \times \Gamma^\star.$$

As usual, we are interested in runs from I to F.

Definition

A transducer is an automaton $\mathcal{T} = \langle \mathcal{S}; I, F \rangle$ where $\mathcal{S}$ is a finite transition system over $\Sigma^\star \times \Gamma^\star$; the acceptance condition is given by $I, F \subseteq Q$.

The transduction associated with $\mathcal{T}$ is the relation

$$\mathcal{L}(\mathcal{T}) = \{ \mathsf{lab}(\pi) \mid \pi \text{ run from } I \text{ to } F \} \subseteq \Sigma^\star \times \Gamma^\star$$

A transduction is rational if it is accepted by some transducer.

One also speaks about the behavior of $\mathcal{T}$, written $[\![\mathcal{T}]\!]$, and we can say that $\mathcal{T}$ recognizes $\mathcal{L}(\mathcal{T})$.

Note that by transition-splitting it suffices to consider labels of the form

$$a{:}b \qquad \text{and} \qquad a{:}\varepsilon \qquad \text{and} \qquad \varepsilon{:}b.$$

A transducer is alphabetic if all labels are of the form

$$a{:}b \qquad \text{where } a, b \in \Sigma$$

These transductions are length-preserving and are much easier to handle than the general ones.

For example, iterating a length-preserving transducer only produces finite orbits.

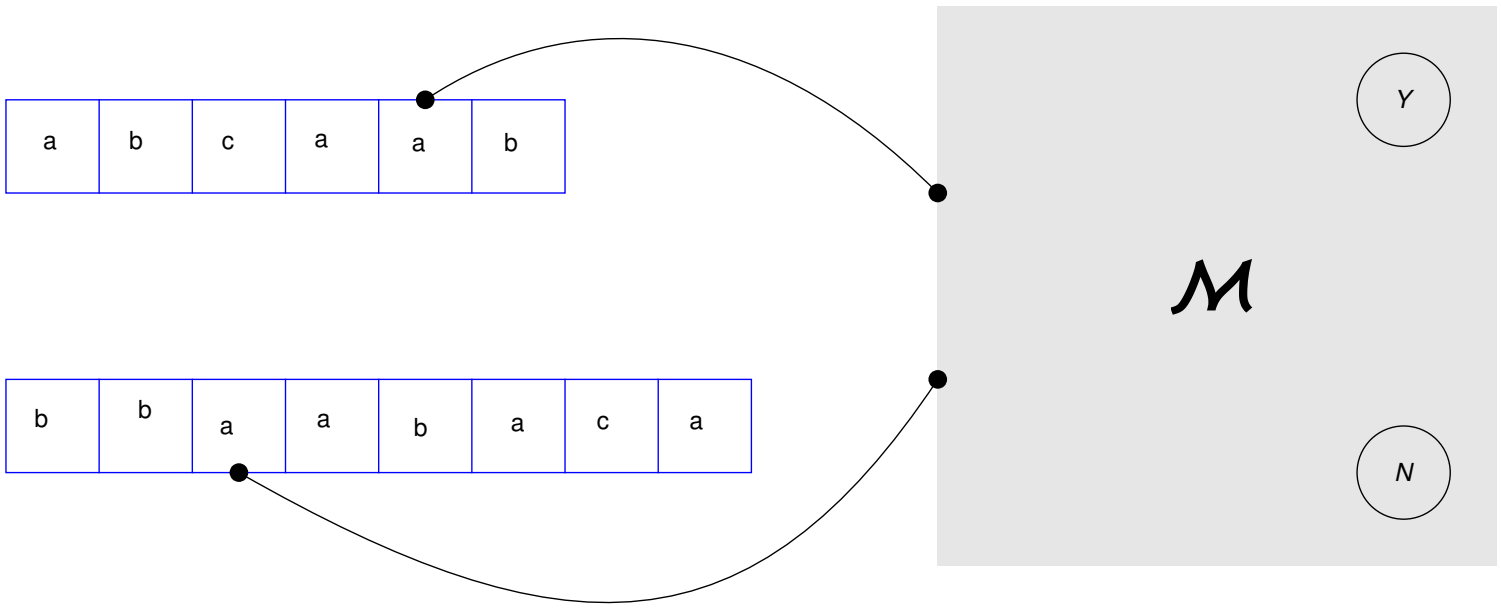
A fairly good intuitive way to think about an acceptor of 2-track words is to modify our old finite state machines:

- Keep the finite state control.
- Allow 2 separate input tapes with separate read-only heads.

The read heads are still one-way, but they can move independently from each other; in particular, one head can get arbitrarily far ahead of another by using $a{:}\varepsilon$ or $\varepsilon{:}b$ transitions.

