Linear Cellular Automata and Automaticity

Klaus Sutner

Abstract

The theory of finite state machines in general, and of synchronous rational relations in particular, plays a central role in the study of one-dimensional cellular automata. We discuss a uniform framework to answer questions about finite cellular automata, as well as one-way and two-way infinite ones. The first-order theory of the global map is decidable in all three cases, and feasible algorithms are available for sufficiently simple questions.

1 Introduction

One-dimensional cellular automata are substantially different from their higher-dimensional counterparts in that they are easily described in terms of finite state machines. For example, the cover language of a two-sided shift associated with a cellular automaton is always regular, and one can measure its complexity in terms of the associated minimal automaton. Putting aside issues of computational complexity, the state complexity of a regular language is fairly easy to compute and there are efficient algorithms that can be brought to bear. While these methods are confined to languages recognized by finite automata, it was already pointed out by Dana Scott in 1967 [44] that the use of finite automata to define functions and relations should be considered to be, at minimum, of equal importance.

> 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.

Klaus Sutner
Department of Computer Science, Carnegie Mellon University, USA, e-mail: sutner@cmu.edu

In fact, in their seminal paper [41], Rabin and Scott already explored some of the properties of functions defined be finite state machines, now commonly referred to as rational transductions. In the context of one-dimensional cellular automata, transductions arise naturally, since we can think of the global map of the automaton as a rational transduction, a function on a suitable class of words. One complicating factor here is that the words in question are naturally one-way or two-way infinite, so the machinery of finite state machines needs to be generalized to these types of words. In addition, some amount of information can still be obtained by studying associated languages of finite words.

Rational transductions themselves are easily computable and sidestep issues of undecidability, but their behavior under iteration can be complicated. To wit, it is easy to see that the one-step relation of a Turing machine is a rational transduction, and indeed a fairly simple one that mostly copies its input to output. Hence, iteration and the study of orbits of cellular automata will necessarily encounter issues of undecidability. In fact, attempts to formalize a classification based on orbits along the lines of Wolfram's four classes leads to decision problems in the first few levels of the arithmetic hierarchy [47]. The same is true for other classifications that are more closely associated with topological dynamics [33, 32]. Worse, as shown by Baldwin and Shelah [1], it seems difficult to formulate any hierarchy of cellular automata based on notions of computational complexity, rather than the classical ideas from symbolic dynamics. As the authors make clear, input/output conventions play a major role when it comes to interpreting symbolic dynamical systems as computational devices. One can try to sidestep these coding issues by focusing instead directly on the complexity of the orbit relation: how difficult is it to determine whether a configuration lies in the orbit of another? This approach was first proposed by Davis [17] in an attempt to develop a more robust notion of computational universality. As it turns out, in the context of cellular automata, orbit complexity produces a highly complicated hierarchy that involves all semidecidable degrees, since one can construct a particular linear cellular automaton whose orbit relation is of exactly that of some given degree $\mathbf{d}$, see [51]. In fact, one can construct the cellular automaton in a way that the orbit relation has some degree $\mathbf{d}_1$, whereas the confluence relation has another given degree $\mathbf{d}_2$. Somewhat surprisingly, one can even build a reversible cellular automaton whose orbit relation has some specified degree (though confluence has necessarily the same degree in this case). Testing membership in these degree-based classes is again undecidable, e.g., it is already Σ_3-complete to check whether the orbit relation is decidable, and it is Σ_4-complete to check whether the orbit relation is r.e.-complete.

By contrast, we will here narrowly focus on properties of linear cellular automata that are necessarily decidable and can be algorithmically determined by exploiting the theory of regular languages and rational transductions. In a sense, this approach goes back to Büchi's application of automata theory to the Entscheidungsproblem for monadic second order logic. Another milestone was the study of automatic groups by Epstein [21]. More generally, one considers automatic structures, i.e., first-order relational structures whose carrier set and relations can be described by finite state machines, see [28, 37]. The use of finite state machines to solve decision problems over various logics has also become a staple in formal verification methods in computer science, see [11] and [27] for extensive bibliographies.

In the context of linear cellular automata there are three types of automatic structures that need to be analyzed: structures over finite words, one-way infinite words and two-way infinite words. We will refer to these scenarios as word automatic, ω-automatic and ζ-automatic, respectively. Unsurprisingly, the methods applied in all three cases are quite similar in principle, but they differ substantially in the level of technical machinery required in the associated algorithms. In particular the two-way infinite cases poses considerable algorithmic problems and typically requires human intervention to obtain feasible solutions. Still, one fundamental property of automatic structures is that they all have a decidable first-order theory by avoiding concepts such as orbits that inherently require a stronger logical framework such as second-order or transitive closure logic. In a sense, the results obtained in this manner are local in nature, one can explore only local structures in phase space such as questions of reversibility, openness, surjectivity and so on. As we will see, the framework provided by finite state machines provides elegant and simple solutions to questions related to linear cellular automata in all the situations where it can be applied.

We are going to outline the basic arguments, but, in the interest of brevity, we refer the reader to the copious literature. For general background on finite state machines see [26, 43, 40]. A discussion of recognizable languages of two-way infinite words can be found in [38] and [39]; in the context of cellular automata, two-way infinite words appear in [15] and [14]. For background on decision problems in logic and the use of automata-theoretic methods [6, 7, 8, 45, 2, 35].

2 Symbolic Dynamics and Automata

2.1 Finite Automata

A *transition system* is a directed graph $T = \langle Q, E \rangle$ whose edges are labeled by letters of some alphabet Σ. Alternatively, we can also think of T as a *transition relation* $T \subseteq Q \times \Sigma \times Q$. Correspondingly, the transition system is *deterministic* if the relation is single-valued, *co-deterministic* if the reversal of the relation is single-valued and *complete* if the relation is total. We are only interested in the case when the *state set* Q is finite; similarly, the alphabet Σ is required to be a finite, non-empty set. By a *word* over Σ we mean an element of Σ^*, the free monoid generated by Σ; a *language* over Σ is any subset of Σ^*.

Paths in T are naturally associated with label sequences, words in Σ^*. A *finite state machine* or *automaton* $\mathcal{A}$ consists of a transition system T together with an *acceptance condition*. The acceptance condition is always expressed in terms of path existence in the transition system. In the most basic case, one chooses two subsets of I and F of Q and selects paths with source in I and target in F. This produces the *acceptance language* $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^*$; one also says that $\mathcal{A}$ *recognizes* the language. A language is *regular* if it is recognized by some automaton. We refer to these machines as *acceptors*, as opposed to the *transducers* introduced in a moment.

In a *semiautomaton* we assume $Q = I = F$, so all paths in the transition system are considered. An automaton is deterministic or co-deterministic whenever the underlying transition system is so deterministic or co-deterministic. The automaton is *ambiguous* if there are two distinct paths in the transition system carrying the same label and with the same source and target. A *partial deterministic finite automaton (PDFA)* has a deterministic transition system and a single initial state. A *deterministic finite automaton (DFA)* is, in addition, required to be complete. The *state complexity* of an automaton is the number of its states. It is well-known that every regular language is recognized by a unique (up to isomorphism) DFA of minimum state complexity, the so-called *minimal automaton* of the language. Hence we can define the state complexity of a regular language as the state complexity of the corresponding minimal DFA. Regular languages enjoy numerous closure properties, they form a Boolean algebra, are closed under concatenation, Kleene star, reversal, homomorphic images and preimages. Moreover, all these closure properties are effective and often polynomial time, the key exception being determinization which can be exponential in time and space, see [26, 43, 40, 5].

Acceptors recognize languages, in order to address relations on words we need a slight generalization. The most basic kind of relation on words has the form

$$R \subseteq \Sigma^* \times \Sigma^*$$

and we will refer to this type of a relation as a *transduction*. We write the elements of a transduction as u/v where $u, v \in \Sigma^*$. A transduction is *rational* if there exists a finite automaton with labels in $\Sigma^* \times \Sigma^*$ that recognizes R: $u/v \in R$ iff there is a path in the transition system from I to F labeled by u/v. We will refer to these automata as *transducers*, as opposed to the language acceptors from above. Needless to say, we insist that the number of transitions is still finite. One can easily verify that the labels in a transducer can be chosen to be of the form a/ε, ε/b and a/b where $a, b \in \Sigma$, without affecting the class of rational relations. By a well-known theorem due to Elgot and Mezei [20], rational transductions are closed under composition. Analogous definitions can be given for k-ary relations in general. It is convenient to visualize the automata recognizing transductions as being equipped with two one-way, read-only input tapes that operate independently and are controlled by a single finite state unit.

As was already pointed out in the Rabin-Scott paper, there are substantial differences between regular languages and rational relations in general. While the latter are closed under union, essentially by definition, they fail to be closed under intersection and complement. In general, nondeterminism is essential for transducers, while it can be eliminated for acceptors—at least if one ignores complexity issues. As a consequence, decision algorithms based on representing the structure in question by finite automata have to avoid rational relations in general.

A way to sidestep these problems is to focus on *synchronous* transductions where all the transition labels are of the form a/b, $a, b \in \Sigma$, also referred to as *alphabetic* or *letter-to-letter* transductions. Synchronous transductions were introduced in [20] and rediscovered in [23]. In essence, a synchronous relation is but a regular language over a product alphabet $\Sigma \times \Sigma$ and enjoys similar closure properties. Note that these relations are trivially length-preserving. By another theorem of Elgot and Mezei, any length-preserving rational transduction is already synchronous in the sense that there is a synchronous transducer that recognizes the transduction, though the corresponding machine may be more difficult to construct. In terms of the automaton model, the tape heads of the two input tapes are moving in lockstep. For two words $x, y \in \Sigma^n$, one refers to

$$x{:}y \;=\; \begin{array}{|c|c|c|c|c|}\hline x_1 & x_2 & \dots & x_{n-1} & x_n \\ \hline y_1 & y_2 & \dots & y_{n-1} & y_n \\ \hline \end{array}$$

as the *convolution* of x and y, a string over the product alphabet $\Sigma \times \Sigma$. This idea naturally extends to strings x and y of differing lengths by padding out the shorter string to the right, using a specially designated padding symbol. A great many important word relations are synchronous in this sense, prime examples being lexicographic order, concatenation and binary addition. Since global maps are naturally length-preserving we have no need for padding.

A *Mealy automaton* is a particularly simple type of synchronous transducer where the projection that omits the second component of all labels produces a deterministic transition system. In other words, for any two transitions $p \xrightarrow{a/b} q$ and $p \xrightarrow{a/b'} q'$ we can conclude that $b = b'$ and $q = q'$. By choosing an initial state, we obtain a partial, length-preserving function on words. These devices are called output modules in Eilenberg [19].

