TUCS

Jarkko Kari (Editor)

Proceedings of JAC 2010

Journées Automates Cellulaires

Turku Centre *for* Computer Science

TUCS Lecture Notes
No 13, Dec 2010

Proceedings of

JAC 2010

Journées Automates Cellulaires

December 15-17, 2010, Turku, Finland

Editor:

Jarkko Kari

TUCS Lecture Notes 13
December 2010

ISBN 978-952-12-2503-1 (printed)
ISBN 978-952-12-2504-8 (electronic)
ISSN 1797-8823 (printed)
ISSN 1797-8831 (electronic)

Foreword

The second Symposium on Cellular Automata "Journées Automates Cellulaires" (JAC 2010) took place in Turku, Finland, on December 15-17, 2010. The first two conference days were held in the Educarium building of the University of Turku, while the talks of the third day were given onboard passenger ferry boats in the beautiful Turku archipelago, along the route Turku–Mariehamn–Turku. The conference was organized by FUNDIM, the Fundamentals of Computing and Discrete Mathematics research center at the mathematics department of the University of Turku.

The program of the conference included 17 submitted papers that were selected by the international program committee, based on three peer reviews of each paper. These papers form the core of these proceedings. I want to thank the members of the program committee and the external referees for the excellent work that have done in choosing the papers to be presented in the conference.

In addition to the submitted papers, the program of JAC 2010 included four distinguished invited speakers: Michel Coornaert (Université de Strasbourg, France), Bruno Durand (Université de Provence, Marseille, France), Dora Giammarresi (Università di Roma Tor Vergata, Italy) and Martin Kutrib (Universität Gießen, Germany). I sincerely thank the invited speakers for accepting our invitation to come and give a plenary talk in the conference. The invited talk by Bruno Durand was eventually given by his co-author Alexander Shen, and I thank him for accepting to make the presentation with a short notice. Abstracts or extended abstracts of the invited presentations appear in the first part of this volume.

The program also included several informal presentations describing very recent developments and ongoing research projects. I wish to thank all the speakers for their contribution to the success of the symposium. I also would like to thank the sponsors and our collaborators: the Finnish Academy of Science and Letters, the French National Research Agency project EMC (ANR-09-BLAN-0164), Turku Centre for Computer Science, the University of Turku, and Centro Hotel. Finally, I sincerely thank the members of the local organizing committee for making the conference possible.

These proceedings are published both in an electronic format and in print. The electronic proceedings are available on the electronic repository HAL, managed by several French research agencies. The printed version is published in the general publications series of TUCS, Turku Centre for Computer Science. We thank both HAL and TUCS for accepting to publish the proceedings.

Turku, December 2010

Jarkko Kari

Program Committee

Enrico Formenti (Université de Nice-Sophia Antipolis, France)
Jarkko Kari (chair, University of Turku, Finland)
Jean Mairesse (Université Paris 7, France)
Ivan Rapaport (Universidad de Chile, Santiago, Chile)
Klaus Sutner (Carnegie Mellon University, Pittsburgh, USA)
Siamak Taati (University of Groningen, the Netherlands)

External referees

Florent Becker, Raimundo Briceño, Silvio Capobianco, Romain Demangeon, Alberto Dennunzio, Amir F. Dana, Anahí Gajardo, Pierre Guillon, Edmund Harriss, Petr Kůrka, Bruno Martin, Pierre-Étienne Meunier, Christophe Papazian, Julien Provillard, Éric Rémila, Ville Salo, Matthias Schulz, Véronique Terrier, Guillaume Theyssier, Michael Weiss, Jean-Baptiste Yunès, Charalampos Zinoviadis.

Organizing Committee

Alexis Ballier
Pierre Guillon
Timo Jolivet
Jarkko Kari (chair)
Arto Lepistö
Charalampos Zinoviadis

Table of Contents

Invited Talks: Abstracts and Extended Abstracts

Regular Papers

SOME EXTENSIONS OF THE MOORE-MYHILL GARDEN OF EDEN THEOREM

MICHEL COORNAERT

Institut de Recherche Mathématique Avancée Université de Strasbourg, 7 rue René Descartes, 67084 Strasbourg Cedex, France
E-mail address: coornaert@math.u-strasbg.fr

Abstract. The Moore-Myhill Garden of Eden theorem asserts that a cellular automaton with finite alphabet over a free abelian group of rank 2 is surjective if and only if it is pre-injective. Here, pre-injectivity means that two configurations which coincide outside of a finite subset of the group must coincide everywhere if they have the same image under the cellular automaton. The Garden of Eden theorem has been extended to amenable groups by Ceccherini-Silberstein, Machi and Scarabotti and to cellular automata defined over strongly irreducible subshifts of finite type by Fiorenzi. I will discuss these extensions as well as linear versions jointly obtained with Ceccherini-Silberstein.

Journées Automates Cellulaires 2010 (Turku), pp. 2-7

1D EFFECTIVELY CLOSED SUBSHIFTS AND 2D TILINGS

BRUNO DURAND [1], ANDREI ROMASHCHENKO [2], AND ALEXANDER SHEN [2]

[1] LIF, CNRS & Aix–Marseille Université
E-mail address: Bruno.Durand@lif.univ-mrs.fr
URL: http://www.lif.univ-mrs.fr/~bdurand

[2] LIF, CNRS & Aix–Marseille Université, on leave from IITP RAS
E-mail address: {Andrei.Romashchenko,Alexander.Shen}@lif.univ-mrs.fr

Abstract. Michael Hochman showed that every 1D effectively closed subshift can be simulated by a 3D subshift of finite type and asked whether the same can be done in 2D. It turned out that the answer is positive and necessary tools were already developed in tilings theory.

We discuss two alternative approaches: first, developed by N. Aubrun and M. Sablik, goes back to Leonid Levin; the second one, developed by the authors, goes back to Peter Gacs.

1. Simulation

Let A be a finite alphabet and let F be an enumerable set of A-strings. Consider all biinfinite A-sequences (i.e., mappings of type $\mathbb{Z} \to A$) that do not contain substrings from F. The set of these sequences is effectively closed (its complement is a union of an enumerable set of intervals in Cantor topology) and invariant under (left and right) shifts. Sets constructed in this way are called *effectively closed 1D subshifts.*

Effectively closed 2D subshifts are defined in a similar way; instead of biinfinite sequences we have configurations, i.e., mappings of type $\mathbb{Z}^2 \to A$, and instead of forbidden strings we have forbidden patterns (rectangles filled with A-letters). Given the set F of forbidden patterns, we consider the set of all configurations where no elements of F appear. This set of configurations is closed under vertical and horizontal shifts. If F is enumerable, we get *effectively closed 2D subshifts*; if F is finite, we get *2D subshifts of finite type.*

2D subshifts of finite type are closely related to tilings. A *tile* is a square with colored sides (colors are taken from some finite set C). A *tile set* is a set of tiles, i.e., a subset of C^4, since each tile is determined by four colors (upper, lower, left, and right). For a tile set τ, we consider all τ-*tilings*, i.e., the tilings of the entire plane by translated copies of τ-tiles with matching colors.

Key words and phrases: effectively closed subshifts, subshifts of finite type, tilings.

The paper was supported part by NAFIT ANR-08-EMER-008-01 grant, part by EMC ANR-09-BLAN-0164-01.

Tilings can be considered as a special case of 2D subshifts of finite type. Indeed, subshift is governed by local rules (forbidden pattern say what is not allowed according to these rules). In tilings the rules are extremely local: they say that the neighbor tiles should have matching colors, i.e., 1×2 rectangles where the colors do not match, are forbidden.

So every tile set determines a subshift of finite type (the alphabet is a tile set). The reverse statement is also true if we allow the extension of an alphabet. (It is natural since we have to simulate any local rule by a more restricted class of matching rules.) Formally, for every alphabet A and subshift S of finite type we can find:

- a set of colors C;
- a tile set $\tau \subset C^4$;
- a mapping $d\colon \tau \to A$

such that every τ-tiling after applying d to each tile becomes an element of S and every element of S can be obtained in this way from some τ-tiling. Such a correspondence between tilings and subshifts of finite type works in any dimension: k-dimensional tilings correspond to k-dimensional subshifts of finite type (modulo the alphabet extension).

Now we want to compare subshifts in different dimensions. Let S be a 1D subshift. We can make a 2D subshift from it by copying each letter vertically. It is easy to see that an effectively closed 1D subshift becomes an effectively closed 2D subshift (we use rules that guarantee the vertical propagation, i.e., require that vertical neighbors should have the same letter, and the rules of the original 1D subshift in horizontal direction). This 2D shift, denoted by $\bar{S}$, is not of finite type, if the original 1D shift was not of finite type. However, $\bar{S}$ is *sofic*, i.e., is a projection by a subshift of finite type in extended alphabet:

Theorem 1.1. *For every effectively closed 1D subshift S in alphabet A there exists an alphabet A', a finite 2D subshift S' in alphabet A', and a mapping $d\colon A' \to A$ such that the image of S' under d (applied in each place) is $\bar{S}$.*

This theorem (with 3D instead of 2D, which makes it easier) was proved by Michael Hochman [10] who asked whether the same is true for 2D. His motivation came from ergodic theory.

It turned out that the tools needed to prove theorem 1.1 for 2D tilings were already developed in the framework of tilings theory when Hochman asked his question. Moreover, there are two different sets of tools that can be used; one was used by Nathalie Aubrun and Mathieu Sablik [1] (and goes back to Leonid Levin [4]), the other one was used in [6] (and goes back to Peter Gács [8]). In the sequel we discuss informally how these tools work, and what are the similarities and the differences.

2. Tools

Let us describe informally our problem. In 2D we have local rules that guarantee that each vertical line contains some letter. We need to add some other rules to guarantee that the emerging horizontal sequence of letters does not have substrings from some enumerable set F. We are allowed to superimpose additional structure to the configuration (by extending the alphabet: we let A' be a product of A and some other finite set). Rules for this extended configuration should guarantee that its base belongs to $\bar{S}$.

So we need to run a computation that generates F and some process that compares generated elements with substrings in the horizontal sequence. It is well known (since the first papers of Wang [14, 15] where the notion of a tile set was introduced) that tile sets can simulate computation easily: indeed, a time-space diagram of a Turing machine (or a cellular automaton) obeys local rules that guarantee that computation is performed correctly when started. The problem is to initiate the computation: there is no special point in the plane where the computation can be started, so we need to "break the translational symmetry" somehow.

This problem was solved by Berger [2] who proved that there exists an aperiodic tile set, i.e., a tile set τ such that τ-tilings exist but all are aperiodic. (A tiling is *periodic* if there is a non-zero translation that does not change it. One can show that if a tile set has a periodic tiling then it has a 2-periodic tiling where some finite block is repeated horizontally and vertically.) Berger used a complicated multi-level construction that was later simplified in different ways by Robinson [13] and others. The simplification made clear that Berger's construction is essentially based on self-similarity: any tiling can be divided into blocks that behave like individual tiles. (In the original construction this similarity was obscured by some irregularities; the cleaned versions could be found in [12] or [3].)

This self-similarity creates some kind of a skeleton that can be used to initiate computations. However, the problem is that we necessarily initiate them in many different places, and these "geometrically parallel" computation should be organized to achieve some goal. Berger used them to prove the undecidability of the domino problem (to determine whether a given tile set has at least one tiling); for that purpose it is enough to initiate multiple copies of the same computation: all are limited in time and space, but among them there are computations of arbitrary length. For that we split the plane into different zones used for different computations. It is possible to find such an arrangement; in each zone the standard local rules for a computation are used but zones are not contiguous. So we need additional efforts to transmit the information from one zone to another one. This all can be done (with limited overlap, so the total density of information in a given cell remains finite).

Then Hanf [9] and Myers [11] proved that there are tile sets that admit only non-recursive tilings (a much stronger statement than the existence of an aperiodic tile set). This was done by embedding a separation problem for two inseparable enumerable sets, and for this we need that all the parallel computations not only share the same (finite) program, but also share the same (infinite) input. Therefore, some additional machinery is needed to synchronize the inputs of all the computations (each computation gets a finite part of the infinite input sequence, but these finite parts are consistent pieces of an infinite input).

When simulating 1D effectively closed subshift, we need more: the input is given to us externally (the contents of the vertical lines that carry A-letters) and we need to check this input against all possible forbidden substrings. This means that we are very limited in space (and cannot distribute pieces of input sparsely over the entire plane as before).

2.1. Robinson-type solution

The way to do this was developed in [4]. At each level of self-similarity we have *computation squares* that are arranged in *computational stripes.* Such a stripe is infinite in vertical direction and carries an infinite computation of a finite-space cellular automaton. (One can wonder whether it makes sense to have an infinite computation in a finite space. Indeed, it is not really infinite; it runs for some time (exponential in the width of the stripe) and then is restarted. The repeated computations are not necessarily identical, since they interact with the other computations which could be different.) Each stripe performs some checks for the part of the horizontal sequence that is near it. When the level (and the size of the stripe) increases, the checked zone and the time allowed for the computation increase. Working together, the stripes can check the horizontal sequence against all forbidded substrings.

In [4] this technique was used for one specific 1D effectively closed subshift with a binary alphabet (for some fixed $\alpha < 1$ we forbid all sufficiently long strings whose Kolmogorov complexity is less than α times length). However, this technique is quite general and can be used for any 1D effectively close subshift modulo some technical problem.

This technical problem is that the underlying self-similar structure may be "degenerate" in the sense that the plane is divided in two parts that have no common ancestors. In this case we need some additional tricks (extending the zone of responsibility of each stripe) that were not needed for the specific subshift of [4]. The reason why they were not needed: if a string of low complexity (compared to length) is split into two parts, one of then has low complexity, too.

So the technique of [4] is not enough. The final construction was discovered (in fact, independently from [4]) by N. Aubrun and M. Sablik.

2.2. Fixed-point solution

There is a different way to organize the computations that uses fixed-point self-similar tiling. The idea of a self-similar fixed-point tile set can be explained as follows. We already know (since Wang papers) that tiling can be used to simulate computations. This computation, in its turn, can be used to guarantee the desired behavior of bigger blocks, called *macro-tiles*. So for a desired behavior of macro-tiles we can construct tiling rules (i.e., tile set) that guarantees this behavior. If, by chance, these tiling rules coincide with the rules for macro-tiles, we get self-similarity as a consequence.

But there is a classical tool to get this coincidence intentionally, not by chance: the Kleene fixed-point construction. It was used by Kleene in the recursion theory and later by von Neumann to construct self-reproducing automata. Usually it is illustrated as follows: for every program p (in fact, for every string p) there exists a program p' that prints the text of p. Kleene's theorem guarantees that one can find p such that p' is equivalent to p, i.e., the program p prints its own text. The same trick (though not just the statement of Kleene's recursion theorem) can be used for 2D computations. This was done first by Gács [8] in a complicated setting (error-correction in 2D computations); we use the same idea in a much simpler environment. For each tile set τ one can construct a set τ' of tiles that force macro-tiles to behave like τ-tiles; Kleene's trick can then be used to make τ isomorphic to τ'. This construction is explained in [5].

Then some additional structure can be superimposed with this self-similar skeleton (by adding some other computations); Kleene's trick can still be used to achieve self-similarity (in some extended sense).

This construction is rather flexible and can be applied to different problems, see [6]. The differences and similiarities between two constructions are summarized in the following comparison table.

2.3. Comparison table

Problem	Solution 1	Solution 2
Breaking the symmetry	Use (modified) Berger–Robinson self-similar construction where self-similarity is guaranteed by geometric arguments	Use fixed-point self-similar construction, where self-similiarity is a byproduct of some computational structure
Placing the computations	Computations of different levels are all performed "on the ground", by individual cells, and the plane is divided into regions allocated to each level	Computations of different levels are performed at different levels of hierarchy: high level computations deal not with individual tiles but with macro-tiles
Arranging arbitrarily long computations	Computations are infinite in the vertical direction but finite in horizontal direction, each computation performs a space-bounded check of some part of the horizontal sequence; the bound increases with the level	Computations are finite in both direction; each computation performs a time-bounded check of some part of the horizontal sequence; the bound increases with the level
Bringing the bits of the horizontal sequence to the computation	Recursively from lower levels; the bits are synchronized explicitly "on the ground"	Recursively from lower levels; each level checks whether the bits at the next level are recorded correctly
Dealing with degenerate case of the self-similar pattern	Using overlapping zones of responsibility	Using overlapping zones of responsibility
Error resistance	Not clear (we first need some error-resistant underlying geometric construction)	Adding redundancy at each level

References

[1] N. Aubrun, M. Sablik, Simulation of recursively enumerable subshifts by two dimensional SFT and a generalization. Preprint, available from M. Sablik's home page.

[2] R. Berger, The undecidability of the domino problem. *Memoirs of the AMS*, v. 66 (1966).

[3] B. Durand, L. Levin, A. Shen, Local rules and global order, or aperiodic tilings, *The Mathematical Intelligencer*, v. 27 (2005), no. 1, p. 64–68.

[4] B. Durand, L. Levin, A. Shen, Complex Tilings. *J. Symbolic Logic*, **73** (2), 593–613, 2008.

[5] B. Durand, A. Romashchenko, A. Shen, Fixed Point and Aperiodic Tilings. *Proc. 12th International Conference of Developments in Language Theory. Kyoto, Japan, 2008*, p. 537–548.

[6] B. Durand, A. Romashchenko, A. Shen, Fixed-point tile sets and their applications. CoRR abs/0910.2415, 2009. `http://arxiv.org/abs/0910.2415`

[7] B. Durand, A. Romashchenko, A. Shen, Effective closed subshifts in 1D can be implemented in 2D. *Fields of Logic and Computation*, Lecture Notes in Computer Science, v. 6300 (2010), p. 208–226.

[8] P. Gács, Reliable Computation with Cellular Automata. J. Comput. Syst. Sci. **32**(1), 15–78, 1986.

[9] W. Hanf, Nonrecursive tilings of the plane, i, *Journal of Symbolic Logic*, v. 39 (1974), no. 2, p. 283–285.

[10] M. Hochman, On the dynamic and recursive properties of multidimensional symbolic systems. *Inventiones mathematicae*, **176**, 131–167 (2009).

[11] D. Myers, Nonrecursive tilings of the plane, ii, *Journal of Symbolic Logic*, v. 39 (1974), no. 2, p. 286–294.

[12] N. Ollinger, Two-by-two Substitution Systems and the Undecidability of the Domino Problem, *Computability in Europe, 2008* (CiE'2008), Lecture Notes in Computer Science, v. 5028, p. 476–485.

[13] R. Robinson, Undecidability and nonperiodicity for tilings of the plane, *Inventiones Mathematicae*, v. 12 (1971), p. 177–209.

[14] H. Wang, Proving theorems by pattern recognition, II, *Bell System Technical Journal*, v. 40 (1961), p. 1–41.

[15] H. Wang, Dominoes and the ∀∃∀ case of the decision problem. *Proceedings of the Symposium on Mathematical Theory of Automata*, Brooklyn Polytechnic Institute, New York, 1962, p. 23–55.

Journées Automates Cellulaires 2010 (Turku), pp. 8-12

TILING-RECOGNIZABLE TWO-DIMENSIONAL LANGUAGES: FROM NON-DETERMINISM TO DETERMINISM THROUGH UNAMBIGUITY

DORA GIAMMARRESI

Dipartimento di Matematica, Università di Roma "Tor Vergata", via della Ricerca Scientifica, 00133 Roma, Italy
E-mail address: giammarr@mat.uniroma2.it

Abstract. Tiling recognizable two-dimensional languages, also known as REC, generalize recognizable string languages to two dimensions and share with them several theoretical properties. Nevertheless REC is not closed under complementation and the membership problem is NP-complete. This implies that this family REC is intrinsically non-deterministic. The natural and immediate definition of unambiguity corresponds to a family UREC of languages that is strictly contained in REC. On the other hand this definition of unambiguity leads to an undecidability result and therefore it cannot correspond to any deterministic notion. We introduce the notion of line-unambiguous tiling recognizable languages and prove that it corresponds or somehow naturally introduces different notions of determinism that define a hierarchy inside REC.

A *picture* (or two-dimensional string) is a two-dimensional arrays of symbols from a finite alphabet. A set of pictures is called *two-dimensional language.* Basic notations and operations can be extended from string to pictures. The size of a picture p is a pair (m, n) corresponding to the number of its rows and columns, respectively. Moreover there can be defined an operation of column-concatenation between pictures with the same number of rows and of row-concatenation between pictures with the same number of columns. By iteration, there can be also defined the corresponding row- and column- star operations.

The first generalization of finite-state automata to two dimensions can be attributed to M. Blum and C. Hewitt who in 1967 introduced the notion of a four-way automaton moving on a two-dimensional tape as the natural extension of a one-dimensional two-way finite automaton (see [7]). They also proved that the deterministic version corresponds to a language class smaller than the corresponding one defined by the non-deterministic model. Four-way automata was not a successful model since the corresponding language class does not satisfies important properties as closure under concatenation and star operations. Since then, many approaches have been presented in the literature in order to find the "right way" to generalize in 2D what regular languages are in one dimension: finite automata, grammars, logics and regular expressions (see for example [8, 14, 25, 18, 27]). Here we focus

Key words and phrases: automata and formal Languages, two-dimensional languages, tiling systems, unambiguity, determinism.

This work was partially supported by ESF Project *"AutoMathA"* (2005-2010).

on the family *REC* of *tiling recognizable picture languages* (see [14, 15] that have been widely investigated and that it is considered as a valid candidate to represent a counter part to 2D of regular string languages.

The definition of REC takes as starting point a characterization of recognizable string languages in terms of local languages and projections (cf. [11]). A picture language L is *local* if it is defined by a finite set of 2×2 pictures, called *tiles* that represent all allowed sub-pictures of size $(2, 2)$ for pictures in L. A pair composed by a local language over an alphabet Γ and an alphabetic projection $\pi : \Gamma \longrightarrow \Sigma$ is called *tiling system.* A picture language L over an alphabet Σ is recognized by a tiling system (given by a local language L' and π) if each picture $p \in L$ can be obtained as projection of a picture $p' \in L'$ (i.e. $p = \pi(p')$). A picture language is tiling recognizable if it is recognized by a tiling system. REC is the family of tiling recognizable picture languages. We point that languages of infinite picture (ω-pictures) were also studied in the setting of tiling systems in [1, 12, 13].

It can be verified that REC is closed under union and intersection, rotation and mirror and under column- and row- concatenation and star operations. Moreover, the definition of REC in terms of tiling systems turns out to be very robust: in [15, 17] it is shown that the family REC has a characterization in terms of logical formulas (a generalization of Büchi's theorem for strings to 2D). In [19], it is proved that REC has a counterpart as machine model in the *two-dimensional on-line tessellation acceptor (OTA)* introduced by K. Inoue and A. Nakamura in [18]. Other models of automata for REC are proposed in [4, 8, 24]. Tiling systems can be also simulated by domino systems [19] and Wang tiles [10] and grammars [9]. Further we remark that when pictures degenerate in strings (i.e. when considering only one-row pictures) recognizability by tiling systems corresponds exactly to recognizability by finite state string automata.

A crucial difference with the one-dimensional case lies in the fact that the definition of recognizability by tiling systems is intrinsically non-deterministic. Deterministic machine models to recognize two-dimensional languages have been considered in the literature: they always accept classes of languages smaller than the corresponding non-deterministic ones (see for example, [7, 18, 26]). This seems to be unavoidable when jumping from one to two dimensions. Further REC family is not closed under complementation and therefore the definition of any constraint to force determinism in tiling systems should necessary result in a class smaller than REC. Strictly connected with this problems are the complexity results on the recognition problem in REC. Let L be a language in REC defined by a tiling system composed by a local picture language L' and a projection π. To recognize that a given picture p of m rows and n columns belongs to L, one has to "rewrite" symbols in all positions in p to get a local picture p' that belongs to L' and such that $\pi(p') = p$. This can be done by scanning all positions of p in some order. The non-determinism implies that, once reached a given position one may eventually backtrack on all positions already visited, that is on $O(mn)$ steps. Moreover in [21] it is proved that the recognition problem for REC languages is NP-complete.

In formal language theory, an intermediate notion between determinism and nondeterminism is the notion of unambiguity. In an unambiguous model, we require that each accepted object admits only one successful computation. Both determinism and unambiguity correspond to the existence of a *unique* process of computation, but while determinism is a "local" notion, unambiguity is a fully "global" one. *Unambiguous tiling recognizable* two-dimensional languages have been introduced in

[14], and their family is referred to as UREC. Informally, a picture language belongs to UREC if it admits an unambiguous tiling system, that is if every picture has a unique pre-image in its corresponding local language. In [5], the proper inclusion of UREC in REC is proved but it is also proved that it is undecidable whether a given tiling system is unambiguous. From a computational side, there are not known algorithms to recognize pictures that exploit the properties of UREC. This implies that, at each step of the recognition computation, it can be necessary to backtrack on all already visited positions.

A relevant goal is then to find subclasses for REC that inherit important properties but also allow feasible computations. Moreover an interesting result would be proving that, as for regular string languages, notions of some kind of unambiguity and determinism coincide.

Remark that another difference between unambiguity and determinism is that determinism is always related to a scanning strategy to read the input. In the string case the scanning is implicitly assumed to be left-to right and in fact deterministic automata are defined related to this direction. Moreover since deterministic, non-ambiguous and non-deterministic models are all equivalent there is no need to consider determinism from right-to-left (referred to as co-determinism). Nevertheless it is worthy to remark that not all regular string languages admits automata that are both deterministic and co-deterministic. In the two-dimensional case we have to consider all the scanning directions from left, right, top and bottom sides.

By exploiting the different possibilities of scanning for a two-dimensional array in [3, 2] there are introduced different notions of unambiguity we call here *line-unambiguity* where a line can be either a column or a row or a diagonal. We consider tiling systems for which the computations to recognize a given picture can have at each position a backtracking on at most $m + n$ steps. Such definitions lie between those of unambiguity and determinism (as long as we consider that a deterministic computation has zero backtracking steps at each position) while they all coincide with determinism when pictures degenerate in strings.

The informal definitions are very simple and natural. A tiling system is *column-unambiguous* if, when used to recognize a picture by reading it along a left-to-right or right-to left direction, once computed a local column, there is only one possible next local column. As consequence in a computation by a column-unambiguous tiling system to recognize a picture with m rows, the backtracking at each step is at most of m steps. Similarly there are defined *row-unambiguous* and *diagonal-unambiguous* tiling systems corresponding to computations that proceed by rows or by diagonals, respectively. The corresponding families of languages are denoted by *Col-UREC*, *Row-UREC* and *Diag-UREC*. In [3, 2] there are proved necessary conditions for a language to be in Col-UREC and in Row-UREC. Using such conditions one can show that families Col-UREC and Row-UREC are strictly contained in UREC. In a different set-up it is also shown that Diag-UREC is strictly included both in Col-UREC and Row-UREC. Moreover all those properties are decidable.

Very interestingly we can prove that diagonal-unambiguous tiling systems are equivalent to some deterministic tiling systems where the uniqueness of computation is guaranteed by certain conditions on the set of local tiles: the corresponding language family is denoted by DREC ([3]). Similar results hold for classes Col-UREC and Row-UREC whose union turns to be equivalent to another "deterministic" class named Snake-DREC [23]. All those classes are closed under complementation [2, 23]. As result, when we consider this line unambiguity we can prove equivalence with deterministic models and therefore we guarantee a recognition algorithm linear in the size (i.e. number of rows times number of columns) of the input.

