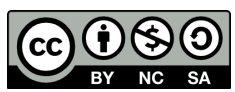
CDM

Model Checking and Rational Relations

Klaus Sutner

Carnegie Mellon University

Spring 2023

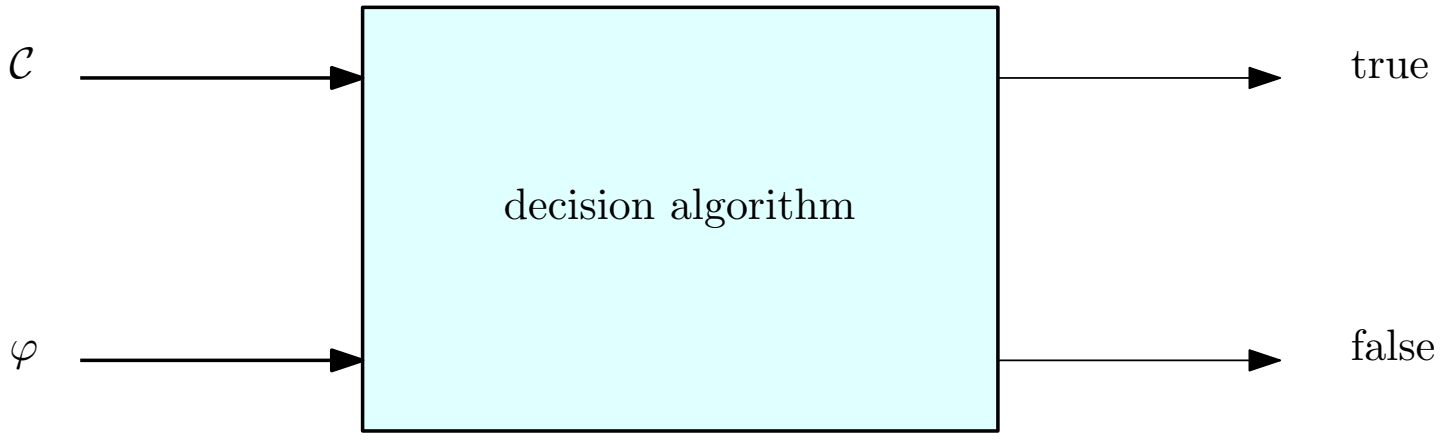


Back to model checking:

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

Note that if we are trying to find algorithms for this problem we will need to be able to specify the structure $\mathcal{C}$ as part of the input. Obviously this is going to be a bit tricky.

But: for finite structures this should be just fine. A computer chip, e.g., is a finite structure.



If $\mathcal{C}$ is finite this is doable (and clearly primitive recursive), but what should this even mean if $\mathcal{C}$ is infinite?

The Trick:
Use a finite representation of $\mathcal{C}$, not the actual structure.

Model checking is CS terminology and fairly recent, in standard mathematical logic one usually speaks about the complete diagram or the first-order theory[†] of a structure.

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

This is nothing but the view of $\mathcal{C}$ through the lense of first-order logic, everything we can express about $\mathcal{C}$ in this framework. Model checking then comes down to determining whether $\varphi \in \mathrm{Th}(\mathcal{C})$.

Since formulae are just syntactic objects (that can easily be coded as natural numbers or strings), $\mathrm{Th}(\mathcal{C})$ might well be, say, decidable. So the underlying language has to be countable; an entirely reasonable assumption.

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

The idea to use finite representations to deal with infinite objects is one of the core ideas in all of math. In fact, in a way, math can be thought of as wordprocessing (manipulation of finite symbol sequences)[†].

In CS, this issue is front and center: not only do we need finite representations, we need them to be well-behaved from an algorithmic point of view. This adds multiple levels of complexity and makes life more interesting (read: difficult).

[†] That's them fighting words, of course. Math is much more than this, but the "wordprocessing" part is absolutely key.

For finite structures, there is a straightforward solution: we can write down lookup tables for all the functions and relations. In practice, a more succinct representation may be preferable, though, even in the finite case: for large structures we get tables that are too big to manage in practice. .

More generally, we can 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 is a huge simplification compared to the standard model where everything is defined in set theory (and computability is a non-issue).

To check validity of a sentence over a finite structure, essentially one can just implement Tarski's definition of truth.

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.

In fact, a closer look shows that PSPACE is already enough. Alas, that is where things end: 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).

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.

Theorem

The theory of the natural numbers with addition and multiplication is undecidable.

Horribly, terribly, hopelessly undecidable, in fact.

So is this all this model 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 see a proof next week. 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.

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.

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.

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

To avoid major technical problems, we will use cellular automata (rather than real program/hardware verification) as a sandbox to play in. Here are the basic definitions.

Definition ((Finite) Elementary Cellular Automaton (ECA))

A ternary Boolean function $f : \mathbf{2}^3 \to \mathbf{2}$ is called an (elementary) local map. The corresponding global map $\widehat{f} : \mathbf{2}^{\mathbb{Z}} \longrightarrow \mathbf{2}^{\mathbb{Z}}$ 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 $0 \leq e < 256$.

Comments

One of thinks of configurations as being given by assigning a particular state to each cell in a biinfinite one-dimensional grid.

Alternatively one can also consider one-way infinitude configurations so that the global map takes the form

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

Pushing a bit harder, we wind up with finite configurations and global maps

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

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 $\boldsymbol{x}$ 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.

These are actually harder to deal with than the biinfinite ones. Trust me.

Exercise

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

Exercise

Come up with another way to fix the missing neighbors problem.

Cellular automata are quite fashionable these days, and they 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.
- Lastly, and unexpectedly, even our exceedingly simple ECA display amazingly complex behavior, against all intuition.

Our methods generalize directly to local maps of the form $\Sigma^w \to \Sigma$ where Σ is an arbitrary alphabet and $w \geq 2$ is any width.

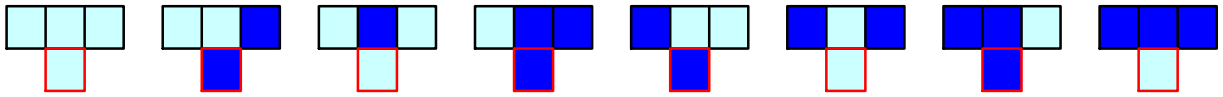
The bit sequences are usually called configurations of type $\mathbf{2}^n$, $\mathbf{2}^{\mathbb{N}}$ or $\mathbf{2}^{\mathbb{Z}}$. One of thinks of a one-dimensional grid of cells, each in a particular state.

To determine the next configuration, a configuration is chopped up into overlapping blocks of length 3, and they are simultaneously replaced by their image under the local map f.