To analyze computations of a finite state machine it is convenient to extend the transition system of the automaton along an additional dimension of time. Suppose we have a transition system $T = \langle Q, E \rangle$ over Σ and a word $x \in \Sigma^*$ of length n. We define the *unfolding of T along x* to be the directed graph with vertex set $Q \times [0, n]$ and edges

$$(p,t) \xrightarrow{a} (q, t{+}1) \qquad \text{for} \quad a = x_{t+1}, p \xrightarrow{a} q \in E$$

We write $\mathsf{unf}(T, x)$ for the unfolding of T along x. For clarity, we will often refer to the index t as *time t*. A *traversal* in the unfolding is a path from a point at time 0 to a point at time n. When T is the transition system of a semiautomaton $\mathcal{A}$, the computations of $\mathcal{A}$ on input x are represented by traversals in the unfolding. Hence, the *multiplicity* of a word x with respect to $\mathcal{A}$ is simply the number of traversals in the unfolding. An example of an unfolding is shown in figure 1; the semiautomaton in question will be discussed in section 2.2.

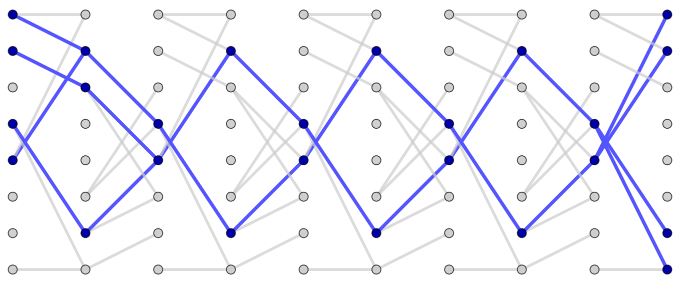

Fig. 1 The unfolding of a de Bruijn semiautomaton along the word 01010101010. There are 8 traversals in two connected components.

So far, our definitions all revolve around finite words. To address the one-way infinite or two-way infinite sequences one encounters in symbolic dynamics, we generalize to *ω-words* of the form $\mathbb{N} \longrightarrow \Sigma$ and *ζ-words* of the form $\mathbb{Z} \longrightarrow \Sigma$. We write Σ^ω and Σ^ζ for the corresponding sets of words. Our definitions related to transition systems can be lifted, *mutatis mutandis*, in a fairly natural fashion to these infinite words. For example, for an ω-word, the unfolding of a transition system is over $\mathbb{N}$ and a traversal is a one-way infinite path with source at time 0; similarly, in the two-way infinite case, a traversal is any two-way infinite path in the unfolding. Difficulties do arise in connection with acceptance conditions suitable to these settings. We will comment section 3.2.1 on appropriate choices that allow us to handle infinite words in a manner that preserves the critical properties in the finite case. For a detailed discussion of infinite words and their associated automata we refer the reader to [39]. Unsurprisingly, reasoning about infinitary acceptance conditions becomes substantially more complicated and generally requires tools of infinite combinatorics such as Kőnig's lemma, e.g., for correctness proofs related to automata-theoretic algorithms.

2.2 Subshifts and Linear Cellular Automata

A *subshift* or *shift space* is a subset $\mathcal{X}$ of Σ^ζ that is topologically closed in the sense of the natural product topology on Σ^ζ and shift-invariant: $\sigma(\mathcal{X}) = \mathcal{X}$ where $\sigma(X)_i = X_{i+1}$. Infinite words X can be associated with a language of finite words, the *cover* $\mathsf{cov}(X) \subseteq \Sigma^*$, also known as the set of blocks of X, consisting of all finite factors of X. For a language L of infinite words, the cover is the union of the individual covers: $\mathsf{cov}(L) = \bigcup_{X \in L} \mathsf{cov}(X)$. Since Σ^ζ is compact, a shift space can be reconstructed from its cover: a configuration X is in $\mathcal{X}$ iff it is the limit of a sequence of words in the cover. Morphisms between shift spaces are continuous maps $f : \mathcal{X} \longrightarrow \mathcal{Y}$ that commute with the shift.

A shift space is *sofic* or a *sofic system* if its cover is a regular language. It is clear that $\Sigma^* - \mathcal{X}$ is a two-sided *ideal*, that is to say, a set $I \subseteq \Sigma^*$ such that $\Sigma^* \, I \, \Sigma^* = I$, the ideal of forbidden blocks. A *basis* of an ideal is any subset that generates it and a subshift that admits a finite bases for this ideal is a *subshift of finite type*. It was shown by Weiss [54] that sofic systems are the smallest class of subshifts that contains all shifts of finite type and is closed under morphisms. Moreover, the morphism can always be chosen to be finite-to-one according to [13].

By the Lyndon-Curtis-Hedlund theorem [25], the morphisms themselves are the *global maps* of a linear cellular automaton. For our purposes, a *linear cellular automaton (LCA)* is represented by its *local map* or *(local) rule* $\rho : \Sigma^w \longrightarrow \Sigma$ where Σ is the alphabet of the cellular automaton and we may safely assume that the *width* w is at least 2. We will always assume that Σ is the digit alphabet $\Sigma_k = \{0, 1, \dots, k{-}1\}$, so it is natural to identify local rules over a Σ_k with rule numbers in the interval $[0, k^{k^w} - 1]$ in the obvious fashion.

In symbolic dynamics and coding theory [32, 34] global maps are also referred to as *sliding block codes* and it is customary to control the application of the local map more carefully. Specifically, one considers two integers α, the *anticipation*, and β, the *memory*, where $\beta \leq \alpha$. For any word $X = (x_i)$, finite or infinite, that have positions $i \leq j$, define the block from i to j to be the factor $X[i{:}j] = x_i x_{i+1} \dots x_j \in \Sigma^{j-i+1}$ of X. Now suppose we have $w = \alpha - \beta + 1 \geq 1$. We can then define the global map G_ρ associated with ρ by

$$G_r(X)(i) = \rho\big(X[i+\beta{:}i+\alpha]\big)$$

provided that the indicated positions in X exist. This is always the case for two-way infinite words, but requires a bit more attention otherwise. First off, for our purposes it is safe to assume that $\beta \leq 0 \leq \alpha$. In fact, for w odd, we can keep the position of the output letter centered be setting $\alpha = \lfloor w/2 \rfloor$ and $\beta = -\alpha$. In the even case we think of adding a phantom last variable to ρ, so that $\alpha = \lfloor w/2 \rfloor - 1$ and $\beta = -\alpha - 1$. We can now handle the missing position problem and associate the following global maps with any local map ρ:

word map	$\rho^* : \Sigma^{\geq w} \longrightarrow \Sigma^+$
periodic word map	$\rho^{\mathsf{p}} : \Sigma^+ \longrightarrow \Sigma^+$
fixed word map	$\rho^{\mathsf{f}} : \Sigma^+ \longrightarrow \Sigma^+$
ω global map	$\rho^\omega : \Sigma^\omega \longrightarrow \Sigma^\omega$
ζ global map	$\rho^\zeta : \Sigma^\zeta \longrightarrow \Sigma^\zeta$

For a finite word X of length n, $\rho^*(X) = \rho(X[1{:}w]) \dots \rho(X[n-w+1{:}n])$ is a word of length $n - w + 1$. Similarly, $\rho^{\mathsf{f}}(X) = G_\rho\big(0^{-\beta} X 0^\alpha\big)$ for fixed boundary conditions and $\rho^{\mathsf{p}}(X) = G_\rho\big(X[n+\beta-1{:}n] \; X \; X[1{:}\alpha]\big)$ for periodic ones. The last two maps are naturally length-preserving. In the infinite scenario we have $\rho^\omega(X) = G_\rho\big(0^{-\beta} \, X\big)$, for one-way infinite words, and $\rho^\omega(X) = G_\rho\big(X\big)$, for two-way infinite words. Note that we exclude the empty word in the finite case, this choice will be visible in the generating functions in examples below.

Since cellular automata ρ are morphisms, we obtain a shift space by setting $\mathcal{S} = \rho^{\zeta}(\Sigma^{\zeta})$. We write $\mathcal{L}(\rho) \subseteq \Sigma^*$ for the cover of this space. It is easy to see that these spaces are always sofic: we can construct a semiautomaton based on a de Bruijn graph whose edges are labeled by the local map of the CA. More precisely, suppose the cellular automaton uses alphabet Σ and has width $w \geq 2$. Then the *de Bruijn graph* over Σ of order $w-1$ has vertex set Σ^{w-1} and directed edges $ax \longrightarrow xb$ where $a, b \in \Sigma$ and $x \in \Sigma^{w-2}$. Attaching the label $\rho(axb)$ to edge $ax \longrightarrow xb$, we obtain a semiautomaton $\mathcal{B}(\rho)$ that recognizes the cover of $\mathcal{S}$.

Moreover, using standard automata-theoretic methods, one can decide whether a space $\rho^{\zeta}(\Sigma^{\zeta})$ is finite type. Specifically, we can compute an automaton accepting a minimal basis for the ideal of forbidden blocks and then test whether the language of the automaton is finite. For example, there are 176 elementary cellular automata producing subshifts of finite type and rule 77 has ideal base $00011, 00111, 11000, 11100, 0001000, 1110111$. The growth rate of the ideal language is given by the recurrence $a_n = 2a_{n-1} - a_{n-3} + a_{n-5}$ with initial conditions $1, 2, 4, 8, 16, 28$. By contrast, the ideal base for rule 18 is of the form $11\left(0^+10^+1\right)^*1$ and the growth rate of the ideal language is given by the recurrence $a_n = 3a_{n-1} - a_{n-2} - a_{n-5}$ with initial conditions $1, 2, 4, 7, 13$.

From the perspective of rational relations, it is clear that all the maps defined above are rational transductions. In fact, all these transductions other than the shrinking word map are synchronous. In the infinite case we consider again the de Bruijn graph, but this time the edge $ax \longrightarrow xb$ is labeled $b/\rho(axb)$, see figure 2. These transducers are of the Mealy type, if one allows for multiple initial states. Strictly speaking, the corresponding transduction involves a shift, see below for a construction that centers the output. In a sense, shifted output is irrelevant for our purposes, but it may affect the size of the machines involved, see section 3.3.