References

[1] J.-H. Altenbernd, W. Thomas, and S. Wöhrle. Tiling systems over infinite pictures and their acceptance conditions. In *Developments in Language Theory 2002*, volume 2450 of *Lecture Notes in Computer Science*, pages 297–306. Springer, 2003.

[2] M. Anselmo, D. Giammarresi, M. Madonia. M. Anselmo, D. Giammarresi, and M. Madonia. Deterministic and unambiguous families within recognizable two-dimensional languages. *Fundamenta Informaticae*, 98(2-3):143–166, 2010.

[3] M. Anselmo, D. Giammarresi, M. Madonia. From determinism to non-determinism in recognizable two-dimensional languages. In *Procs. DLT 07*, T. Harju, J. Karhumaki and A. Lepisto (Eds.), LNCS 4588, Springer-Verlag, Berlin 2007.

[4] M. Anselmo, D. Giammarresi, M. Madonia. A computational model for tiling recognizable two-dimensional languages. *Theoretical Computer Science*, Vol. 410-37, 3520–3529 Elsevier 2009.

[5] M. Anselmo, D. Giammarresi, M. Madonia, A. Restivo. Unambiguous Recognizable Two-dimensional Languages. *RAIRO: Theoretical Informatics and Applications*, Vol. 40, 2, pp. 227-294, EDP Sciences 2006.

[6] M. Anselmo, M. Madonia. Deterministic and unambiguous two-dimensional languages over one-letter alphabet. *Theoretical Computer Science*, Vol. 410-16, 1477–1485 Elsevier 2009.

[7] M. Blum, C. Hewitt. Automata on a two-dimensional tape. *IEEE Symposium on Switching and Automata Theory*, pages 155–160, 1967.

[8] S. Bozapalidis, A. Grammatikopoulou, Recognizable picture series, *Journal of Automata, Languages and Combinatorics*, special vol. on *Weighted Automata, 2004.*

[9] S. Crespi Reghizzi and M. Pradella. Tile rewriting grammars and picture languages. *Theoretical Computer Science*, vol 340, n.2, pp. 257-272, Elsevier 2005.

[10] De Prophetis, L., Varricchio, S.: Recognizability of rectangular pictures by wang systems. Journal of Automata, Languages, Combinatorics. **2** (1997) 269-288

[11] S. Eilenberg. *Automata, Languages and Machines.* Vol. A, Academic Press, 1974.

[12] O. Finkel. On recognizable languages of infinite pictures. *Int. J. Found. Comput. Sci.*, 15(6):823–840, 2004.

[13] O. Finkel. Highly undecidable problems about recognizability by tiling systems. *Fundam. Inform.*, 91(2):305–323, 2009.

[14] D. Giammarresi, A. Restivo. Recognizable picture languages. *Int. Journal Pattern Recognition and Artificial Intelligence.* Vol. 6, No. 2& 3, pages 241 –256, 1992.

[15] D. Giammarresi, A. Restivo. Two-dimensional languages. *Handbook of Formal Languages*, G.Rozenberg, *et al.* Eds, Vol. III, pag. 215–268. Springer Verlag, 1997.

[16] D. Giammarresi, A. Restivo. Matrix-based complexity functions and recognizable picture languages. In *Logic and Automata: History and Perspectives.* E. Grader, J.Flum, T. Wilke Eds. ,pag 315-337. Texts in Logic and Games 2. Amsterdam University Press, 2007.

[17] D. Giammarresi, A. Restivo, S. Seibert, W. Thomas. Monadic second order logic over pictures and recognizability by tiling systems. *Information and Computation*, Vol 125, 1, pag 32–45, 1996.

[18] K. Inoue, A. Nakamura. Some properties of two-dimensional on-line tessellation acceptors. *Information Sciences*, Vol. 13, pages 95–121, 1977.

[19] K. Inoue, I. Takanami. A characterization of recognizable picture languages. In *Proc. Second International Colloquium on Parallel Image Processing*, A. Nakamura et al. (Eds.), LNCS 654, Springer-Verlag, Berlin 1993.

[20] M. Latteux and D. Simplot. Recognizable Picture Languages and Domino Tiling. *Theorethical Computer Science* 178(1-2): 275-283, 1997.

[21] K. Lindgren, C. Moore, M. Nordahl. Complexity of two-dimensional patterns. *Journal of Statistical Physics*, 91 (5-6), pag. 909–951, 1998.

[22] O. Matz.Regular expressions and Context-free Grammars for picture languages. *Proc. STACS'97* - LNCS 1200 pag. 283-294 - Springer Verlag 1997.

[23] V. Lonati, M. Pradella. Snake-Deterministic Tiling Systems. In *Proc. MFCS 2009*, LNCS, Vol. 5734, 549-560, Springer 2009.

[24] V. Lonati and M. Pradella. Picture-recognizability with automata based on Wang tiles. In *Proc. SOFSEM 2010*, LNCS, vol. 5901, 576-587. Springer, 2010.

[25] O. Matz. On piecewise testable, starfree, and recognizable picture languages. In *Foundations of Software Science and Computation Structures*, M. Nivat Ed., vol. 1378, Springer, 1998.

[26] A. Potthoff, S. Seibert, W. Thomas. Nondeterminism versus determinism of finite automata over directed acyclic graphs. *Bull. Belgian Math. Soc.* 1, 285–298, 1994.

[27] R. Siromoney. Advances in array languages. In *Graph-Grammars and Their Applications to Computer Science*, Ehrig et al. (Eds.), pages 549–563. Lecture Notes in Computer Science 291, Springer-Verlag, Berlin, 1987.

Journées Automates Cellulaires 2010 (Turku), pp. 13-30

MEASURING COMMUNICATION IN CELLULAR AUTOMATA

MARTIN KUTRIB AND ANDREAS MALCHER

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
E-mail address: {kutrib,malcher}@informatik.uni-giessen.de

Abstract. Cellular automata and iterative arrays are one-dimensional arrays of interconnected interacting finite automata which work synchronously at discrete time steps. In this paper, the focus lies on the resource of communication which naturally takes place between cells. Communication is measured here with regard to qualitative and quantitative aspects. More detailed, the amount of communication in cellular automata is measured by limiting the bandwidth of the communication links between the cells, as well as limiting the number of messages allowed to be sent between cells. An overview on recent results on the computational capacity of as well as on decidability problems in such restricted cellular automata and iterative arrays is given.

1. Introduction

Parallel computational models are appealing and widely used in order to describe, understand, and manage parallel processes occurring in real life. One principal task in order to employ a parallel computational model in an optimal way is to understand how cooperation of several processors is organized optimally. To this end, it is essential to know which communication and which amount of communication must or should take place between several processors. From the viewpoint of energy and the costs of communication links, it would be desirable to communicate a minimal number of times with a minimum amount of information transmitted. On the other hand, it would be interesting to know how much communication is necessary in a certain parallel model to accomplish a certain task.

Here, we consider the model of cellular automata with parallel and sequential input mode. The latter model is also known as iterative array. In both models the state of each cell is communicated to its neighbors in every time step. That is, on the one hand the state is sent regardless of whether it is really required, and on the other hand, the number of different messages that may be sent is determined by the number of states. Thus, it seems to be natural to restrict this “unbounded” communication by limiting the number of possible different messages between cells. This brings us to cellular automata and iterative arrays where the bandwidth of the communication links between two cells is bounded by some fixed constant. In the most restricted setting this is one bit bandwidth. However, these automata are

2000 ACM Subject Classification: F.1.1, F.1.2, F.4.3.

Key words and phrases: cellular automata, iterative arrays, message complexity, limited communication, decidability, formal languages.

still powerful enough to accept unary as well as non-unary non-context-free (even non-semilinear) languages in real time. For some classes it is additionally known that almost all of the commonly investigated decidability problems are undecidable. Furthermore, we obtain proper hierarchies with regard to the resources bandwidth, dimension, and time. Finally, we get a complete picture on the relations between cellular automata and iterative arrays in the case of restricted inter-cell bandwidth.

For real-time one-way cellular automata where the communication is quantitatively measured by counting the number of uses of the communication links between cells, the total sum of all communications or the maximal number of communications that may appear between each two cells can be considered. Reducing the number of communications in such a way that each two neighboring cells may communicate constantly often only, leads to devices which also still can accept non-context-free (even non-semilinear) languages. Again, almost all of their decidability questions can be shown to be undecidable. An interesting additional restriction is to consider inputs of a certain form only. For such bounded languages it is known that in other computational models, such as certain variants of multi-head finite automata, undecidable problems become decidable. However, all commonly investigated decidability questions remain undecidable for communication-restricted real-time one-way cellular automata accepting bounded languages.

Finally, both limitations on the communication between cells are combined. It turns out that even this restriction does not lead to positive decidability results. The known undecidability results can be translated to the case of one and two message bandwidth. Thus, the resource communication makes the model in a way inherently complex since even a limitation to a very small amount of communication does not reduce the computational complexity of the model's undecidability problems.

2. Preliminaries and Definitions

We denote the rational numbers by $\mathbb{Q}$, and the non-negative integers by $\mathbb{N}$. The empty word is denoted by λ, the reversal of a word w by w^R, and for the length of w we write $|w|$. The set of words over some alphabet A whose lengths are at most $l \in \mathbb{N}$ is denoted by $A^{\leq l}$. We write $\subseteq$ for set inclusion, and $\subset$ for strict set inclusion. The cardinality of a set M is denoted by $|M|$.

A one-dimensional iterative array is a linear, semi-infinite array of identical deterministic finite automata, sometimes called cells. The finite automata work synchronously at discrete time steps. Except for the leftmost cell each one is connected to its both nearest neighbors (see Figure 1). For convenience we identify the cells by their coordinates, that is, by non-negative integers. The distinguished leftmost cell at the origin is connected to its right neighbor and, additionally, equipped with a one-way read-only input tape. At the outset of a computation the input is written on the input tape with an infinite number of end-of-input symbols to the right, and all cells are in the so-called quiescent state. The state transition of all cells but the input cell depends on the current state of the cell itself and the current states of its neighbors. The state transition of the input cell additionally depends on the input symbol to be read next. The head of the one-way input tape is moved at any step to the right. With an eye towards recognition problems the machines have no extra output tape but the states are partitioned into accepting and rejecting states.

In an iterative array with *k-message restricted inter-cell communication*, the state transition depends on the current state of each cell and on the messages that are currently sent by its neighbors, where the possible messages are formalized as a set of possible communication symbols. The messages to be sent by a cell depend on its current state and are determined by so-called communication functions.

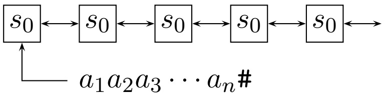


Figure 1: Initial configuration of an iterative array.

Definition 2.1. A *one-dimensional iterative array with k-message restricted inter-cell communication* (*IA$_k$*) is a system $\langle S, A, B, F, \texttt{\#}, s_0, b_l, b_r, \delta, \delta_0\rangle$, where

(1) S is the finite, nonempty set of *cell states*,
(2) A is the finite, nonempty set of *input symbols*,
(3) B with $|B| = k$ is the finite set of *communication symbols*,
(4) $F \subseteq S$ is the set of *accepting states*,
(5) $\texttt{\#} \notin A$ is the *end-of-input symbol*,
(6) $s_0 \in S$ is the *quiescent state*,
(7) $b_l, b_r : S \to B \cup \{\perp\}$ are *communication functions* which determine the information *to be sent* to the left and right neighbors, where $\perp$ means *nothing to send*,
(8) $\delta : (B \cup \{\perp\}) \times S \times (B \cup \{\perp\}) \to S$ is the *local transition function for all but the input cells* satisfying $\delta(b_r(s_0), s_0, b_l(s_0)) = s_0$,
(9) $\delta_0 : S \times (A \cup \{\texttt{\#}\}) \times (B \cup \{\perp\}) \to S$ is the *local transition function for the input cell.*

Let $\mathcal{M}$ be an IA$_k$. A configuration of $\mathcal{M}$ at some time $t \geq 0$ is a description of its global state which is a pair (w_t, c_t), where $w_t \in A^*$ is the remaining input sequence and $c_t : \mathbb{N} \to S$ is a mapping that maps the single cells to their current states. The configuration (w_0, c_0) at time 0 is defined by the input word w_0 and the mapping c_0 that assigns the quiescent state to all cells, while subsequent configurations are chosen according to the global transition function Δ: Let (w_t, c_t), $t \geq 0$, be a configuration. Then its successor configuration $(w_{t+1}, c_{t+1}) = \Delta(w_t, c_t)$ is as follows.

$$c_{t+1}(i) = \delta(b_r(c_t(i-1)), c_t(i), b_l(c_t(i+1)))$$

for all $i \geq 1$, and $c_{t+1}(0) = \delta_0(c_t(0), a, b_l(c_t(1)))$ where $a = \texttt{\#}$ and $w_{t+1} = \lambda$ if $w_t = \lambda$, as well as $a = a_1$ and $w_{t+1} = a_2 \cdots a_n$ if $w_t = a_1 \cdots a_n$. Thus, the global transition function Δ is induced by δ and δ_0.

A *two-way cellular automaton with k-message restricted inter-cell communication* is similar to an iterative array. The main difference is that the cell at the origin does not fetch the input but the input is supplied in parallel to the cells. That is, an input $a_1 \cdots a_n$ is fed to the cells $1, \ldots, n$ such that initially cell i is in state a_i. Cells 0 and $n+1$ are initially in a permanent so-called *boundary* state # (see Figure 2). Cell 1 indicates acceptance or rejection, and the array is bounded to the n cells which are initially active.

Definition 2.2. A *cellular automaton with k-message restricted inter-cell communication* (*CA$_k$*) is a system $\langle S, A, B, F, \texttt{\#}, b_l, b_r, \delta\rangle$, where

(1) S is the finite, nonempty set of *cell states*,
(2) $A \subseteq S$ is the finite, nonempty set of *input symbols*,
(3) B with $|B| = k$ is the finite set of *communication symbols*,
(4) $F \subseteq S$ is the set of *accepting states*,
(5) $\# \notin S$ is the *boundary state*,
(6) $b_l, b_r : (S \cup \{\#\}) \to (B \cup \{\bot\})$ are *communication functions* which determine the information *to be sent* to the left and right neighbors, where $\bot$ means *nothing to send*,
(7) $\delta : (B \cup \{\bot\}) \times S \times (B \cup \{\bot\}) \to S$ is the *local transition function.*

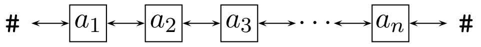


Figure 2: A two-way cellular automaton.

A *one-way cellular automaton* (OCA_k) is a cellular automaton in which each cell receives information from its immediate neighbor to the right only. So, the flow of information is restricted to be from right to left. Formally, δ is a mapping from $S \times (B \cup \{\bot\})$ to S.

A *configuration* of a cellular automaton $\langle S, A, B, F, \#, b_l, b_r, \delta \rangle$ at time $t \geq 0$ is a mapping $c_t : \{1, \ldots, n\} \to S$, for $n \geq 1$. For a given input $w = a_1 \cdots a_n \in A^+$ we set the initial configuration $c_{0,w}(i) = a_i$, for $1 \leq i \leq n$. Successor configurations are computed according to the global transition function Δ:

Let c_t, $t \geq 0$, be a configuration. Then its successor configuration $c_{t+1} = \Delta(c_t)$ is as follows.

$$\begin{aligned}
&c_{t+1}(1) = \delta(b_r(\#), c_t(1), b_l(c_t(2)))\\
&c_{t+1}(i) = \delta(b_r(c_t(i-1)), c_t(i), b_l(c_t(i+1))), i \in \{2, \ldots, n-1\}\\
&c_{t+1}(n) = \delta(b_r(c_t(n-1)), c_t(n), b_l(\#))
\end{aligned}$$

for CA$_k$ and

$$\begin{aligned}
&c_{t+1}(i) = \delta(c_t(i), b_l(c_t(i+1))), i \in \{1, \ldots, n-1\}\\
&c_{t+1}(n) = \delta(c_t(n), b_l(\#))
\end{aligned}$$

for OCA$_k$.

An input w is accepted by an IA$_k$ ((O)CA$_k$) $\mathcal{M}$ if at some time i during the course of its computation the input cell (cell 1) enters an accepting state. The *language accepted by* $\mathcal{M}$ is denoted by $L(\mathcal{M})$. Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n+1$ ($t(n) \geq n$ for (O)CA$_k$) be a mapping. If all $w \in L(\mathcal{M})$ are accepted with at most $t(|w|)$ time steps, then $L(\mathcal{M})$ is said to be of time complexity t.

The family of all languages which are accepted by some device X with time complexity t is denoted by $\mathscr{L}_t(X)$. If t is the function $n+1$, for IA$_k$, or the function n, for (O)CA$_k$, acceptance is said to be in *real time* and we write $\mathscr{L}_{rt}(X)$. Since for nontrivial computations an IA$_k$ has to read at least one end-of-input symbol, real time has to be defined as $(n+1)$-time. The *linear-time* languages $\mathscr{L}_{lt}(X)$ are defined according to $\mathscr{L}_{lt}(X) = \bigcup_{r \in \mathbb{Q},\, r \geq 1} \mathscr{L}_{r \cdot n}(X)$.

3. What Arrays with Limited Inter-Cell Bandwidth Can Do

For iterative arrays and cellular automata where the bandwidth of the communication links between two cells is bounded by some fixed constant, the most restricted setting is one message bandwidth. However, these devices are still powerful enough to solve problems such as the firing squad synchronization problem in optimal time [22]. Moreover, it is known [28, 29] that one-message IA_1 can accept rather complicated unary languages in real time, for example, words whose lengths are powers of two $\{\, a^{2^n} \mid n \geq 1 \,\}$, words whose length are square numbers $\{\, a^{n^2} \mid n \geq 1 \,\}$, words whose length are prime numbers $\{\, a^p \mid p \text{ is prime} \,\}$, or words whose lengths are Fibonacci numbers. Clearly, every deterministic finite automaton can be simulated in the input cell of an iterative array. Thus, we obtain the strict inclusion

$$\mathrm{REG} \subset \mathscr{L}_{rt}(\mathrm{IA}_1)$$

where REG denotes the regular languages.

The next lemma shows that an additive speed-up in IA_k is always possible [17]. The benefit of this lemma is that we do not have to care about additive constant factors which may arise in constructions. The situation is different for cellular automata where we have an infinite, strict, and tight hierarchy depending on the additive constant (see [32] for OCA_k, and Theorem 5.1 for CA_k).

Lemma 3.1. *$\mathscr{L}_{rt+p}(IA_k) = \mathscr{L}_{rt}(IA_k)$, for any constant number $p \geq 1$.*

In [17] it is shown that one-message IA_1 can simulate binary counters in real time. This construction is often a useful tool for the modular design of algorithms. For example, by exploiting counters it is possible to check certain properties of distances between two designated cells, to verify that something happens at a certain time step, to implement clocks and pulses, and much else besides. In general, a specific application of a counter requires some additional mechanisms. For example, it might be of interest to recognize the time step at which a counter overflows or becomes zero, instead of knowing the real counter value. The proofs of some of the next results make extensively use of the ability of one-message IA_1 to simulate binary counters attached to the input cell, which stores the least significant bit. The counter either can be incremented or decremented, and can be tested for zero. Moreover, it is possible to use two counters and to switch from one counter to another with the minimal resources time and communication bandwidth.

The following language is known not to be regular. On the other hand, it can be accepted by pushdown automata making at most one turn.

Lemma 3.2. *$\{\, a^n b^n \mid n \geq 0 \,\} \in \mathscr{L}_{rt}(IA_1)$.*

The proof of the previous lemma is based on a construction that allows to switch between an increasing counter and a decreasing counter by using some signal. The switching costs four additional time steps. In general, any *constant* number of such switchings can be realized. The resulting device can be made real-time again by Lemma 3.1. Furthermore, we can do more complicated computations based on the same technique. For example, if there are some subroutines sharing the same communication, then the switching signals can be interpreted individually by the routines. As before, the input cell controls the time steps at which the signals are sent. This idea is applied in the construction showing the next result. The language $\{\, a^n b^n c^n \mid n \geq 0 \,\}$ is not accepted by any pushdown automaton, but it is accepted by some one-message IA_1 in real time.

Lemma 3.3. *$\{\, a^n b^n c^n \mid n \geq 0 \,\} \in \mathscr{L}_{rt}(IA_1)$.*

By similar constructions one can set up real-time one-message IA_1 that recognize languages of the form $\{\, a^nb^nc^nd^n \mid n \geq 0 \,\}$ or $\{\, a^nb^mc^nd^m \mid n, m \geq 0 \,\}$. The important observation is that the number of switching signals to be sent is constant. Glimpsing at the language $\{\, a^n(b^n)^m \mid n, m \geq 0 \,\}$ one receives the impression that the number of switches is about m, which is no longer constant. So, the language might be too hard to be accepted by real-time one-message IA_1. In fact, in order to show the converse we cannot use the techniques developed so far. But instead we can *reuse counter values* in a sense, that once a counter is decremented to zero, it restarts to count from its previous initial value. To this end, a master copy of the original counter value has to be kept in another counter. The construction is sketched in [17].

Lemma 3.4. $\{\, a^n(b^n)^m \mid n, m \geq 0 \,\} \in \mathscr{L}_{rt}(IA_1)$.

Next, we turn to some integer calculations, where the problem instances are suitably encoded and consider the problems of "adding" negative and positive integers as well as "multiplying" non-negative integers [17].

Lemma 3.5. $\{\, a^nb^mc^l \mid n, m, l \geq 0 \textit{ and } -n + m = l \,\} \in \mathscr{L}_{rt}(IA_1)$.

In order to construct a real-time one-message IA_1 for the next languages we have to handle counters that do not share the same communication. If we have a constant number of counters to handle, then the *simulation can be time-shared*, that is, the single steps of the counters are simulated cyclically one after the other. Clearly, in general this may violate the real-time constraint. But for our purposes it works fine.

Lemma 3.6. $\{\, a^nba^mb(ba)^{n\cdot m} \mid n, m \geq 0 \,\} \in \mathscr{L}_{rt}(IA_1)$.

Let $a_1, a_2, \ldots, a_k, \$, a, b$ be $k+3$ different symbols. Obvious generalizations of the construction (using several counters) show that, for example, the languages

$\{\, a_1^n a_2^m \$ (ba)^{n\cdot m} \mid n, m \geq 0 \,\}$,
$\{\, a_1^n a_2^m a_3^l \$ (baa)^{n\cdot m\cdot l} \mid n, m, l \geq 0 \,\}$,
$\{\, a_1^{\alpha_1} a_2^{\alpha_2} \cdots a_k^{\alpha_k} \$ (ba^{k-1})^{\alpha_1\cdot\alpha_2\cdots\alpha_k} \mid \alpha_1, \alpha_2, \ldots, \alpha_k \geq 0 \,\}$,
$\{\, a^nba^mb(baaaa)^{n^2\cdot m^3} \mid n, m \geq 0 \,\}$,
$\{\, a_1^n a_2^m a_3^l \$ (baaaaaa)^{n^2\cdot m^3\cdot l^2} \mid n, m, l \geq 0\}$ and, given constants $j_1, j_2, \ldots, j_k \geq 0$,
$\{a_1^{\alpha_1} a_2^{\alpha_2} \cdots a_k^{\alpha_k} \$ (ba^{j_1+j_2+\cdots+j_k-1})^{\alpha_1^{j_1}\cdot\alpha_2^{j_2}\cdots\alpha_k^{j_k}} \mid \alpha_1, \alpha_2, \ldots, \alpha_k \geq 0 \,\}$

are accepted by one-message IA_1 in real-time.

4. What Some of the Arrays with Limited Inter-Cell Bandwidth Can Do

To obtain a valuable tool to show that certain languages cannot be accepted by iterative arrays with limited inter-cell bandwidth, we define two equivalence relations and derive upper bounds on the number of equivalence classes which can be distinguished by IA_k. If the number of equivalence classes induced by a certain language exceeds this upper bound, the language is not accepted by any IA_k.

Definition 4.1. Let $L \subseteq A^*$ be a language over an alphabet A and $l \geq 1$ be a constant.

(1) Two words $w \in A^*$ and $w' \in A^*$ are l-right-equivalent with respect to L if for all $y \in A^{\leq l}$: $wy \in L \iff w'y \in L$.
(2) $N_r(l, L)$ denotes the number of l-right-equivalence classes with respect to L.
(3) Two words $w \in A^{\leq l}$ and $w' \in A^{\leq l}$ are l-left-equivalent with respect to L if for all $y \in A^*$: $wy \in L \iff w'y \in L$.
(4) $N_\ell(l, L)$ denotes the number of l-left-equivalence classes with respect to L.

The next lemma provides upper bounds for the number of equivalence classes distinguished by IA_k.

Lemma 4.2. *Let $k \geq 1$ be a constant.*

(1) *If $L \in \mathscr{L}_{rt}(IA_k)$, then there exists a constant $p \geq 0$ such that*

$$N_r(l, L) \leq p^{(l+1)}$$

and

(2) *if $L \in \mathscr{L}_t(IA_k)$, then there exists a constant $p \geq 0$ such that*

$$N_\ell(l, L) \leq p \cdot (k+1)^l$$

for all $l \geq 1$ and all time complexities $t : \mathbb{N} \to \mathbb{N}$.

Now, we fix the time complexity to real time and consider the number of different messages. For any number of messages $k \geq 1$ we define an alphabet $A_k = \{a_0, \dots, a_k\}$ and a language

$$L_{bit}(k) = \{\, e^x\$u_1u_2\cdots u_m \mid x \geq 1 \text{ and } m \geq 2x-1 \\ \text{and } u_i \in A_k, 1 \leq i \leq m, \text{ and } u_j = u_{j+2x-1}, 1 \leq j \leq m-(2x-1)\,\}.$$

The languages $L_{bit}(k)$ are witnesses for an infinite, strict, and tight hierarchy dependent on the number of different messages [14].

Theorem 4.3. *Let $k \geq 1$ be a constant. The language $L_{bit}(k+1)$ belongs to the difference $\mathscr{L}_{rt}(IA_{k+1}) \setminus \mathscr{L}_{rt}(IA_k)$. Therefore, $\mathscr{L}_{rt}(IA_k) \subset \mathscr{L}_{rt}(IA_{k+1})$.*

The previous result can be extended to iterative arrays of any higher dimension [14]. Without formal definition we mention that a d-dimensional iterative array IA^d is an iterative array, where the cells are arranged as d-dimensional grid ($\mathbb{N}^d$) as extention of a line ($\mathbb{N}^1$). Each cell is connected to its immediate neighbors in any dimension. Moreover, we fix the time complexity to real time, the number of different messages to constants $k \geq 1$, and consider the dimension. For any dimension $d \geq 2$ we define a language $L_{dim}(d)$ as follows. We start with a series of regular sets:

$$X_1 = \$\{a, b\}^+, \qquad X_{i+1} = \$X_i^+, \text{ for } i \geq 1.$$

Due to the separator symbol \$, every word $u \in X_{i+1}$ can uniquely be decomposed into its subwords from X_i. So, we can define the projection on the jth subword as usual: Let $u = \$u_1 \cdots u_m$, where $u_j \in X_i$, for $1 \leq j \leq m$. Then $u[j]$ is defined to be u_j, if $1 \leq j \leq m$, otherwise $u[j]$ is undefined. Now define the language

$$M(d) = \{\, u¢e^{x_d}\$ \cdots \$e^{x_1}\$e^{2x}\$v \mid u \in X_d \text{ and } x_i \geq 1, 1 \leq i \leq d, \\ \text{and } x = x_1 + \cdots + x_d \text{ and } v = u[x_d][x_{d-1}]\cdots[x_1] \text{ is defined}\,\}.$$

Finally, the language $L_{dim}(d)$ is given as homomorphic image of $M(d)$. More precisely, $L_{dim}(d) = h(M(d))$, where $h : \{a, b, e, \$, ¢\}^* \to \{a, b\}^*$ is defined by: $h(a) = ba$, $h(b) = bb$, $h(e) = b$, $h(\$) = ab$, $h(¢) = aa$.