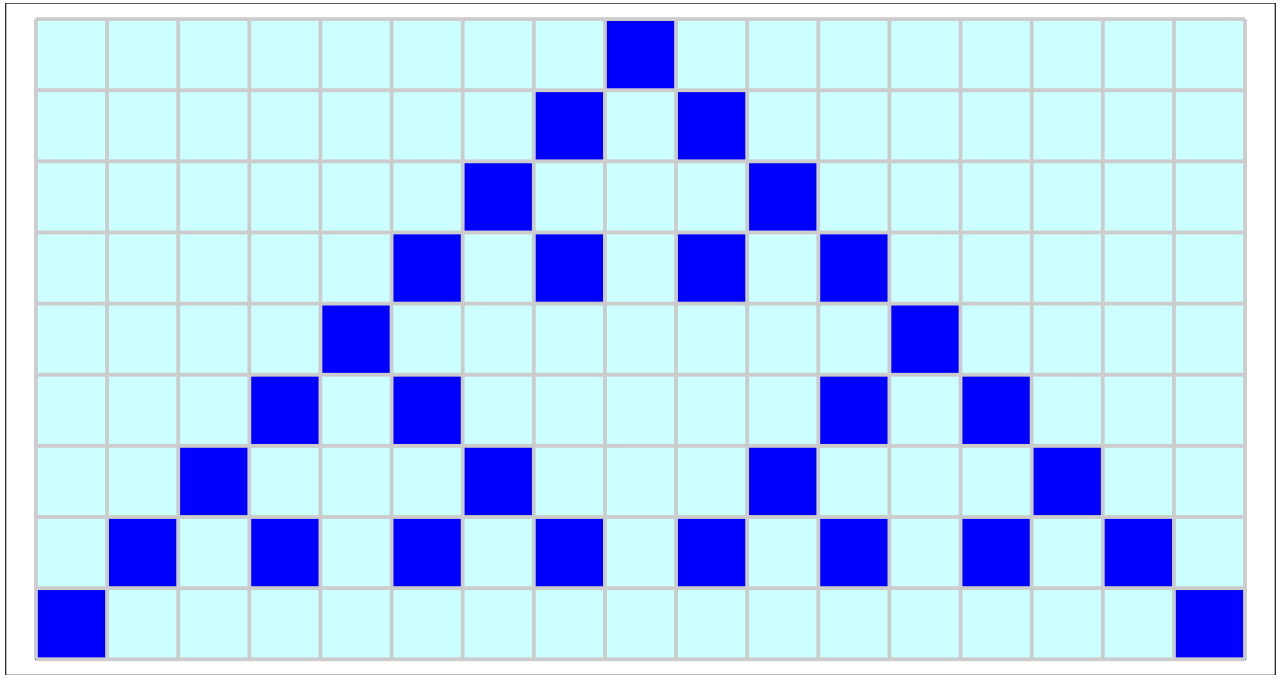
One can think of cellular automata as very simple models of physics[†] : space is one-dimensional, has been chopped up into discrete cells, each cell requires only one bit to describe its current state. Time is also discrete, and this system evolves according to a simple local map (no action at a distance) that is uniform and synchronized throughout the space. Accordingly, one often speaks of a local rule and global rule.

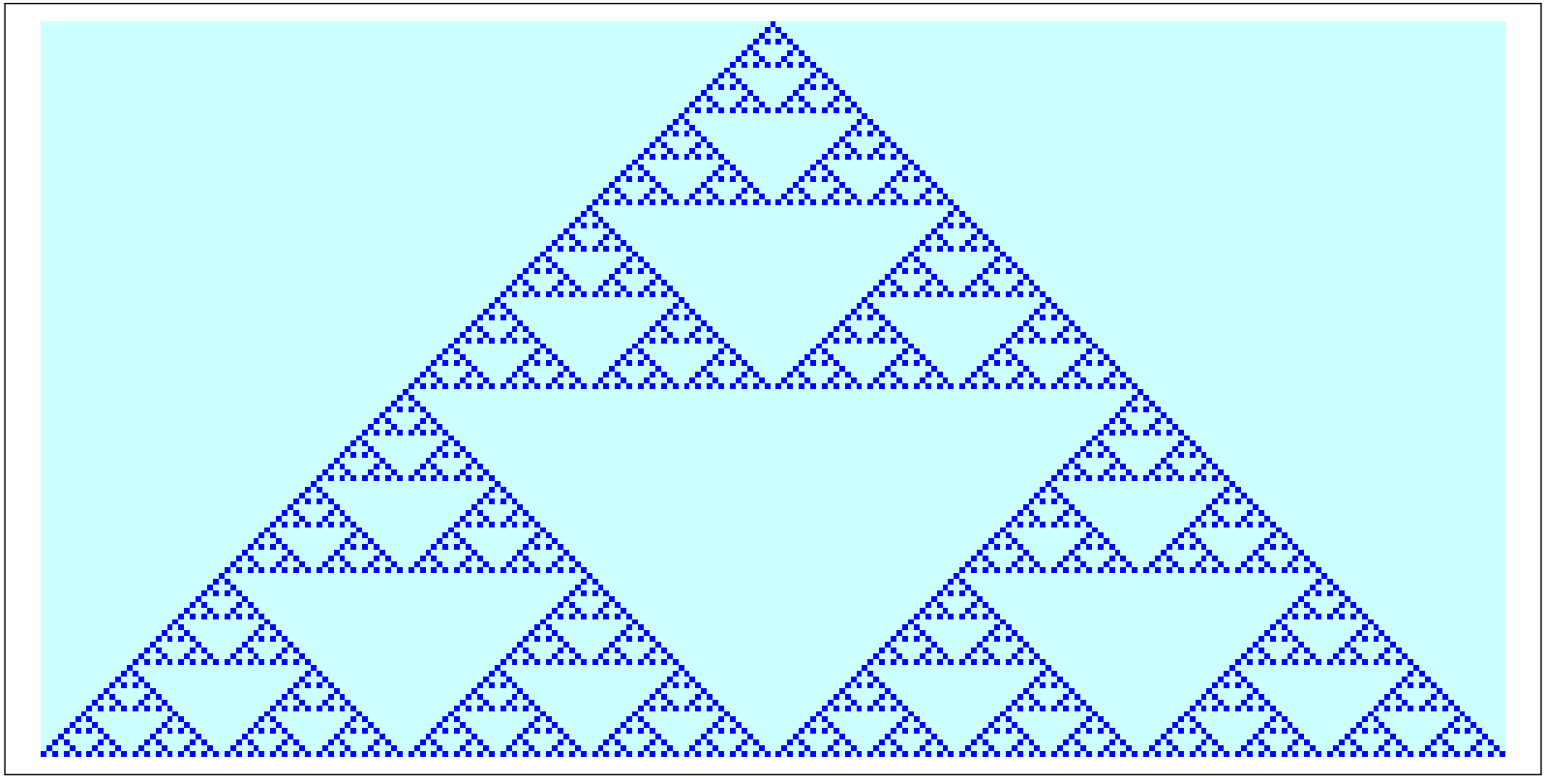
[†] Admittedly, this makes much more sense for 2- and 3-dimensional cellular automata.