For finite words we need to modify the de Bruijn transducer slightly. For simplicity, let us only consider periodic boundary conditions for width 3, the argument can be easily adjusted to other widths and the fixed boundary case. The transducer for ρ has states of the form $(xyz, lr) \in \Sigma^3 \times \Sigma^2$ where l and r are used to nondeterministically guess the leftmost and rightmost letter in the input, respectively. The transitions take the form

$$\begin{aligned} \bot &\xrightarrow{l/e} rlz, lr && e = \rho(r, l, z) \\ xyz, lr &\xrightarrow{z/e} yzu, lr && e = \rho(y, z, u) \\ xyr, lr &\xrightarrow{r/e} \top && e = \rho(y, r, l) \end{aligned}$$

While this transducer is nondeterministic as defined, it can be determinized and turned into a Mealy automaton. The result for elementary cellular automaton number 150, the exclusive or of all three bits, is shown in figure 3. The given construction is based on using memory -1 and anticipation of 1, but one could also choose memory -2 with no anticipation. As far as efficiency is concerned, there seems to be no clear advantage to either approach.

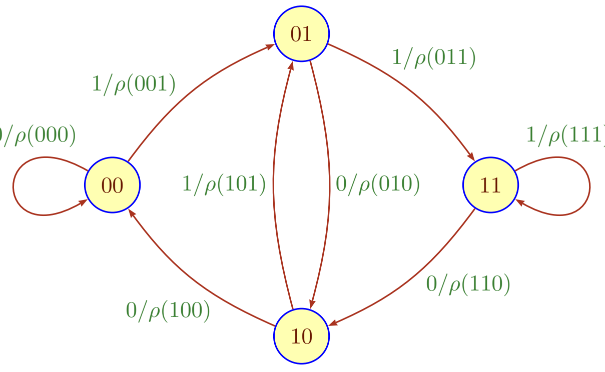

Fig. 2 The standard synchronous de Bruijn transducer for an elementary cellular automaton with local rule ρ. The corresponding de Bruijn acceptor is obtained by removing the labels in the upper track.

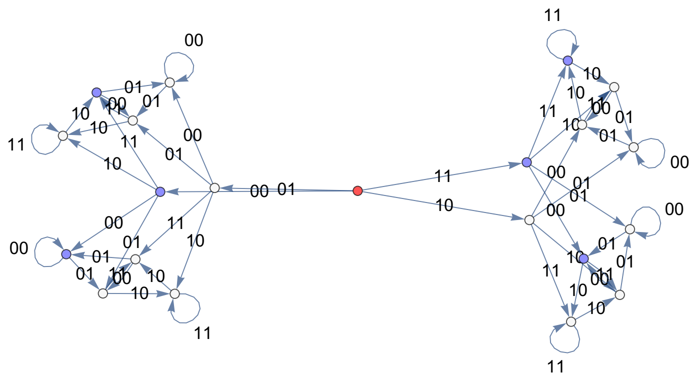

Fig. 3 A Mealy machine that recognizes ρ^{p} associated with elementary cellular automaton number 150. To avoid more visual clutter, we have written transition labels a/b as ab.

The cover languages $\mathcal{L}(\rho)$ have a number of special properties: in particular, they are *transitive*, *factorial* and *extensible*. Let $L \subseteq \Sigma^*$, $L \neq \emptyset, \varepsilon$ be a language. Then L is factorial if L is closed with respect to factors: $uxv \in L$ implies that $x \in L$. Language L is transitive iff $u, v \in L$ implies that $uxv \in L$ for some x; likewise, a finite state machine is called transitive iff its transition system has only one strongly connected component. Lastly, L is extensible if for all $x \in L$ there exist $a, b \in \Sigma$ such that $axb \in L$. We refer to such languages as TFE languages. It was shown by Darji and Seif [16] that it is decidable whether a given regular language is a cover language.

As a consequence, the standard minimal DFA associated with any regular language is arguably not a good representation for cover languages. We refer to a transitive semiautomaton as a *Fischer* automaton. Clearly, the acceptance language of a Fischer automaton is always TFE. These automata were first introduced by Fischer [22] and discovered independently by Beauquier [3] in the guise of the 0-minimal ideal in the syntactic semigroup of $\mathcal{L}(\rho)$. The minimal Fischer automaton is unique up to isomorphism, and is the smallest deterministic semiautomaton accepting a given TFE regular language. As in the case of standard minimal DFA, behavioral equivalence provides a natural epimorphism from any deterministic semiautomaton to the minimal Fischer automaton. Since de Bruijn graphs are strongly connected and even Hamiltonian, our de Bruijn automata $\mathcal{B}(\rho)$ are in particular Fischer automata, albeit nondeterministic ones in general.

From a computational perspective, a simple way to determine the minimal Fischer automaton is to first compute the standard minimal DFA $\mathcal{A}$, see [49]. We may safely assume the language is not equal to Σ^*, so the diagram of $\mathcal{A}$ has a unique sink. Moreover, the diagram contains a unique strongly connected component that that has outgoing edges only to the sink. The subautomaton induced by this strongly connected component is the minimal Fischer automaton. In the context of cellular automata, the only potentially non-polynomial step in this computation is the determinization of the original de Bruijn automaton. Unfortunately, exponential blowup is possible in these circumstances and occurs with some frequency. More precisely, consider a de Bruijn automaton over a binary alphabet that is both deterministic and co-deterministic; we will refer to these automata as *permutation automata* since every letter in the alphabet induces a permutation of the state set. The language of a permutation automaton is trivially Σ^*, but changing the label of a single transition produces a nondeterministic machine, a 1-permutation automaton in reference to the Hamming distance of the labeling to the nearest permutation labeling. If the switch occurred at a self-loop, the 1-permutation automaton will exhibit full exponential blowup during determinization. Indeed, in this case, the corresponding minimal DFA consists of a single large strongly connected component, the Fischer automaton, plus a sink. It seems that this occurs for exactly one half of the 1-permutation automata obtained from $(w,2)$ cellular automata. This conjecture has been computationally verified up to width 5, but remains open and seems difficult.

2.3 Surjectivity and Injectivity

De Bruijn automata and their unfoldings provide a simple framework to explain many of the classical results in symbolic dynamics, with an emphasis on attendant algorithms. Since we are dealing with several types of global maps, we will refer, say, to a local rule being n-surjective, ω-surjective and ζ-surjective, depending on whether we consider words in Σ^n, Σ^ω, Σ^ζ. In the finite case we assume periodic boundary conditions by default and will not further burden our notation. The goal here is to characterize all n for which the property in question obtains. For example, we can establish the characterization of surjectivity from Hedlund's seminal paper [25] as follows.

A semiautomaton is *d-deterministic* for some $d \in \mathbb{N}$ if for any two computations π and π' of length $d+1$ with the same label and the same source we have $\pi(1) = \pi'(1)$. Thus, a 0-deterministic automaton is simply a deterministic automaton. The definition for d-codeterministic automata is analogous. A universal semiautomaton is *balanced* if all words have the same multiplicity in the automaton. This somewhat peculiar property is of interest for the semiautomata $\mathcal{B}(\rho)$, where a simple path counting argument shows that, if $\mathcal{B}(\rho)$ is indeed balanced, then the uniform multiplicity must already be k^{w-1}.

Theorem 1 *Let ρ be a (k, w) linear cellular automaton. Then ρ is ζ-surjective iff the de Bruijn automaton $\mathcal{B}(\rho)$ is balanced iff $\mathcal{B}(\rho)$ is unambiguous.*

Theorem 2 *Let ρ be a (k, w) linear cellular automaton. Then ρ is ζ-open iff iff the de Bruijn automaton $\mathcal{B}(\rho)$ is d^+-deterministic and d^--deterministic for some d^+ and d^-.*

Theorem 3 *Let ρ be a (k, w) linear cellular automaton. Then ρ is ζ-injective iff the non-transient part of $\mathcal{B}(\rho)^2$ is the diagonal.*

We will sketch some of the arguments from the perspective of automata theory. The multiplicity condition clearly implies surjectivity by compactness. For the opposite direction, let x be a word of minimum multiplicity $m \geq 1$. Since the out-degree of every state in the de Bruijn automaton $\mathcal{B}(\rho)$ is k, all extensions xy must also have multiplicity m. In particular, all extensions $x\Sigma^{w-1}x$ have multiplicity m. Now consider the unfolding of any such word xzx. The number of traversals in the slice of $\mathsf{unf}\big(\mathcal{B}(\rho), xzx\big)$ from n to $n + w - 1$ has cardinality k^{w-1}, the same as the cardinality of Σ^{w-1}; hence we must have $m = k^{w-1}$. Thus any word must have multiplicity at least k^{w-1}, and another path counting argument shows that it must in fact be equal to k^{w-1}.

As another example, suppose all words have uniform multiplicity. If $\mathcal{B}(\rho)$ were ambiguous, and by the transitivity of the automaton, we find a word x so that the unfolding $\mathsf{unf}\big(\mathcal{B}(\rho), x\big)$ contains two separate traversals. But then x^r has multiplicity at least 2^r, a contradiction. On the other hand, suppose the de Bruijn automaton is unambiguous and consider all unfoldings $\mathsf{unf}\big(\mathcal{B}(\rho), x\big)$ where $x \in \Sigma^n$, $n \geq w - 1$. Since $\mathcal{B}(\rho)$ is k-regular, there are altogether k^{n-w+1} traversals from $(p, 0)$ to (p, n) for any state p of $\mathcal{B}(\rho)$.

Multiplicity can also be expressed in terms of a monoid homomorphism $\mu : \Sigma^* \longrightarrow \mathbb{N}^{Q\times Q}$ that maps finite words into Q by Q matrices of nonnegative integers. Here $\mu(x)(p, q)$ is the number of computations carrying label x with source p and target q. In a semiautomata, the standard definition of multiplicity is then simply the 1-norm of these matrices. The multiplicity matrices afford an alternative way to develop many of the basic results about shift morphisms, see [24]. In particular, it follows from the Frobenius-Perron theorem that the matrix semigroup generated by $\mu(a)$, $a \in \Sigma$, is finite iff it fails to contain the null matrix, i.e., when the morphism is surjective. The matrices of minimal rank form an ideal in the semigroup, that provides information about the number of predecessors of configurations.