Theorem 4.4. *Let $d \geq 1$ and $k \geq 1$ be constants. The language $L_{dim}(d+1)$ belongs to the difference $\mathscr{L}_{rt}(IA_1^{d+1}) \setminus \mathscr{L}_{rt}(IA_k^d)$. Therefore, $\mathscr{L}_{rt}(IA_k^d) \subset \mathscr{L}_{rt}(IA_k^{d+1})$.*

Interestingly, $L_{dim}(d+1)$ belongs to $\mathscr{L}_{lt}(\mathrm{IA}_1^1)$, that is, one can trade all dimensions and messages for a slow-down from real time to linear time.

Theorem 4.5. *Let $d \geq 1$ and $k \geq 1$ be constants. Then $\mathscr{L}_{rt}(IA_k^d) \subset \mathscr{L}_{lt}(IA_k^d)$.*

From Theorem 4.4 and Theorem 4.3 we obtain a double hierarchy concerning messages and dimensions which is depicted in Figure 3.

$$\begin{array}{ccccccccc}
\vdots & & \vdots & & & & \vdots & & \\
\cup & & \cup & & & & \cup & & \\
\mathscr{L}_{rt}(\mathrm{IA}_1^d) & \subset & \mathscr{L}_{rt}(\mathrm{IA}_2^d) & \subset & \cdots & \subset & \mathscr{L}_{rt}(\mathrm{IA}_k^d) & \subset & \cdots \\
\cup & & \cup & & & & \cup & & \\
\vdots & & \vdots & & & & \vdots & & \\
\cup & & \cup & & & & \cup & & \\
\mathscr{L}_{rt}(\mathrm{IA}_1^2) & \subset & \mathscr{L}_{rt}(\mathrm{IA}_2^2) & \subset & \cdots & \subset & \mathscr{L}_{rt}(\mathrm{IA}_k^2) & \subset & \cdots \\
\cup & & \cup & & & & \cup & & \\
\mathscr{L}_{rt}(\mathrm{IA}_1^1) & \subset & \mathscr{L}_{rt}(\mathrm{IA}_2^1) & \subset & \cdots & \subset & \mathscr{L}_{rt}(\mathrm{IA}_k^1) & \subset & \cdots
\end{array}$$

Figure 3: Double hierarchy of fast IA with restricted inter-cell communication.

5. What Arrays with Limited Inter-Cell Bandwidth Cannot Do

The main difference between cellular automata and iterative arrays is that the input is processed in parallel by the former model and processed sequentially by the latter model. An interesting variant of cellular automata is the restriction to one-way information flow. The relations between iterative arrays and cellular automata with two-way and one-way information flow are summarized in the left part of Figure 4. Here, we will clarify the relation between the discussed language classes in the case of restricted communication. It turns out that we obtain a finer hierarchy than in the unrestricted case. The results are depicted in the right part of Figure 4.

$$\begin{tikzcd}
 & \mathscr{L}_{lt}(\mathrm{CA}) = \mathscr{L}_{lt}(\mathrm{IA}) & \\
 & \mathscr{L}_{rt}(\mathrm{CA}) \arrow[u, dotted] & \\
\mathscr{L}_{rt}(\mathrm{OCA}) \arrow[ur] & & \mathscr{L}_{rt}(\mathrm{IA}) \arrow[ul] \\
 & \mathrm{REG} \arrow[ul] \arrow[ur] &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \mathscr{L}_{lt}(\mathrm{CA}_k) & \\
\mathscr{L}_{rt}(\mathrm{CA}_k) \arrow[ur] & & \mathscr{L}_{lt}(\mathrm{IA}_k) \arrow[ul] \\
\mathscr{L}_{rt}(\mathrm{OCA}_k) \arrow[u] & & \mathscr{L}_{rt}(\mathrm{IA}_k) \arrow[u] \\
 & & \mathrm{REG} \arrow[u]
\end{tikzcd}$$

Figure 4: Relations between unrestricted (left) and restricted (right) language families. Solid arrows are strict inclusions and dotted arrows are inclusions. Families which are not connected by any path are incomparable.

The first difference between language classes with and without communication restrictions is that there are regular languages which cannot be accepted by any k-message cellular automaton CA_k even if we add a constant number of time steps to real time, whereas all regular languages are accepted in the unrestricted case. Moreover, the witness languages can be accepted by a real-time CA_{k+1}. Thus, we obtain a strict message hierarchy for two-way real-time cellular automata. Furthermore, the witness languages are accepted by $(n+r+1)$-time CA_k. Thus, we obtain a very dense strict time hierarchy. If we allow just one more time step, we obtain a strictly more powerful device.

Theorem 5.1. *Let $k \geq 1$ and $p \geq 0$ be constants.*

(1) *There is a regular language which is not accepted by any $(n+p)$-time CA_k.*
(2) *Then $\mathscr{L}_{rt+p}(CA_k) \subset \mathscr{L}_{rt+p}(CA_{k+1})$.*
(3) *Then $\mathscr{L}_{rt+p}(CA_k) \subset \mathscr{L}_{rt+p+1}(CA_k)$.*

If the communication channels of the CA have a sufficient capacity, the regular languages are accepted.

Lemma 5.2. *Let $k \geq 1$ be a constant. Then every regular language over a k-letter alphabet is accepted by some real-time CA_k.*

Together we obtain a two-dimensional infinite hierarchy for cellular automata with restricted communication concerning the number of messages communicated and the number of time steps performed, which is depicted in Figure 5.

$$\begin{array}{ccccccccc}
\vdots & & \vdots & & & & \vdots & & \\
\cup & & \cup & & & & \cup & & \\
\mathscr{L}_{rt+r}(\mathrm{CA}_1) & \subset & \mathscr{L}_{rt+r}(\mathrm{CA}_2) & \subset & \cdots & \subset & \mathscr{L}_{rt+r}(\mathrm{CA}_k) & \subset & \cdots \\
\cup & & \cup & & & & \cup & & \\
\vdots & & \vdots & & & & \vdots & & \\
\cup & & \cup & & & & \cup & & \\
\mathscr{L}_{rt+1}(\mathrm{CA}_1) & \subset & \mathscr{L}_{rt+1}(\mathrm{CA}_2) & \subset & \cdots & \subset & \mathscr{L}_{rt+1}(\mathrm{CA}_k) & \subset & \cdots \\
\cup & & \cup & & & & \cup & & \\
\mathscr{L}_{rt}(\mathrm{CA}_1) & \subset & \mathscr{L}_{rt}(\mathrm{CA}_2) & \subset & \cdots & \subset & \mathscr{L}_{rt}(\mathrm{CA}_k) & \subset & \cdots
\end{array}$$

Figure 5: Two-dimensional infinite hierarchy of CA with restricted communication.

The next theorem clarifies the relation between iterative arrays and one-way cellular automata.

Theorem 5.3. *Let $k \geq 1$ be a constant. There is a language belonging to the difference $\mathscr{L}_{rt}(OCA_1) \setminus \mathscr{L}_{lt}(IA_k)$.*

Next, we show proper inclusions between language families that are related by inclusions for structural reasons. In [5] an unrestricted real-time iterative array accepting prime numbers in unary has been constructed. In [29] the result has been improved to a one-message iterative array. However, while in the unrestricted case any real-time iterative array can be simulated by a real-time two-way cellular automaton, here we have seen that both devices define incomparable language families. By a different witness language the next result follows.

Theorem 5.4. *Let $k \geq 1$ be a constant. Then $\mathscr{L}_{rt}(OCA_k) \subset \mathscr{L}_{rt}(CA_k)$.*

The next result says that for cellular automata with restricted communication a parallel processing of the input is more powerful than a sequential processing under linear time conditions. This is in contrast to the unrestricted case where both input modes imply the same computational capacity.

Theorem 5.5. *Let $k \geq 1$ be a constant. Then $\mathscr{L}_{lt}(IA_k) \subset \mathscr{L}_{lt}(CA_k)$.*

We complement our considerations with the following theorem.

Theorem 5.6. *Let $k \geq 1$ be a constant. Then the language families $\mathscr{L}_{rt}(OCA_k)$ and $\mathscr{L}_{rt}(CA_k)$ are incomparable with REG, $\mathscr{L}_{rt}(IA_k)$, and $\mathscr{L}_{lt}(IA_k)$.*

It is a long-standing open problem whether linear-time cellular automata are more powerful than real-time cellular automata. Since linear-time CA_k can accept all regular languages whereas real-time CA_k cannot, we can answer this question in the affirmative for the case of restricted communication. On the other hand, Theorem 5.5 says that in the case of restricted communication parallel input mode implies a more powerful model than sequential input mode whereas both input modes are equally powerful in the unrestricted case.

In Figure 6 we summarize the relations between cellular automata with restricted and unrestricted communication [13].

$$\begin{tikzcd}
\mathscr{L}_{lt}(\mathrm{CA}) \arrow[r, equal] & \mathscr{L}_{lt}(\mathrm{CA}_k) \\
\mathscr{L}_{lt}(\mathrm{OCA}) \arrow[u, dashed] \arrow[r, equal] \arrow[d, equal] & \mathscr{L}_{lt}(\mathrm{OCA}_k) \arrow[u, dashed] \\
\mathscr{L}_{rt}(\mathrm{CA})^R & \mathscr{L}_{rt}(\mathrm{CA}_k)^R \arrow[l] \arrow[u] \\
\mathscr{L}_{rt}(\mathrm{OCA}) \arrow[u] & \mathscr{L}_{rt}(\mathrm{OCA}_k) \arrow[l] \arrow[u] \\
\mathrm{REG} \arrow[u] &
\end{tikzcd}$$

Figure 6: Relations between unrestricted and restricted language families. Solid arrows are strict inclusions, dashed arrows are inclusions, and double lines denote equivalence.

Decidability Questions

For real-time IA with unrestricted communication it is known that many decidability questions are undecidable [21, 27]. One may ask under which additional conditions undecidable questions become decidable. Since the recognizing power of real-time IA_k is weaker than general real-time IA, it is a natural question whether undecidable questions for real-time IA are decidable for IA_k. But this question has been answered negatively [17]. In fact, several of the common decidability questions are even *non-semidecidable* for real-time IA_1. A formal problem is *semidecidable*, if the algorithm halts on all instances for which the answer is *yes*. Thus, non-semidecidability results for real-time IA_1 show that even real-time IA with a minimum amount of communication are very powerful devices.

A well-known non-semidecidable problem is emptiness for Turing machines (or algorithms) [4, 26]. The proof of the non-semidecidability results relies on a reduction of the emptiness problem for Turing machines. To this end, Turing machine computations are encoded into small grammars [7]. Roughly speaking, *valid computations of Turing machines* are histories of accepting Turing machine computations. It suffices to consider deterministic Turing machines with a single tape and a single read-write head. Without loss of generality and for technical reasons, one can assume that any accepting computation has at least three and, in general, an odd number of steps. Therefore, it is represented by an even number of configurations. Moreover, it is assumed that the Turing machine cannot print blanks, and that a configuration is halting if and only if it is accepting.

Let S be the state set of some Turing machine $\mathcal{M}$, where s_0 is the initial state, $T \cap S = \emptyset$ is the tape alphabet containing the blank symbol, $A \subset T$ is the set of input symbols, and $F \subseteq S$ is the set of accepting states. Then a configuration of $\mathcal{M}$ can be written as string of the form T^*ST^* such that $t_1 \cdots t_i s t_{i+1} \cdots t_n$ is used to express that $\mathcal{M}$ is in state s, scanning tape symbol t_{i+1}, and t_1 to t_n is the non-blank part of the tape inscription. The set of *valid computations* VALC($\mathcal{M}$) is now defined to be the set of patterns of the form $w_1\$w_3\$\cdots\$w_{2k-1}¢w_{2k}^R\$\cdots\$w_4^R\$w_2^R$, where w_i are configurations, \$ and ¢ are symbols not appearing in w_i, w_1 is an initial configuration of the form s_0A^*, w_{2k} is an accepting configuration of the form T^*FT^*, and w_{i+1} is the successor configuration of w_i, for $1 \leq i \leq 2k$. The set of *invalid computations* INVALC($\mathcal{M}$) is the complement of VALC($\mathcal{M}$) with respect to the coding alphabet $\{\$, ¢\} \cup T \cup S$.

The simple but nevertheless important and fruitful observation is that the set VALC($\mathcal{M}$) is empty if and only if the set accepted by the Turing machine $\mathcal{M}$ is empty. The following lemma is the starting point of the reductions. It has been shown in [21], see also [12].

Lemma 5.7. *Given a Turing machine $\mathcal{M}$, (general) real-time IA can effectively be constructed that accept VALC($\mathcal{M}$) and INVALC($\mathcal{M}$).*

To establish non-semidecidability results for real-time IA$_1$ we need some variation of the set of valid computations which still has the property that the set is empty if and only if the corresponding Turing machine accepts the empty set. Thus, we proceed as follows. Given a set of symbols $A = \{a_1, a_2, \ldots, a_n\}$, for all $k \geq 1$ we define a homomorphism h_k by $h_k(a_i) = a_i^k$, $1 \leq i \leq n$.

Lemma 5.8. *Given a Turing machine $\mathcal{M}$, there exists a number $k \geq 1$ such that real-time IA$_1$ can effectively be constructed that accept h_k(VALC($\mathcal{M}$)) and h_k(INVALC($\mathcal{M}$)).*

Clearly, the application of the homomorphism preserves the property mentioned before: The set h_k(VALC($\mathcal{M}$)) is empty if and only if the set accepted by the Turing machine $\mathcal{M}$ is empty. So, we obtain the next result immediately.

Theorem 5.9. *Emptiness, universality, finiteness, infiniteness, equivalence, and inclusion are undecidable for real-time IA$_1$.*

With respect to language theory there are more relations between (in)valid computations and non-semidecidable problems. The following reasonings originate from [7], see also [12, 21].

Let $\mathcal{M}$ be some Turing machine, and assume that $\mathcal{M}$ accepts a finite set. Then VALC($\mathcal{M}$) is finite and, clearly, context free. If conversely $\mathcal{M}$ accepts an infinite set, then an application of the pumping lemma shows that VALC($\mathcal{M}$) is not context free. Therefore, $\mathcal{M}$ accepts a finite set if and only if VALC($\mathcal{M}$) is context free.

Moreover, if $\mathcal{M}$ accepts a finite set, then VALC($\mathcal{M}$) is finite and its complement INVALC($\mathcal{M}$) is regular. Conversely, if INVALC($\mathcal{M}$) is regular, then VALC($\mathcal{M}$) is regular since the regular languages are closed under complementation. Therefore, VALC($\mathcal{M}$) is context free which implies that $\mathcal{M}$ accepts a finite set. Therefore, $\mathcal{M}$ accepts a finite set if and only if INVALC($\mathcal{M}$) is regular.

Corollary 5.10. *Regularity and context-freeness are undecidable for real-time IA_1.*

Now, the previous results can be applied to show that there is no general pumping lemma and no minimization algorithm for real-time IA_1. In general, a family of languages possesses a *pumping lemma in the narrow sense* if for each language L from the family there exists a constant $n \geq 1$ computable from L such that each $z \in L$ with $|z| > n$ admits a factorization $z = uvw$, where $|v| \geq 1$ and $u'v^iw' \in L$, for infinitely many $i \geq 0$. The prefix u' and the suffix w' depend on u, w and i.

Theorem 5.11.

(1) *The family of languages accepted by real-time IA_1 does not possess a pumping lemma (in the narrow sense).*

(2) *There is no minimization algorithm converting some real-time IA_1 to an equivalent real-time IA_1 having a minimal number of states.*

6. Limiting the Number of Messages

In the following we turn to measure the communication in cellular automata by the number of uses of the links between cells. It is understood that whenever a communication symbol not equal to $\perp$ is sent, a communication takes place. Here we do not distinguish whether either or both neighboring cells use the link. More precisely, the number of communications between cell i and cell $i+1$ up to time step t is defined by

$$\text{com}(i,t) = |\{\, j \mid 0 \leq j < t \text{ and } (b_r(c_j(i)) \neq \perp \text{ or } b_l(c_j(i+1)) \neq \perp)\,\}|\,.$$

For computations we now distinguish the maximal number of communications between two cells and the total number of communications. Let $c_0, c_1, \ldots, c_{t(|w|)}$ be the sequence of configurations computed on input w by some cellular automaton with time complexity $t(n)$, that is, the *computation on w*. Then we define

$$\begin{aligned} \text{mcom}(w) &= \max\{\, \text{com}(i, t(|w|)) \mid 1 \leq i \leq |w|-1 \,\} \text{ and} \\ \text{scom}(w) &= \sum_{i=1}^{|w|-1} \text{com}(i, t(|w|)). \end{aligned}$$

Let $f : \mathbb{N} \to \mathbb{N}$ be a mapping. If all $w \in L(\mathcal{M})$ are accepted with computations where $\text{mcom}(w) \leq f(|w|)$, then $\mathcal{M}$ is said to be *max communication bounded by f*. Similarly, if all $w \in L(\mathcal{M})$ are accepted with computations where $\text{scom}(w) \leq f(|w|)$, then $\mathcal{M}$ is said to be *sum communication bounded by f*. In general, it is not expected to have tight bounds on the exact number of communications but tight bounds on their numbers in the order of magnitude. For the sake of readability we denote the class of CA that are max communication bounded by some function $g \in O(f)$ by MC(f)-CA, where it is understood that f gives the order of magnitude. In addition, we use the notation *const* for functions from $O(1)$. Corresponding notations are used for OCA and sum communication bounded CA and OCA. (SC(f)-CA and SC(f)-OCA).

Computational Capacity

In order to identify the computational power of communication bounded real-time devices we begin by describing the relationship to previous works. In [30, 31] two-way cellular automata are considered where the number of proper state changes is bounded. Similar as in the present paper the sum of all state changes or the maximal number of the state changes of single cells are bounded. By applying the technique of saving communication steps by storing the last signal received in the state and to interpret an arriving $\perp$ suitably [19], it is not hard to see, that such a device can be simulated by the corresponding communication bounded device. Whether or not state change bounded devices are strictly weaker than communication bounded ones is an open problem. However, we adapt some of the results shown in connection with state changes in the next theorem.

Theorem 6.1 ([30, 31])**.**

(1) $\mathscr{L}_{rt}(\mathit{MC}(\mathit{const})\text{-}\mathit{CA}) \subset \mathscr{L}_{rt}(\mathit{SC}(n)\text{-}\mathit{CA})$.

(2) $\mathit{REG} \subset \mathscr{L}_{rt}(\mathit{MC}(\mathit{const})\text{-}\mathit{CA}) \subset \mathscr{L}_{rt}(\mathit{MC}(\sqrt{n})\text{-}\mathit{CA}) \subset \mathscr{L}_{rt}(\mathit{MC}(n)\text{-}\mathit{CA})$.

(3) $\mathscr{L}_{rt}(\mathit{MC}(\mathit{const})\text{-}\mathit{CA}) \subset \mathit{NL}$.

In order to clarify our notion and to show the power of the devices in question, we give some examples of languages which can be accepted by MC(*const*)-OCA. It has been shown in [19] that the family MC(*const*)-OCA contains the non-context-free languages $\{\, a_1^n a_2^n \cdots a_k^n \mid n \geq 1 \,\}$ for $k \geq 2$, $\{\, a^n b^m c^n d^m \mid n, m \geq 1 \,\}$, as well as the languages $\{\, a^n w \mid n \geq 1 \wedge w \in (b^* c^*)^k b^* \wedge |w|_b = n \,\}$, for all constants $k \geq 0$. All of these languages are either semilinear or non-bounded. But in contrast to many other computational devices, for example certain multi-head finite automata, parallel communicating finite automata, and certain parallel communicating grammar systems, MC(*const*)-OCA can accept non-semilinear bounded languages. This is shown in the next example.

Example 6.2. The language $L = \{\, a^n b^{n+\lfloor\sqrt{n}\rfloor} \mid n \geq 1 \,\}$ belongs to the family $\mathscr{L}_{rt}(\text{MC}(\mathit{const})\text{-OCA})$.

In [23] a CA is constructed such that its cell n enters a designated state exactly at time step $2n+\lfloor\sqrt{n}\rfloor$, and at most n cells are used for the computation. In fact, the CA constructed is actually an OCA. Additionally, each cell performs only a finite number of communication steps. Thus, the CA constructed is an MC(*const*)-OCA.

An MC(*const*)-OCA accepting L implements the mirror of the above construction on the a-cells of the input $a^n b^m$. Thus, the leftmost cell enters the designated state q at time step $2n + \lfloor\sqrt{n}\rfloor$. Additionally, in the rightmost cell a signal s with maximum speed is sent to the left. When this signal arrives in an a-cell exactly at a time step at which the cell would enter the designated state q, the cell changes to an accepting state instead. So, if $m = n + \lfloor\sqrt{n}\rfloor$, then s arrives at time $2n + \lfloor\sqrt{n}\rfloor$ at the leftmost cell and the input is accepted. In all other cases the input is rejected. Clearly, the OCA constructed is an MC(*const*)-OCA. ∎

Next, we turn to an infinite strict hierarchy of real-time SC(f)-CA families [19]. The top of the hierarchy is given by the next theorem.

Theorem 6.3. *Let $f : \mathbb{N} \to \mathbb{N}$ be a function. If $f \in o(n^2/\log(n))$, then language $L = \{\, wcw^R \mid w \in \{a,b\}^+ \,\}$ is not accepted by any real-time SC(f)-CA.*

In order to define witness languages that separate the levels of the hierarchy, for all $i \geq 1$, the functions $\varphi_i : \mathbb{N} \to \mathbb{N}$ are defined by $\varphi_1(n) = 2^n$, and $\varphi_i(n) = 2^{\varphi_{i-1}(n)}$, for $i \geq 2$, and we set $L_i = \{\, w\$^{\varphi_i(|w|)-2|w|}w^R \mid w \in \{a,b\}^+ \,\}$.

Lemma 6.4. *Let $i \geq 1$ be an integer and $f : \mathbb{N} \to \mathbb{N}$ be a function.*

(1) *If $f \in o((n\log^{[i]}(n))/\log^{[i+1]}(n))$ then language L_i is not accepted by any real-time SC(f)-CA.*

(2) *Language L_i is accepted by some real-time SC($n\log^{[i]}(n)$)-CA.*

So, we can derive the infinite hierarchy.

Theorem 6.5. *Let $i \geq 0$ be an integer. Then $\mathscr{L}_{rt}(SC(n\log^{[i+1]}(n))\text{-}CA)$ is properly included in $\mathscr{L}_{rt}(SC(n\log^{[i]}(n))\text{-}CA)$.*

Decidability Questions

As is the case for devices with limited inter-cell bandwidth, various problems are undecidable even for the weakest non-trivial device with limited messages, that is, for real-time MC(*const*)-OCA.

Two of the common techniques to show undecidability results are reductions of Post's Correspondence Problem or reductions of the emptiness and finiteness problem on Turing machines using the set of valid computations. Both techniques have been used successfully to obtain results for variants of cellular automata [15, 20, 21, 27]. Taking a closer look at these known techniques, it is not clear yet whether they can be applied to MC(*const*)-OCA. In [19] it is shown that emptiness is undecidable for real-time MC(*const*)-OCA by reduction of Hilbert's tenth problem which is known to be undecidable. The problem is to decide whether a given polynomial $p(x_1,\dots,x_n)$ with integer coefficients has an integral root. That is, to decide whether there are integers $\alpha_1,\dots,\alpha_n$ such that $p(\alpha_1,\dots,\alpha_n) = 0$. In [10] Hilbert's tenth problem has been used to show that emptiness is undecidable for certain multi-counter machines. As is remarked in [10], it is sufficient to restrict the variables $x_1,\dots,x_n$ to take non-negative integers only.

If $p(x_1,\dots,x_n)$ contains a constant summand, then we may assume that it has a negative sign. Otherwise, we continue with $p(x_1,\dots,x_n)$ multiplied with -1, whose constant summand now has a negative sign and which has the same integral roots as $p(x_1,\dots,x_n)$.

Such a polynomial then has the following form:

$$p(x_1,\dots,x_n) = t_1(x_1,\dots,x_n) + \cdots + t_r(x_1,\dots,x_n)$$

where each $t_j(x_1,\dots,x_n)$ $(1 \leq j \leq r)$ is of the form $t_j(x_1,\dots,x_n) = s_j x_1^{i_{j,1}} \cdots x_n^{i_{j,n}}$ with $i_{j,1},\dots,i_{j,n} \geq 0$. If $|s_j| > 1$ we replace $s_j x_1^{i_{j,1}} \cdots x_n^{i_{j,n}}$ by s_j copies of $x_1^{i_{j,1}} \cdots x_n^{i_{j,n}}$. So, we may assume without loss of generality that all constant factors are either 1 or -1.

Sketchy speaking, the reduction is as follows. For a polynomial $p(x_1,\dots,x_n)$ with integer coefficients that has the above form, languages $L(t_j)$ for every term

t_j are defined that evaluate the terms t_j. The next step is to simulate an evaluation of the given polynomial p. To this end, the evaluations of the single terms have to be put together, which is done by concatenating certain regular languages around each language $L(t_j)$ resulting in $\tilde{L}(t_j)$, and intersecting the reversals of all these languages. This gives a language $\tilde{L}(p) = \bigcap_{j=1}^{r} \tilde{L}(t_j)^R$. From $\tilde{L}(p)$ another language $L(p)$ is constructed which is empty if and only if $p(x_1, \ldots, x_n)$ has no solution in the non-negative integers. In a long proof it is shown that $L(p)$ belongs to $\mathscr{L}_{rt}(\mathrm{MC}(\mathit{const})\text{-OCA})$. So, emptiness is undecidable for $\mathscr{L}_{rt}(\mathrm{MC}(\mathit{const})\text{-OCA})$ and we obtain the next theorem.

Theorem 6.6. *Emptiness, finiteness, infiniteness, equivalence, inclusion, regularity, and context-freeness are undecidable for real-time MC(const)-OCA.*

Moreover, even the restrictions itself are not decidable:

Theorem 6.7. *It is undecidable for an arbitrary real-time OCA whether it is a real-time MC(const)-OCA.*

So, even for the weakest non-trivial device with limited messages, that is, for real-time MC(*const*)-OCA all the mentioned properties are undecidable.