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 $\ldots 0001000 \ldots$.





Exercise

Prove that this fractal has dimension $\log_2 3$.

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

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

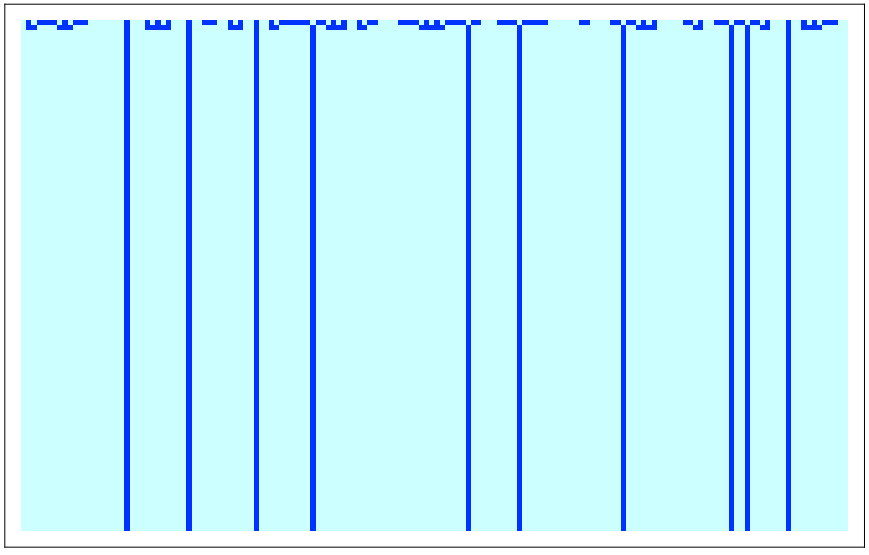
Of course, we can just iterate $\widehat{f}$ on X and be done with it.

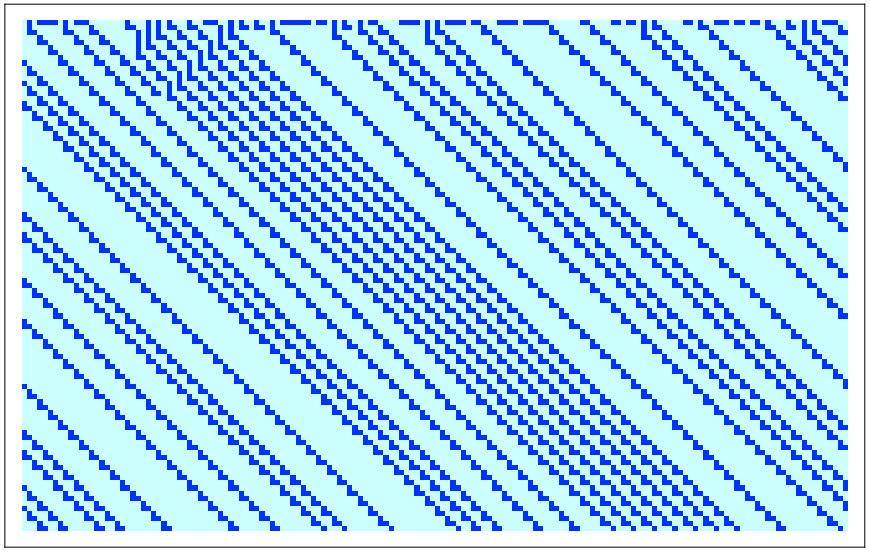
The real question is this: can we compute Y without computing all t configurations in the orbit. For example, can we get away with $O(\log t)$ configurations?

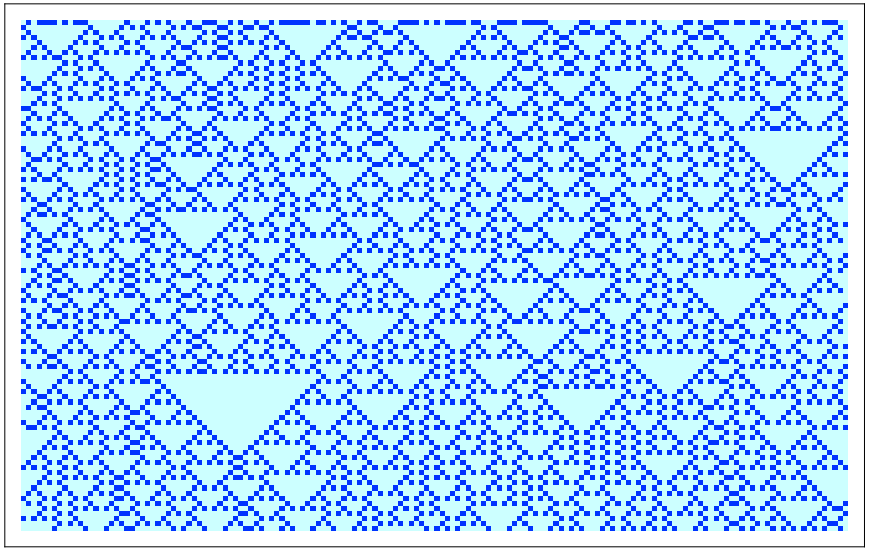
Exercise

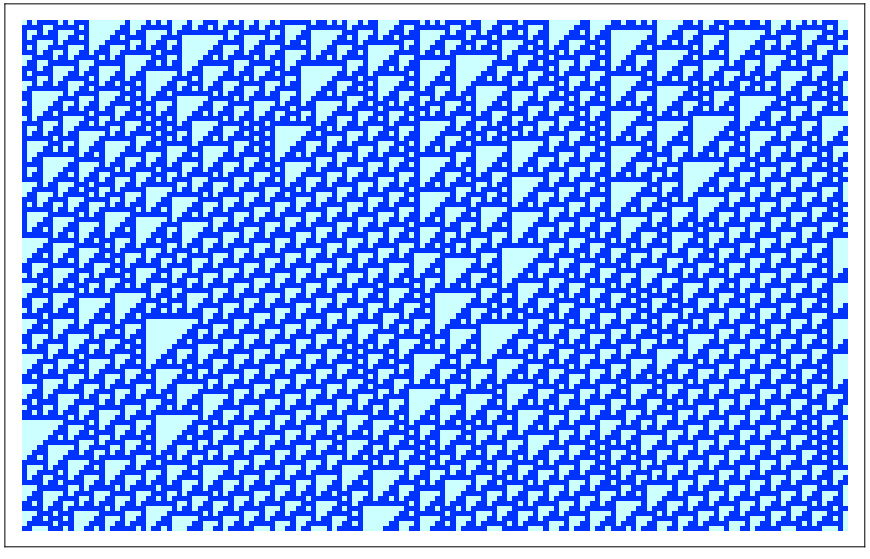
Consider ECA 90, local map $f(x, y, z) = x \oplus z$.
Show that a $\log t$ shortcut exists for this ECA.