Turning to shift spaces with a surjective LCA, the number of predecessors of any word in Σ^{ζ} is obviously bounded by k^{w-1}, but my well be smaller. To analyze the number of predecessors, recall that for any nondeterministic finite state machine $\mathcal{A}$ on state set Q we can think of the transition relation as a right semigroup action on the ambient set $\mathfrak{P}(Q)$. On this view, the classical Rabin-Scott determinization method in [41] produces the closure of the initial state set under this action. Similarly, the *kernel* automaton of $\mathcal{A}$ is obtained by instead closing the set $\{\, \{p\} \mid p \in Q \,\}$. We may safely assume that the potential sink $\emptyset$ is removed from the kernel automaton, so we are dealing with a partial deterministic automaton. Thus, computations in the kernel automaton keep track of all computations of $\mathcal{A}$ that start at any single state.

Now suppose ρ is a surjective cellular automaton and $\mathcal{B}(\rho)$ the associated semiautomaton. With a view towards determining reversibility, i.e., deciding whether the global map in injective, it is natural to consider $W_{\mathsf{m}}(\rho)$, the least number of predecessors of any word $X \in \Sigma^{\zeta}$. In other words, $W_{\mathsf{m}}(\rho)$ is the smallest possible number of traversals in any unfolding $\mathsf{unf}\big(\mathcal{B}(\rho), X\big)$. Checking whether $W_{\mathsf{m}}(\rho) = 1$ is easily translated into a purely graph-theoretic condition on the diagram of the Cartesian product automaton $\mathcal{B}(\rho) \times \mathcal{B}(\rho)$. Now consider the non-transient part of the product automaton, i.e., the subgraph induced by all points that lie on a two-way infinite path. Since $\mathcal{B}(\rho)$ is transitive, the diagonal $\Delta = \{\, (x, x) \mid x \in \mathcal{B}(\rho) \,\}$ must lie in a single strongly connected component. Moreover, that strongly connected component must be Δ itself, and there cannot be any other non-trivial strongly connected components.

It follows from our previous remarks about the multiplicity of extensions of a word that the kernel automaton of $\mathcal{A}$ has a terminal strongly connected component that consist of sets of equal cardinality. The subautomaton induced by this strongly connected component is the *right Welch automaton* of ρ and we can define $W_{\mathsf{r}}(\rho)$ to be the cardinality of the state sets of this automaton. Analogously we can define the left Welch automaton by starting with reversal $\mathcal{A}^{\mathsf{op}}$ of $\mathcal{A}$; $W_{\mathsf{l}}(\rho)$ is defined analogously. The diagrams of these automata for $(4, 2)$ CA number 7230 can be seen in figure 4.

One can show that $W_{\mathsf{l}}(\rho)\, W_{\mathsf{m}}(\rho)\, W_{\mathsf{r}}(\rho)$ is k^{w-1}, as a consequence rule ρ is ζ-injective iff $W_{\mathsf{l}}(\rho)\, W_{\mathsf{r}}(\rho) = k^{w-1}$. Of course, this method requires calculation of the Welch automata and is inefficient, see section 3.3 for a better approach. On the other hand, paths in the Welch automata contain all the information needed to determine how long a finite word w needs to be in order for the unfolding of w to contain a bottleneck, a place where all the computations pass through the same node in $\mathcal{B}(\rho)$. But then we can effectively construct an inverse cellular automaton, essentially by mapping w to, say, the first symbol in the bottleneck; some slight optimizations are possible. For example, the $(4, 2)$ LCA number 3915 has a $(4, 2)$-inverse with rule number 11535.

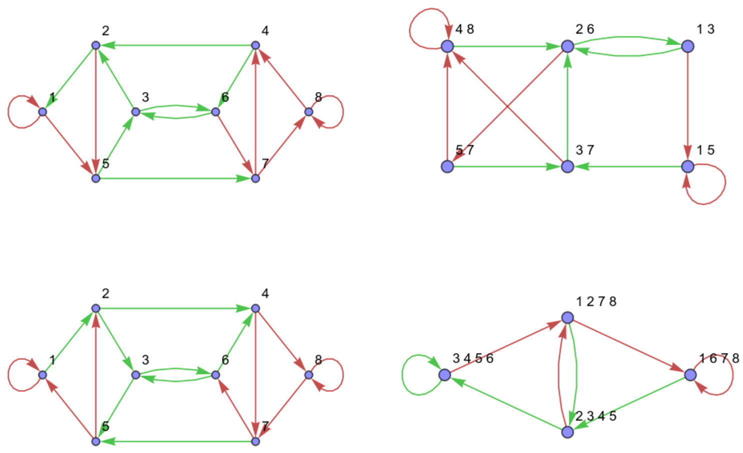

Fig. 4 The semiautomata and Welch automata associated with the $(4,2)$ CA number 7230.

The characterization of basic properties of the global map for ζ-words is fairly straightforward, but becomes more involved for ω-words: The existence of one-sided boundary conditions removes symmetry. For finite words the situation changes again since the answer now typically depends on the length of the words in question. The approach discussed in the next section deals with all these problems.

3 Decision Procedures and Automaticity

Decision algorithms for the local theory of linear cellular automata are based on first-order structures consisting of a space of words, together with a suitable global map as defined in section 2.2. We will refer to these structures as *phase space*. In this section we will outline the general approach to handling decision problems, but we point out that more information can be gleaned from the automata that arise during the execution of the algorithm, see section 3.3.

3.1 The First-Order Theory

The following classical result is the key for the decision method for the local theory of cellular automata. Suppose we have some relational first-order language $L = \mathcal{L}(R_1, \ldots, R_\ell)$ and a first-order structure $\mathfrak{A} = \langle A; \widehat{R}_1, \ldots, \widehat{R}_\ell \rangle$ where $A \subseteq \Sigma^*$ is a regular language and $\widehat{R}_i$ is a synchronous relation on A of the same arity as relation symbol R_i. We refer to such a structure as being *automatic* or *synchronous*. The goal is to determine whether a arbitrary prenex-normal form formula

$$\Phi = Q_1 x_1\, Q_2 x_2 \ldots Q_k x_k\, \phi(x_1, \ldots, x_k)$$

is valid over $\mathfrak{A}$. Here the Q_i denote existential or universal quantifiers and the matrix ϕ is quantifier-free; we always allow for equality.

Theorem 4 *The first-order theory of an automatic structure is decidable.*

This follows readily from standard results in automata theory, see [30, 29, 28, 53, 52]. We refer the reader to the references for technical details and will here only comment on the salient parts of the decision algorithm and then limit our discussion to the context of cellular automata. A few more details can also be found in section 3.3.1.

Given the matrix of the formula, we can effectively construct a k-track transducer $\mathcal{A}_\phi$ with the property that

$$\mathcal{L}(\mathcal{A}_\phi) = \{\, u_1{:}u_2{:}\ldots{:}u_k \in (\Sigma^k)^* \mid \mathfrak{A} \models \phi(u_1, \ldots, u_k)\,\}$$

where $u_1{:}u_2{:}\ldots{:}u_k$ denotes a word over the product alphabet Σ^k, u_i represents the ith track of the word. The machine $\mathcal{A}_\phi$ is built from the given transducers for the relations $\widehat{R}_i$ and equality, using standard Boolean operations. To determine the validity of Φ, we successively remove all the quantifiers, starting at the last quantifier Q_k and working to the left. Existential quantifiers are handled by simply removing the corresponding track. Universal quantifiers, on the other hand, are converted into an existential one plus two negations, $\forall \equiv \neg\exists\neg$. Note that determinization is usually needed to handle the first negation, but the subsequence removal of a track is likely to reintroduce nondeterminism, so the second negation again requires determinization. Once all the quantifiers have been removed, we obtain a machine $\mathcal{A}_\Psi$, really a unlabeled directed graph that we interpret as a machine over some one-symbol tally alphabet. Thus, the final query is a test for emptiness of the finite state machine $\mathcal{A}_\Phi$: we have $\mathfrak{A} \models \Phi$ if, and only if, $\mathcal{L}(\mathcal{A}_\Phi) \neq \emptyset$. Once the machine has been constructed, the corresponding test is linear time.

Focusing on cellular automata, the automatic structures in question take the following shape. There are three cases to consider depending on whether the words in question are finite, ω-infinite or ζ-infinite. The same general framework applies, but the automata in question require substantial modifications that we will comment on in section 3.2. The three kinds of structures are

$$\mathfrak{C}^*_\rho = \langle\, \Sigma^*, \rho^*\,\rangle \qquad \mathfrak{C}^\omega_\rho = \langle\, \Sigma^\omega, \rho^\omega\,\rangle \qquad \mathfrak{C}^\zeta_\rho = \left\langle\, \Sigma^\zeta, \rho^\zeta\,\right\rangle$$

As already mentioned, for technical reasons, it is customary in this context to think of the map as a binary relation which we will denote by $\rightarrow$. Clearly, any function can be expressed in terms of the corresponding relation, but the latter approach makes it somewhat easier to handle logical concepts. For example, while it is convenient to allow terms of the form $f(f(x)) = y$, there is a hidden existential quantifier that becomes visible in the relational setting: $\exists\, z\, (x \to z \wedge z \to y)$. In the case of finite words, it is also of interest to study the finite substructures $\mathfrak{C}_\rho^{(n)}$ obtained by restriction to Σ^n. Hence, the language used to describe properties of these structures is the first-order language $\mathcal{L}(\to)$ with equality, but see section 3.3.2 for natural generalizations.

To organize the conversion algorithm from formula to automaton is convenient to assume that the matrix is given in disjunctive normal form, i.e., a disjunction of conjunctions of basic atomic formulae $x \to y$ or $x = y$, plus their negations. Observe that the negation $\neg(x \to y)$ is equivalent to $\exists\, z\, (x \to z \wedge z \neq y)$ and can thus be eliminated, at the cost of more existential quantifiers before the matrix. Unnegated equalities $x = y$ can often be eliminated in the construction of the automaton $\mathcal{A}_\phi$ by identifying tracks related to $\to$-terms in the same disjunct. At any rate, in each disjunct we are left with conjuncts of the form $x \to y$, $x = y$ or $x \neq y$ which can be handled by a standard product machine construction. Assembling the disjuncts is easy as long as we use nondeterminism, the disjoint sum of the components works fine.

Since the global map is length-preserving, the discussion for finite words is usually rephrased slightly. As already mentioned, individual finite structures $\mathfrak{C}_\rho^{(n)}$ per se are of little interest, instead one is trying to determine the so-called *spectrum* of the property in question. Given any first order sentence Φ, we define its spectrum to be

$$\mathsf{spec}(\Phi) = \{\, n \in \mathbb{N}_+ \mid \mathfrak{C}_\rho^{(n)} \models \Phi \,\}$$

Given our convention that removal of the quantifiers produces an automaton over a tally alphabet, $\mathcal{A}_\Phi$ recognizes the spectrum of Φ, where the natural numbers are written in unary. Minimization of $\mathcal{A}_\Phi$ produces a lasso-shaped machine, so that the spectrum is always ultimately periodic.