An approach often investigated and widely accepted is to consider a given type of device for special purposes only, for example, for the acceptance of languages having a certain structure or form. From this point of view it is natural to start with unary languages (for example, [1, 2, 11, 24, 25]). For general real-time one-way cellular automata it is known that they accept only regular unary languages [27]. Since the proof is constructive, we derive that the borderline between decidability and undecidability has been crossed. So, we generalize unary languages to bounded languages. For several devices it is known that they accept non-semilinear languages in general, but only semilinear bounded languages. Since for semilinear sets several properties are decidable [6], constructive proofs lead to decidable properties for these devices in connection with bounded languages [3, 8, 9, 10]. Example 6.2 witnesses the following theorem.

Theorem 6.8. *The language family $\mathscr{L}_{rt}(MC(const)\text{-}OCA)$ contains non-semilinear bounded languages.*

So, it is natural to consider decidability problems for the devices under consideration accepting *bounded* languages. In [16] it has been shown that also with this additional restriction none of the problems becomes decidable as long as the number of messages allowed is not too small.

Theorem 6.9. *Emptiness, finiteness, infiniteness, equivalence, inclusion, regularity, and context-freeness are undecidable for arbitrary real-time SC(n)-OCA and MC($\log n$)-OCA accepting* bounded *languages.*

7. Both Restrictions At Once

Since it turned out that neither limiting the bandwidth nor limiting the number of messages, even for bounded languages, reduces the computational capacity of the devices such that certain properties become decidable, we next combine both approaches and study real-time one-way cellular automata which are allowed to communicate only a fixed, finite number of messages per time step. Additionally, the communication links between two cells are allowed to be used constantly often only. In the most restricted case, we consider real-time one-way cellular automata which can communicate one message between two cells exactly once. The next example reveals that real-time MC(*const*)-OCA$_1$ accept non-regular context-free languages. More precisely, only two messages per cell are used.

Example 7.1. We describe the computation of an MC(*const*)-OCA$_1$ $\mathcal{M}$ on inputs of the form $a^n bc^m$. The acceptance of the language is governed by two signals. During the first time step the rightmost cell receives the message 1 associated with the boundary symbol and identifies itself to be the rightmost cell. During the second time step the cell with input symbol b (b-cell) sends a message. In this way, the unique a-cell with right neighboring b-cell can identify itself. Subsequently, the a-cell sends the message 1 with speed 1/2, and the rightmost cell sends the message 1 with maximal speed to the left. So, the slow signal starts at time step 2 in the rightmost a-cell, takes $2n-2$ further time steps to reach the leftmost cell, and thus stays at time steps $2n$ and $2n+1$ in the leftmost cell. The fast signal is set up at time step 1 and takes $n+m$ further time steps to reach the leftmost cell. When both signals meet in a cell, that is, $n+m+1 = 2n+1$, an accepting state is entered. Therefore, the leftmost cell accepts if and only if $n = m$. Moreover, each cell sends at most two messages, and can be set up to send no more messages even for inputs of another form.

Now assume that the language accepted by $\mathcal{M}$ is regular. Since regular languages are closed under intersection, so is the language $L(\mathcal{M}) \cap a^*bc^* = a^n bc^n$, which is non-regular but context free. ■

Turning to our key question, we present a lemma which relates languages accepted by real-time one-way cellular automata with a constant number of communications with languages accepted by real-time one-way cellular automata with a constant number of communications and one-message communication [18]. In this way, undecidability results can be derived from the undecidability results known.

Lemma 7.2. *Let* $\mathcal{M} = \langle S, F, A, B, \#, b_l, \delta\rangle$ *be a real-time MC(const)-OCA and* $\$ \notin A$ *be a new symbol. Then a real-time MC(const)-OCA*$_1$ $\mathcal{M}'$ *accepting the language* $\{\, w\$^{(|B|+2)(|w|+1)}v \mid v \in \{\$, A(A \cup \{\$\})^*\}, w \in L(\mathcal{M})\,\}$ *can effectively be constructed.*

It is straightforward to generalize this construction for cellular automata with two-way communication. Furthermore, the construction increases the number of communication steps per cell only linearly. Therefore, the construction also works fine for SC(f)-CA and MC(f)-CA where f is not necessarily a constant function. Now, we can use Lemma 7.2 in order to reduce the undecidability problems for MC(*const*)-OCA to MC(*const*)-OCA$_1$.

Theorem 7.3. *Emptiness, finiteness, infiniteness, equivalence, inclusion, regularity, and context-freeness are undecidable for arbitrary real-time MC(const)-OCA*$_1$.

Clearly, the undecidability carries over to, for example, MC(*const*)-CA$_k$ with two-way communication and the models SC(n)-OCA$_k$ and SC(n)-CA$_k$. The undecidability results for real-time SC(n)-OCA and MC($\log n$)-OCA accepting bounded languages cannot be translated directly to the corresponding automata with one communication symbol by using the construction of Lemma 7.2, since the construction does not preserve the boundedness of the languages. However, if we allow an additional communication symbol, then we obtain undecidability results also for bounded languages accepted by real-time SC(n)-OCA and MC($\log n$)-OCA with restricted communication alphabet of size two.

Lemma 7.4. *Let $\mathcal{M} = \langle S, F, A, B, \#, b_l, \delta\rangle$ be a real-time MC(const)-OCA and $\$ \notin A$ be a new symbol. Then a real-time MC(const)-OCA$_2$ $\mathcal{M}'$ accepting the language $\{w\$^{(|B|+2)(|w|+1)}\$ \mid w \in L(\mathcal{M})\}$ can effectively be constructed.*

It is obvious that $L(\mathcal{M}')$ is a bounded language if $L(\mathcal{M})$ is. Thus, we obtain the following undecidability results.

Theorem 7.5. *Emptiness, finiteness, infiniteness, inclusion, equivalence, regularity, and context-freeness are undecidable for arbitrary real-time SC(n)-OCA$_2$ and MC($\log n$)-OCA$_2$ accepting bounded languages.*

References

[1] Book, R.V.: Tally languages and complexity classes. Inform. Control **26** (1974) 186–193

[2] Chrobak, M.: Finite automata and unary languages. Theoret. Comput. Sci. **47** (1986) 149–158

[3] Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G.: Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach, Yverdon (1984)

[4] Cudia, D.F., Singletary, W.E.: Degrees of unsolvability in formal grammars. J. ACM **15** (1968) 680–692

[5] Fischer, P.C.: Generation of primes by a one-dimensional real-time iterative array. J. ACM **12** (1965) 388–394

[6] Ginsburg, S.: The Mathematical Theory of Context-Free Languages. McGraw Hill, New York (1966)

[7] Hartmanis, J.: Context-free languages and Turing machine computations. Proc. Symposia in Applied Mathematics **19** (1967) 42–51

[8] Ibarra, O.H.: Simple matrix languages. Inform. Control **17** (1970) 359–394

[9] Ibarra, O.H.: A note on semilinear sets and bounded-reversal multihead pushdown automata. Inform. Process. Lett. **3** (1974) 25–28

[10] Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. J. ACM **25** (1978) 116–133

[11] Klein, A., Kutrib, M.: Cellular devices and unary languages. Fund. Inform. **78** (2007) 343–368

[12] Kutrib, M.: Cellular automata and language theory. In: Encyclopedia of Complexity and System Science. Springer (2009) 800–823

[13] Kutrib, M., Malcher, A.: Cellular automata with limited inter-cell bandwidth. Theoret. Comput. Sci., to appear

[14] Kutrib, M., Malcher, A.: Fast cellular automata with restricted inter-cell communication: Computational capacity. In: Theoretical Computer Science (IFIP TCS2006). Volume 209 of IFIP, Springer (2006) 151–164

[15] Kutrib, M., Malcher, A.: Fast reversible language recognition using cellular automata. Inform. Comput. **206** (2008) 1142–1151

[16] Kutrib, M., Malcher, A.: Bounded languages meet cellular automata with sparse communication. In: Descriptional Complexity of Formal Systems (DCFS 2009), Otto-von-Guericke-Universität Magdeburg (2009) 211–222

[17] Kutrib, M., Malcher, A.: Computations and decidability of iterative arrays with restricted communication. Parallel Process. Lett. **19** (2009) 247–264

[18] Kutrib, M., Malcher, A.: On one-way one-bit O(one)-message cellular automata. Electron. Notes Theor. Comput. Sci. **252** (2009) 77–91

[19] Kutrib, M., Malcher, A.: Cellular automata with sparse communication. Theoret. Comput. Sci. **411** (2010) 3516–3526

[20] Malcher, A.: Descriptional complexity of cellular automata and decidability questions. J. Autom., Lang. Comb. **7** (2002) 549–560

[21] Malcher, A.: On the descriptional complexity of iterative arrays. IEICE Trans. Inf. Syst. **E87-D** (2004) 721–725

[22] Mazoyer, J.: A minimal time solution to the firing squad synchronization problem with only one bit of information exchanged. Technical Report TR 89-03, Ecole Normale Supérieure de Lyon (1989)

[23] Mazoyer, J., Terrier, V.: Signals in one-dimensional cellular automata. Theoret. Comput. Sci. **217** (1999) 53–80

[24] Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata. SIAM J. Comput. **30** (2001) 1976–1992

[25] Pighizzini, G., Shallit, J.O.: Unary language operations, state complexity and Jacobsthal's function. Int. J. Found. Comput. Sci. **13** (2002) 145–159

[26] Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)

[27] Seidel, S.R.: Language recognition and the synchronization of cellular automata. Technical Report 79-02, Department of Computer Science, University of Iowa (1979)

[28] Umeo, H., Kamikawa, N.: A design of real-time non-regular sequence generation algorithms and their implementations on cellular automata with 1-bit inter-cell communications. Fund. Inform. **52** (2002) 257–275

[29] Umeo, H., Kamikawa, N.: Real-time generation of primes by a 1-bit-communication cellular automaton. Fund. Inform. **58** (2003) 421–435

[30] Vollmar, R.: On cellular automata with a finite number of state changes. Computing **3** (1981) 181–191

[31] Vollmar, R.: Some remarks about the 'efficiency' of polyautomata. Internat. J. Theoret. Phys. **21** (1982) 1007–1015

[32] Worsch, T.: Linear time language recognition on cellular automata with restricted communication. In: LATIN 2000: Theoretical Informatics. Volume 1776 of LNCS, Springer (2000) 417–426

Journées Automates Cellulaires 2010 (Turku), pp. 31-42

A QUANTUM GAME OF LIFE

PABLO ARRIGHI [1] AND JONATHAN GRATTAGE [2]

[1] Université de Grenoble, LIG
E-mail address: pablo.arrighi@imag.fr
URL: http://membres-lig.imag.fr/arrighi

[2] École Normale Supérieure de Lyon, LIP
E-mail address: jonathan.grattage@ens-lyon.fr
URL: http://www.grattage.co.uk/jon

Abstract. This research describes a three dimensional quantum cellular automaton (QCA) which can simulate all other 3D QCA. This intrinsically universal QCA belongs to the simplest subclass of QCA: Partitioned QCA (PQCA). PQCA are QCA of a particular form, where incoming information is scattered by a fixed unitary U before being redistributed and rescattered. Our construction is minimal amongst PQCA, having block size $2 \times 2 \times 2$ and cell dimension 2. Signals, wires and gates emerge in an elegant fashion.

1. Introduction

(Quantum) Cellular Automata. Cellular automata (CA), first introduced by von Neumann [41], consist of an array of identical cells, each of which may take one of a finite number of possible states. The entire array evolves in discrete time steps by iterating a function G. This global evolution G is shift-invariant (it acts the same everywhere) and causal (information cannot be transmitted faster than some fixed number of cells per time step). Because this is a physics-like model of computation [21], Feynman [18], and later Margolus [22], suggested early in the development of quantum computation (QC) that quantising this model was important. This was for two main reasons. Firstly, in QCA computation occurs without extraneous (unnecessary) control, hence eliminating a source of decoherence, which is problem for QC implementation. Secondly, they are a good framework in which to study the quantum simulation of a quantum system, which is predicted to be a major application of QC. From a Computer Science perspective there are other reasons to study QCA, such as studying space-sensitive problems in computer science (*e.g.* 'machine self-reproduction', 'Firing Squad Synchronisation', ...) in the quantum setting, or having more complete and formal models of physics ability to compute. There is also the theoretical physics perspective, where CA are used as toy models of quantum space-time [20]. The first approach to defining QCA [2, 16, 42] was later superseded by a more axiomatic approach [9, 10, 35] together with more operational approaches [12, 27, 32, 33, 39, 42].

Key words and phrases: cellular automata, quantum computation, universality.

Intrinsic universality. Probably the most well known CA is Conway's 'Game of Life', a 2D CA which has been shown to be universal for computation, in the sense that any Turing Machine (TM) can be encoded within its initial state and then executed by evolution of the CA. As TM are often regarded as a robust definition of 'what an algorithm is' in classical computer science, this provides a key result in CA research. However, more can be achieved using CA than just running any algorithm. They run distributed algorithms in a distributed manner, model phenomena together with their spatial structure, and allow the use of the spatial parallelism inherent to the model. These features, modelled by CA and not by TM, are all worthy of investigation, and so the concept of universality should be revisited in this context to account for space. This is achieved by returning to the original meaning of the word *universality* [1, 11, 13], namely the ability for one instance of a computational model to be able to simulate other instances of the same computational model. Intrinsic universality formalises the ability of a CA to simulate another in a space-preserving manner [23, 30, 37], and has been extended to the quantum setting [4, 6, 7]. In previous work [7] it was shown that, when the dimension of space is greater than one, the problem of intrinsic universality reduces to the ability to code for signals, wires and a universal set of quantum gates, as expected from the classical CA case [30].

Related work. There are several related results in the current CA literature. For example, several works [21, 25, 26] provide computation universal Reversible Partitioned CA constructions, and [24] deals with their ability to simulate any CA in the one-dimensional case. The problem of minimal intrinsically universal CA has been addressed [31], and for Reversible CA (RCA) the issue of intrinsic universality has been tackled [14, 19]. The difficulty is in having an n-dimensional RCA simulate all other n-dimensional RCA and not, say, the $(n-1)$-dimensional RCA, otherwise a history-keeping dimension could be used, as shown by Toffoli [38]. Concerning QCA, Watrous [42] proved that QCA are universal in the sense of QTM. Shepherd, Franz and Werner [36] defined a class of QCA where the scattering unitary U_i changes at each step i (CCQCA). Universality in the circuit-sense has been achieved by Van Dam [39], Cirac and Vollbrecht [40], Nagaj and Wocjan [27] and Raussendorf [33]. In the bounded-size configurations case, circuit universality coincides with intrinsic universality, as noted by Van Dam [39]. QCA intrinsic universality in the one-dimensional case has been resolved [5], and also in the general $n > 1$-dimensional case [7]. Both results rely on recent work [6], where it was shown that a simple subclass of QCA, namely Partitioned QCA (PQCA), are intrinsically universal. This enables the focus here to be on this simple, natural class of QCA, as previously proposed ([33, 35, 39, 42] *etc.*). PQCA are QCA of a particular form, where incoming information is scattered by a fixed unitary U before being redistributed and rescattered. Hence the problem of finding an intrinsically universal PQCA is reduced to finding some scattering unitary U that supports the implementation of signals, wires and a universal set of quantum gates.

Game of Life. As the more general case of finding a general n-dimensional intrinsically universal QCA has been resolved [6], one could question the necessity of focusing on the particular 3D case of this problem. Our answer to this question is threefold.

- In the study of models of computation, the definition of the model is often followed by the search for a universal instance, and then for a minimal universal instance. Reaching this final step is generally regarded as a sign of maturity of the model. It shows that the entire model can be reduced to a particular simplest instance. Moreover, sometimes a particular instance stands out, either for its elegance and simplicity, or the richness of its behaviour. This was clearly the case with the Game of Life within the realm of Classical CA, and it is echoing this impression that we have named this QCA the 'Quantum Game of Life'. This is not because of a resemblance in the construction of the local rule, which clearly owes more to the Billiard Ball Model CA [19].
- Dimension three is particularly interesting, not only because of the relevance to physics, but because intrinsic universality QCA constructions lend themselves to a striking simplification. In dimension one the simplest known intrinsically universal QCA has cell dimension 36 [5]. In dimension two it has cell dimension 4 [7], and seems reducible to 3 via a costly and inelegant variation of the scheme [7], but probably no further. The solution we present here is in dimension three and has cell dimension 2; it is therefore minimal.
- Animations of the QCA operating, plus an implementation of this QCA, are available on the companion website [8]. Although it lacks a friendly way of preparing the initial configuration, it turns out to be quite interesting and entertaining already, as is the Game of Life. We plan to implement the QCA in a more graphical, interactive, computationally efficient fashion. The end result could be a pedagogical tool, both for discussing quantum theory and in order to start exploring the dynamical behaviour of QCA, such as in [28], perhaps using methods inspired by similar work on Probabilistic CA [34].

2. The Universal QCA

Definitions. Configurations hold the basic states of an entire array of cells, and hence denote the possible basic states of the entire QCA:

Definition 2.1 (Finite configurations)**.** A *(finite) configuration* c over Σ is a function $c : \mathbb{Z}^3 \longrightarrow \Sigma$, with $(i_1, i_2, i_3) \longmapsto c(i_1, i_2, i_3) = c_{i_1,i_2,i_3}$, such that there exists a (possibly empty) finite set I satisfying $(i_1, i_2, i_3) \notin I \Rightarrow c_{i_1,i_2,i_3} = q$, where q is a distinguished *quiescent* state of Σ. The set of all finite configurations over Σ will be denoted $\mathcal{C}_f^\Sigma$.

As this work relates to QCA rather than classical CA, the global state can be a superposition of these configurations. To construct the separable Hilbert space of superpositions of configurations the set of configurations must be countable. This is why finite, unbounded, configurations are considered; the quiescent state of a CA is analogous to the blank symbol of a TM tape.

Definition 2.2 (Superpositions of configurations)**.** Let $\mathcal{H}_{\mathcal{C}_f^\Sigma}$ be the Hilbert space of configurations. Each finite configuration c is associated with a unit vector $|c\rangle$, such that the family $(|c\rangle)_{c\in\mathcal{C}_f^\Sigma}$ is an orthonormal basis of $\mathcal{H}_{\mathcal{C}_f^\Sigma}$. A *superposition of configurations* is then a unit vector in $\mathcal{H}_{\mathcal{C}_f^\Sigma}$.

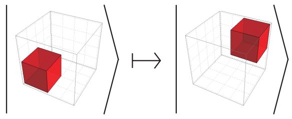


Figure 1: Signals propagate diagonally across the partition.

Detailed explanations of these definitions, as well as axiomatic definitions of QCA, are available [9, 10, 35]. Building upon these works, we have shown [6, 7] that Partitioned QCA (PQCA) are intrinsically universal. Since they are the most canonical description of QCA, and since the aim of this paper is to construct the most canonical, and yet universal, QCA, we will assume that all QCA are PQCA throughout this work.

Definition 2.3 (Partitioned QCA)**.** A partitioned three-dimensional quantum cellular automaton (PQCA) is defined by a *scattering unitary*, a unitary operator, U such that $U : \mathcal{H}_\Sigma^{\otimes 2^3} \longrightarrow \mathcal{H}_\Sigma^{\otimes 2^3}$, and $U\,|qq\ldots qq\rangle = |qq\ldots qq\rangle$, *i.e.* that takes a cube of 2^3 cells into a cube of 2^3 cells and preserves quiescence. Consider $G = (\bigotimes_{2\mathbb{Z}^3} U)$, the operator over $\mathcal{H}$. The induced global evolution is G at odd time steps, and σG at even time steps, where σ is a translation by one unit in all directions.

Here we provide a specific instance of a U-defined PQCA which is capable of intrinsically simulating any V-defined PQCA, for any V. In order to describe such a U-defined PQCA in detail, two things are required: the dimensionality of the cells (including the meaning attached to each of the states they may take); and the way in which the scattering unitary U acts upon these cells.

2.1. Signals and wires

Classical CA studies often refer to ‘signals’ without an explicit definition. In this context, a signal refers to the state of a cell which may move to a neighbouring cell consistently, from one step to another, by the evolution of the CA. Therefore a signal would appear as a line in the space-time diagram of the CA.

The U-defined PQCA constructed in this paper is minimal, in the sense that it is a 3D PQCA with the smallest non-trivial block-size and cell-dimension. Hence, each cell has only two possible states, either empty or occupied; it is a binary automaton, with scattering unitary U acting on $2 \times 2 \times 2$ cell neighbourhoods (a cube). The description of the scattering unitary U that defines the behaviour of the automaton will now be provided in a case by case manner.

2.1.1. *Signals.* Signals in our scheme will be encoded as isolated occupied cells. If a neighbourhood trivially contains no signals, then no change is made. If a neighbourhood contains only one signal, *i.e.* only one cell is occupied, and the other seven are empty, then the signal is shifted by one in every direction, yielding a 3D diagonal propagation. This rule is given in Fig. 1. These two rules, together with their rotations, define the action of the scattering unitary U upon the subspace with zero or one occupied cells. Notice that U can easily be seen to be unitary upon this subspace, as it performs a permutation of the basis states.

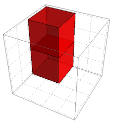


Figure 2: Two occupied cells which share one face form a static barrier.

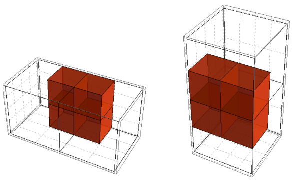


Figure 3: Two vertical barriers placed in adjacent partitions horizontally form a $2 \times 2 \times 1$ barrier (*left*). At the next time step, the repartitioning is still stable, with two horizontal barriers in adjacent vertical partitions formed from those same cells (*right*).

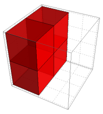


Figure 4: Four occupied cells occupying a face of the partition neighbourhood form a wall.

2.1.2. *Barriers.* Two or more occupied cells that share a common face form a barrier, which is intended to be a stationary pattern. The rule is given in Fig. 2. Such a barrier is stable for one partition, but in the next partition it appears as two signals moving away from each other, and hence it will scatter. This can be avoided by extending beyond the current partition, using at least four occupied cells. For example, a $2 \times 2 \times 1$ block of cells which overlaps a partition and forms a barrier in each partition, as shown in Fig. 3, is stable. This rule, together with its rotations, defines the action of the scattering unitary U upon the subspace of two adjacent occupied cells. Upon this subspace the scattering unitary U acts like the identity, which is unitary.

2.1.3. *Walls.* When two barriers, made of four occupied cells, form a square that fits within a partition, as in Fig. 4, we say that they form a wall. A wall is able to redirect a fifth occupied cell, that shares only one face with the wall, which represents the incoming signal. Again a wall in one partition appears as signals moving away from each other in the next partition, but again they can be made stable following Fig. 3, by extending them across partitions. The behaviour required is for an incoming signal to "bounce" off the wall, and the rule producing this behaviour is given in Fig. 5. This rule suffices to define signal redirection, or "rewiring". These rules, together with their rotations, define the action of the scattering unitary U upon the subspace of walls and walls with a signal. Notice that U is unitary upon this subspace, as it performs a permutation of the basis states.

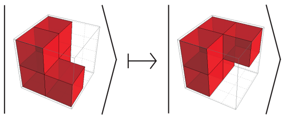


Figure 5: A signal which interacts with a wall in a neighbourhood "bounces" off the wall, causing it to change direction along one axis. Compare with the unhindered signal propagation shown in Fig. 1.

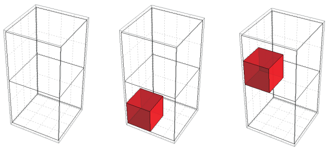


Figure 6: Qubits are implemented as parallel tracks of signals. No qubit is modelled as no signal (*left*), $|0\rangle$ is modelled as a signal on the lower track (*centre*), while a signal on the upper track models a qubit in state $|1\rangle$ (*right*).

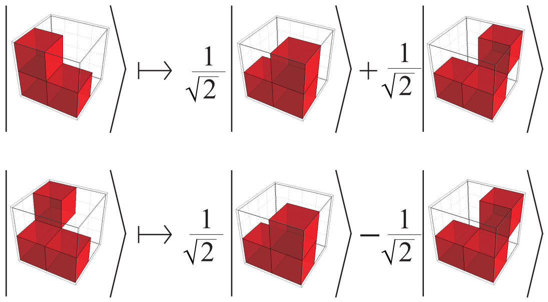

↦ $\frac{1}{\sqrt{2}}$ + $\frac{1}{\sqrt{2}}$
↦ $\frac{1}{\sqrt{2}}$ − $\frac{1}{\sqrt{2}}$

Figure 7: The rules implementing the Hadamard operation. The first gives $|0\rangle \mapsto \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ while the second gives $|1\rangle \mapsto \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$.

2.2. Qubits and quantum gates

To allow a universal set of gates to be implemented by the PQCA, certain combinations of signals and barriers need to be given special importance. To implement qubits, pairs of signals will be used. Indeed in our scheme a qubit is formed by two parallel "tracks" of signals, as shown in Fig. 6. A signal on the bottom track indicates a $|0\rangle$ state, whereas the top track indicates a $|1\rangle$ state. Qubits in superpositions are modelled by appropriate superpositions of the configurations, as in Def. 2.2. We can now define rules which implement quantum gates on those qubits[1].

2.2.1. *Hadamard.* The Hadamard gate is a requirement for universal quantum computation, where $H : |0\rangle \mapsto \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, |1\rangle \mapsto \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$. In order to achieve this typically quantum behaviour, a special meaning is attached to the interaction of a signal with a single two-cell barrier (as defined in section 2.1.2). The corresponding rule is given by Fig. 7.

[1] In Quantum Mechanics, a qubit can be encoded into any 2 degrees of freedom of a system. Some experiments in quantum optics, for example, use the spin degree of freedom, whereas others use a spatial degree of freedom. This model is analogous to the latter case. See the conclusion section for a discussion of relevance to physical systems.

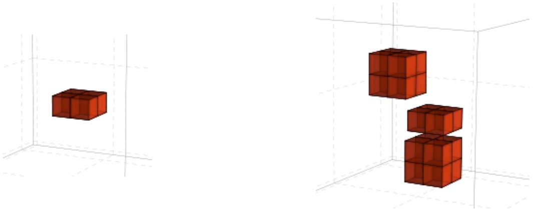


Figure 8: A stable Hadamard configuration, formed by two barriers extended across partitions (*left*). The Hadamard configuration can be extended with two cubed "walls". The first cube, on the top left, redirects the top track $|1\rangle$ signal, incoming from the bottom left and travelling to the far top right, into the barrier. The second cube, bottom right, redirects the output bottom $|0\rangle$ track so that it is again moving parallel to the $|1\rangle$ track, keeping the qubit representation consistent (*right*).

As all the rules we provide are rotation invariant, the unambiguity of the Hadamard rule may not be clear. To be unambiguous, the $|0\rangle$ and $|1\rangle$ cases need be to distinguished from each other, and the signal distinguishable from the barrier. This can be done as follows: if the cube can be rotated such that all three signals are on a single face, forming an "L" shape, then the input to the Hadamard is $|0\rangle$, and the signal is at the top of the L. If instead a lone occupied cell is on the top left of the furthest face, with a two-cell barrier along the bottom of the closest face, forming a dislocated L shape, then the input to the Hadamard is $|1\rangle$ and the signal is the isolated, top left cell.