Some ECA Behaviors









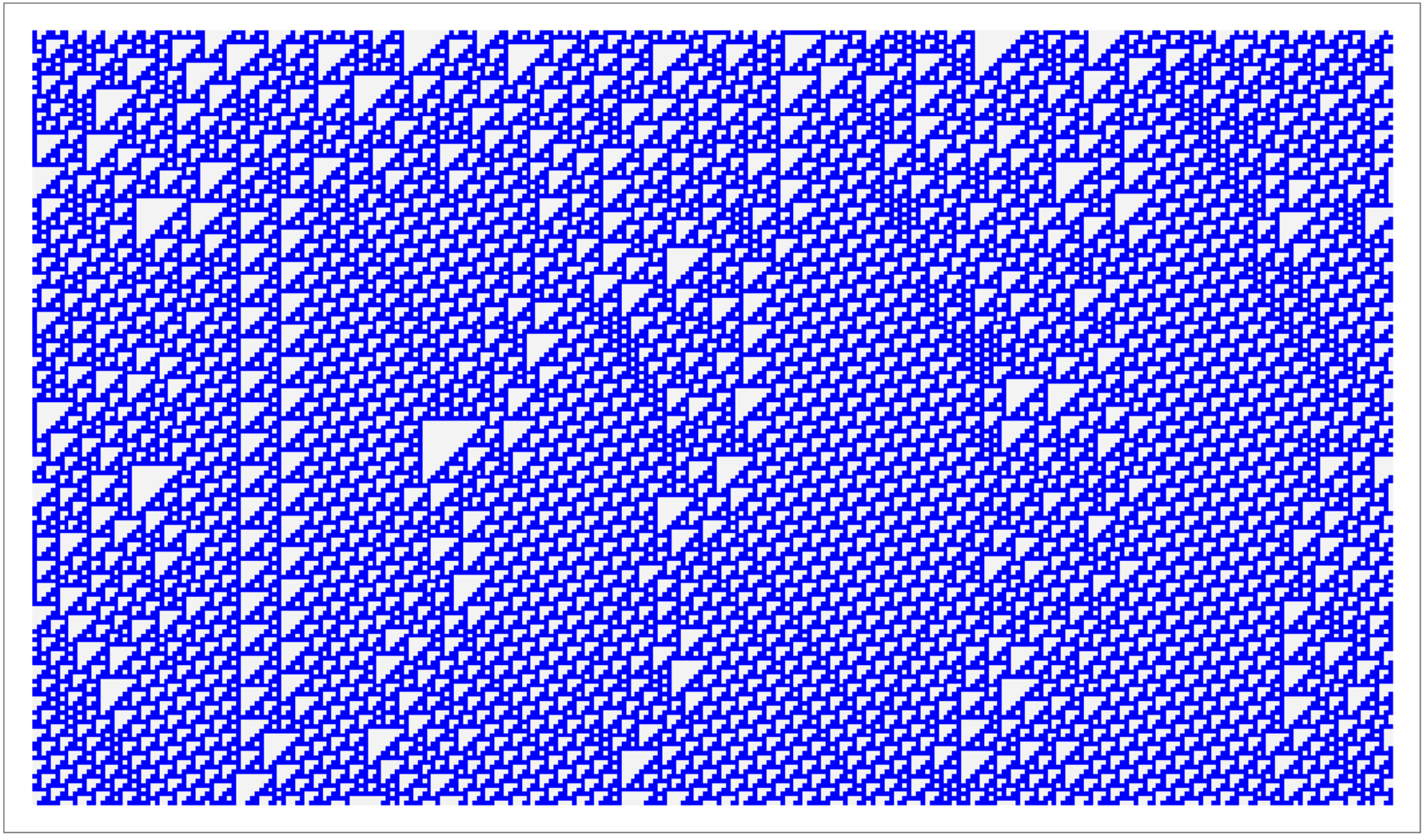
Somewhat surprisingly, elementary cellular automata seem to display roughly 4 types of behavior.

- All configurations die out 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.

Note, though, that Wolfram's classification is entirely heuristical, it seems hopeless to formalize it in terms of computability theory (one can construct cellular automata that mix up the four classes).



And things get even more interesting when we generalize a bit: local maps look like