Theorem 5 *Any sentence in the logic $\mathcal{L}(\to)$ has ultimately periodic spectrum. A corresponding representation is effectively computable.*

In other words, we can effectively compute finite sets $A, B \subseteq \mathbb{N}$ such that the spectrum has the form $A \cup (B + t + p\mathbb{N})$. Even though the individual structures $\mathfrak{C}_\rho^{(n)}$ are all trivially decidable even for stronger logics such as second order or transitive closure logic, there is no hope for any sort of analogue to the last theorem. For example, it is Π_1-complete to determine uniformly in n whether $\mathfrak{C}_\rho^{(n)}$ contains a unique fixed point that is reachable from everywhere [48].

As an example, consider the existence of a proper 3-cycle in $\mathfrak{C}_\rho^{(n)}$. The elementary cellular automata with open global map are rules number 60, 90, 102, 105, 150, 153, 165, 195. The spectra of the 3-cycle property for all these rules are periodic and the periods can be found in the table below. However, we can extract more information from the automata that appear during the construction. In particular we can compute the growth functions of the automata that recognizes all the triples that form a proper 3-cycle. From there one can determine the actual number of 3-cycles which numbers may be constant, or change periodically themselves.

period	rules	cycle count
3	60, 102, 153, 195	1
5	90, 165	5, 5, 20
5	150	10, 20
20	105	20

More generally, counting the number of occurrences of particular subgraphs in phase space can be handled in a similar manner, though it may be very difficult to handle the growth functions involved. For example, rule 73 has the very simple spectrum $7+\mathbb{N}$, but the first few non-zero cycle counts are $7, 16, 18, 20, 44, 68, 65, 119, 250, 368, \ldots$ and appear to have no reasonable description. Algorithmic details can be found in section 3.3.

3.2 Infinite Words

The automata-based decision algorithm for automatic structures applies, as a matter of principle, to phase spaces on infinite words. The only difference is that we now need to employ ω-transducers or ζ-transducers, depending on the space in question. We emphasize the qualifier "in principle," since, as a practical matter, the algorithms manipulating these machines often have substantially worse performance. Of course, this is not a new phenomenon, state space explosion has been the bane of automata-based model checking methods for quite some time [11, 10].

3.2.1 ω-Automaticity

We first consider one-sided shift spaces where we can think of the global map as being given by a synchronous ω-transducer. For simplicity we will only consider 0-boundary conditions and odd widths, no substantial changes occur in the general case. The corresponding synchronous transducer is easily obtained from the standard de Bruijn transducer $\mathcal{T}(\rho)$ by restricting the initial states to be of the form $0^r\Sigma^r$ where r is the radius of the local map. The acceptance condition for $\mathcal{T}(\rho)$ is straightforward: all states are final, and we simply require an infinite path starting at one of the initial states. By construction, $\mathcal{T}(\rho)$ is deterministic, so the one-step relation $\to$ is deterministic ω-regular. The same holds for compound relations built from $\to$ using only conjunctions and disjunctions; alas, negation cannot be handled this way. Similar comments apply to the basic acceptor $\mathcal{B}(\rho)$, except that this automaton fails to be deterministic in general; it always is a multi-entry automaton.

Finding a reasonable definition of regularity for ω-languages in general requires substantially more effort. As suggested by Büchi, one should keep a finite, nondeterministic transition system $\langle Q, \Sigma, \tau \rangle$ in place and modify the standard acceptance condition as follows. Suppose we have a collection of initial and final states $I, F \subseteq Q$ as before in the finite word case. Given an ω-word X, a *run* of $\mathcal{A}$ on X is an infinite sequence $\pi = (p_i)_{i \in \mathbb{Z}}$ of states in Q such that $\tau(p_i, x_i, p_{i+1})$ for all $i \in \mathbb{N}$. Define the *recurrent set* of the run to be $\mathsf{rec}(\pi) = \{ p \in Q \mid \exists^\infty i \in \mathbb{N} (p_i = p) \}$, the collection of all states the appear infinitely often in the run labeled X. Then $\mathcal{A}$ is said to accept X if there is some run π such that $\mathsf{rec}(\pi) \cap F \neq \emptyset$. A language $W \subseteq \Sigma^\omega$ is *ω-regular* if it is accepted by an Büchi automaton. From the definition of a Büchi automaton it follows immediately that there is a close link between regular languages and ω-regular languages: the latter can always be characterized as the collection of ω-languages of the form

$$W = \bigcup_{i \leq n} U_i V_i^\omega$$

where the $U_i, V_i \subseteq \Sigma^+$ are all regular. The notation V^ω indicates all ω-words that can be obtained by concatenating together infinitely many non-empty words from V.

There are several essential differences between ordinary word automata and Büchi automata. A major application of word automata is to use fast acceptance testing for pattern matching. On the other hand, acceptance testing of ω-words is algorithmically meaningless unless the word in question has a finitary description. For ultimately period words acceptance is easily decidable, but becomes undecidable for computable words, and even primitive recursive ones. Instead, the key decision problem for Büchi automata in an algorithmic context is whether they accept any string, the so-called Emptiness problem. The problem is solvable in time linear the size of the given automaton. Another major issue is the lack of direct analogue to Rabin-Scott determinization: some ω-regular languages can only be accepted by nondeterministic Büchi automata, a typical example being the language $\{a, b\}\, a^\omega$ of all words containing on finitely many as. One refers to the languages that can be recognized by a deterministic Büchi automaton as a *deterministic* ω-regular language. These turn out to be exactly the *adherences* of ordinary regular languages: they consist of ω-words that have infinitely many finite prefixes in a given regular language. Note the ω-languages $\rho^\omega(\Sigma^\omega)$ naturally are the adherences of the regular language of all finite prefixes of ω-words in the range of ρ^ω. Deterministic ω-regular languages are closed under union and intersection, but not under complementation. The proof that ω-regular languages in general form a Boolean algebra is quite difficult. In fact, the first practical complementation algorithm was developed by Safra in [42], almost three decades after the introduction of Büchi automata.

The definition carries over to synchronous ω-transductions. Ignoring algorithmic complexity issues for the time being, we can define an *ω-automatic* structure to be a first-order structure $\mathfrak{A} = \langle A; R_1, \ldots, R_n \rangle$ where $A \subseteq \Sigma^{\mathbb{N}}$ is an ω-regular set and all the relations $R_i \subseteq (\Sigma^\omega)^{k_i}$ are ω-regular transductions.

Theorem 6 *The first-order theory of ω-automatic structures is decidable.*

On the face of it, it may seem surprising that finite state machines should be sufficient to handle first-order logic over uncountable spaces like Σ^ω. Ultimately, the argument goes through, because first-order logic is too weak to express properties that cannot already be determined in a far less complicated substructure. Specifically, given a first-order structure $\mathfrak{A}$, a substructure $\mathfrak{B}$ is *elementary* if for any first-order sentence $\phi(\mathbf{b})$ with parameters $\mathbf{b}$ in $\mathfrak{B}$ is valid in $\mathfrak{A}$ iff it is valid in $\mathfrak{B}$. In other words, from a strictly first-order perspective, there is no difference between the two structures. In particular the automatic structures $\mathfrak{C}_\rho^\omega$ have an elementary, countable substructure that ultimately serves as the ambient background for decision algorithm.

Theorem 7 *Ultimately periodic ω-words form an elementary substructure of $\mathfrak{C}_\rho^\omega$.*

To see why the theorem holds, consider ultimately periodic ω-words. In order to apply the Tarski-Vaught test, see [9], it suffices to consider a first-order formula $\exists x\, \varphi(x, \mathbf{b})$ where all the parameters are ultimately periodic. For simplicity, assume there is only a single parameter b, and let $\mathfrak{C}_\rho^\omega \models \exists x\, \varphi(x, b)$.

By automaticity, we may assume we have a two-track Büchi automaton $\mathcal{A}$ that accepts some word $x{:}b$. By the definition of acceptance for Büchi automata, there is some initial state p, a final state q and a strongly connected component C, $q \in C$, in the automaton such that an accepting computation of $\mathcal{A}$ starts at p, ultimately stays in C and touches q infinitely often. Since b is ultimately periodic, say, with primitive period $u \in \Sigma^*$, there must be a conjugate of some power u^t of u that labels the second track in $\mathcal{A}$ in a cycle anchored at q. We can read off a corresponding label in the first track to produce an ultimately periodic witness x that demonstrates that $\exists x\, \varphi(x, b$ also holds over the substructure.

The last result fails for the type of "finite" configurations $u0^\omega$ often employed in computability arguments, even if we insist that the rule is quiescent in the sense that 0^ω is a fixed point. On the other hand, it is noteworthy that the theorem does hold for the substructure of computable words. In this case, the parameter b in the existential formula in the proof is a computable function and there is no reason for for the computation to ever settle down on a fixed cycle in a strongly connected component. However, one can use a more involved argument to show that it is still possible to find a computable witness, see [50]. Alas, the witness can no longer be effectively determined from an index for the parameter.

Returning to the automata associated in particular with cellular automata, the basic machines that check for weak k-cycles are still rudimentary. The qualifier "weak" here is meant to indicate that no inequality tests are involved, so that k may not be minimal. Including these inequality conditions requires the separation of initial and final states, and uses multiple copies of the weak machines arranged along a Boolean lattice. This structure is made necessary by the fact that inequality conditions between different pairs of tracks can be satisfied in many different sequences. Figure 5 shows the structure of the machine for testing proper 4-cycles.

Similar comments apply for paths of length k or the existence of k predecessors. The machines all have a fairly straightforward structure and the acceptance condition

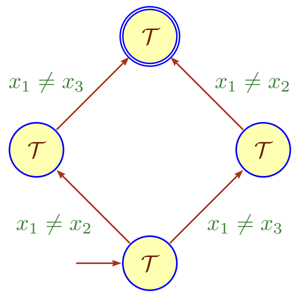

Fig. 5 A 4-track transducer checking for proper 4-cycles. The initial states are located in the bottom copy of $\mathcal{T}$, and the final states, in the top.

is not substantially different from the finite case: initial state play the same role as before, and for final states we have to make sure that at least one is touched infinitely often. For example, it is easy to construct ultimately periodic witnesses, in case they exist at all. Ultimately it is the handling of negation in the removal of universal quantifiers that produce truly complicated machines.