On its own, a barrier is not enough to implement the Hadamard gate, because it does not respect the parallel track convention of qubits defined earlier. Rather, inputs and outputs are orthogonal. To rectify this, the barrier can be combined with two walls, which redirect the signals appropriately, as shown in Fig. 8. Further rewiring can be added so the qubit can arrive and leave in any direction, but these are not easily represented in 2D projections.

These two rules, together with their rotations, define the action of the scattering unitary U upon the subspace of two-cell barriers plus a signal. Notice that U is unitary upon this subspace, as it is a composition of a Hadamard gate upon the input states, together with the permutation of the basis states that perform the movement described in Fig. 7.

2.2.2. *Controlled rotations.* A two qubit controlled gate is another requirement for universal quantum computation, and in this case we choose the controlled-R($\frac{\pi}{4}$)gate, where cR$(\pi/4)$: $|11\rangle \mapsto e^{\frac{i\pi}{4}}|11\rangle$, and is the identity otherwise. To encode a two qubit controlled gate, signal collisions, where two moving signals cross each other diagonally, will be interpreted as adding a global phase of $e^{\frac{i\pi}{4}}$. This rule is given by Fig. 9.

This allows a controlled-R($\frac{\pi}{4}$) operation to be defined, by redirecting the $|1\rangle$ (true) signal track so that the tracks will cross each other in this way, and then recreating the qubits by rejoining the $|1\rangle$ tracks back with the appropriate $|0\rangle$ tracks, using walls to redirect and delay signals as in the Hadamard case. Again, this 3D configuration cannot easily be presented in 2D.

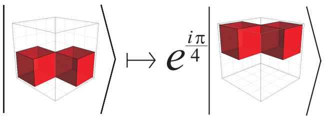


Figure 9: When two signals cross each other diagonally, a complex phase is added to the configuration state.

This same rule can also be used to implement a single qubit phase change operation by causing a control signal to loop such that it will intersect with the true track of the qubit, thus adding the global phase to the configuration. This needs to be correctly timed so that the tile implementing the single qubit rotation can be iterated, following the same reasoning as in [7].

This rule defines, together with its rotations, the action of the scattering unitary U upon the subspace of two non-adjacent signals on face. U can again be seen to be unitary upon this subspace, as it is a permutation of the basis states, with a phase.

2.3. Universality, erosion and dynamics

We have seen that the scattering unitary U is a unitary on the subspaces upon which it has been defined. Additionally, following arguments given previously [7], it may be concluded that the U-defined PQCA given here is intrinsically universal:

- In space dimension greater than one, it suffices to implement signals, wires and a universal set of gates to achieve intrinsic universality;
- The controlled-R($\frac{\pi}{4}$) and the Hadamard gate are universal for Quantum Computation.

Indeed the standard set of cNot, H, R($\frac{\pi}{4}$) can be recovered as follows:

$$\textsc{cNot}\,|\psi\rangle = (\mathbb{I}\otimes H)(\textsc{c}\mathrm{R}(\pi/4))^4(\mathbb{I}\otimes H)\,|\psi\rangle$$

where $\textsc{c}\mathrm{R}(\frac{\pi}{4})^4$ denotes four applications of the controlled-R($\frac{\pi}{4}$) gate, giving the controlled-Phase operation.

We could let the action of the scattering unitary U be the identity in all other subspaces. However, some patterns would then be unmovable and indestructible, such as the wall of Fig. 4. This would arguably make the dynamics of the QCA presented here a little plain. Moreover, since walls could be built through simultaneous signal collision, it would be natural to be able to dismantle them. If we are interested in the dynamics, different ways of completing the definition of U upon the remaining subspaces should be considered. A natural way is to consider that all of the other cases are made of non-interacting signals, where the occupied cells propagate past each other diagonally. This allows, for instance, for the demolition of walls and the creation of new stable patterns.

Animated examples and an implementation of this cellular automaton can be found online [8]. We hope to explore its dynamics in future simulations.

3. Comparison with Two-Dimensional Rules & Tiles

In previous work we provided a generic construction of an n-dimensional intrinsically universal QCA [7]. In this section we will compare the two dimensional instance given previously with the scheme presented here.

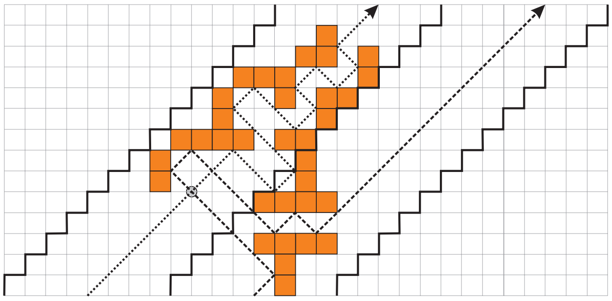


Figure 10: The 2D 'controlled-R($\frac{\pi}{4}$) gate' tile, with a signal interaction at the highlighted cell.

Objects. In [7] the cell-dimension is 4. A cell can be either empty, a wall, or a binary signal (0 and 1). In a sense the same objects are recovered in the scheme presented here, but as the cell dimension is 2 these are conveyed as patterns of cells. Walls are now coded as 2^2 squares as in Fig. 4. Signals no longer have a bit attached to them; instead the location of the signal is used to encode this bit of information.

Rules. Once the above observations have been made; it can be seen that the two dimensional, cell-dimension four scheme previously described [7] is almost a projection of the scheme presented here. Indeed:

- the 2D signal travelling rule of [7]; $\left|\begin{array}{|c|c|}\hline & \\ \hline s & \\ \hline\end{array}\right\rangle \mapsto \left|\begin{array}{|c|c|}\hline & s \\ \hline & \\ \hline\end{array}\right\rangle$, is a height-projection of the corresponding rule in Fig. 1;
- the 2D signal bouncing rule; $\left|\begin{array}{|c|c|}\hline \blacksquare & \blacksquare \\ \hline \mathrm{s} & \\ \hline\end{array}\right\rangle \mapsto \left|\begin{array}{|c|c|}\hline \blacksquare & \blacksquare \\ \hline & s \\ \hline\end{array}\right\rangle$, again a height-projection of the corresponding rule in Fig. 5;
- the 2D control-phase rule;

$$\left|\begin{array}{|c|c|}\hline 1 & \\ \hline 1 & \\ \hline\end{array}\right\rangle \mapsto e^{\frac{i\pi}{4}} \left|\begin{array}{|c|c|}\hline & 1 \\ \hline & 1 \\ \hline\end{array}\right\rangle, \quad \left|\begin{array}{|c|c|}\hline \mathrm{x} & \\ \hline \mathrm{y} & \\ \hline\end{array}\right\rangle \mapsto \left|\begin{array}{|c|c|}\hline & \mathrm{y} \\ \hline & \mathrm{x} \\ \hline\end{array}\right\rangle \textit{ otherwise},$$

is a width-depth projection of the corresponding rule in Fig. 9;
- only the 2D Hadamard rule of [7] differs slightly:

$$\left|\begin{array}{|c|c|}\hline \blacksquare & \\ \hline 0 & \blacksquare \\ \hline\end{array}\right\rangle \mapsto \frac{1}{\sqrt{2}} \left|\begin{array}{|c|c|}\hline \blacksquare & 0 \\ \hline & \blacksquare \\ \hline\end{array}\right\rangle + \frac{1}{\sqrt{2}} \left|\begin{array}{|c|c|}\hline \blacksquare & 1 \\ \hline & \blacksquare \\ \hline\end{array}\right\rangle$$

$$\left|\begin{array}{|c|c|}\hline \blacksquare & \\ \hline 1 & \blacksquare \\ \hline\end{array}\right\rangle \mapsto \frac{1}{\sqrt{2}} \left|\begin{array}{|c|c|}\hline \blacksquare & 0 \\ \hline & \blacksquare \\ \hline\end{array}\right\rangle - \frac{1}{\sqrt{2}} \left|\begin{array}{|c|c|}\hline \blacksquare & 1 \\ \hline & \blacksquare \\ \hline\end{array}\right\rangle$$

Tiles. Previously, not only were the above rules provided, but a set of fixed-sized (16×14) tiles, taking a fixed number of time steps (24) to be travelled through, and each accomplishing one of the universal quantum gates upon the incoming signals were also given [7]. The example of the controlled-R($\frac{\pi}{4}$) tile is reproduced in Fig. 10 for concreteness. The difference between such tiles and the corresponding rule (the controlled-R($\frac{\pi}{4}$) rule in this case) is that tiles need only to be placed next to one another so as to make up a circuit, with all signal wiring and synchronisations being accounted for. It would be cumbersome to describe the equivalent of all these tiles in the scheme presented here, particularly due to their 3D nature. Some are shown on the companion website [8]. The reader may convince himself of their feasibility through the following arguments: since those tiles exist in 2D, and since the 2D rules are projections of the 3D rules, there must exist some 3D tiles for which the 2D ones are, in some sense, projections.

4. Conclusion

This paper presents a minimal 3D PQCA which is capable of simulating all other PQCA, preserving the topology of the simulated PQCA. This means that the initial configuration and the forward evolution of any PQCA can be encoded within the initial configuration of this PQCA, with each simulated cell encoded as a group of adjacent cells in the PQCA, *i.e.* intrinsic simulation. The main, formal result of this work can therefore be stated as:

Claim 4.1. There exists an 3D U-defined PQCA, with block size 2 and cell dimension 2, which is an intrinsically universal PQCA. Let H be a 3-dimensional V-defined PQCA such that V can be expressed as a quantum circuit C made of gates from the set Hadamard, Cnot, and $\mathrm{R}(\frac{\pi}{4})$. Then G is able to intrinsically simulate H.

Any finite-dimensional unitary V can always be approximated by a circuit $C(V)$ with an arbitrary small error $\varepsilon = \max_{|\psi\rangle} ||V\,|\psi\rangle - C\,|\psi\rangle\,||$. Assuming instead that G simulates the $C(V)$-defined PQCA, for a region of s cells over a period t, the error with respect to the V-defined PQCA will be bounded by $st\varepsilon$. This is due to the general statement that errors in quantum circuits increase, at most proportionally with time and space [29]. Combined with the fact that PQCA are universal [7, 6], this means that G is intrinsically universal, up to this unavoidable approximation.

Discussion. This PQCA is definitely minimal amongst the 3D PQCA. Reducing the block size or the cell dimension necessarily results in a trivial PQCA. Moreover, PQCA are the simplest and most natural class of QCA [3]. Nevertheless, it is not so clear that this PQCA is minimal amongst all intrinsically universal 3D QCA. Indeed, PQCA-cells are really QCA-subcells, and PQCA-blocks need to be composed with them shifted in order to yield the QCA neighbourhood. Based on these observations our QCA has QCA-cell dimension 2^8 and the 3D radius–$\frac{1}{2}$ neighbourhood. Whether there are some further simplifications in this more general setting is an open question.

An important source of inspiration, along with the original Game of Life, was the the simplest known intrinsically universal classical Partitioned CA [22], which has cell dimension 2. Called the BBM CA, it was itself directly inspired by the Billiard Ball Model [19]. Our scheme also features signals (billiard balls, or particles) bouncing and colliding. This analogy with physics is a desirable feature; Feynman's sole purpose for the invention of QC [17] was that quantum computers would be able to simulate quantum systems efficiently, and hence predict their behaviour. It is still thought that quantum simulation will be one of most important uses of quantum computers for society, with expected and potentially unexpected impact in quantum chemistry, biochemistry or nanotechnologies (*e.g.* in terms of the synthesis of specific purpose molecules). This was also one of the reasons why Feynman immediately promoted the QCA model of QC [18]; they are a natural mathematical setting in which to encode the continuous-time and space, sometimes complex, behaviour of elementary physical elements (such as wave equations of particles, possibly interacting, under some field, for example) into the more discrete-data discrete-time framework of quantum computation. This issue of encoding is non-trivial, but has largely been solved for the one-dimensional case, by using QCA as a mathematical framework in which to model the system of interest. For example, several works simulate quantum field theoretical equations in the continuous limit of a QCA dynamics (such as [32], *etc.*) These results do not extend to more-than-one dimensions due to the problem of isotropy. Whilst in one-dimension the division of space into a line of points is an acceptable model, because it does not privilege the left over the right direction, in two-dimensions the grid seems to inevitably favour the cardinal directions (N, E, S, W) over the diagonal ones. That this construction is similar to the Billiard Ball Model CA suggests a Quantum Billiard-Ball Model could be defined, moving away from the underlying grid, which would remedy this problem. This is planned this for future work.

Acknowledgements

The authors would like to thank Jérôme Durand-Lose, Jarkko Kari, Simon Martiel, Kenichi Morita, Guillaume Theyssier and Philippe Jorrand.

References

[1] J. Albert and K. Culik. A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, 1:1–16, 1987.

[2] P. Arrighi. Algebraic characterizations of unitary linear quantum cellular automata. In *Proceedings of MFCS, LNCS*, volume 4162, page 122. Springer, 2006.

[3] P. Arrighi. Quantum cellular automata. Thèse d'habilitation à diriger les recherches, University of Grenoble, 2009.

[4] P. Arrighi and R. Fargetton. Intrinsically universal one-dimensional quantum cellular automata. In *Proceedings of DCM*, 2007.

[5] P. Arrighi, R. Fargetton, and Z. Wang. Intrinsically universal one-dimensional quantum cellular automata in two flavours. *Fundamenta Informaticae*, 21:1001–1035, 2009.

[6] P. Arrighi and J. Grattage. Partitioned quantum cellular automata are intrinsically universal. Accepted for publication, Post-proceedings of the Physics and Computation workshop, 2009.

[7] P. Arrighi and J. Grattage. A Simple n-Dimensional Intrinsically Universal Quantum Cellular Automaton. *LATA 2010, LNCS*, 6031:70–81, 2010.

[8] P. Arrighi and J. Grattage. Companion website: www.grattage.co.uk/jon/3DQCA, 2010.

[9] P. Arrighi, V. Nesme, and R. Werner. Unitarity plus causality implies localizability. *QIP 2010 and Journal of Computer and System Sciences, ArXiv preprint: arXiv:0711.3975*, 2010.

[10] P. Arrighi, V. Nesme, and R. F. Werner. Quantum cellular automata over finite, unbounded configurations. In *Proceedings of MFCS, LNCS*, volume 5196, pages 64–75. Springer, 2008.

[11] E. R. Banks. Universality in cellular automata. In *SWAT '70: Proceedings of the 11th Annual Symposium on Switching and Automata Theory (SWAT 1970)*, pages 194–215, Washington, DC, USA, 1970. IEEE Computer Society.

[12] G. K. Brennen and J. E. Williams. Entanglement dynamics in one-dimensional quantum cellular automata. *Phys. Rev. A*, 68(4):042311, Oct 2003.

[13] B. Durand and Z. Roka. The Game of Life: universality revisited Research Report 98-01. Technical report, Ecole Normale Suprieure de Lyon, 1998.

[14] J. O. Durand-Lose. Reversible cellular automaton able to simulate any other reversible one using partitioning automata. In *In LATIN'95: Theoretical Informatics, number 911 in LNCS*, pages 230–244. Springer, 1995.

[15] J. O. Durand-Lose. Intrinsic universality of a 1-dimensional reversible cellular automaton. In *Proceedings of STACS 97, LNCS*, page 439. Springer, 1997.

[16] C. Durr, H. Le Thanh, and M. Santha. A decision procedure for well-formed linear quantum cellular automata. In *Proceedings of STACS 96, LNCS*, pages 281–292. Springer, 1996.

[17] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1982.

[18] R. P. Feynman. Quantum mechanical computers. *Foundations of Physics (Historical Archive)*, 16(6):507–531, 1986.

[19] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3):219–253, 1982.

[20] S. Lloyd. A theory of quantum gravity based on quantum computation. ArXiv preprint: quant-ph/0501135, 2005.

[21] N. Margolus. Physics-like models of computation. *Physica D: Nonlinear Phenomena*, 10(1-2), 1984.

[22] N. Margolus. Parallel quantum computation. In *Complexity, Entropy, and the Physics of Information: The Proceedings of the 1988 Workshop on Complexity, Entropy, and the Physics of Information Held May-June, 1989, in Santa Fe, New Mexico*, page 273. Perseus Books, 1990.

[23] J. Mazoyer and I. Rapaport. Inducing an order on cellular automata by a grouping operation. In *Proceedings of STACS'98, in LNCS*, volume 1373, pages 116–127. Springer, 1998.

[24] K. Morita. Reversible simulation of one-dimensional irreversible cellular automata. *Theoretical Computer Science*, 148(1):157–163, 1995.

[25] K. Morita and M. Harao. Computation universality of one-dimensional reversible (injective) cellular automata. *IEICE Trans. Inf. & Syst., E*, 72:758–762, 1989.

[26] K. Morita and S. Ueno. Computation-universal models of two-dimensional 16-state reversible cellular automata. *IEICE Trans. Inf. & Syst., E*, 75:141–147, 1992.

[27] D. Nagaj and P. Wocjan. Hamiltonian Quantum Cellular Automata in 1D. ArXiv preprint: arXiv:0802.0886, 2008.

[28] V. Nesme and J. Gütschow. On the fractal structure of the space-time diagrams of clifford cellular automata. manuscript, 2009.

[29] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, October 2000.

[30] N. Ollinger. Universalities in cellular automata a (short) survey. In B. Durand, editor, *First Symposium on Cellular Automata "Journées Automates Cellulaires" (JAC 2008). Proceedings*, pages 102–118. MCCME Publishing House, Moscow, 2008.

[31] N. Ollinger and G. Richard. A Particular Universal Cellular Automaton. In EPTCS 1, 2009, pp. 205-214, pages 205–214.

[32] C.A. Pérez-Delgado and D. Cheung. Local unreversible cellular automaton ableitary quantum cellular automata. *Physical Review A*, 76(3):32320, 2007.

[33] R. Raussendorf. Quantum cellular automaton for universal quantum computation. *Physical Review A*, 72(2):22301, 2005.

[34] D. Regnault, N. Schabanel, and É. Thierry. Progresses in the analysis of stochastic 2D cellular automata: a study of asynchronous 2D minority. *Theoretical Computer Science*, 410(47-49):4844–4855, 2009.

[35] B. Schumacher and R. Werner. Reversible quantum cellular automata. ArXiv pre-print quant-ph/0405174, 2004.

[36] D. J. Shepherd, T. Franz, and R. F. Werner. A universally programmable quantum cellular automata. *Phys. Rev. Lett.*, 97(020502), 2006.

[37] G. Theyssier. Captive cellular automata. In *Proceedings of MFCS 2004, in LNCS*, volume 3153, pages 427–438. Springer, 2004.

[38] T. Toffoli. Computation and construction universality of reversible cellular automata. *J. of Computer and System Sciences*, 15(2), 1977.

[39] W. Van Dam. Quantum cellular automata. Masters thesis, University of Nijmegen, The Netherlands, 1996.

[40] K. G. H. Vollbrecht and J. I. Cirac. Reversible universal quantum computation within translation-invariant systems. *New J. Phys Rev A*, 73:012324, 2004.

[41] J. von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Champaign, IL, USA, 1966.

[42] J. Watrous. On one-dimensional quantum cellular automata. *Complex Systems*, 5(1):19–30, 1991.

Journées Automates Cellulaires 2010 (Turku), pp. 43-53

THE BLOCK NEIGHBORHOOD

PABLO ARRIGHI [1] AND VINCENT NESME [2]

[1] Université de Grenoble, LIG, 220 rue de la chimie, 38400 Saint-Martin-d'Hères, France
E-mail address: pablo.arrighi@imag.fr
URL: http://membres-lig.imag.fr/arrighi/

[2] Quantum information theory, Universität Potsdam, Karl-Liebknecht-Str. 24/25, 14476 Potsdam, Germany
E-mail address: vnesme@gmail.com
URL: http://www.itp.uni-hannover.de/~nesme/

Abstract. We define the block neighborhood of a reversible CA, which is related both to its decomposition into a product of block permutations and to quantum computing. We give a purely combinatorial characterization of the block neighborhood, which helps in two ways. First, it makes the computation of the block neighbourhood of a given CA relatively easy. Second, it allows us to derive upper bounds on the block neighborhood: for a single CA as function of the classical and inverse neighborhoods, and for the composition of several CAs. One consequence of that is a characterization of a class of "elementary" CAs that cannot be written as the composition of two simpler parts whose neighborhoods and inverse neighborhoods would be reduced by one half.

Introduction

Otherwise decent people have been known to consider reversible cellular automata (RCAs) and look for ways to decompose them into a product of reversible blocks permutations. One big incentive for doing so is to ensure structural reversibility, as was the concern in [Mar84], as it helps to design RCAs (see for instance [MH89, MU92]), whereas determining from its local transition function whether a CA is reversible is undecidable [Kar90].

Sadly, the relation is not clearly understood between both frameworks; several articles tackle this problem [Kar96, Kar99, DL01], whose conclusion, in a nutshell, is the following. It is always possible, by increasing the size of the alphabet, to simulate a d-dimensional CA by a reversible block CA of depth at most $d+1$. In the case of dimensions 1 and 2, up to shifts, no additional space and no coding is needed; it is still an open problem whether the same can be said in higher dimensions.

We will be here concerned with the size of the blocks, or rather, with the information on the neighborhood that is deducible purely geometrically from a block structure decomposition.

2000 ACM Subject Classification: F.1.1.
Key words and phrases: cellular automata, neighborhood, quantum, block representation.

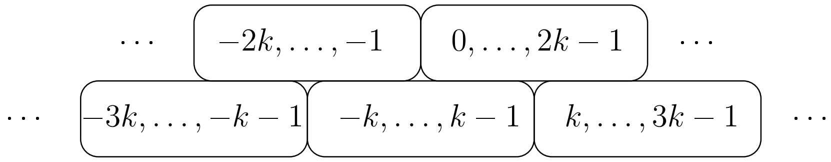


Figure 1: The geometric neighborhood in a block structure.

If we just know that the CA is defined by such a structure, we can deduce, for instance, that the cell 0 has an influence only on the cells $-2k,\ldots,2k-1$, which means the neighborhood of this CA has to be included in $[\![-2k+1;2k]\!]$. But it is also true that the cells $-k$ and $k-1$ influence only the cells $-2k,\ldots,2k-1$, so the translation invariance tells us more: we can deduce that the neighborhood of this CA is included in $[\![-k;k]\!]$. Another way to look at it is to modify slightly the block structure, and update cell 0 once and for good on the first step, so that the new structure would look something like Figure 2.

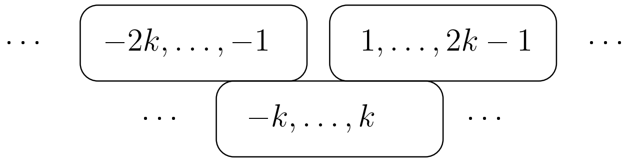


Figure 2: Block structure with a tooth gap.

If we concentrate only on the central block on the first line and the fact that no block of the second line acts on cell 0 and ask what can then be the minimal size of the central block, we get to Figure 3 and our definition of the block neighborhood in Definition 1.4. Section 1 is devoted to the basic properties of this neighborhood, in particular Proposition 1.5 gives an expression of it in terms of combinatorics on words.

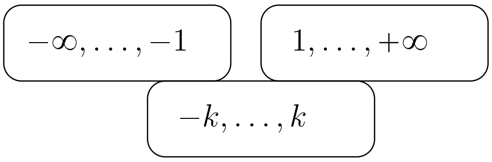


Figure 3: Simplified block structure.

So, how large must this central block be? Since it does all the work updating the state of cell 0, it should at least include the neighborhood of this cell. But there is a dual way to look at Figure 2 when it is turned upside-down. What we know see is a block decomposition of the inverse CA, where the first step updates the complement of $[\![-k;k]\!]$, so we also have a condition involving the neighborhood of the inverse CA, which has little to no relation to that of the CA itself. Hence, there is something non trivial to say about that, and these considerations will be developed in Section 2, where the two results needed for bounding block neighborhoods are stated. Proposition 2.1 is not new — although it is the first time that it is put in direct relation with block decomposition — but Corollary 2.2 and Proposition 2.3 are.

A last caveat: while this article is written in a purely classical perspective, everything it deals with also has to do with quantum CAs (QCAs). The definition of QCAs we are dealing with was introduced in [SW04] as the natural extension of the usual definition of CAs to a universe ruled by reversible quantum laws. It is founded on the same principles that rule usual CAs: discrete space-time, translation invariance, locality; in particular, QCAs have a similar notion of neighborhood. It was already proven in that first article that reversible CAs can be naturally embedded into a quantum setting, turning them into QCAs. However, curiously enough, the neighborhood of these QCAs — the quantum neighborhood — was not shown to be equal to that of the original CAs; rather, a nontrivial bound was given (which is to be found as Proposition 2.1 of the present article). It was then made explicit in [ANW08] that the quantum neighborhood can indeed, and typically will, be strictly larger than the original one.

The authors tried to translate into purely classical terms a definition of the quantum neighborhood of quantized reversible CAs, and found the expression of Proposition 1.5, before realizing the close connection to block structures. In retrospect, the link is hardly surprising, since a construction was given in [ANW] that uses auxiliary space to write a CA in a block structure, where each block acts on exactly the quantum neighborhood — the construction is given in the quantum case, but applies to the classical case, mutatis mutandis. Notions such as semicausality (Definition 1.3) and semilocalizability (Definition 1.4) are also imported from the quantum world, cf. [ESW02].

So, in good conscience, the block neighborhood could be called the quantum neighborhood, but since in the final version no explicit reference to the quantum model needs to be made, the name sounded a bit silly. Nevertheless, if other natural neighborhoods were to be defined in relation to block structures, let it be said that the neighborhood we define and study in this article will always deserve “quantum” as a qualifier.

Notations

- Σ is the alphabet.
- $a.b$ denotes the concatenation of words a and b.
- $a|_X$ is the restriction of word a on a subset of indices X.
- $a = b|_X$ means that words a and b coincide on X.
- $\bar{X}$ denotes the complement of X, usually in $\mathbb{Z}$.
- For $A, B \subseteq \mathbb{Z}$, $A + B$ is their Minkowski sum $\{a + b \mid a \in A, b \in B\}$; similarly with $A - B$.
- $[\![x; y]\!]$ is the integer interval $[x; y] \cap \mathbb{Z}$.
- fg denotes the composition of CAs f and g.
- $\bigstar$ denotes the operation reversing the order in a tuple : $\bigstar(x_1, x_2, \ldots, x_n) = (x_n, x_{n-1}, \ldots, x_1)$. It acts similarly on $\Sigma^{\mathbb{Z}}$ by $\bigstar(a)_n = a_{-n}$.

1. Definitions

Definition 1.1. For a bijection f whose domain and range are written as products, its dual is defined by $\tilde{f} = \bigstar f^{-1} \bigstar$. This applies in particular to the case where f is a CA. In this case $\tilde{f}$ is the conjugation of f by the central symmetry.

For instance, shifts are self-dual. Clearly, $f \mapsto \tilde{f}$ is an involution. In the remainder of this article, each time a notion (like a function or a property) is defined in term of a CA f, its dual, denoted by adding a tilde, is defined in the same way in term of $\tilde{f}$.

Definition 1.2. The (classic) neighborhood $\mathcal{N}(f)$ is the smallest subset A of $\mathbb{Z}$ such that $v|_A$ determines $f(v)|_0$.

The dual neighborhood $\tilde{\mathcal{N}}$ is thus defined by $\tilde{\mathcal{N}}(f) = \mathcal{N}(\tilde{f})$. The following two definitions are imported from [ESW02], where they are shown to be equivalent in the quantum case.