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

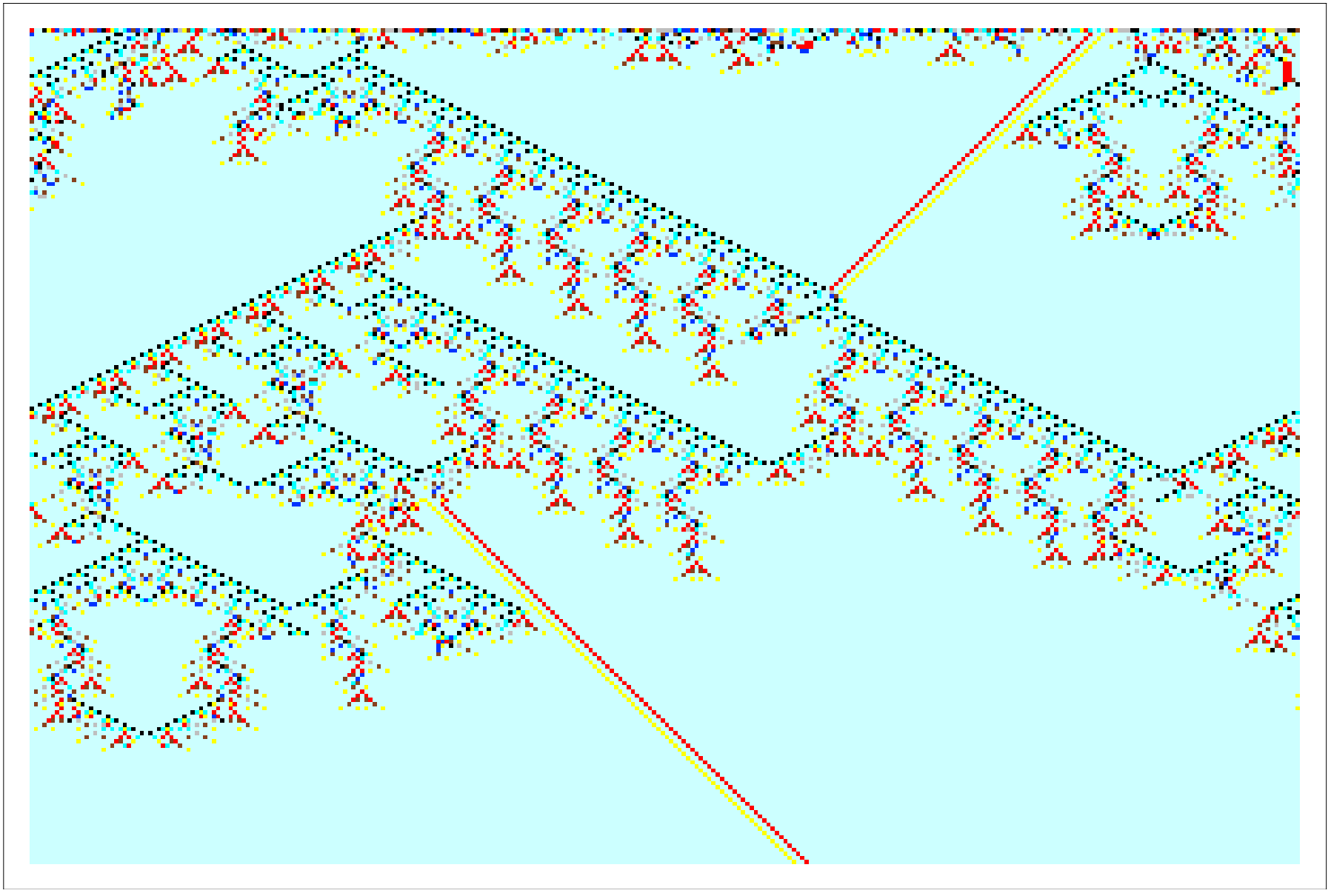
and the global maps are $\Sigma^{\mathbb{Z}} \longrightarrow \Sigma^{\mathbb{Z}}$ (r is the radius of the CA and $k = |\Sigma|$).

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.

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.

A FO Structure

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.

Sad Story: Annoyingly, with a little effort this even yields the algorithms from the last paper—a fact that I was totally unaware of when I wrote the paper. In my mind, automata-based decision methods were pure theory, very elegant but totally unimplementable.

GANS really is your friend (within bounds, of course).

How about the other properties? Surjectivity is easy:

$$\forall\, x \,\exists\, z \left(z \rightarrowtriangle x \right)$$

But openness is more difficult: it's a topological property dealing with open sets and there is no direct first-order definition.

Here is a trick: define $x \stackrel{*}{=} y$ to mean that configurations x and y differ in only finitely many places. Then the global map is open iff

$$\forall\, x, y, z \,(x \rightarrowtriangle z \wedge y \rightarrowtriangle z \wedge x \stackrel{*}{=} y \Rightarrow x = y)$$

Alas, $\stackrel{*}{=}$ is not first-order definable over $\langle \mathbf{2}^{\mathbb{Z}}; \rightarrowtriangle \rangle$, so we cannot rewrite this formula into one that does not contain $\stackrel{*}{=}$.

But: we can model check the larger structure $\langle \mathbf{2}^{\mathbb{Z}}; \rightarrowtriangle, \stackrel{*}{=} \rangle$.

To avoid all sorts of technical arduousness, let us start with 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).

Our goal is solve the expression model checking problem for these structures.

Note that there is a very simple, succinct description of a phase-space: for the global map, all we need is the local map which can be written as 8 bits for ECA; so we are really given some $\log n + c$ bits.

Since there are infinitely many finite spaces $\mathcal{C}_n$, it is natural to ask for which values of the parameter n a certain proposition holds (this is data model checking, in essence):

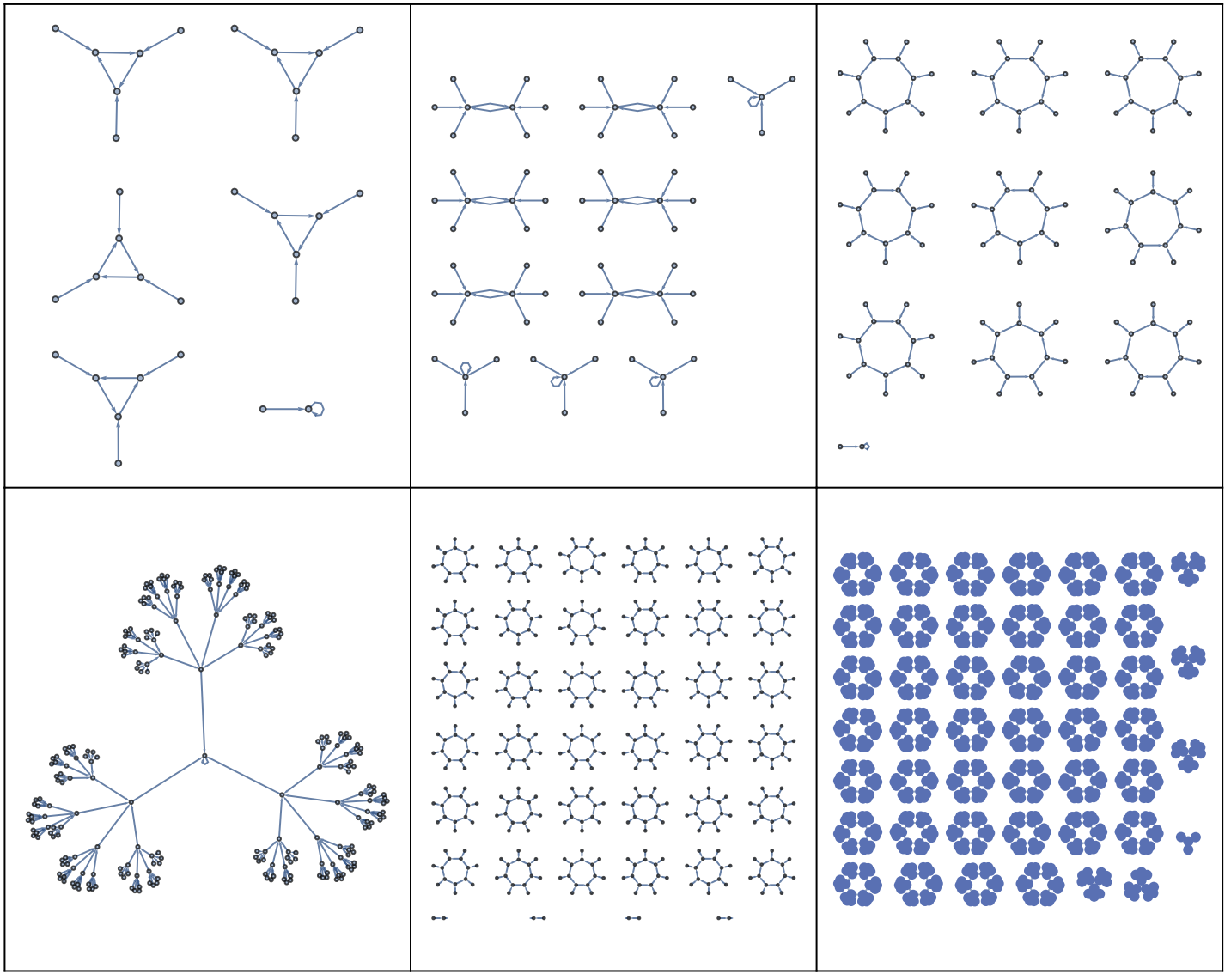
$$\operatorname{spec} \varphi = \{\, n \in \mathbb{N} \mid \mathcal{C}_n \models \varphi \,\}$$