3.2.2 ζ-Automaticity

To handle classical, two-sided shift spaces in a similar manner one needs to develop a notion of regularity for ζ-languages. Naturally one would like to exploit the machinery of the ω-automata from the last section to this end. To handle words that extend infinitely far to the left, first define *co-ω-words* as maps $X : \mathbb{Z}_- \longrightarrow \Sigma$, where $\mathbb{Z}_-$ denotes the negative integers. These words can be concatenated with ω-words Y in the obvious fashion to produce ζ-words $XY : \mathbb{Z} \longrightarrow \Sigma$. Any co-$\omega$-word X can be converted into an ω-word X^{op} by setting $X^{\mathsf{op}} : \mathbb{N} \longrightarrow \Sigma$ by $X^{\mathsf{op}}(i) = X(-i-1)$; the same approach works in the opposite direction and we have $(X^{\mathsf{op}})^{\mathsf{op}} = X$.

Now suppose we have a transition system together with initial and final state sets, $\mathcal{A} = \langle Q, \Sigma, \tau; I, F \rangle$. In analogy to the ω-case, define the *negatively recurrent set* of a run to be $\mathsf{rec}_-(\pi) = \{ p \in Q \mid \exists^\infty i \in \mathbb{N} \, (p_{-i} = p) \}$. We can then define a ζ-Büchi automaton to be the same data structure as a Büchi automaton, but with the acceptance condition modified to require $\mathsf{rec}_-(\pi) \cap I \neq \emptyset$ as well as $\mathsf{rec}(\pi) \cap F \neq \emptyset$. A language is *$\zeta$-regular* if it is accepted by a ζ-Büchi automaton; in a similar manner we can define *synchronous ζ-transductions*. One can check that this definition yields a Boolean algebra of languages and that decomposition results similar to the ω-case obtain: a language $W \subseteq \Sigma^\zeta$ is ζ-regular if, and only if, it is of the form

$$\begin{aligned} W &= \bigcup (X_i)^\omega \, Y_i \, Z_i^\omega \\ &= \bigcup U_i^{\mathsf{op}} \, V_i \end{aligned}$$

where the unions are finite, X_i, Y_i and Z_i are regular subsets of Σ^+, and U_i, V_i are ω-regular languages.

It is clear from the definition that we can decompose a ζ-Büchi automaton into a collection of pairs of Büchi automata of the form $\mathcal{A}_{p,-} = \langle Q, \Sigma, \tau; I, \{p\} \rangle$ and $\mathcal{A}_{p,+} = \langle Q, \Sigma, \tau; \{p\}, F \rangle$ where p ranges over Q. We may assume that all states in ζ-Büchi automaton are non-transient and lie on paths from nontrivial strongly connected components containing initial and final states, respectively. Also, it suffices to choose states p so that every computation passes through at least one such point.

The basic acceptor $\mathcal{B}(\rho)$ and transducer $\mathcal{T}(\rho)$ in figure 2 fit this framework simply by considering all states to be initial and final, the usual convention for semiautomata. In this case, acceptance simply means there is a two-way infinite run with the appropriate label and shift invariance holds. ζ-Büchi automata make it possible to generalize the decidability result from the ω-case, but note that, in the context of shift spaces, one is actually interested in words modulo a shift, there is no artificial coordinate system as in the split-and-glue method just outlined. In other words, our basic machines accept X if, and only if, they accept $\sigma(X)$. This naturally leads to the question of whether shift-invariant ζ-languages can be handled by special ζ-automata. To shed light on this question, define a ζ-Büchi automaton to be *ζ-unambiguous* if the automaton has at most one accepting run on every input. Figure 6 shows an ambiguous as well as an unambiguous transducer for the property of almost-equality, see section 3.3.2 for an application. Unambiguous automata can be considered to be a first step towards deterministic automata; in the context of ζ-Büchi automata they provide a weaker version of a full determinization algorithm. It is shown in [39] that for shift-invariant ζ-regular languages there always is a ζ-unambiguous automaton that accepts the language.

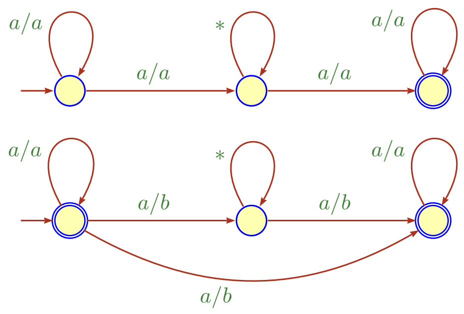

Fig. 6 A ζ-transducer automaton that recognizes all the almost-equal relation on Σ^ζ and its unambiguous counterpart. Here $a \neq b$ are supposed to be arbitrary but distinct letters in Σ and $*$ denotes all possible labels.

Ignoring all questions of computational complexity, the fundamental decidability result carries over from ω-automatic structures to ζ-automatic ones.

Theorem 8 *The first-order theory of ζ-automatic structures is decidable.*

One might suspect that, as before for ω-automaticity, the use of finite state machines in the decision algorithm establishes a simple limitation on the expressiveness of the logic that can be explained in terms of a suitable elementary substructure of the uncountable space. In this case, we wind up with a class of ζ-words that are known as *backgrounds* or *ultimately periodic*. These words take the form ${}^{\omega}u\,v\,w^{\omega}$ where $u, v, w \in \Sigma^*$. Backgrounds are a natural generalization of both locally finite words, i.e., words of the form ${}^{\omega}0\,v\,0^{\omega}$ and periodic words ${}^{\omega}u^{\omega}$.

Theorem 9 *Ultimately periodic ζ-words form an elementary substructure of $\mathfrak{C}_{\rho}^{\zeta}$.*

Recall Cook's proof of the computational universality of elementary cellular automaton number 110, see [12]. The construction relies on the orbit of backgrounds rather than finite configurations as in standard computability arguments. In fact, backgrounds are used to simulate cyclic tag systems using rule 110. By contrast, the reachability problem for ECA 110 is trivially decidable for finite and one-way infinite words, but it is r.e.-complete for backgrounds. In light of the substructure theorem, the choice of configurations in the proof seems fairly natural.

3.3 Algorithmic Issues

The choice of first-order logic as a query language avoids all issues of undecidability, but there still is a price to pay in terms of computational complexity. Indeed, feasibility problems become dominant in the infinite case and in particular for ζ-automata. In this section we comment on various methods to combat algorithmic problems and obtain actual results at least under limited circumstances.

3.3.1 Machine Constructions

The basic transducers $\mathcal{T}(\rho)$ have size linear in the size of the rule table for ρ, and we have obvious machines for equality and inequality. All the automata constructions involved in the algorithms are justified by the standard closure properties of regular languages: Boolean operations handle logical conjunction, disjunction and negation; closure under homomorphisms and inverse homomorphisms is used for embeddings and projections that address the need to deal with product alphabets of the form Σ^k, corresponding to formulae with k free variables. The exponential size of these alphabets requires particular attention, it is often mandatory to find ways to rearrange the given formula into logically equivalent forms, typically not in prenex-normal form, that make it possible to keep the number of tracks reasonably small. As to the Boolean operations, union can be handled in linear time using nondeterminism. Intersections are typically quadratic and rely on product machine constructions.

Lastly, negation in general admits no polynomial time bounds; in particular for infinite words the cost may even be super-exponential.

Two key components in the construction of transducers representing compound logical formulae are *embeddings* and *projections*. More precisely, the embedding operation takes the form $\mathsf{emb}_{\mathbf{t}}^{(n)} : m\text{-track} \longrightarrow n\text{-track}$ where $m \leq n$, $\mathbf{t} = t_1, \ldots, t_m$, $t_i \in [n]$, all distinct. Thus $\mathsf{emb}_{\mathbf{t}}^{(n)}(\mathcal{A}^{(m)}) = \mathcal{B}^{(n)}$ means that track i of $\mathcal{A}^{(m)}$ is identified with track t_i in $\mathcal{B}^{(n)}$. The other tracks are free, i.e., all possible transitions are included. While this operation does not affect the state set, it can cause potentially very large alphabets and, correspondingly, large numbers of transitions in the embedded automaton.

The natural dual to embeddings are projections $\mathsf{prj}_{\mathbf{t}}^{(n)} : n\text{-track} \longrightarrow m\text{-track}$ where $\mathbf{t} = t_1, \ldots, t_{n'}$, $n' \leq n$, $t_i \in [n]$, all distinct, $m = n - n'$. Here $\mathsf{prj}_{\mathbf{t}}^{(n)}(\mathcal{A}^{(n)}) = \mathcal{B}^{(m)}$ means that, for all transitions in $\mathcal{A}^{(n)}$, the tracks t_i of the transition labels have been erased, producing $\mathcal{B}^m$. Again, the state set is unaffected but the number of transitions shrinks. The special case $n = n'$ is important. As already pointed out, the removal of all tracks results in an unlabeled digraph with a collection of source and target nodes. It is convenient to adopt the convention that the edges are labeled by a special tally symbol, so that the language of the resulting acceptor can naturally be considered as a subset of $\mathbb{N}$, written in unary. The formula is valid if, and only if, this set is non-empty. As a practical matter, if the leading quantifier in the original formula is universal, one always avoids the standard conversion to an existential quantifier plus two negations and simply negates the question, leaving an existential question and a single negation.

In all three scenarios, the basic transducers for the one-step relation or equality are easily handled. Using embeddings and product constructions one can then build up machines that handle the weak versions of k-cycles, k-paths and k-predecessors. Introducing inequality constraints is a first step towards more complicated machines, and the removal of universal quantifiers exacerbates the situation. As an example, consider the question of the existence of proper 3-cycles in phase space. The straightforward way to express the assertion that such a cycle exists in $\mathcal{L}(\to)$ is the following existential formula.

$$\exists\, x, y, z \left(x \to y \wedge y \to z \wedge z \to x \wedge x \neq y \wedge x \neq z \wedge y \neq z\right)$$

A moment's thought reveals that this can be rewritten as

$$\exists\, x, y, z \left(x \to y \wedge y \to z \wedge z \to x \wedge x \neq y\right)$$

since $\to$ is functional. As a general principle, prenex normal form makes it easy to describe the algorithm but is computationally inefficient. It is preferable to use equivalent formulae that make it possible to project away tracks corresponding to existential quantifiers as early as possible. In this case, it is preferable to rewrite the last formula as

$$\exists\, x, y\ \big(x \to y \wedge \exists\, z\, (y \to z \wedge z \to x \wedge x \neq y)\big)$$

to take a advantage of the projection. The construction of the machine then takes the form

$$\mathcal{A}' = \mathsf{prj}_3^{(3)}\big(\mathsf{emb}_{1,2}^{(3)}(\mathcal{T}_\rho) \times \mathsf{emb}_{2,3}^{(3)}(\mathcal{T}_\rho) \times \mathsf{emb}_{3,1}^{(3)}(\mathcal{T}_\rho)\big)$$
$$\mathcal{A} = \mathsf{prj}_{1,2}^{(0)}\big(\mathsf{prj}_3^{(3)}\big(\mathcal{A}'\big) \times \mathcal{U}^{(2)}\big)$$

where $\mathcal{U}^{(2)}$ is a two-track machine checking for inequality.