Definition 1.3. A function $f : A \times B \to C \times D$ is semicausal if its projection C depends only on A, i.e. if there exists $g : A \to C$ such that $f(a,b)_C = g(a)$.

Definition 1.4. A bijection $f : A \times B \to C \times D$ is (reversibly) semilocalizable if there exists a seevit E and bijections $g : A \to C \times E$ and $h : D \to B \times E$ such that $f(a,b) = (g_C(a), \tilde{h}(g_E(a), b))$, as illustrated in Figure 4.

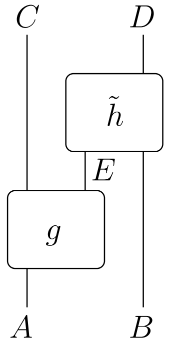


Figure 4: a semilocalizable bijection

Semilocalizability is, as the name suggests, an asymmetric property, in the sense that applying a symmetry on Figure 4 along a vertical axis, i.e. swapping A with B and C with D, breaks the semilocalizability. However, a transformation that preserves the property is the central symmetry, which corresponds to taking the dual of f: the notion of semilocalizability is self-dual.

Proposition 1.5. *$f : A \times B \to C \times D$ is semilocalizable if and only if the three following conditions are met:*

1. *f is semicausal;*
2. *$\tilde{f}$ is semicausal;*
3. *for every $a, a' \in A$ and $b, b' \in B$, if $f(a,b)_D = f(a',b)_D$ then $f(a,b')_D = f(a',b')_D$.*

Proof. Suppose f is semilocalizable. Then obviously from Figure 4 both f and $\tilde{f}$ are semicausal. Let $a, a' \in A$ and $b, b' \in B$ such that $f(a,b)_D = f(a',b)_D$. That means $h^{-1}(b, g_E(a)) = h^{-1}(b, g_E(a'))$ therefore $g_E(a) = g_E(a')$, and it follows immediately $f(a,b')_D = f(a',b')_D$.

Suppose now conditions (1), (2), (3) are met. Let $\sim_A$ be the binary relation on A defined by $a \sim_A a'$ iff $\forall b \in B$ $f(a,b) = f(a',b)$. Note that because of (3) this is equivalent to $\exists b \in B\ f(a,b) = f(a',b)$ (except if $B = \emptyset$, which is too trivial a case to worry about), from which we deduce

$$\forall c, c' \in C\ \forall d \in D \quad \tilde{f}(d,c) \sim_A \tilde{f}(d,c') \tag{1.1}$$

It is clearly an equivalence relation, so using the fact that f is semicausal we can define $g : A \to C \times (A/\sim_A)$ by $g(a) = (f(a,b)_C, [a])$, where b is an arbitrary element of B and $[a]$ is the class of a in $A/\sim_A$. One can define dually $\sim_D$ on D and define $h : D \to B \times (D/\sim_D)$ by $h(d) = (\tilde{f}(d,c)_B, [d])$.

It remains to be proven that $\alpha : \begin{pmatrix} A/\sim_A & \to & D/\sim_D \\ [a] & \mapsto & [f(a,b)_D] \end{pmatrix}$ is a well-defined bijection. To prove that it is well-defined, we need to show that for every $a, a' \in A$ such that $a \sim_A a'$ and every $b, b' \in B$, $f(a,b)_D \sim_D f(a',b')_D$, which is easily done in two small steps. First, by definition of $\sim_A$, $f(a,b)_D = f(a',b)_D$. Then, by the dual of (1.1), $f(a',b)_D \sim_D f(a',b')$. We now prove that α is bijection by showing that its inverse is its dual, defined by $\tilde{\alpha}([d]) = [\tilde{f}(d,c)_A]$, so that $\tilde{\alpha}\alpha([a]) = [\tilde{f}(f(a,b)_D, c)_A]$. Since this value is independent of c, we can try in particular with $c = f(a,b)_C$, where it is clear that we get $[a]$. ■

Definition 1.6. For a CA f on the alphabet Σ and X, Y two subsets of $\mathbb{Z}$, let $\mathbf{Q}_X^Y(f)$ be the property: "f seen as a function from $\Sigma^X \times \Sigma^{\bar{X}}$ to $\Sigma^Y \times \Sigma^{\bar{Y}}$ is semilocalizable".

Some property are obvious from the definition of semilocalizability, especially from Figure 4. Let us give two basic examples.

Lemma 1.7. *If $\mathbf{Q}_X^Y(f)$ holds, then so does $\mathbf{Q}_{X'}^{Y'}(f)$ for every $X' \supseteq X$ and $Y' \subseteq Y$.*

- For a CA seen as a function from $\Sigma^X \times \Sigma^{\bar{X}}$ to $\Sigma^Y \times \Sigma^{\bar{Y}}$, being semicausal means $X \supseteq Y + \mathcal{N}(f)$; the semicausality of $\tilde{f}$ means $X \supseteq Y + \tilde{\mathcal{N}}(f)$.

The following property, however, is easier to prove with Proposition 1.5 in mind.

Lemma 1.8. *If $\mathbf{Q}_X^Y(f)$ and $\mathbf{Q}_{X'}^Y(f)$, then $\mathbf{Q}_{X\cap X'}^Y(f)$.*

Proof. Let a, b be words on $X \cap X'$, and u, v words on $\overline{X \cap X'}$, and suppose $f(a.u) = f(b.u)|_{\bar{Y}}$. Let u' be the word on $\overline{X \cap X'}$ that is equal to u on $X \backslash X'$, v elsewhere. According to $\mathbf{Q}_X^Y(f)$, $f(a.u') = f(b.u')|_{\bar{Y}}$; we then conclude from $\mathbf{Q}_{X'}^Y(f)$ that $f(a.v) = f(b.v)|_{\bar{Y}}$. ■

Not that we get immediately the following corollary from the selfduality of semilocalizability: if $\mathbf{Q}_X^Y(f)$ and $\mathbf{Q}_X^{Y'}(f)$, then $\mathbf{Q}_X^{Y\cup Y'}(f)$.

We have now established all the properties on $\mathbf{Q}_X^Y(f)$ required to define the block neighborhood.

Definition 1.9. The block neighborhood $\mathcal{BN}(f)$ of f is the smallest X such that $\mathbf{Q}_X^{\{0\}}(f)$ holds.

The word "neighborhood" is not gratuitous. In fact, $\mathbf{Q}_X^Y(f)$ behaves exactly like "X includes the neighborhood of Y for some CA f' such that $\mathcal{N}(f') = \mathcal{BN}(f)$", as stated in the next lemma. The idea is that $\mathcal{BN}$ characterizes a notion of dependency that is very similar to the usual one characterized by $\mathcal{N}$.

Lemma 1.10. *$\mathbf{Q}_X^Y(f)$ is equivalent to $X \supseteq Y + \mathcal{BN}(f)$.*

Proof. Suppose $X \supseteq Y + \mathcal{BN}(f)$. By translation invariance, for all $y \in Y$, we have $\mathbf{Q}^{\{y\}}_{\{y\}+\mathcal{BN}(f)}(f)$, which, according to Lemma 1.7, implies $\mathbf{Q}^{\{y\}}_{X}(f)$. Invoking now the dual of Lemma 1.8, we get $\mathbf{Q}^{Y}_{X}(f)$.

For the reciprocal, suppose now $\mathbf{Q}^{Y}_{X}(f)$. According to Lemma 1.7, we have, for every $y \in Y$, $\mathbf{Q}^{\{y\}}_{X}(f)$; but that is, by definition of $\mathcal{BN}$, equivalent to $X \supseteq \{y\} + \mathcal{BN}(f)$, so we must have $X \supseteq Y + \mathcal{BN}(f)$. ∎

So, the block neighborhood is just one kind of neighborhood. However, $\mathcal{BN}$ has by contruction one property that $\mathcal{N}$ does not share: it is self-dual. It is not enough to make interesting, and many questions are left open at this point. We just know $\mathcal{BN} \supseteq \mathcal{N} \cup \tilde{\mathcal{N}}$, but do we have a lower bound on $\mathcal{BN}$? Is it always finite? The answer is in Corollary 2.2. How do the block neighborhoods compose? It can be easily inferred from Figure 4 that $\mathcal{BN}(gf) \subseteq \mathcal{BN}(g) + \mathcal{BN}(f)$, which is certainly good news, but Proposition 2.3 provides much more interesting bounds.

2. Main theorem

Proposition 2.1. $\mathcal{BN} \subseteq \mathcal{N} - \mathcal{N} + \tilde{\mathcal{N}}$.

Sanity check: $\mathcal{N} - \mathcal{N} + \tilde{\mathcal{N}}$ contains indeed $\mathcal{N}$ because $\mathcal{N} \cap \tilde{\mathcal{N}} \neq \varnothing$, which follows from $\{0\} = \mathcal{N}(ff^{-1}) \subseteq \mathcal{N}(f) - \tilde{\mathcal{N}}(f)$.

Proof. The proof of this proposition can be essentially found in [SW04], where the result is stated as Lemma 4, albeit in a foreign formalism. Another avatar of the proposition and its proof can be found in the form of Lemma 3.2 of [AN08]. In order to keep this article self-contained, we give yet another proof.

Let us then prove $\mathbf{Q}^{\{0\}}_{\mathcal{N}(f)-\mathcal{N}(f)+\tilde{\mathcal{N}}(f)}(f)$. Let a, b be words on $\mathcal{N}(f) - \mathcal{N}(f) + \tilde{\mathcal{N}}(f)$ and u, v words on its complement such that $f(a.u) = f(b.u)|_{\overline{\{0\}}}$. Applying f^{-1} to that equality we get $a.u = b.u|_{\overline{\tilde{\mathcal{N}}(f)}}$, which implies of course $a.v = b.v|_{\overline{\tilde{\mathcal{N}}(f)}}$, from which we obtain $f(a.v) = f(b.v)|_{\overline{-\mathcal{N}(f)+\tilde{\mathcal{N}}(f)}}$. On the other hand, $f(w)|_{-\mathcal{N}(f)+\tilde{\mathcal{N}}(f)}$ is a function of $w|_{\mathcal{N}(f)-\mathcal{N}(f)+\tilde{\mathcal{N}}(f)}$, so $f(a.u) = f(a.v)|_{-\mathcal{N}(f)+\tilde{\mathcal{N}}(f)}$ and $f(b.u) = f(b.v)|_{-\mathcal{N}(f)+\tilde{\mathcal{N}}(f)}$, which in the end proves $f(a.v) = f(b.v)|_{\overline{\{0\}}}$. ∎

Rather surprisingly, this bound is not self-dual, which allows us to reinforce it immediately.

Corollary 2.2. $\mathcal{BN} \subseteq (\mathcal{N} - \mathcal{N} + \tilde{\mathcal{N}}) \cap (\tilde{\mathcal{N}} - \tilde{\mathcal{N}} + \mathcal{N})$.

Proposition 2.3. *Let $f_1, \ldots f_n$ be reversible CAs. Then*

$$\mathcal{BN}(f_n \cdots f_1) \subseteq \bigcup_{k=1}^{n} \left(\tilde{\mathcal{N}}(f_n \cdots f_{k+1}) + \mathcal{BN}(f_k) + \mathcal{N}(f_{k-1} \cdots f_1) \right).$$

This formula could seem at first glance not to be self-dual, and therefore obviously suboptimal, but there is more to the duality than just putting and removing tildes. Since $\widetilde{fg} = \tilde{g}\tilde{f}$, we have $\widetilde{\mathcal{BN}}(f_n \cdots f_1) = \mathcal{BN}(\tilde{f}_1 \cdots \tilde{f}_n)$; it is from here straightforward to check that the formula is indeed self-dual.

Proof. Let $\mathcal{V} = \bigcup_{k=1}^{n} \left(\tilde{\mathcal{N}}(f_n \cdots f_{k+1}) + \mathcal{BN}(f_k) + \mathcal{N}(f_{k-1} \cdots f_1) \right)$; we have to prove $\mathbf{Q}^{\{0\}}_{\mathcal{V}}(f)$. Let a, b be words on $\mathcal{V}$, u, v words on $\bar{\mathcal{V}}$, and assume $f_n \cdots f_1(a.u) = f_n \cdots f_1(b.u)|_{\overline{\{0\}}}$.

For $k \in [\![0; n]\!]$, let $\mathcal{C}_k = \tilde{\mathcal{N}}(f_n \cdots f_{k+1})$; for $k \in [\![1; n]\!]$, let $\mathcal{K}_k = \mathcal{C}_k + \mathcal{BN}(f_k)$ and $\mathcal{D}_k = \mathcal{N}(f_{k-1} \cdots f_1)$. For $k \in [\![1; n]\!]$, let $\mathcal{V}_k = \mathcal{K}_k + \mathcal{D}_k$; by definition, $\mathcal{V} = \bigcup_{k=1}^{n} \mathcal{V}_k$.

We will prove by induction the following hypothesis $(\mathcal{H}_k)$ for $k \in [\![0; n]\!]$:

- $f_k \cdots f_1(a.u) = f_k \cdots f_1(b.u)|_{\overline{\mathcal{C}_k}}$ and
- $f_k \cdots f_1(a.v) = f_k \cdots f_1(b.v)|_{\overline{\mathcal{C}_k}}$.

Since we already know $a.u = b.u|_{\overline{\mathcal{C}_0}}$, it follows immediately $a.v = b.v|_{\overline{\mathcal{C}_0}}$, so $(\mathcal{H}_0)$ is true.

Suppose $(\mathcal{H}_k)$ for some $k \in [\![0; n-1]\!]$. Let $a' = f_k \cdots f_1(a.u)|_{\mathcal{K}_{k+1}}$ and $b' = f_k \cdots f_1(b.u)|_{\mathcal{K}_{k+1}}$; since $\mathcal{K}_{k+1} + \mathcal{D}_{k+1} \subseteq \mathcal{V}$, a' and b' are respectively equal to $f_k \cdots f_1(a.v)|_{\mathcal{K}_{k+1}}$ and $f_k \cdots f_1(b.v)|_{\mathcal{K}_{k+1}}$. Let us define $u' = f_k \cdots f_1(a.u)|_{\overline{\mathcal{K}_{k+1}}}$ and $v' = f_k \cdots f_1(a.v)|_{\overline{\mathcal{K}_{k+1}}}$. We have

$$\mathcal{C}_k = \tilde{\mathcal{N}}(f_n \cdots f_{k+1}) \subseteq \tilde{\mathcal{N}}(f_n \cdots f_{k+2}) + \tilde{\mathcal{N}}(f_{k+1}) \subseteq \mathcal{C}_{k+1} + \mathcal{BN}(f_{k+1}) = \mathcal{K}_{k+1}.$$

We can therefore deduce from $(\mathcal{H}_k)$ that u' and v' are respectively equal to $f_k \cdots f_1(b.u)|_{\overline{\mathcal{K}_{k+1}}}$ and $f_k \cdots f_1(b.v)|_{\overline{\mathcal{K}_{k+1}}}$. By definition of $\mathcal{C}_{k+1}$, since $f_n \cdots f_1(a.u) = f_n \cdots f_1(b.u)|_{\overline{\{0\}}}$, we have $f_k \cdots f_1(a.u) = f_k \cdots f_1(b.u)|_{\overline{\mathcal{C}_k}}$, which is the first point of $(\mathcal{H}_{k+1})$. Since $\mathcal{K}_{k+1} = \mathcal{C}_{k+1} + \mathcal{BN}(f_{k+1})$, according to Lemma 1.10, we have $\mathbf{Q}^{\mathcal{C}_{k+1}}_{\mathcal{K}_{k+1}}(f_{k+1})$. We therefore deduce the second point of $(\mathcal{H}_{k+1})$.

So in the end we get $(\mathcal{H}_n)$, which concludes the proof because $\mathcal{C}_n = \{0\}$. ■

Corollary 2.4. *Suppose $\mathcal{N}(f) \subseteq [\![-\alpha; \beta]\!]$ and $\tilde{\mathcal{N}}(f) \subseteq [\![-\gamma; \delta]\!]$. Then $\mathcal{BN}(f^k) \subseteq [\![-(k+1)\max(\alpha,\gamma) - \min(\beta,\delta)\,; (k+1)\max(\beta,\delta) + \min(\alpha,\gamma)]\!]$.*

For $X \subseteq \mathbb{R}$, let X^* be its convex hull and for $\lambda \in \mathbb{R}$, $\lambda X = \{\lambda x \mid x \in X\}$. We get, for any reversible CA, the asymptotic relation $\lim_{k \to +\infty} \frac{1}{k}\mathcal{BN}(f^k)^* \subseteq \mathcal{N}(f)^* \cup \tilde{\mathcal{N}}(f)^*$. Let us assume we are in the case $\lim_{k \to +\infty} \frac{1}{k}\mathcal{N}(f^k)^* = \mathcal{N}(f)^*$ and $\lim_{k \to +\infty} \frac{1}{k}\tilde{\mathcal{N}}(f^k)^* = \tilde{\mathcal{N}}(f)^* \cup \tilde{\mathcal{N}}(f)^*$. Then what this means informally is that condition (3) in Proposition 1.5 applied to f^k becomes less restrictive as k grows, and fades at the limit.

It is interesting to note the relation with Kari's constructions in [Kar96] and [Kar99]. We will briefly discuss the latter; it is of course stated in dimension 2, but that is not an obstacle to comparison, as the same construction can be made in dimension 1, or our analysis generalized to dimension 2 (cf. section 3.2). Let us place ourselves in dimension 1. Let f be a CA whose neighborhood and dual neighborhood are both included in $[\![-1; 1]\!]$. In this case, Corollary 2.4 implies $\mathcal{BN}(f^k) \subseteq [\![-(k+2); k+2]\!]$. Kari proves that there is an embedding φ and a CA g such that $f = \varphi g \varphi^{-1}$, where g fulfills by construction $\mathcal{BN}(g) \subseteq [\![-1; 1]\!]$. Kari's construction therefore contains an asymptotically optimal bound on $\mathcal{BN}(f^k)$.

Corollary 2.5. *If the neighborhoods and dual neighborhoods of f and g are included in $[\![-n; n]\!]$, then $\mathcal{BN}(fg) \subseteq [\![-4n; 4n]\!]$.*

The contraposition is actually more interesting. Consider h, whose neighborhood and dual neighborhood are both included in $[\![-n; n]\!]$; its block neighborhood has to be contained in $3[\![-n; n]\!]$. It seems perfectly reasonable to assume that h could be a composition of two more elementary reversible cellular automata f and g having strictly smaller neighborhoods, containing $\frac{1}{2}[\![-n; n]\!]$ but close to it. Actually, if no restriction is imposed on the behaviour of f^{-1} and g^{-1}, maybe even allowing f

and g to be nonreversible, it is certainly possible to decompose h in such a way by increasing the size of the alphabet. However, if the dual neighborhoods of f and g are also required to be close to $\frac{1}{2}[\![-n;n]\!]$, then such a decomposition will not be possible if $\mathcal{BN}(h)$ is too large. For instance, if $\mathcal{BN}(h)$ is not contained in $\frac{5}{2}[\![n;n]\!]$, then $\mathcal{N}(f)$, $\tilde{\mathcal{N}}(f)$, $\mathcal{N}(g)$ and $\tilde{\mathcal{N}}(g)$ cannot all be included in $\frac{5}{8}[\![-n;n]\!]$. In this sense, h can be considered "elementary".

3. Remarks

We gather in this section several unrelated observations about the block neighborhood.

3.1. Subtraction Automata

Suppose Σ can be provided with a binary operation $\cdot - \cdot$ such that:

- there exists an element of Σ denoted 0 such that $x = y$ is equivalent to $x - y = 0$;
- f is an endomorphism of $(\Sigma^{\mathbb{Z}}, -)$, where $-$ is defined component-wise on $\Sigma^{\mathbb{Z}}$.

We say in this case f admits a subtraction. For instance, linear automata as defined in [GNW10] admit subtractions.

Proposition 3.1. *Automata with subtractions have minimal block neighborhoods. In other words, for any automaton f admitting a subtraction, $\mathcal{BN}(f) = \mathcal{N}(f) \cup \tilde{\mathcal{N}}(f)$.*

Proof. Let A be any subset of $\mathbb{Z}$, a, b be words on A, u, v words on $\bar{A}$, and suppose $f(a.u) = f(b.u)$. Then $f(a.v) - f(b.v) = f(a.v - b.v) = f((a-b).0) = f(a.u - b.u) = f(a.u) - f(b.u) = 0$. ■

3.2. Generalization

We can actually drop many properties of the CAs that are irrelevant to the notions developed in this article. We don't need translation invariance. We don't need the alphabet to be finite. We don't need the neighborhoods to be finite. We don't need the domain and range cell structures to be identical. In this abstract setting, a "reversible automaton" is a bijection from $\prod_{i \in I} X_i$ to $\prod_{j \in J} Y_j$ and $\mathcal{N}(f)$ is a function from $\mathcal{P}(J)$ to $\mathcal{P}(I)$ which to $B \subseteq J$ associates the minimal subset A of I such that $f(x)|_B$ depends only on $x|_A$. In general, a function $\alpha : \prod_{i \in I} X_i \to \prod_{j \in J} Y_j$ is a *neighborhood scheme* if for all Y, $\alpha(Y) = \bigcup_{X \subseteq Y} \alpha(X)$; $\mathcal{N}(f)$ is of course one example of a neighborhood scheme. The usual definition of the neighborhood in the case of a cellular automaton corresponds here to $\mathcal{N}(f)(\{0\})$. Any function $\alpha : \mathcal{P}(J) \to \mathcal{P}(I)$ has a transpose $\alpha^\dagger : \mathcal{P}(I) \to \mathcal{P}(J)$ defined by $\alpha^\dagger(A)$ being the largest subset B of J such that $\alpha(B) \subseteq A$. We have indeed $(\alpha^\dagger)^\dagger = \alpha$, and for usual one-dimensional CAs, † corresponds to $\mathcal{N} \mapsto -\mathcal{N}$.

There is not anymore any good notion of duality on automata, but $\tilde{\mathcal{N}}(f)$ can be defined as $\mathcal{N}^\dagger(f^{-1})$. Of course the definition of $\mathcal{BN}(f)$ cannot make any reference to 0, instead $\mathcal{BN}(f)(B)$ is now the smallest subset of I fulfilling $\mathbf{Q}_A^B(f)$. The self-duality of $\mathcal{BN}$ is of course still valid. Lemmas 1.7 and 1.8 state respectively that $\mathcal{BN}$ is well-defined and that it is a neighborhood scheme.

Lemma 1.10 (used once at the end of the proof of Proposition 2.3) becomes "$\mathbf{Q}_X^Y(f)$ is equivalent to $X \supseteq \bigcup_{y \in Y} \mathcal{BN}(f)(Y)$", which is precisely the definition of $\mathcal{BN}$; it can therefore be forgotten, as the triviality it is now. Proposition 2.1 becomes $\mathcal{BN} \subseteq \mathcal{N} \circ \mathcal{N}^\dagger \circ \tilde{\mathcal{N}}$, Corollary 2.2 changes accordingly, and Proposition 2.3 remains true when "+" is substituted with "$\circ$". It follows that indecomposability results such as Corollary 2.5 are extremely robust: they cannot be overcome by increasing the size of the alphabet or relaxing the translational invariance. It also shows of course the limitations of this method, namely that it is utterly unable to exploit these parameters.

3.3. Optimality

The bounds presented in Corollary 2.2 and Proposition 2.3 seem peculiar enough as to be suspect of non-optimality. However, we have been unable to come up with a better approximation, and would rather tend to think that they cannot be improved. We will concentrate on Corollary 2.2 alone, whose optimality is conjectured in the following statement.

Conjecture 3.2. For any subsets X, Y and Z of $\mathbb{Z}$ such that $X \cup Y \subseteq Z \subseteq (X-X+Y) \cap (Y-Y+X)$, if there exists a CA f such that $\mathcal{N}(f) = X$ and $\tilde{\mathcal{N}}(f) = Y$, then there exists a CA g such that $\mathcal{N}(g) = X$, $\tilde{\mathcal{N}}(g) = Y$ and $\mathcal{BN}(g) = Z$.

This section will be devoted to proving the following weaker version:

Proposition 3.3. *Conjecture 3.2 is true when $Z \subseteq \{2y - x \mid x, y \in X \cap Y\}$. In particular it is true if X and Y are equal intervals.*

Proof. Given that there is by hypothesis a CA f such that $\mathcal{N}(f) = X$ and $\tilde{\mathcal{N}}(f) = Y$, we only need to prove that for every $z \in Z$ there exists a CA g_z such that $\mathcal{N}(g_z) \subseteq X$, $\tilde{\mathcal{N}}(g_z) \subseteq Y$ and $z \in \mathcal{N}(g_z) \subseteq Z$. Then the proposition is proven by considering the direct sum of f and all these g_z's.

The Toffoli automaton presented in [ANW08] (definition 12), defined by $\Sigma = (\mathbb{Z}/2\mathbb{Z})^2$ and $T(v)_0 = (v_0^2 + v_0^1 v_1^1, v_1^1)$, will serve as the basic constructing tool for g_z. Its inverse is given by $T^{-1}(v)_0 = \left(v_{-1}^2, v_0^1 + v_{-1}^2 v_0^2\right)$, so we clearly have $\mathcal{N}(T) = \tilde{\mathcal{N}}(T) = \{0; 1\}$. Let us prove $\mathcal{BN}(T) = [\![0; 2]\!]$, by proving first that $\mathbf{Q}_{\mathbb{N}}^{\mathbb{N}}(T)$ is true, and then that $\mathbf{Q}_{\{0;1\}}^{\{0\}}(T)$ is false. Let then a, b be words on $\mathbb{N}$ and u, v words on its complement, and suppose $T(a.u) = T(b.u)_{\bar{\mathbb{N}}}$. In particular, $T(a.u)_{-1}^2 = T(b.u)_{-1}^2$, which implies $a_0^1 = b_0^1$, so we get immediately $T(a.v) = T(b.v)_{\bar{\mathbb{N}}}$, which proves $\mathbf{Q}_{\mathbb{N}}^{\mathbb{N}}(T)$. Consider now the words $a = (0,0)(0,0)$ and $b = (0,0)(1,1)$ on $\{0; 1\}$, and u, v the words on its complement that are $(0,0)$ everywhere except in position 2, where $u_2 = (1,0)$. We have $T(a.u) = T(b.u)_{\overline{\{0\}}}$ but $T(a.v)_1 = (0,0)$ while $T(b.v)_1 = (1,0)$, therefore $\mathbf{Q}_{\{0;1\}}^{\{0\}}(T)$ is false.

This CA can be obviously expanded into an automaton T_l such that $\mathcal{N}(T_l) = \tilde{\mathcal{N}}(T_l) = \{0; l\}$ and $\mathcal{BN}(T_l) = \{0; l; 2l\}$.