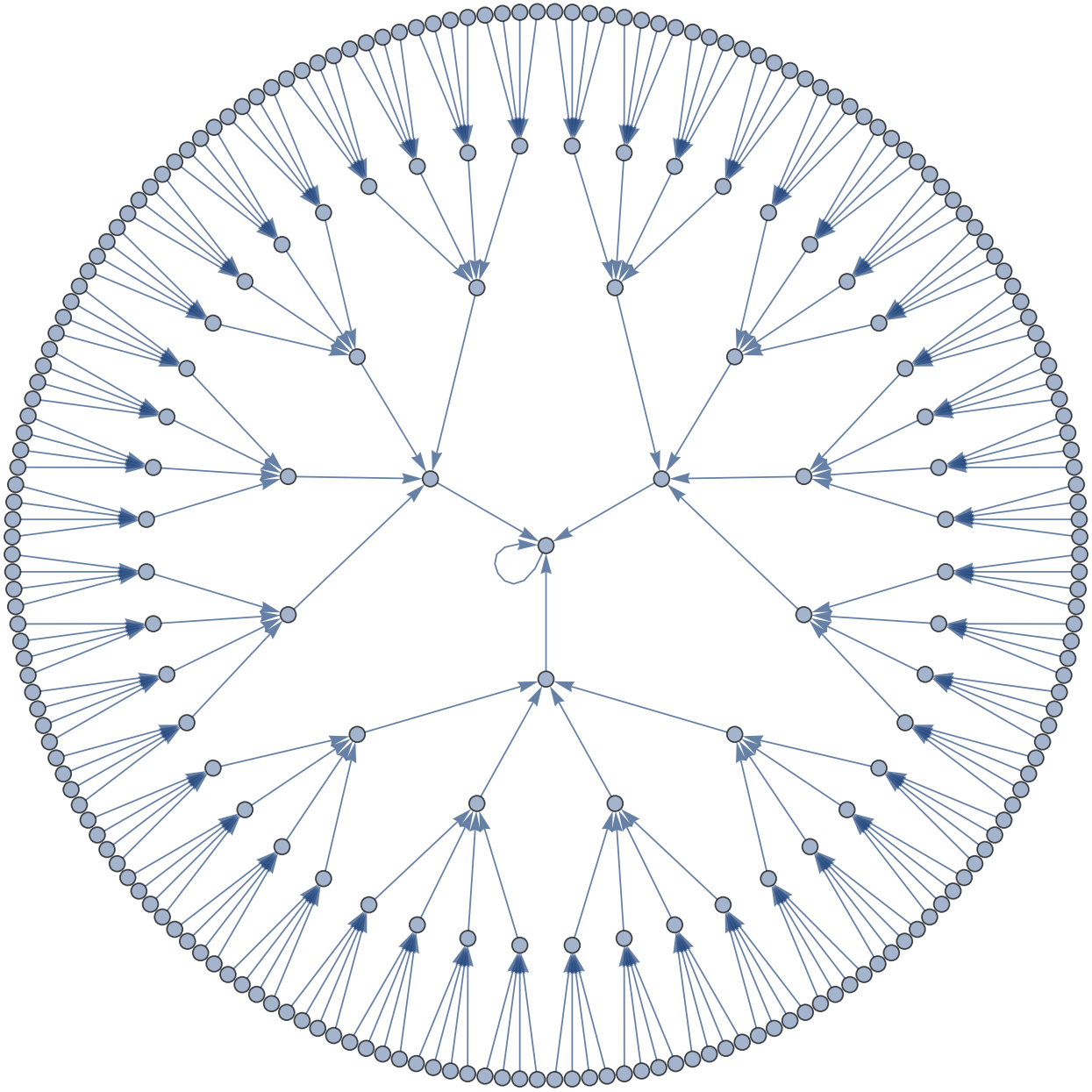
This set of "good" grid sizes is called the spectrum of φ.

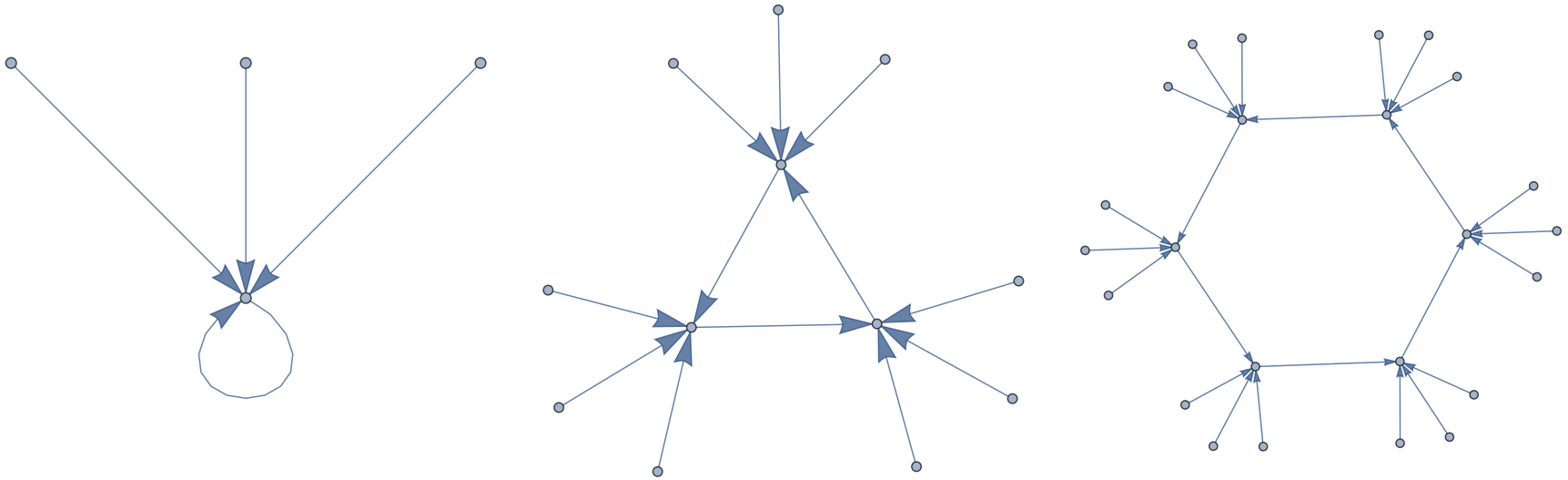
For example, we might want to know for which n a fixed local rule produces a reversible global rule on $\mathbf{2}^n$, say, under cyclic boundary conditions.