Another example where the use of prenex-normal form is to be avoided is iteration of the global map. In general, given any synchronous, length-preserving map f represented by a transducer $\mathcal{A}$, The direct construction of a transducer $\mathcal{B}$ that represents the k-fold iterate of the global map would rely on a product of machines obtained by k+1-track embeddings of $\mathcal{T}_\rho$, followed by projections. However, after initializing $\mathcal{A} = \mathsf{emb}_{1,2}^{(3)}(\mathcal{T}_\rho)$ and $\mathcal{T}_{23} = \mathsf{emb}_{2,3}^{(3)}(\mathcal{T}_\rho)$ we can simply repeat the construction

$$\mathcal{A} = \mathsf{emb}_{1,2}^{(3)}\big(\mathsf{prj}_2^{(3)}\big(\mathcal{A} \times \mathcal{T}_{23}\big)\big)$$

k times and in the end return $\mathsf{prj}_3^{(3)}\big(\mathcal{A}\big)$. In all such constructions, minimization should be applied regularly to keep state complexities low.

While the general decision algorithm ultimately determines a yes/no answer, it is often useful to analyze the machines that are constructed during the execution of the algorithm directly. In fact, one can often avoid an explicit construction of the decision automaton and instead argue about machines that are logically related to the ones that appear in the decision algorithm. As an example, consider questions about in-degrees for finite phase spaces. We certainly can express the assertion that every node has indegree 0 or 2 as a first-order sentence

$$\forall\, x, y\, \exists\, u, v\, \forall\, z\ \big(y \to x \Rightarrow (u, v \to x \wedge u \neq v) \wedge (z \to x \Rightarrow z = u \vee z = v)\big)$$

The alternating quantifiers and the length of the matrix of the formula would produce rather unwieldy automata. Instead, we can build very simple machines and reason about the growth functions of their languages. For example, a machine that accepts all pairs of words that are predecessors of the same word looks like

$$\mathcal{A} = \mathsf{prj}_3^{(3)}\big(\mathsf{emb}_{1,3}^{(3)}(\mathcal{T}_\rho) \times \mathsf{emb}_{2,3}^{(3)}(\mathcal{T}_\rho)\big)$$

A priori, this machine only determines a point in phase space of in-degree at least one. However, after determinizing and minimizing $\mathcal{A}$, we can easily compute the generating function representing the growth function of $\mathcal{A}$: this comes down to solving a linear system of equations over $\mathbb{Z}[z]$. In the case of elementary cellular automaton number 90 with cyclic boundary conditions, this function is $\frac{4(1+4z)}{1-4z^2}$ and from there the number of pairs can be seen to be $2^n(3 + (-1)^n)$. Similarly we can determine the number of nodes with positive indegree to be $2^{n-3}(3 - (-1)^n)$

and the number of quadruples associated with nodes of degree at least 4 to be $2^{n+2}(9 + 7(-1)^n)$. A little counting argument then shows that the degree statement from above holds in $\mathfrak{C}_{90}^{(n)}$ exactly for all odd n, and the corresponding statement for in-degree 4 holds for all even n.

If the decision algorithm is used directly to obtain a yes/no answer, we wind up with the Emptiness problem for ζ-Büchi automata, analogous to the finite and one-way scenario.

Proposition 1 *The Emptiness problem for ζ-Büchi automata is solvable in time linear in the size of the automaton.*

To see why, note that we can use, say, Tarjan's algorithm for strongly connected components to make sure that the automaton is non-transient. Next, we can check that there is at least one state p on a path from some nontrivial strongly connected component C_- to another C_+ so that C_- contains an initial state and C_+ contains a final state. All of this can be handled with standard graph algorithms in linear time.

Of course, the real algorithmic challenge occurs in the construction of the ζ-Büchi automaton in the first place. In fact, even for plain Büchi automata that are used in decision procedures for, say, Presburger arithmetic, algorithmic issues are dominant. As a consequence, the literature offers a number of alternative kinds of ω-automata: Rabin automata, Muller automata, Streett automata and so on, see Kupferman's chapter in [10]. The key obstruction is expressed in the next proposition.

Proposition 2 *Determinization of ζ-Büchi automata is super-exponential time in the worst case.*

More precisely, there are examples of ω-regular languages with an n state Büchi automaton whose complement requires a Büchi automaton of size $(n-1)!$, much worse that the plain exponential blowup in the finite word case. At any rate, the determinization of a Büchi automaton produces a Rabin or a Muller automaton, that can be converted back into an equivalent Büchi automaton. Recall, though, that the determinization of a ζ-Büchi automaton requires its decomposition into linearly many pairs of Büchi automata, each of which needs to be determinized separately. As a result, one should expect successful determinization only for reasonably small machines. Using unambiguous machines as introduced in section 3.2.2 can also help control the size of machines, at least in some cases. Still, the problem of state explosion imposes substantial bounds on actual computations.

3.3.2 Automatic Extensions

Another way to avoid unduly large machines is to change the underlying language itself. To wit, we can add synchronous relations to the language $\mathcal{L}(\rightarrow, R_1, \ldots, R_\ell)$ and accordingly provide suitable basic transducers that recognize these relations.

Since the structure $\langle W; \to, \widehat{R}_1, \dots, \widehat{R}_\ell \rangle$ is still automatic, the algorithms carry over directly. Automatic extensions of this kind are particularly useful when the relations fail to be first-order definable over the basic structure $\langle W; \to \rangle$, but they also may be algorithmically advantageous if the predicate in question is already definable, simply because a there is no need to recompute the corresponding machines from scratch in each run of the algorithm. Perhaps the most important examples of such extensions are related to modified notions of equality on infinite words. For ζ-words X and Y, define $X =_L Y$ to mean that $\exists n \forall i \geq n \, (x_{-i} = y_{-i})$. Similarly, $X =_R Y$ means that $\exists n \forall i \geq n \, (x_i = y_i)$ and $X \overset{*}{=} Y$ if $X =_L Y \wedge Y =_L X$. Thus $X \overset{*}{=} Y$ expresses the fact that X are almost equal, they differ in at most finitely many places. These predicates are easily ζ-automatic, for example, figure 6 shows a 2-track ζ-Büchi automaton recognizing the almost-equal relation. Hence we can work in the extended structure $\mathfrak{C}_\rho = \langle \Sigma^{\mathbb{Z}}, \to, \overset{*}{=}, =_L, =_R \rangle$.

It was shown by Hedlund that the global map over Σ^ζ is surjective if, and only if, the following holds:

$$\forall x, y, z \left(x \to z \wedge y \to z \wedge x \overset{*}{=} y \Rightarrow x = y \right)$$

In a similar vein, openness of the global map in the topological sense is equivalent to the universal formula

$$\forall x, y, z \left(x \to z \wedge y \to z \wedge (x =_L y \vee x =_R y) \Rightarrow x = y \right)$$

Hence, in the presence of our additional predicates, surjectivity is expressed by a Π_1 statement, as opposed to the usual Π_2 statement. Negation produces a Σ_1 formula, the ideal input for the decision algorithm. Openness is even more complicated to express directly; again by Hedlund, it is equivalent to the assertion that the global may is k-to-one for some fixed k. Since there are only boundedly many choices for k there is a first-order formula for openness, but it is hopelessly large. Another interesting extension is the shift operator (except in fixed-boundary condition scenario). For example, consider the additive elementary cellular automata number 90 and 150. Both are open and non-injective over $\mathbf{2}^\zeta$ and their kernel of dimension is 2. Their non-injectivity spectrum is $\mathbb{N}$ and $3\mathbb{N}$, respectively. In the presence of a shift predicate we can further express the assertion that there are two distinct, non-constant predecessors with spatial period 2; this assertion produces spectra $4\mathbb{N}$ and $3\mathbb{N}$. Additional distinctions become possible if we enlarge the vocabulary by a summation operator, again in relational form $s(x, y, z)$ iff $x + y = z$.

It is unclear how far the extension mechanism can be pushed. As a consequence of assorted undecidability results it is clearly impossible in general to add a synchronous reachability or orbit relation to our structures. Unfortunately, the orbit relation is rational only in trivial cases, even a plain shift produces an irrational orbit relation. By contrast, in the context of invertible, length-preserving transducers used in group theory, there are some interesting examples of rational orbit relations [4]. This is somewhat surprising, since the transductions in the context of group theory carry state whereas cellular automata operate in a uniform fashion. It seems that the ability of cellular automata to propagate information in a spatial way causes insurmountable obstacles.

With a few towards feasible algorithms it is often useful to analyzed the critical properties of the machines in question “by hand,” in order to find computational shortcuts. As an example, the tests for surjectivity, openness and injective that are based on the direct application of the general decision algorithm are far from optimal. However, a moment’s thought reveals the only way the whole surjectivity implication can hold is when the strongly connected component containing the diagonal in the square of $\mathcal{B}(\rho)$ consists of the diagonal alone, the algorithm proposed in [50] that runs in time quadratic in the size of the rule table. Similar comments apply to the test for openess and injectivity.

4 Conclusion

Interpreting linear cellular automata as synchronous transducers over various sets of words makes it possible to apply the fairly well-developed theory of regular languages and rational transductions to the study of these cellular automata. In particular we obtain a general decision algorithm for various local properties of the phase spaces determined by the global maps. In light of the plethora of undecidability results around cellular automata, it is clear that the general method is necessarily limited in scope. On the other hand, it does provide a simple, algorithmically useful framework to discuss a variety of properties of cellular automata. Beyond outright undecidability there is also the question of efficiency and it is currently unclear exactly how far the methods carry. As a consequence of negation, the running time of the general algorithm fails to be elementary, making it unlikely that the method can be applied to complicated assertions. On the positive side, it is conceivable that methods developed in model checking to avoid determinization could be brought to bear on some instances of our problem, see [31]. Similarly, the use of Boolean decision diagrams, another staple of model checking algorithms, may help to deal with larger machines and thus enlarge the practical reach of the decision algorithm.