More generally, for any nonempty intervals X and Z of $\mathbb{Z}$ such that $X \subseteq Z \subseteq X - X + X$, there is a CA f such that $\mathcal{N}(f) = \tilde{\mathcal{N}}(f) = X$ and $\mathcal{BN}(f) = Z$. We can engineer such an f by considering the direct sum of several CAs. First, for each element $x \in X$, consider the shift by $-x$: the sum of all these shifts is a CA g such that $\mathcal{N}(g) = \tilde{\mathcal{N}}(g) = \mathcal{BN}(g) = X$. Then, for each element $z \in Z$, choose x and y in

$X \cap Y$ such that $z = 2y - x$. The automaton $g_z = \sigma^x T_{y-x}$, where σ is the elementary shift to the left, is then such that $\mathcal{N}(T_l) = \tilde{\mathcal{N}}(T_l) = \{x; y\}$ and $\mathcal{BN}(T_l) = \{x; y; z\}$, which concludes the proof. ■

Conclusion

Of course a lot of questions remain. Are the upper bounds on the block neighborhood given in this article optimal under all circumstances? And is it possible to make these bounds more efficient by including as parameters the size of the alphabet and the requirement that the transformations be translation invariant? This would probably require a whole different technique.

Something happened in this article that is increasingly common: after a theory grows a quantum extension (in this case QCAs join the family of CAs) and new tools and techniques are invented to study the quantum setup, they come back to the classical setup (semilocalizability comes to mind, and a lot of others are disguised as combinatorial properties) and bring various insights, simpler proofs and/or new results.

The block neighborhood is nothing else than the quantum neighborhood. It shows what had been grasped until then only intuitively: whereas CAs can be defined by their local transition functions, QCAs are intrisically block-structured. In that sense, working on QCAs is a lot like working on CAs with a restricted bag of tools that includes only local permutations — duplication or destruction of information are stricly forbidden. It also means that, even staying in a purely classical framework, finding this kind of constructions is worthwhile and meaningful, even in the case where a result is already known to be attainable by another method. Not only will the construction be nicer in a purely abstract way, because it will employ only elementary means: it will also have the benefit of being immediately transposable to the quantum case.

Acknowledgements

The authors would like to thank Jarkko Kari for showing them, to their amazement, how the neighborhood and the inverse neighborhood of CAs depend so little on each other, even in the iterated dynamics. They also gratefully acknowledge the support of the Deutsche Forschungsgemeinschaft (Forschergruppe 635) and the EU (project QICS).

References

[AN08] Pablo Arrighi and Vincent Nesme. Quantization of cellular automata. In Bruno Durand, editor, *Proceedings of the First Symposium on Cellular Automata "Journées Automates Cellulaires" JAC 2008*, Exploratory paper track, pages 204–215, Uzès France, 04 2008. Издательство МЦНМО. ISBN 978-5-94057-377-7.

[ANW] Pablo Arrighi, Vincent Nesme, and Reinhard F. Werner. Unitarity plus causality implies localizability. To appear in *Journal of Computer and System Sciences*. `arXiv:0711.3975v3`.

[ANW08] Pablo Arrighi, Vincent Nesme, and Reinhard F. Werner. One-dimensional quantum cellular automata over finite, unbounded configurations. In *Language and Automata Theory and Applications: Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, pages 64–75, Berlin, Heidelberg, 2008. Springer-Verlag.

[DL01] Jérôme Durand-Lose. Representing reversible cellular automata with reversible block cellular automata. In Robert Cori, Jacques Mazoyer, Michel Morvan, and Rémy Mosseri, editors, *Discrete Models: Combinatorics, Computation, and Geometry, DM-CCG '01*, volume AA of *Discrete Mathematics and Theoretical Computer Science Proceedings*, pages 145–154, 2001.

[ESW02] T. Eggeling, Dirk Schlingemann, and Reinhard F. Werner. Semilocal operations are semilocalizable. *Europhysics Letters*, 57(6):782–788, 2002.

[GNW10] Johannes Gütschow, Vincent Nesme, and Reinhard F. Werner. The fractal structure of cellular automata on abelian groups. 2010.

[Kar90] Jarkko Kari. Reversibility of 2d cellular automata is undecidable. *Physica D*, 45(1-3):386–395, 1990.

[Kar96] Jarkko Kari. Representation of reversible cellular automata with block permutations. *Mathematical Systems Theory*, 29(1):47–61, 1996.

[Kar99] Jarkko Kari. On the circuit depth of structurally reversible cellular automata. *Fundam. Inf.*, 38(1-2):93–107, 1999.

[Mar84] Norman Margolus. Physics-like models of computation. *Physica D*, 10:81–95, 1984.

[MH89] Ken'ichi Morita and Masateru Harao. Computation universality of one-dimensional reversible (injective) cellular automata. *IEICE Transactions on Information and Systems, E*, 72:758–762, 1989.

[MU92] Ken'ichi Morita and Satoshi Ueno. Computation-universal models of two-dimensional 16-state reversible cellular automata. *IEICE Transactions on Information and Systems, E*, 75:141–147, 1992.

[SW04] Benjamin Schumacher and Reinhard F. Werner. Reversible quantum cellular automata. 05 2004. `arXiv:quant-ph/0405174`.

Journées Automates Cellulaires 2010 (Turku), pp. 54-64

COMPUTING (OR NOT) QUASI-PERIODICITY FUNCTIONS OF TILINGS

ALEXIS BALLIER AND EMMANUEL JEANDEL

Laboratoire d'Informatique Fondamentale de Marseille, CMI, 39 rue Joliot-Curie, F-13453
Marseille Cedex 13, France
E-mail address, A. Ballier: alexis.ballier@lif.univ-mrs.fr
E-mail address, E. Jeandel: emmanuel.jeandel@lif.univ-mrs.fr

Abstract. We know that tilesets that can tile the plane always admit a quasi-periodic tiling [4, 8], yet they hold many uncomputable properties [3, 11, 21, 25]. The quasi-periodicity function is one way to measure the regularity of a quasi-periodic tiling. We prove that the tilings by a tileset that admits only quasi-periodic tilings have a recursively (and uniformly) bounded quasi-periodicity function. This corrects an error from [6, theorem 9] which stated the contrary. Instead we construct a tileset for which any quasi-periodic tiling has a quasi-periodicity function that cannot be recursively bounded. We provide such a construction for 1−dimensional effective subshifts and obtain as a corollary the result for tilings of the plane *via* recent links between these objects [1, 10].

Tilings of the discrete plane as studied nowadays have been introduced by Wang in order to study the decidability of a subclass of first order logic [26, 27, 5]. After Berger proved the undecidability of the domino problem [3], interest has grown for understanding how complex are these simply defined objects [11, 21, 9, 6]. Despite being able to have complex tilings, any tileset that can tile the plane admits a *quasi-periodic* tiling [4, 8]; roughly speaking, a quasi-periodic tiling is a tiling in which every finite pattern can be found in any sufficiently large part of the tiling. It is therefore natural to define the quasi-periodicity function of a quasi-periodic tiling: it associates to an integer n the minimal size in which we are certain to find any pattern of size n [8, 6]. This is one way to measure the complexity of a quasi-periodic tiling and, to some extent, of a tileset τ since τ must admit at least one quasi-periodic tiling. We start by proving in Section 2 that tilings by tilesets that admit only quasi-periodic tilings have a recursively (and uniformly) bounded quasi-periodicity function (Theorem 1.4). Remark that there exists non-trivial tilesets that admit only quasi-periodic tilings [23, 19, 22] and that the property of having only such tilings can be reduced to the domino problem [3, 23] and is thus undecidable[1].

Both authors are partly supported by ANR-09-BLAN-0164. A. Ballier has been partly supported by the Academy of Finland project 131558. We thank Pierre Guillon for discussions that lead to the constructions provided in Section 3.

[1] Take a tileset τ_u that admits only one uniform tiling (and thus only quasi-periodic tilings), a tileset τ_f that admits non quasi-periodic tilings (*e.g.*, a fullshift on $\{0, 1\}$) then it is clear that $(\tau \times \tau_f) \cup \tau_u$ admits only quasi-periodic tilings if and only if τ does not tile the plane.

With the aim to study discretization of dynamical systems, 1−dimensional subshifts have been extensively studied in symbolic dynamics [18, 16]. Quasi-periodic tilings correspond to almost periodic sequences [12] or uniformly recurrent sequences in this context. Again, the existence of complex uniformly recurrent sequences has been shown [20]. In Section 3 we show, given a partial recursive function φ, how to construct an effective subshift in which every uniformly recurrent configuration has a quasi-periodicity function greater than φ where it is defined (Theorem 3.3). This allows us to correct the error from [6] as we obtain as a corollary (using recent links between tilings and effective 1−dimensional subshifts [1, 10]) that there exists tilesets for which no quasi-periodic tiling can have a quasi-periodicity function that is recursively bounded (Theorem 3.5).

1. Definitions

A *configuration* is an element of $\mathbf{Q}^{\mathbb{Z}^2}$ where $\mathbf{Q}$ is a finite set or, equivalently, a mapping from $\mathbb{Z}^2$ to $\mathbf{Q}$. A *pattern* $\mathbf{P}$ is a function from a finite domain $\mathcal{D}_P \subseteq \mathbb{Z}^2$ to $\mathbf{Q}$. The *shift* of vector v ($v \in \mathbb{Z}^2$) is the function denoted by σ_v from $\mathbf{Q}^{\mathbb{Z}^2}$ to $\mathbf{Q}^{\mathbb{Z}^2}$ defined by $\sigma_v(c)(x) = c(v+x)$. A pattern $\mathbf{P}$ *appears* in a configuration c (denoted $\mathbf{P} \in c$) if there exists $v \in \mathbb{Z}^2$ such that $\sigma_v(c)_{|\mathcal{D}_P} = \mathbf{P}$. Similarly, we can define the shift of vector v of a pattern $\mathbf{P}$ by the function $\sigma_v(\mathbf{P})(x) = \mathbf{P}(v+x)$; then we can say that a pattern $\mathbf{P}$ appears in another pattern $\mathbf{M}$ if there exists $v \in \mathbb{Z}^2$ such that $\sigma_v(\mathbf{M})_{|\mathcal{D}_P} = \mathbf{P}$ and denote it by $\mathbf{P} \in \mathbf{M}$. We use the same vocabulary and notations for both notions of shift and appearance but there should not be any confusion since configurations are always denoted by lower case letters and patterns by upper case letters.

Given a finite set of colors $\mathbf{Q}$, a *tileset* is defined by a finite set of patterns $\mathcal{F}$; we say that a configuration c is a *valid tiling* for $\mathcal{F}$ if none of the patterns of $\mathcal{F}$ appear in c. We denote by $\mathbf{T}_{\mathcal{F}}$ the set of valid tilings for $\mathcal{F}$. If $\mathbf{T}_{\mathcal{F}}$ is non-empty we say that $\mathcal{F}$ can tile the plane. A set of configurations $\mathbf{T}$ is said to be a *set of tilings* if there exists some finite set of patterns $\mathcal{F}$ such that $\mathbf{T} = \mathbf{T}_{\mathcal{F}}$. This notion of set of tilings corresponds to subshifts of finite type [16, 15]. When we impose no restriction on $\mathcal{F}$ these are subshifts and when $\mathcal{F}$ is recursively enumerable we say that $\mathbf{T}_{\mathcal{F}}$ is an *effective subshift* (see, *e.g.*, [7, 14, 13, 1, 10]).

A periodic configuration c is a configuration such that the set $\{\sigma_v(c), v \in \mathbb{Z}^2\}$ is finite. It is well known (since Berger [3]) that there exists tilesets that do not admit a periodic tiling but can still tile the plane. On the other hand, quasi-periodicity is the correct regularity notion if we always want a tiling with this property. Periodic configurations are quasi-periodic but the converse is not true. Several characterizations of quasi-periodic configurations exist [8], we give one here that we use for the rest of the paper.

Definition 1.1 (Quasi-periodic configuration)**.** A configuration $c \in \mathbf{Q}^{\mathbb{Z}^2}$ is quasi-periodic if any pattern that appears in c appears in any sufficiently large pattern of c.

More formally, if a pattern $\mathbf{P}$ appears in c then there exists $n \in \mathbb{N}$ such that for every pattern $\mathbf{M}$ defined on $[-n;n]^2$ that appears in c, $\mathbf{P}$ appears in $\mathbf{M}$.

We denote by $n_{(\mathbf{P},c)}$ the smallest such n for finding a pattern $\mathbf{P}$ in the quasi-periodic configuration c.

Theorem 1.2 ([4, 8])**.** *Any non-empty set of tilings contains a quasi-periodic configuration.*

For an integer n, the set of patterns defined on a square domain $[-n;n]^2$ is finite, it is therefore natural to define the quasi-periodicity function of a quasi-periodic configuration.

Definition 1.3 (Quasi-periodicity function)**.** The quasi-periodicity function of a quasi-periodic configuration c, denoted by $\mathcal{Q}_c$, is the function from $\mathbb{N}$ to $\mathbb{N}$ that maps a given integer n to the smallest integer m such that any pattern of domain $[-n;n]^2$ that appears in c appears in any pattern of c of domain $[-m;m]^2$.

$$\mathcal{Q}_c(n) = \max\left\{n_{(\mathbf{P},c)}, \mathbf{P} \in c, \mathcal{D}_P = [-n;n]^2\right\}$$

The function $\mathcal{Q}_c$ measures in some sense the complexity of the quasi-periodic configuration c: the faster it grows, the more complex c is. Since one can construct tilesets whose tilings have many uncomputable properties (*e.g.*, such that every tiling is uncomputable as a function from $\mathbb{Z}^\mathbf{2}$ to $\mathbf{Q}$ [11, 21] or such that every pattern that appears in a tiling has maximal Kolmogorov complexity [9]), it is natural to expect the quasi-periodicity function to inherit the non-recursive properties of tilings. This is what had been proved in [6].

In some particular cases it is easy to prove that this function is actually computable. Consider a tileset such that any pattern that appears in a tiling appears in every tiling; in that case every tiling is quasi-periodic and the quasi-periodicity function is the same for every tiling. Moreover there exists an algorithm that decides if a pattern can appear in a tiling or not (this has been proven by different ways, either by considering the fact that the first order theory of the tileset is finitely axiomatizable and complete therefore decidable [2] or by using a direct compactness argument [14]). Given this algorithm, it is easy to compute the quasi-periodicity function (that does not depend on the tiling): for a given p, compute all the $[-p;p]^2$ patterns that appear in a tiling and then compute all the $[-n;n]^2$ patterns for $n \geq p$ until every $[-p;p]^2$ pattern appears in every $[-n;n]^2$ pattern and output the smallest such n.

In the remainder of this paper, we improve this technique to obtain a less restrictive condition on the tileset while proving that the quasi-periodicity function is recursively bounded:

Theorem 1.4. *If a tileset (defined by $\mathcal{F}$) admits only quasi-periodic tilings then there exists a computable function $q : \mathbb{N} \to \mathbb{N}$ such that for any tiling c of $\mathbf{T}_\mathcal{F}$, c has a quasi-periodicity function bounded by q,* i.e., *$\forall c \in \mathbf{T}_\mathcal{F}, \forall n \in \mathbb{N}, \mathcal{Q}_c(n) \leq q(n)$.*

Note that this result is contrary to a result in [6] stating that there exists tilesets admitting only quasi-periodic tilings with quasi-periodicity functions with no computable upper bound. There is indeed a mistake in [6] that will be examined later.

2. Computable bound on the quasi-periodicity function

In this section we consider a tileset defined by a finite set of forbidden patterns $\mathcal{F}$ such that every tiling by $\mathcal{F}$ is quasi-periodic. The only hypothesis we have is the following: For any tiling $c \in \mathbf{T}_\mathcal{F}$ and for any pattern $\mathbf{P}$ that appears in c, there exists an integer $n_{(\mathbf{P},c)}$ such that any $[-n_{(\mathbf{P},c)}, n_{(\mathbf{P},c)}]^2$ pattern that appears in c contains

P. In order to prove Theorem 1.4, we first have to prove that there exists a bound that does not depend on the tiling:

Lemma 2.1. *If a tileset $\mathcal{F}$ admits only quasi-periodic tilings then, for any pattern* $\mathbf{P}$ *that appears in some tiling of* $\mathbf{T}_{\mathcal{F}}$*, there exists an integer n such that any tiling that contains* $\mathbf{P}$ *also contains* $\mathbf{P}$ *in all its* $[-n;n]^2$ *patterns.*

We define $n_{(\mathbf{P},\mathcal{F})}$ to be the smallest integer with this property.

Remark that the converse of this lemma is obviously true by definition: if for any pattern there exists such an integer then all the tilings are quasi-periodic.

Proof. Suppose this is not true: there exists a pattern $\mathbf{P}$ and a sequence $(c_n)_{n\in\mathbb{N}}$ of configurations that contain $\mathbf{P}$ and such that c_n also contains a $[-n;n]^2$ pattern that does not contain $\mathbf{P}$.

For a given n, consider $\mathbf{O}_n$, one of the largest square patterns of c_n that does not contain $\mathbf{P}$. Since c_n is quasi-periodic and contains $\mathbf{P}$ by hypothesis, there does not exist arbitrary large square patterns that do not contain $\mathbf{P}$ and thus $\mathbf{O}_n$ is well defined. Note that $\mathbf{O}_n$ is defined on at least $[-n;n]^2$. Since we supposed $\mathbf{O}_n$ of maximal size, there must be a pattern $\mathbf{P}$ adjacent to it like depicted on Figure 1.

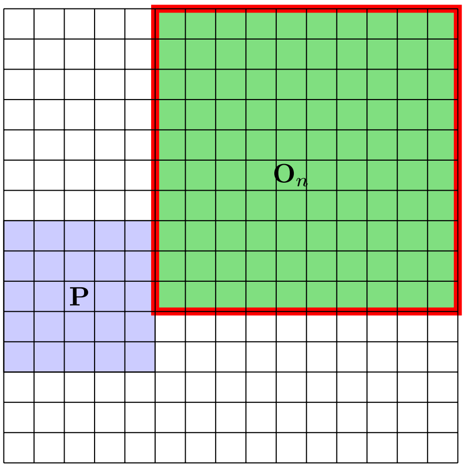


Figure 1: $\mathbf{O}_n$ near $\mathbf{P}$.

Now if we center our view on this $\mathbf{P}$ adjacent to $\mathbf{O}_n$, for infinitely many n's the largest part of $\mathbf{O}_n$ always appears in the same quarter of plane (with origin $\mathbf{P}$). Since $\mathbf{O}_n$ is defined on at least $[-n;n]^2$, by compactness we obtain a tiling with $\mathbf{P}$ at its center and a quarter of plane without $\mathbf{P}$. Such a tiling cannot be quasi-periodic. ∎

Lemma 2.1 shows that if all the tilings that are valid for $\mathcal{F}$ are quasi-periodic then there exists a global bound on the quasi-periodicity function of any tiling: define $f(n) = \max\left\{n_{(\mathbf{P},\mathcal{F})}, \mathcal{D}_P = [-n;n]^2, \mathbf{P} \text{ appears in a tiling by } \mathcal{F}\right\}$; for any tiling $c \in \mathbf{T}_{\mathcal{F}}$ and any integer n, we have $\mathcal{Q}_c(n) \leq f(n)$. The only part left in the proof of Theorem 1.4 is to prove that f is computably bounded.

In a quasi-periodic tiling, if a pattern $\mathbf{P}$ defined on $[-n;n]^2$ appears in it then it must appear close to $\mathbf{P}$ (at distance less than $f(n)+n$) in each of the four quarters of plane starting from the corners of $\mathbf{P}$. In general, we cannot compute whether a pattern will appear in some tiling or not, however, we can compute whether a pattern is valid with respect to $\mathcal{F}$.

Lemma 2.2. *If a tileset $\mathcal{F}$ admits only quasi-periodic tilings then, for any pattern $\mathbf{P}$ defined on $[-n;n]^2$ that appears in some tiling of $\mathbf{T}_\mathcal{F}$, there exists an integer m such that any pattern $\mathbf{R}$ defined on $[-n-m;n+m]^2$ that is valid with respect to $\mathcal{F}$ and contains $\mathbf{P}$ at its center (*i.e.*, $\mathbf{R}_{|[-n;n]^2} = \mathbf{P}$) is such that the four patterns $\mathbf{R}_{|[-n-m;-n]^2}$, $\mathbf{R}_{|[-n-m;-n]\times[n;n+m]}$, $\mathbf{R}_{|[n;n+m]\times[-n-m;-n]}$, $\mathbf{R}_{|[n;n+m]^2}$ all contain $\mathbf{P}$.*

We define $m_{(\mathbf{P},\mathcal{F})}$ to be the smallest integer m with this property.

Those four patterns may seem obscure at a first read, they are depicted on Figure 2.

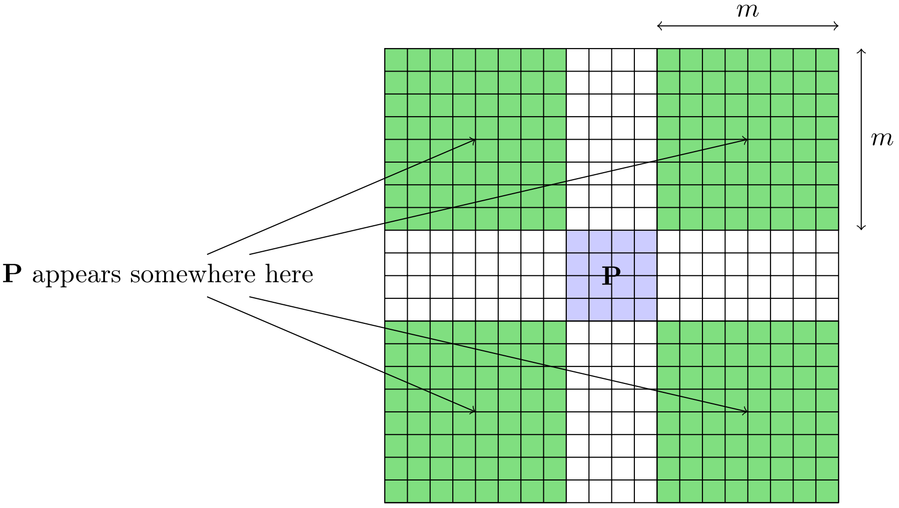


Figure 2: The four patterns in which we must find another occurrence of $\mathbf{P}$.

Proof. For a given pattern $\mathbf{P}$, suppose that there exists no such m. This means that there exist arbitrarily large m and valid patterns $\mathbf{R_m}$ (defined on $[-n-m;n+m]^2$) such that one of the four patterns $\mathbf{R_m}_{|[-n-m;-n]^2}$, $\mathbf{R_m}_{|[-n-m;-n]\times[n;n+m]}$, $\mathbf{R_m}_{|[n;n+m]\times[-n-m;-n]}$, $\mathbf{R_m}_{|[n;n+m]^2}$ does not contain $\mathbf{P}$.

Without loss of generality, we can assume that this always happens in the same quarter of plane. By extracting a tiling centered on the pattern $\mathbf{P}$ at the center of $\mathbf{R_m}$ (which we can do by compactness), there exists a tiling c of $\mathbf{T}_\mathcal{F}$ that contains $\mathbf{P}$ and a quarter of plane without $\mathbf{P}$, contradicting the quasi-periodicity of c. ∎

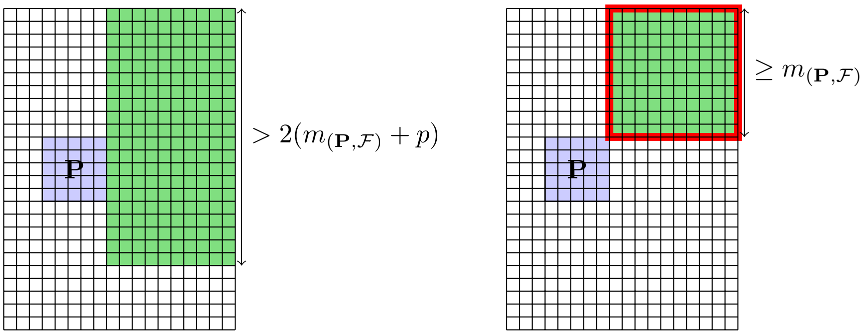
Note that the converse of Lemma 2.2 is also true: if, for any pattern $\mathbf{P}$, there exists such an $m_{(\mathbf{P},\mathcal{F})}$ then all the tilings of $\mathbf{T}_\mathcal{F}$ are quasi-periodic.

Lemma 2.3. *If $\mathcal{F}$ is a tileset that allows only quasi-periodic tilings then, for any pattern $\mathbf{P}$ defined on $[-p;p]^2$ that appears in some tiling of $\mathbf{T}_\mathcal{F}$, we have:*

$$n_{(\mathbf{P},\mathcal{F})} \leq 2(m_{(\mathbf{P},\mathcal{F})} + p)$$

Proof. Let c be a (quasi-periodic) tiling of $\mathbf{T}_\mathcal{F}$ that contains $\mathbf{P}$ and a pattern $\mathbf{O}$ defined on $[-k;k]^2$ that does not contain $\mathbf{P}$ with $k > 2(m_{(\mathbf{P},\mathcal{F})}+p)$. Without loss of generality, we may assume that $\mathbf{O}$ is of maximal size. That is, there is a pattern $\mathbf{P}$ adjacent to $\mathbf{O}$. Let $\mathbf{R}$ be the pattern defined on $[-p-m_{(\mathbf{P},\mathcal{F})};p+m_{(\mathbf{P},\mathcal{F})}]^2$ centered on the pattern $\mathbf{P}$ adjacent to $\mathbf{O}$ in c. Since $k > 2(m_{(\mathbf{P},\mathcal{F})}+p)$ and $\mathbf{O}$ does not contain $\mathbf{P}$, at least one of the four patterns $\mathbf{R}_{|[-p-m_{(\mathbf{P},\mathcal{F})};-p]^2}$, $\mathbf{R}_{|[-p-m_{(\mathbf{P},\mathcal{F})};-p]\times[p;p+m_{(\mathbf{P},\mathcal{F})}]}$, $\mathbf{R}_{|[p;p+m_{(\mathbf{P},\mathcal{F})}]\times[-p-m_{(\mathbf{P},\mathcal{F})};-p]}$, $\mathbf{R}_{|[p;p+m_{(\mathbf{P},\mathcal{F})}]^2}$ does not contain $\mathbf{P}$ as depicted on Figure 3;

since $\mathbf{R}$ is a valid pattern with respect to $\mathcal{F}$, this contradicts the definition of $m_{(\mathbf{P},\mathcal{F})}$. ∎



$\mathbf{P}$
$> 2(m_{(\mathbf{P},\mathcal{F})} + p)$
$\mathbf{P}$
$\geq m_{(\mathbf{P},\mathcal{F})}$

Figure 3: Bounding the size of the patterns not containing $\mathbf{P}$.

Now that we have a bound that deals only about locally valid patterns instead of patterns that appear in tilings (and therefore is computably checkable), we can proceed to the proof of Theorem 1.4:

Proof of Theorem 1.4. $\mathcal{F}$ is a tileset that admits only quasi-periodic tilings. For an integer n, compute all the patterns $\mathbf{P_1}, \ldots, \mathbf{P_k}$ defined on $[-n;n]^2$ that are valid for $\mathcal{F}$.

For each of these $\mathbf{P_j}$ use the following algorithm: For each integer i, compute the set $\mathbf{R_1}, \ldots, \mathbf{R_p}$ of patterns defined on $[-i-n;i+n]^2$ that contain $\mathbf{P_j}$ at their center and are valid with respect to $\mathcal{F}$.