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

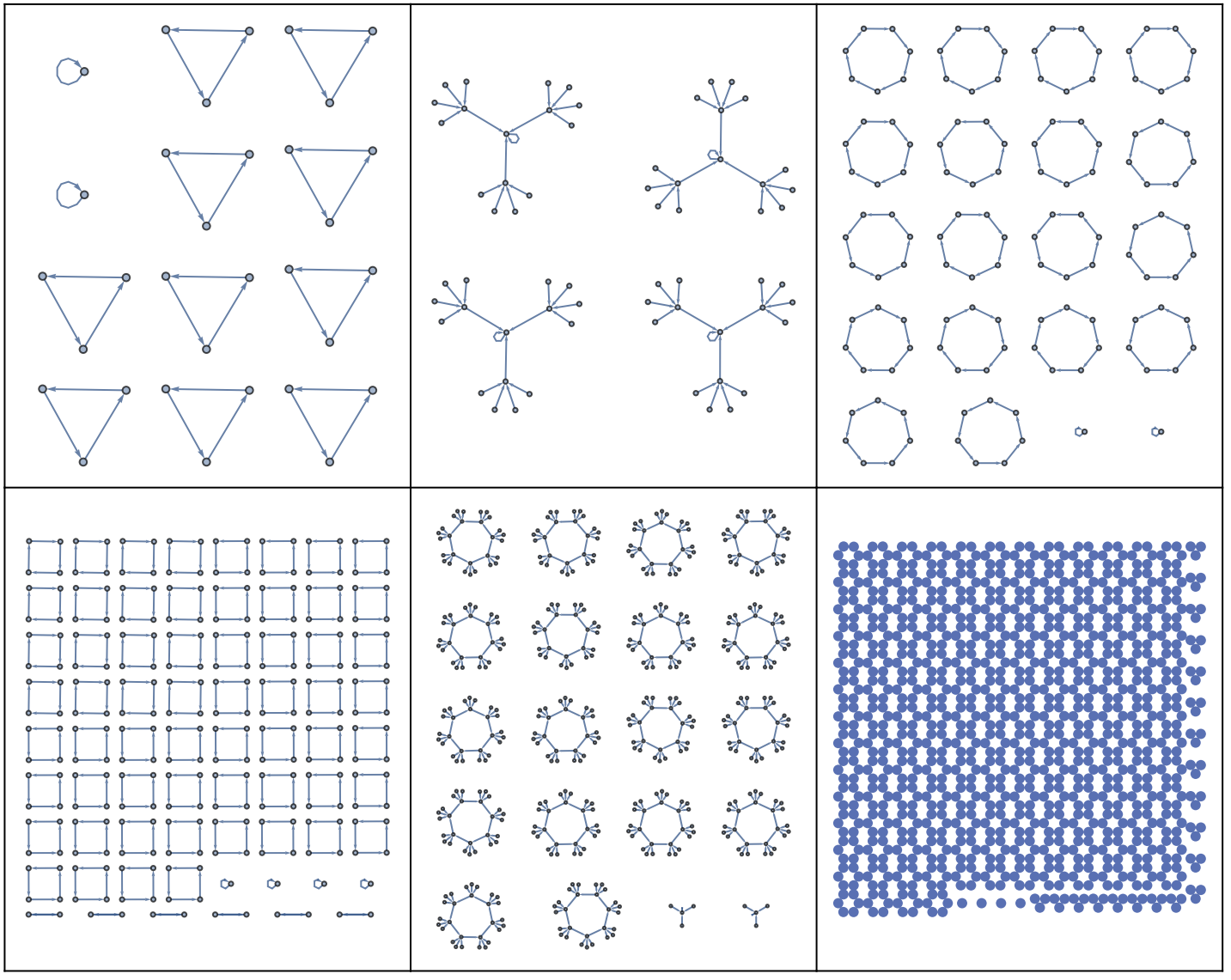
ECA 90, Phase Space $n = 5, \ldots, 10$

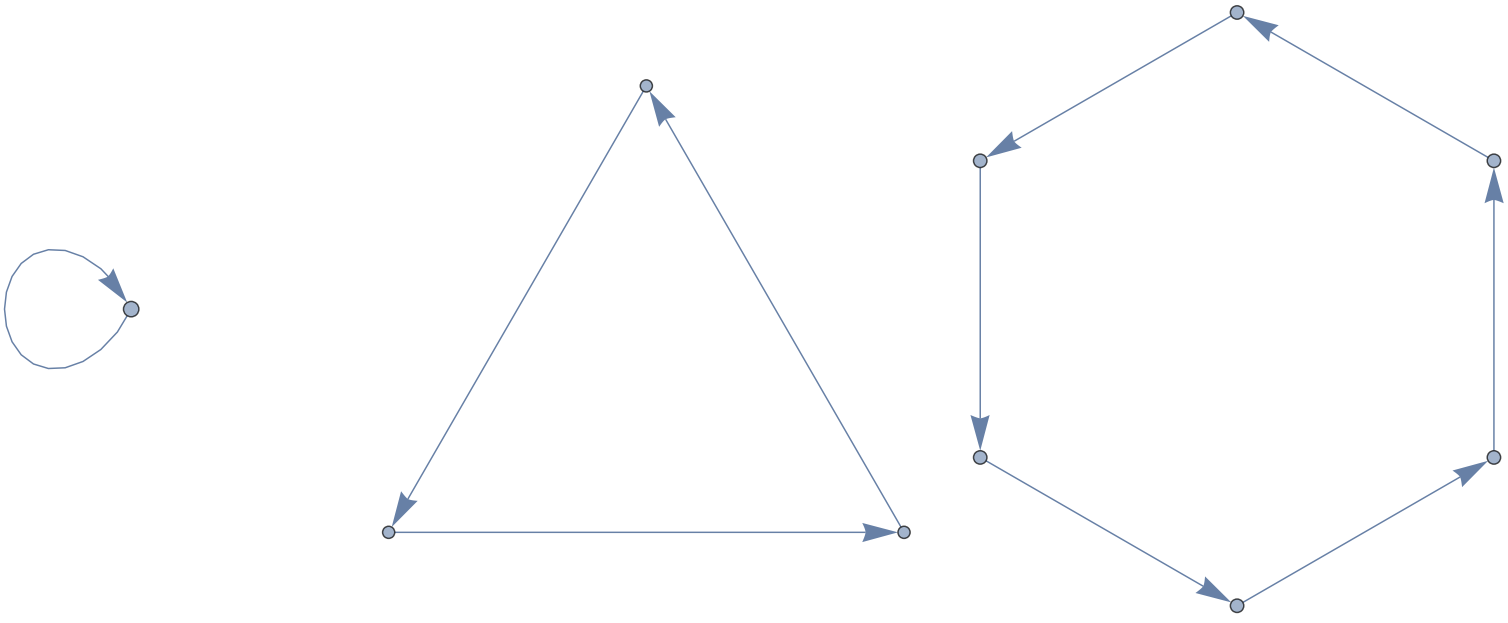






Counts: 1, 5, 40.





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 $\mathcal{A}$.

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.

This may seem too radical, but remember that 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.

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

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

where

- $A \subseteq \Sigma^\star$ is a regular set of words, and
- $R_i \subseteq A^{k_i}$, are regular relations on words of arity k_i.

We already know how to handle the carrier set, but we do not have anything like a "regular 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 regular languages according to our (entirely reasonable) definitions.

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

D. Scott, *Some Definitional Suggestions for Automata Theory*, 1967

So the next project is to generalize regular languages to some reasonable class of regular relations, also known as 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.
- Exploit Kleene's algebraic characterization of regular languages.

We'll start with the machine model and then develop the corresponding algebraic approach.

Instead of single words over an alphabet we will use 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 multi-words or k-track 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.

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 multi-words).

On occasion it is useful to think of such a relation as a map

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

where $\rho(x) = \{\, y \mid \rho(x, y) \,\}$.

What would such a finite state machine $\mathcal{A}$ describing a transduction look like? There are two useful perspectives:

- Language of multi-words.
- Partial function to sets of words.

For the language interpretation it is natural to build an acceptor for such a language. In essence, we simply change the transition labels from words to multi-words:

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

So we allow arbitrary words as labels (unlike letters in the plain finite state machine case). More on this later.

Other than that, we simply copy the definitions from our discussion of regular languages.