When both heads have consumed all of their input, acceptance depends on whether the machine is in a final states.

Two-Tape FSM



After careful and detailed study of the last definition, and all the earlier CDM notes, Wurzelbrunft has come to the following profound conclusion:

> All the results from the language scenario carry over to the transduction scenario, with essentially the exact same proofs.

He has asked one of his grad students to write a little macro in Microsoft Word that performs the necessary adjustments.

The identity relation on $\Sigma^\star$ $x = y$ is rational: do a letter by letter comparison.

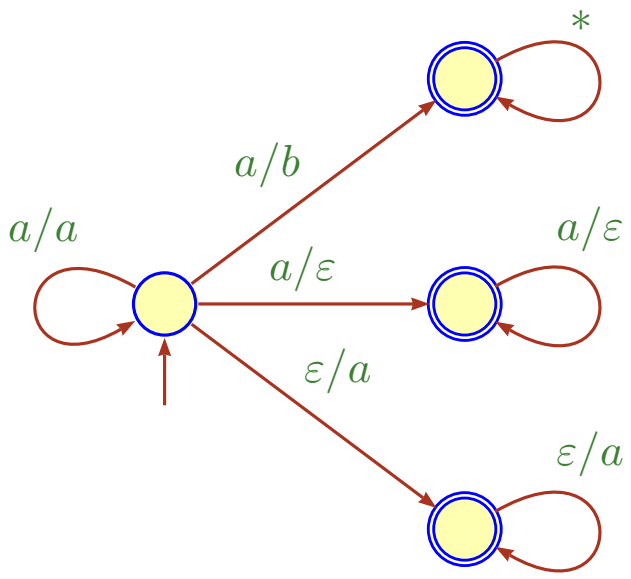
The unequal relation $x \neq y$ is rational: see below.

The prefix relation x is a prefix of y is rational: do a letter by letter comparison until x ends, then skip over the rest of y.

Similarly we can check that x is a suffix or a factor of y: nondeterministically skip to the right place in y, then do a letter by letter comparison.

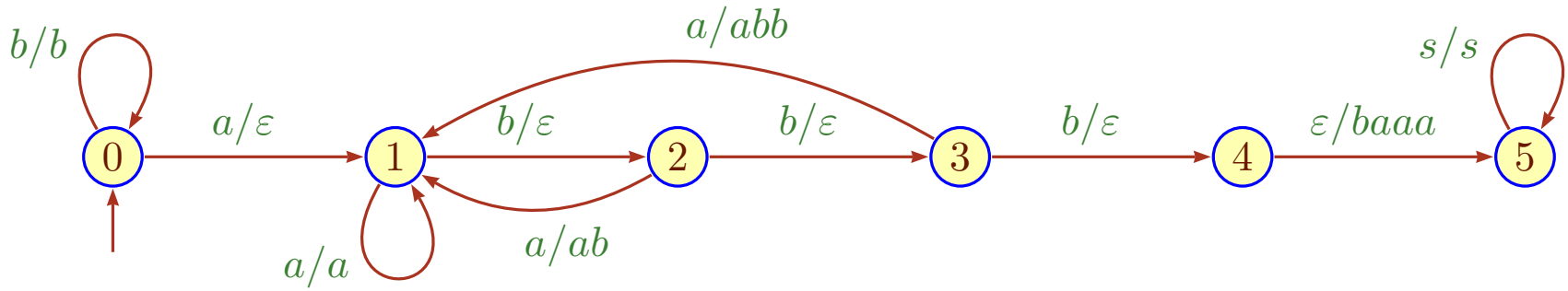
BTW, it is often helpful to attach a special endmarker (typically $\#$) to the end of all words. The machines are a little cleaner this way.

Here is a transducer whose behavior is the relation $x \neq y$.



In the diagram, a and b are supposed to range over Σ, and $a \neq b$.
$*$ means eternal bliss.

A transducer that (essentially) replaces the first occurrence of $abbb$ by $baaa$.



Exercise

Why does this transducer not quite work? Fix the problem. Then change the machine so that all occurrences are replaced.

As already mentioned, we can also think of a transition $p \xrightarrow{u:v} q$ as indicating that input u is transformed into output v[†].

Some authors also write $p \xrightarrow{u/v} q$, in analogy to the usual way of expressing substitutions.

Warning: We explicitly allow for a single input to be associated with many outputs (or perhaps with none at all).

As we will see shortly, for transducers (as opposed to ordinary acceptors) this sort of nondeterminism is absolutely critical, there is no general method to get rid of it.

[†] This is only a question of intuition and psychology, not some new insight.