We can define the ζ-theory of a local rule ρ to be the collection of all first-order sentences in $\mathcal{L}(\rightarrow)$ that are valid over the phase space Σ^{ζ} define by the global map defined by ρ^{ζ}. It seems to be quite a challenge to determine all ζ-theories even for a rather limited class such as elementary cellular automata. In particular we do not know how many distinct theories there are. More generally, consider an arbitrary local rule ρ and use a formal parameter $t = *, \omega, \zeta$ to indicate the type of phase space under consideration: Finite, ω-infinite or ζ-infinite, with local rule ρ. Define the theory $\mathsf{Th}(\rho, t)$ to be the collection of all sentences in $\mathcal{L}(\rightarrow)$ valid over the corresponding state space. To be of interest in the finite case, it should be understood that one wants to associate each sentence with its spectrum rather than a plain true/false value as in the infinite scenario. There are two natural ways to organize the task of analyzing these theories. The first is to fix t and study various local rules ρ. Alternatively, we can fix the local rule ρ and study the relationship between $\mathsf{Th}(\rho, *)$, $\mathsf{Th}(\rho, \omega)$ and $\mathsf{Th}(\rho, \zeta)$. For example, any ω-surjective rule is always ζ-surjective, but the converse is false in general. On the other hand, a rule with infinite surjectivity spectrum is always ζ-surjective; an infinite surjectivity spectrum is implied by ζ-injectivity, the converse fails to hold in general. Lastly, ζ-injectivity implies an infinite surjectivity spectrum. On the other hand, ζ-injectivity does not imply ω-surjectivity; in fact, a rule is ω-surjective iff all words have multiplicity k^{w-2} in the appropriate automaton. A major challenge here is to determine whether a classification of cellular automata based on their first-order theories is decidable: Given two (w, k) cellular automata ρ and τ, is $\mathsf{Th}(\rho, t) = \mathsf{Th}(\tau, t)$? For example, elementary cellular automata 160 and 250, corresponding to logical conjunction and disjunction of the two neighbors, have the same theory; in fact, the two shift spaces are isomorphic. Since the theories themselves are decidable, the classification problem is no worse than Π_1, but decidability is far from clear.

Ultimately we would like to understand whether there is any reasonable taxonomy of these theories in general. Any such classification would refine the usual surjective, open, injective hierarchy, but it is far from obvious what shape this hierarchy might take. Beyond the plain theories using only the global map relation, we can use the extension mechanism discussed in the last section to obtain more fine-grained distinctions in some language $\mathcal{L}(\to, R_1, \ldots, R_\ell)$ where the R_i are synchronous but not first-order definable in the plain language. While these questions should be expected to be fairly difficult in general, it is conceivable that they become manageable for restricted types of cellular automaton, such as those given by linear or affine rules over some finite field. In a similar vein, one can ask if there are any interesting types of cellular automata for which a stronger logical framework still remains decidable. It seems that if one were to try to handle transitive closure logic the answer is no, only trivial local rules could be handled. Alas, we have no proof for this assertion.

Pushing beyond the limitations of automata-based decision method, there is the question to which degree a formal verification of properties of cellular automata is feasible. For example, on might consider the use of a theorem prover or SMT solver, given some plausible axiomatization of our domain. A perfect test case for any such framework is Cook's theorem on the universality of elementary cellular automaton number 110. The arguments there are in a sense local, since the evolution of configurations required to simulate 2-tag systems is spacially periodic. The required backgrounds of the form ${}^{\omega}u\, w\, v^{\omega}$ have on obvious finitary description and might well be amenable to formal methods. On the other hand, the sheer size of the configurations may make formal correctness arguments too challenging for currently available methods. Needless to say, the automata-theoretic approach discussed here is entirely insufficient for this sort of task. In a different, but closely related context, it would be interesting to construct a similar kind of formal proof for Smith's argument that establishes universality of the well-known $(2, 3)$ Turing machine suggested by Wolfram [46].

A Mathematica package the implements most of the algorithms discussed here can be found at https://resources.wolframcloud.com/PacletRepository/ under the name

Automata. This implementation is meant as a proof-a-concept, a lower level language is needed to handle instances of compelling size.

References

1. J. T. Baldwin and S. Shelah. On the classifiability of cellular automata. *Theoretical Computer Science*, 230(1-2):117–129, 2000.
2. J. Barwise and J. Keisler, editors. *Handbook of Mathematical Logic*. Number 90 in Studies in Logic and Found. of Math. Elsevier, Amsterdam, 1977.
3. D. Beauquier. Minimal automaton for a factorial, transitive, rational language. *Theoretical Computer Science*, 67:65–73, 1989.
4. T. Becker and K. Sutner. Orbits of abelian automaton groups. In C. Martin-Vide, A. Okhotin, and D. Shapira, editors, *LATA St. Petersburg*, volume 11417 of *SLNCS*, 2019.
5. J. Berstel. Transductions and context-free languages. http://www-igm.univ-mlv.fr/~berstel/LivreTransductions/LivreTransductions.html, 2009.
6. J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik and Grundl. Math.*, 6:66–92, 1960.
7. J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Logic, Methodology and Philosophy of Science*, Proc. 1960 Internat. Congr., pages 1–11, 1962.
8. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.
9. C. C. Chang and H. J. Keisler. *Model Theory*. Studies in Logic and the Foundations of Mathematics. Elsevier, 1990.
10. E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem. *Handbook of Model Checking*. Springer Verlag, 2018.
11. Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2000.
12. M. Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.
13. E. M. Coven and M. E. Paul. Sofic systems. *Israel J. of Mathematics*, 20(2):165–177, 1975.
14. K. Culik. Global cellular automata. *Complex Systems*, 9:251–266, 1995.
15. K. Culik and Sheng Yu. Cellular automata, $\omega\omega$-regular sets, and sofic systems. *Discrete Applied Mathematics*, 32:85–101, 1991.
16. U. B. Darji and S. W. Seif. A note on decidability of cellularity. *J. Cell. Autom.*, 7(5-6):509–514, 2012.
17. M. Davis. The definition of universal Turing machines. *Proc. of the American Mathematical Society*, 8:1125–1126, 1957.
18. M. Delorme and J. Mazoyer. *Cellular Automata: A Parallel Model*, volume 460 of *Mathematics and Its Applications*. Kluwer Academic Publishers, 1999.
19. S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.
20. C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9:47–68, January 1965.
21. D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Patterson, and W. P. Thurston. *Word Processing in Groups*. Jones and Bartlett, Burlington, 1992.
22. R. Fischer. Sofic systems and graphs. *Monatshefte für Mathematik*, 80:179–186, 1975.
23. C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108(1):45–82, 1993.
24. A. Gleason. Semigroups of shift register counting matrices. *Math. Systems Theory*, 25:253–267, 1992. Revised by F. Kochman, L. Neuwirth.
25. G. A. Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Math. Systems Theory*, 3:320–375, 1969.

26. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
27. M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge UP, 2000.
28. B. Khoussainov and A. Nerode. Automatic presentations of structures. In *LCC '94: Int. Workshop on Logical and Computational Complexity*, pages 367–392, London, UK, 1995. Springer-Verlag.
29. B. Khoussainov and A. Nerode. *Automata Theory and its Applications*. Birkhäuser, 2001.
30. B. Khoussainov and S. Rubin. Automatic structures: overview and future directions. *J. Autom. Lang. Comb.*, 8(2):287–301, 2003.
31. O. Kupferman. Avoiding determinization. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, 2006.
32. P. Kůrka. *Topological and Symbolic Dynamics*. Number 11 in Cours Spécialisés. Societe Mathematique de France, Paris, 2003.
33. P. Kůrka. *Encyclopedia of Complexity and System Science*, chapter Topological Dynamics of Cellular automata. Volume Encyclopedia of Complexity and System Science of Meyers [36], 2009.
34. D. Lind and B. Marcus. *Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.
35. D. Marker. *Model Theory: An Introduction*. Springer Verlag, 2002.
36. R. A. Meyers, editor. *Encyclopedia of Complexity and System Science*. Springer, Berlin, 2009.
37. V. Nekrashevych. *Self-Similar Groups*, volume 117 of *Math. Surveys and Monographs*. AMS, 2005.
38. M. Nivat and D. Perrin. Ensembles reconnaisable de mots biinfinis. *Canad. J. Math.*, 38:513–537, 1986.
39. D. Perrin and J.-E. Pin. *Infinite Words*, volume 141 of *Pure and Applied Math.* Elsevier, Amsterdam, 2004.
40. J. E. Pin. *Varieties of Formal Languages*. Foundations of Computer Science. Plenum Publishing Corporation, 1986.
41. M. O. Rabin and D. S. Scott. Finite automata and their decision problems. *IBM Jour. Research*, 3(2):114–125, 1959.
42. S. Safra. On the complexity of ω-automata. In *Proc. 29th FOCS*, pages 319–327, Washington, 1988. IEEE Computer Soc. Press.
43. J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
44. D. Scott. Some definitional suggestions for automata theory. *J. Comput. Syst. Sci.*, 1(2):187–212, August 1967.
45. J. R. Shoenfield. *Mathematical Logic*. Addison Wesley, Boston, 1967.
46. A. Smith. Universality of wolfram's 2, 3 turing machine. *Complex Systems*, 29(1):1–44, 2020.
47. K. Sutner. A note on Culik-Yu classes. *Complex Systems*, 3(1):107–115, 1989.
48. K. Sutner. Classifying circular cellular automata. *Phys. D*, 45(1–3):386–395, 1990.
49. K. Sutner. Linear cellular automata and Fischer automata. *Parallel Computing*, 23(11):1613–1634, 1997.
50. K. Sutner. *Linear Cellular Automata and De Bruijn Automata*, pages 303–320. Volume 460 of *Mathematics and Its Applications* [18], 1999.
51. K. Sutner. *Classification of Cellular Automata*, chapter Classification of Cellular Automata. Volume Encyclopedia of Complexity and System Science of Meyers [36], 2009.
52. K. Sutner. Model checking one-dimensional cellular automata. *J. Cellular Automata*, 4(3):213–224, 2009.
53. K. Sutner. Computational classification of cellular automata. *Int. J. General Systems*, 41(6):1–13, 2012.
54. B. Weiss. Subshifts of finite type and sofic systems. *Monatshefte für Mathematik*, 77:462–474, 1973.