A good way to think about an acceptor of multi-words is to modify our old finite state machines:

- Keep the finite state control.
- Allow multiple tapes.

Each tape has a separate, one-way read head. Importantly, the heads move independently from each other; in particular, one head can get arbitrarily far ahead of another.

We can express the two-tape situation via a transition relation of the type

$$\tau \subseteq Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \times Q$$

We can check that $x = y$: do a letter by letter comparison.

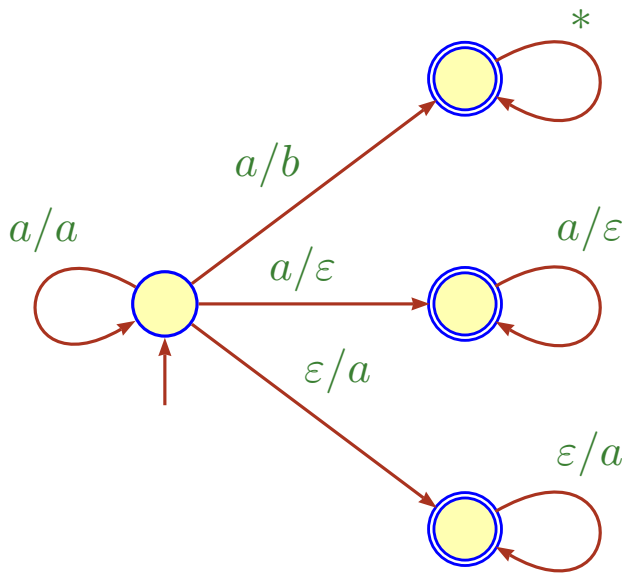
We can check that $x \neq y$: see below.

We can check that x is a prefix of y: do a letter by letter comparison until x ends, then skip over the rest of y.

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

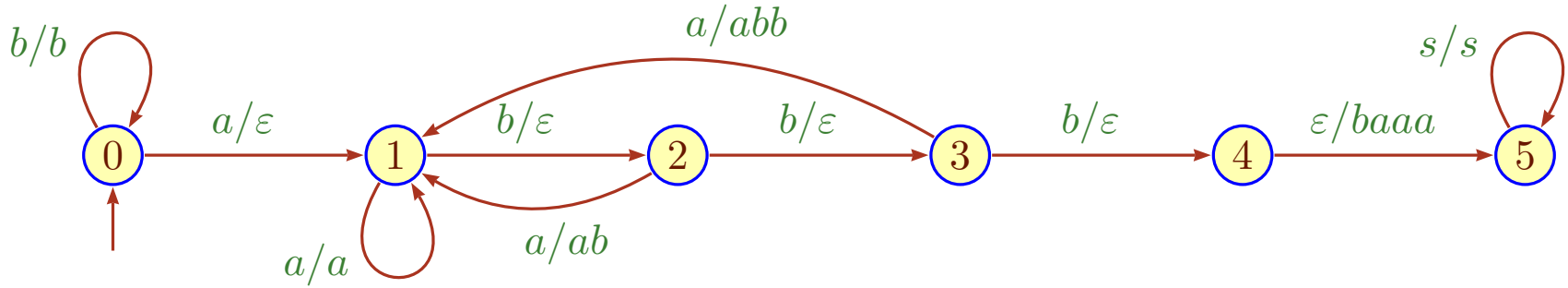
BTW, it is often helpful to attach a special endmarker (typically $\#$) to the end of all words. The machines are a little cleaner this way.

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.

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.

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.

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

Recall: Kleene's Theorem

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.

It is easy to see that we could also allow generators that are (singletons) of arbitrary multi-words:

$$x_1{:}x_2{:}\ldots{:}x_{k-1}{:}x_k$$

where $x_i \in \Sigma_i^\star$.

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

Unfortunately, because of the multiple tracks, this notation system is not as useful as in the case of languages.

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

$$M = \Sigma^\star \times \Gamma^\star$$

The operation here is $x{:}y \cdot u{:}v = xu{:}yv$ and the neutral element is $\varepsilon{:}\varepsilon$. In algebraic terms, this is just a product monoid.

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 better to the idea of having two tracks with one word in each track.

Note that multiplication here is componentwise:

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

Let $\Sigma = \{a, b\}$.

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

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

Exercise

Construct an expression for the un-equal relation.

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.