Here is a more algebraic approach towards a definition of "regular relation." Recall Kleene's theorem on regular languages.

Theorem (Kleene 1956)

Every regular language over Σ can be constructed from $\emptyset$ and singletons $\{a\}$, $a \in \Sigma$, using only the operations union, concatenation and Kleene star.

It follows that there is a convenient notation system (regular expressions) for regular languages that is radically different from finite state machines: we can use an algebra (albeit a slightly weird one) to concoct regular languages.

One direction is easy, given the closure properties of regular languages we already have: every regular expression denotes a regular language.

The opposite direction is handled by dynamic programming. Unfortunately, the regular expressions involved grow exponentially, so the algorithm is not practical.

Still, one very nice feature of Kleene's characterization is that a good definition often generalizes. In this case, the monoid $\Sigma^\star$ is perhaps the most natural setting, but there are other plausible choices.

In particular we could use the product monoid $\Sigma^\star \times \Sigma^\star$ instead: since we are dealing with sets of pairs of strings we naturally obtain binary relations this way.

The relevant algebraic structures are called Kleene algebras. We will not study them in any detail and just pull out the pieces that we need for our project.

Suppose $\langle M, \cdot, 1 \rangle$ is a monoid. Here is a general way to construct a Kleene algebra on top of M. The carrier set is $\mathfrak{P}(M)$ and the operations are

- set theoretic union,
- pointwise multiplication, and
- Kleene star.

More precisely, define

$$K \cdot L = \{ x \cdot y \mid x \in K, y \in L \}$$

$$K^0 = \{1\}, \qquad K^{n+1} = K \cdot K^n$$

$$K^\star = \bigcup_{n \geq 0} K^n$$

$K^\star$ always exists by set theory. Define $K^+ = KK^\star$.

Definition

A k-ary Kleene rational relation is a relation $R \subseteq M$ where

$$M = \Sigma_1^\star \times \Sigma_2^\star \times \ldots \times \Sigma_k^\star$$

and R is generated in the Kleene algebra over M from elements

$$\varepsilon{:}\ldots{:}\varepsilon{:}a{:}\varepsilon{:}\ldots{:}\varepsilon$$

Strictly speaking, the last multi-word should be a singleton set, but in this context it is best not to distinguish between z and $\{z\}$. Trust me.

In the special case $k = 1$ we get back ordinary regular languages.

We will mostly deal with the case $k = 2$ and consider the monoid

$$M = \Sigma^\star \times \Gamma^\star$$
$$x{:}y \cdot u{:}v = xu{:}yv$$

The neutral element is $\varepsilon{:}\varepsilon$. In algebraic terms, this is just a product monoid.

As in the language case, the key importance of Kleene's theorem is that it provides a convenient notation system, we do not need to bother with machines to communicate a recognizable language or a rational relation.

Using the generators and customary operation symbols $+$, $\cdot$ (often implicit) and * we obtain rational expressions that provide a notation system for rational relations.

For applications, this little fact is quintessential.

Writing rational expressions here can be a bit confusing, In particular in the case $k = 2$ it may be better to use vector notation as in $\binom{x}{y}$ rather than $x{:}y$. This corresponds directly to the idea of having two tracks with one word in each track.

Using the 2-track notation we could write the monoid multiplication like so:

$$\binom{x}{y} \cdot \binom{u}{v} = \binom{xu}{yv}$$

More Examples

Let $\Sigma = \{a, b\}$ and $M = \Sigma^\star \times \Sigma^\star$.

The universal relation on $\Sigma^\star$ is given by

$$\left(\binom{\varepsilon}{a} + \binom{\varepsilon}{b} + \binom{a}{\varepsilon} + \binom{b}{\varepsilon}\right)^* = \{\, \binom{x}{y} \mid x, y \in \Sigma^\star \,\}$$

The identity relation on $\Sigma^\star$ is given by

$$\left(\binom{a}{a} + \binom{b}{b}\right)^* = \{\, \binom{x}{x} \mid x \in \Sigma^\star \,\}$$

The un-equal relation is given by

$$\left(\binom{a}{a}^* \left(\binom{a}{b} M + \binom{a}{\varepsilon} \binom{\Sigma}{\varepsilon}^* + \binom{\varepsilon}{b} \binom{\varepsilon}{\Sigma}^* \right)\right.$$

Theorem

A relation is Kleene rational if, and only if, it is the behavior of a (finite) transducer.

The proof is an exact re-run of the argument for regular languages.

A careful inspection of the argument shows that all one needs is labels chosen from a monoid; the fact that the monoid in the language case is the free monoid $\Sigma^\star$ plays no role.

Exercise

Write out a detailed proof of the theorem.