(1) If there is no such pattern $\mathbf{R}$, claim that $\mathbf{P_j}$ cannot appear in any tiling by $\mathcal{F}$, and define *e.g.*, $b_{\mathbf{P_j}} = 0$. Then continue with $\mathbf{P_{j+1}}$

(2) If all these patterns $\mathbf{R}$ restricted to either $[-n-i;-n]^2$, $[-n-i;-n]\times[n;n+i]$, $[n;n+i]\times[-n-i;n]$ or $[n;n+i]^2$ all contain $\mathbf{P}$ then define $b_{\mathbf{P_j}} = 2(i+n)$ and continue with $\mathbf{P_{j+1}}$[2].

For any pattern, one of these cases always happens: If $\mathbf{P_j}$ appears in at least one tiling of $\mathbf{T}_\mathcal{F}$ then, by Lemma 2.2, for $i = m_{(\mathbf{P_j},\mathcal{F})}$ we are in case 2. If $\mathbf{P_j}$ does not appear in any tiling of $\mathbf{T}_\mathcal{F}$ then case 1 must happen, otherwise we would have arbitrary large extensions of $\mathbf{P_j}$ and hence a tiling containing $\mathbf{P_j}$ by compactness. Note that we may halt in case 2 even if $\mathbf{P_j}$ does not appear in any tiling.

Now compute $q(n) = \max\left\{b_{\mathbf{P_j}}, \mathcal{D}_{P_j} = [-n;n]^2\right\}$.

For any tiling $c \in \mathbf{T}_\mathcal{F}$ and any pattern $\mathbf{P}$ defined on $[-n;n]^2$ that appears in c we have:

$$\begin{array}{rcll} n_{(\mathbf{P},c)} & \leq & n_{(\mathbf{P},\mathcal{F})} & \text{by definition of } n_{(\mathbf{P},\mathcal{F})} \\ & \leq & 2(m_{(\mathbf{P},\mathcal{F})} + n) & \text{by Lemma 2.3} \\ & \leq & b_{\mathbf{P}} & \text{by minimality of } m_{(\mathbf{P},\mathcal{F})} \\ & \leq & q(n) & \text{by definition of } q \end{array}$$

Therefore, for any configuration c and any integer n, we have $\mathcal{Q}_c(n) \leq q(n)$ and q is the computable function that completes the proof of Theorem 1.4. ∎

[2] Remark that these patterns are exactly those depicted in Figure 2.

We remark that all the arguments used in the proofs of the lemmas involve only compactness and the fact that we can decide if a given pattern is valid for $\mathcal{F}$. Hence, we may remove some restrictions on $\mathcal{F}$: $\mathbf{T}_{\mathcal{F}}$ is still compact if $\mathcal{F}$ is infinite and we can still decide if a given pattern is valid for $\mathcal{F}$ when $\mathcal{F}$ is recursive. Moreover, if $\mathcal{F}$ is recursively enumerable then there exists a recursive set of patterns $\mathcal{F}'$ such that $\mathbf{T}_{\mathcal{F}} = \mathbf{T}_{\mathcal{F}'}$: consider the (computable) enumeration $f(0), f(1), \dots$ of $\mathcal{F}$; when enumerating $f(i)$, we can compute an integer n such that all the previously enumerated patterns are defined on a domain included in $[-n; n]^2$; then we enumerate all the extensions of $f(i)$ defined on $[-n-1; n+1]^2 \cup \mathcal{D}_{f(i)}$. This enumeration enumerates a new set of patterns $\mathcal{F}'$ that is now recursive since they are enumerated by increasing sizes. It is straightforward that $\mathbf{T}_{\mathcal{F}} = \mathbf{T}_{\mathcal{F}'}$. We conclude that $\mathcal{F}$ needs not to be finite in order for Theorem 1.4 to be valid but we may assume that it is only recursively enumerable. Sets of tilings with a recursively enumerable set of forbidden patterns are usually called *effective subshifts* in the literature [7, 14, 13, 1, 10] or also Π_1^0 subshifts [25, 17] and are a special case of effectively closed sets as studied in computable analysis (see *e.g.*, [28])[3].

3. Large quasi-periodicity functions

In this section we prove that we can construct tilesets whose every quasi-periodic tiling has a large quasi-periodicity function. We start from a 1-dimensional effective subshift $\mathbf{X}$ over an alphabet Σ and then build an effective subshift over the alphabet $\Sigma \times \{0, 1\}$, and the complexity of the quasi-periodicity function will come from the top layer. For this, consider all occurrences of a word u in the subshift $\mathbf{X}$. There are infinitely many of them, so the top layer restricted to occurrences of u will contain a bi-infinite word over $\{0, 1\}$. If we can find infinitely many words in the subshift $\mathbf{X}$ so that occurrences of different words do not somehow overlap in a configuration c, then this would give us an infinite number of bi-infinite words within a single configuration c, in which we could code something.

The following lemma tells us how to find such words in the general case of minimal effective subshifts; a *minimal subshift* is a subshift in which every pattern that appears in a configuration appears in every configuration, or equivalently, a subshift that does not admit a proper non-empty subshift. In this case, all configurations are of course quasi-periodic.

Lemma 3.1. *For any (non-empty) $1-$dimensional minimal effective subshift $\mathbf{X} \subseteq \Sigma^{\mathbb{Z}}$ that has no periodic configuration there exists a computable sequence $(u_n)_{n\in\mathbb{N}}$ of words in the language of $\mathbf{X}$ such that no u_n is prefix of another one.*

Proof. We build recursively a sequence $(u_0, \dots, u_n)$ and a word v_n such that the set $\{u_k, k \leq n\} \cup \{v_n\}$ is prefix-free. For $n = 0$, take two different letters in Σ ($|\Sigma| > 1$ comes from the hypothesis as $\mathbf{X}$ is non-empty and does not contain any periodic configuration).

Now suppose we obtain $(u_0, \dots u_n)$ and v_n. Since $\mathbf{X}$ is supposed to be minimal, v appears in an uniformly recurrent way in a configuration of $\mathbf{X}$ and since $\mathbf{X}$ contains no periodic configuration, there exists two different right-extensions of v: w and w' of the same length. Taking $u_{n+1} = w$ and $v_{n+1} = w'$ ends the recurrence. ■

[3] The definitions are usually given in dimension one, *i.e.*, for (bi-)infinite, words even though they are the same for multi-dimensional configurations.

To obtain our theorem, we will need a subshift $\mathbf{X}$ for which we control precisely the sequence u_n.

Lemma 3.2. *There exists a (non-empty) 1-dimensional minimal effective subshift* $\mathbf{X}$ *and a computable sequence* $(u_n)_{n\in\mathbb{N}}$ *of words in the language of* $\mathbf{X}$ *so that* $|u_n| \leq n$ *and no* u_n *is prefix of another one.*

Proof. We will use a construction based on Toeplitz words. Let p be an integer. For an integer n, denote by $\phi_p(n)$ the first non-zero digit in the writing of n in base p, *e.g.*, $\phi_3(15) = 2$.

Let $w_p = \phi_p(1)\phi_p(2)\ldots$. For example $w_4 = 12311232123312311231123\ldots$.

Now let $\mathbf{X}_p$ be the shift of all configurations c so that all words of c are words of w_p. Note that any word of size n appearing in w_p appears at a position less than p^n so that $\mathbf{X}_p$ is an effective subshift.

Now the following statements are clear:

- For every word w in w_p, there exists k so that for every configuration $c \in \mathbf{X}_p$, w appears periodically in c of period p^k (w might appear in some other places)
- $\mathbf{X}_p$ is minimal (a consequence of the previous statement)

If u_1 and u_2 are two words over Σ_1 and Σ_2 of the same size, we write $u_1 \otimes u_2$ for the word over $\Sigma_1 \times \Sigma_2$ whose ith projection is u_i ($i \in \{1, 2\}$).

Now let $\mathbf{X} = \mathbf{X}_7 \otimes \mathbf{X}_8$. $\mathbf{X}$ is a shift, and $\mathbf{X}$ is minimal[4]: If $c_1 \otimes c_2 \in \mathbf{X}_7 \otimes \mathbf{X}_8$ and u_1 and u_2 are two patterns resp. of w_7 and w_8 of the same size, then u_1 appears periodically in c_1 of period 7^{k_1} and u_2 appears periodically in c_2 of period 8^{k_2}. As these two numbers are relatively prime, there exists a common position i so that u_1 (resp. u_2) appears in position i in c_1 (resp c_2), so that $u_1 \otimes u_2$ appears in $c_1 \otimes c_2$.

Now we can find the sequence u_n.

Let u be a word in $\{5, 6, 7\}^\star\{1, 2, 3, 4\}$. We define $f_8(u)$ inductively as follows:

- If $|u| = 1$, then $f_8(u) = u$.
- If $u = xu_1$ then let $v = f_8(u_1)$ and n be the length of v.
  - If $x = 5$ then $f_8(u) = 567v_1 1234567v_2 1234567v_3 1\ldots v_n 1234$
  - If $x = 6$ then $f_8(u) = 67v_1 1234567v_2 1234567v_3 1\ldots v_n 12345$
  - If $x = 7$ then $f_8(u) = 7v_1 1234567v_2 1234567v_3 1\ldots v_n 123456$

Now it is clear that each $f_8(u)$ is in w_8 and by a straightforward induction, no $f_8(u)$ is prefix of another. Let $S_8 = \{f_8(u)|u \in \{5, 6, 7\}^\star\{1, 2, 3, 4\}\}$ Note that $f_8(u)$ is of length $8^{|u|-1}$. In particular we have $4 \times 3^{n-1}$ words of length 8^{n-1} in S_8.

We do the same with w_7, with words $u \in \{4, 5, 6\}^\star\{1, 2, 3\}$, to obtain a set S_7 containing $3 \times 3^{n-1}$ words of length 7^{n-1}. We can always enlarge all words in S_7 to obtain a set S'_7 containing $3 \times 3^{n-1}$ words of length 8^{n-1}.

Now take $S = S'_7 \otimes S_8$. This set contains $12 \times 9^{n-1} > 8^n$ words of size 8^{n-1} for each n and no word of S is prefix of one another. Now an enumeration in increasing order of S gives the sequence $(u_n)_{n\in\mathbb{N}}$.

The whole construction is clearly effective. ■

Theorem 3.3. *Given a partial computable function* φ*, there exists a* $1-$*dimensional effective subshift* $\mathbf{X}_\varphi$ *such that any quasi-periodic configuration* c *in* $\mathbf{X}_\varphi$ *has a quasi-periodicity function* $\mathcal{Q}_c$ *such that* $\mathcal{Q}_c(n) \geq \varphi(n)$ *when* $\varphi(n)$ *is defined.*

Proof. Consider the subshift $\mathbf{X}$ and the computable sequence $(u_n)_{n\in\mathbb{N}}$ that are given by Lemma 3.2. Since Lemma 3.2 ensures that $|u_n| \leq n$, a sequence $(u_n)_{n\in\mathbb{N}}$ with

[4] Note that the Cartesian product of two minimal shifts is not always minimal [24].

the additional property that $|u_n| = n$ is also computable since we can compute an extension of the words u_n in $\mathbf{X}$ since it is minimal and effective and the prefix-free property is retained while taking extensions. We assume this additional property in this proof.

Let $\Sigma' = \Sigma \times \{0,1\}$. We define $\mathbf{X}_\varphi$ as a subshift of $\mathbf{X} \times \{0,1\}^{\mathbb{Z}}$.

Compute in parallel all the $\varphi(n)$. When $\varphi(n)$ is computed we add the following additional constraints: On the $\{0,1\}$ layer of Σ' we force a 1 to appear on the first letter of u_n once every $\varphi(n)+1$ occurrences of u_n, the first letter of all other occurrences of u_n being 0. There is no ambiguity since no u_n is prefix of another one. This defines $\mathbf{X}_\varphi$ as an effective subshift since $\mathbf{X}$ is effective and $(u_n)_{n\in\mathbb{N}}$ is computable.

Every u_n appears in every configuration of $\mathbf{X}$ since it is minimal. If $\varphi(n)$ is defined, then every u_n with a 1 on the $\{0,1\}$ layer appears exactly every $\varphi(n)$ occurrences of u_n's with a 0 on its $\{0,1\}$ layer in every configuration of $\mathbf{X}_\varphi$. Therefore, for any quasi-periodic configuration c of $\mathbf{X}_\varphi$ we have that $\mathcal{Q}_c(n) \geq \varphi(n)$ where $\varphi(n)$ is defined which completes the proof. ∎

Corollary 3.4. *There exists a 1-dimensional effective subshift $\mathbf{X}$ such that every quasi-periodic configuration c in $\mathbf{X}$ has a quasi-periodicity function which is not bounded by any computable function.*

Proof. Let $(\varphi_n)_{n\in\mathbb{N}}$ be an effective enumeration of partial computable functions.

Let $\varphi(n) = \varphi_n(n) + 1$; φ is also a partial computable function; we can therefore find an effective one dimensional subshift $\mathbf{X}_\varphi \subseteq \Sigma^{\mathbb{Z}}$ via Theorem 3.3 such that any quasi-periodic configuration c of $\mathbf{X}_\varphi$ is such that $\mathcal{Q}_c \geq \varphi$ where φ is defined, hence $\mathcal{Q}_c$ is not recursively bounded. ∎

Theorem 3.5. *There exists a tileset such that every quasi-periodic tiling has a quasi-periodicity function that is not recursively bounded.*

Proof. Take the effective 1−dimensional subshift of the previous corollary (as a subshift of $\Sigma^{\mathbb{Z}}$): $\mathbf{X}_\varphi$. There exists a set of tilings (or 2−dimensional SFT) $\mathbf{X}^2_\varphi \subseteq (Q \times \Sigma)^{\mathbb{Z}^2}$ encoding it [1, 10] in the following way:

In any configuration of $\mathbf{X}^2_\varphi$, the rows of the $\Sigma-$layer are identical, that is, if we write this configuration as $c_Q \times c_\Sigma \in Q^{\mathbb{Z}^2} \times \Sigma^{\mathbb{Z}^2}$, for any i, j in $\mathbb{Z}$, $c_\Sigma(i,j) = c_\Sigma(i,j+1)$. Moreover, the projection:

$$\begin{array}{rccl} p: & (Q \times \Sigma)^{\mathbb{Z}^2} & \to & \Sigma^{\mathbb{Z}} \\ & c_Q \times c_\Sigma & \to & \begin{array}{ccc} \mathbb{Z} & \to & \Sigma \\ n & \to & c_\Sigma(n,0) \end{array} \end{array}$$

of $\mathbf{X}^2_\varphi$ is exactly $\mathbf{X}_\varphi$ (*i.e.*, $p(\mathbf{X}^2_\varphi) = \mathbf{X}_\varphi$). Since the configurations of $\mathbf{X}^2_\varphi$ are the Cartesian product of a construction layer (the $Q^{\mathbb{Z}^2}$ part) and the effective 1−dimensional subshift $\mathbf{X}_\varphi$ repeated on the rows, the quasi-periodicity function of any quasi-periodic configuration of $\mathbf{X}^2_\varphi$ is greater or equal to the quasi-periodicity function of the quasi-periodic 1-dimensional configuration it represents. ∎

Note that quasi-periodicity configurations obtained in the constructions in [1, 10] are rather benign. If we start from a 1−dimensional quasi-periodic configuration c, then the quasi-periodic tilings x that are projected onto c have a quasi-periodicity function that is computable knowing the quasi-periodicity function of c.

4. Note

Theorem 9 in [6] stated the contrary of Theorem 1.4: "there exists a tileset such that all its tilings are quasi-periodic and none of its quasi-periodicity function is computably bounded". Besides some errors that can be easily corrected, there is a big problem in the construction they claim to give. They encode K, a recursively enumerable but not recursive set, in every tiling in a way such that if $i \in K$ then it must appear in every tiling in a pattern of size $g(i)$ where g is a computable function. This property allows by itself to decide K: For an integer i, compute $g(i)$ and all the possible encodings of i if it were to appear in a tiling; patterns that do not appear in a tiling of the plane are recursively enumerable[5] and thus, when we have enumerated all the patterns coding i we know that $i \notin K$. Since K is supposed recursively enumerable, this allows to decide K.

References

[1] Nathalie Aubrun and Mathieu Sablik. Simulation of effective subshifts by two-dimensional SFT and a generalization. *preprint*, 2010.

[2] Alexis Ballier and Emmanuel Jeandel. Tilings and model theory. *First Symposium on Cellular Automata Journées Automates Cellulaires.*, 2008.

[3] Robert Berger. The undecidability of the domino problem. *Memoirs of the American Mathematical Society*, 66, 1966.

[4] George David Birkhoff. Quelques théorèmes sur le mouvement des systèmes dynamiques. *Bulletin de la Société Mathématique de France*, 1912.

[5] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer, 1997.

[6] Julien Cervelle and Bruno Durand. Tilings: recursivity and regularity. *Theoretical computer science*, 310(1-3):469–477, 2004.

[7] Jean-Charles Delvenne, Petr Kůrka, and Vincent D. Blondel. Computational universality in symbolic dynamical systems. *Machines, Computations, and Universality 4th International Conference, MCU 2004*, pages 104–115, 2004.

[8] Bruno Durand. Tilings and quasiperiodicity. *Theoretical Computer Science*, 221(1-2):61–75, 1999.

[9] Bruno Durand, Leonid A. Levin, and Alexander Shen. Complex Tilings. *Journal of Symbolic Logic*, 73(2):593–613, 2008.

[10] Bruno Durand, Andrei Romashchenko, and Alexander Shen. Effective Closed Subshifts in 1D Can Be Implemented in 2D. In *Fields of Logic and Computation*, number 6300 in Lecture Notes in Computer Science, pages 208–226. Springer, 2010.

[11] William P. Hanf. Nonrecursive Tilings of the Plane. I. *Journal of Symbolic Logic*, 39(2):283–285, 1974.

[12] Gustav Arnold Hedlund. Endomorphisms and automorphisms of the shift dynamical systems. *Mathematical Systems Theory*, 3(4):320–375, 1969.

[13] Michael Hochman. A note on universality in multidimensional symbolic dynamics. *Discrete and Continuous Dynamical Systems S*, 2(2), 2009.

[14] Michael Hochman. On the dynamics and recursive properties of multidimensional symbolic systems. *Inventiones Mathematica*, 176(1), April 2009.

[15] Douglas Lind. Multidimensional Symbolic Dynamics. In *Symbolic dynamics and its applications*, volume 11 of *Proceedings of Symposia in Applied Mathematics*, pages 61–80, 2004.

[16] Douglas A. Lind and Brian Marcus. *An Introduction to Symbolic Dynamics and Coding.* Cambridge University Press, New York, NY, USA, 1995.

[17] Joseph S. Miller. Two notes on subshifts.

[5] Simply try to tile arbitrary big patterns around it and if it is not possible claim that the pattern does not appear in a tiling.

[18] Marston Morse and Gustav Arnold Hedlund. Symbolic Dynamics. *American Journal of Mathematics*, 60(4):815–866, October 1938.

[19] Shahar Mozes. Tilings, substitution systems and dynamical systems generated by them. *Journal d'analyse mathématique*, 53:139–186, 1988.

[20] An. Muchnik, A. Semenov, and M. Ushakov. Almost periodic sequences. *Theoretical Computer Science*, 304(1-3):1–33, 2003.

[21] Dale Myers. Nonrecursive Tilings of the Plane. II. *Journal of Symbolic Logic*, 39(2):286–294, 1974.

[22] Nicolas Ollinger. Two-by-two substitution systems and the undecidability of the domino problem. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *CiE*, volume 5028 of *Lecture Notes in Computer Science*, pages 476–485. Springer, 2008.

[23] R. M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12:177–209, 1971.

[24] Pavel Salimov. On Uniform Recurrence of a Direct Product. AutoMathA 2009, To Appear in DMTCS.

[25] Steve Simpson. Medvedev degrees of 2-dimensional subshifts of finite type. *Ergodic Theory and Dynamical Systems*, 2007.

[26] Hao Wang. Proving Theorems by Pattern Recognition I. *Communications of the ACM*, 3(4):220–234, April 1960.

[27] Hao Wang. Proving theorems by pattern recognition II. *Bell system technical journal*, 40:1–41, 1961.

[28] Klaus Weihrauch. *Computable analysis*. Springer, 2000.

A SIMULATION OF OBLIVIOUS MULTI-HEAD ONE-WAY FINITE AUTOMATA BY REAL-TIME CELLULAR AUTOMATA

ALEX BORELLO

Laboratoire d'Informatique Fondamentale de Marseille, 39 rue Frédéric Joliot-Curie, 13453 Marseille, France
E-mail address: alex.borello@lif.univ-mrs.fr

Abstract. In this paper, we present the simulation of a simple, yet significantly powerful, sequential model by cellular automata. The simulated model is called oblivious multi-head one-way finite automata and is characterised by having its heads moving only forward, on a trajectory that only depends on the length of the input. While the original finite automaton works in linear time, its corresponding cellular automaton performs the same task in real time, that is, exactly the length of the input. Although not truly a speed-up, the simulation may be interesting and reminds us of the open question about the equivalence of linear and real times on cellular automata.

1. Introduction

Cellular automata (CA for short), first introduced by J. von Neumann [7] as self-replicating systems, are recognised as a major model of massively parallel computation since A. R. Smith, in 1969, used this Turing-complete model to compute functions [8]. Their simple and homogeneous description as well as their ability to distribute and synchronise the information in a very efficient way contribute to their success. However, to determine to what extent CA can fasten sequential computation is not a simple task.

As regards specific sequential problems, the gain in speed by the use of CA is manifest [1, 2, 3]. But when we try to get general simulations, we have to face the delicate question of whether parallel algorithms are always faster than sequential ones. An inherent difficulty arises from the fact that efficient parallel algorithms make often use of techniques that are radically different from the sequential ones. There might also exist a faster CA for each singular sequential solution whereas no general simulation exists.

Hence, no surprise: the known simulations of Turing machines by CA provide no parallel speed-up. The early construction of Smith [8] simulates one step of the Turing machine by one step of the CA. Furthermore, no faster simulations have been reported yet, even for almost all restricted variants. In particular, we do not

2000 ACM Subject Classification: F.1.1, F.1.2.

Key words and phrases: simulation, oblivious multi-head one-way finite automata, cellular automata, real time.

know whether any finite automata with k heads can be simulated on CA in less than $O(n^k)$ steps, which is the sequential time complexity.

We will not give answers to such issues here, but we shall examine in this context a simple sequential model, called oblivious multi-head finite automata. This device was introduced by M. Holzer in [4] as multi-head finite automata with an additional constraint of obliviousness: the trajectory of the heads only depends on the length of the input. As emphasised in [4], such finite automata lead to significant computational power: they characterise parallel complexity NC^1. Their properties have been further discussed in [5].

We will focus on the one-way version of this model, that is, for which the reading heads can only move forward (that makes it strictly less powerful). While no true speed-up can be hoped for, as these one-way finite automata already perform their task in linear time, we will describe a simulation of them by real-time CA, that is, CA working in linear time with a multiplicative constant equal to 1. Whereas specifying this constant is usually irrelevant, CA represent a particular case amongst models of computation, as we do not know whether linear and real times are equivalent for it.

The article is organised as follows: section 2 introduces the two models considered, section 3 displays some of their features and abilities and section 4 presents the simulation algorithm.

2. Definitions

2.1. Multi-head finite automata

Given an integer $k \geq 1$, a one-way k-head finite automaton is a finite automaton reading an input word using k heads that can move to the right or stand still.

Definition 2.1. A (deterministic) *one-way multi-head finite automaton* (1DFA(k) for short) is a septuple $(\Sigma, Q, \lhd, q_0, Q_\mathrm{a}, k, \delta)$, where Σ is a finite set of *input symbols* (or *letters*), Q is a finite set of *states*, $\lhd \notin \Sigma$ is the (right) *end-marker*, $q_0 \in Q$ is the *initial* state, $Q_\mathrm{a} \subseteq Q$ is the set of the *accepting* states, $k \geq 1$ is the *number of heads* and $\delta : Q \times (\Sigma \cup \{\lhd\})^k \to Q \times \{0,1\}^k$ the *transition function*; 1 means to move the head one letter to the right and 0 to keep it on its current letter. For the heads to be unable to move beyond the end-marker, we require that if $\delta(q, a_1, \ldots, a_k) = (q', m_1, \ldots, m_k)$, then for any $i \in [\![1, k]\!]$, $a_i = \lhd \Rightarrow m_i = 0$.

A *configuration* of a 1DFA(k) on an input word $w \in \Sigma^n$ at a certain time $t \geq 0$ is a couple (p, q) where $p \in [\![0, n]\!]^k$ is the position of the multi-head and q the current state. The computation of such a device on this input word starts with all heads on the first letter, and ends when all heads have reached the end-marker. If the current state is then within Q_a, the word is said to be accepted, otherwise it is rejected. The language $L(\mathcal{F})$ *recognised* by a 1DFA(k) $\mathcal{F}$ is the set of the words accepted by $\mathcal{F}$. One can notice a 1DFA(k) ends its computation in linear time.

We will focus now on data-independent 1DFA (1DIDFA), a particular class of 1DFA for which the path followed by the heads only depends on the length of the input word, not on the letters thereof.

Definition 2.2. Given $k \geq 1$, a 1DFA(k) $\mathcal{F}$ is said to be *oblivious* (or *data-independent*) if there exists a function $f_{\mathcal{F}} : \mathbb{N}^2 \to \mathbb{N}^k$ such that the position of its multi-head at time $t \in \mathbb{N}$ on any input word w is $f_{\mathcal{F}}(|w|, t)$.

2.2. Cellular automata

A cellular automaton is a parallel synchronous computing model consisting of an infinite number of finite automata called *cells* which are distributed on $\mathbb{Z}$ and share the same transition function, depending on the considered cell's previous state as well as its two neighbours'.

Definition 2.3. A *cellular automaton* is a quintuple $(\Sigma, Q, \#, Q_{\mathrm{a}}, \delta)$, where Σ is the finite set of *input symbols* (or *letters*), $Q \supset \Sigma$ is the finite set of *states* and $\delta : Q^3 \to Q$ the *transition function*[1]. $\# \in Q \setminus \Sigma$ is a particular *quiescent* state, verifying $\delta(\#, \#, \#) = \#$. $Q_{\mathrm{a}} \subseteq Q$ is the set of the *accepting* states.

A *configuration* is a function $\mathfrak{C} : \mathbb{Z} \to Q$. A *site* is a cell at a certain time step of the computation we consider; $\langle c, t \rangle$ will denote the state of the site $(c, t) \in \mathbb{Z} \times \mathbb{N}$. The computation of a CA $\mathcal{C}$ on an input word w of size $n \geq 1$ starts at time 0 with all cells in state # except cells 0 to $n - 1$ where the letters of the word are written. This is the initial configuration $\mathfrak{C}_w$ associated to w. Then the cells update in parallel their respective states according to δ: for all $(c, t) \in \mathbb{Z} \times \mathbb{N}$, $\langle c, t + 1 \rangle = \delta(\langle c - 1, t \rangle, \langle c, t \rangle, \langle c + 1, t \rangle)$.

This input word is accepted in time $t \geq n$ if and only if cell 0 (the origin) is in an accepting state at time t. The language $L_\tau(\mathcal{C})$ *recognised* by the automaton in time $\tau : \mathbb{N} \to \mathbb{N}$ is the set of the words w it accepts in time $\tau(|w|)$. If τ is the identity function Id, $L_\tau(\mathcal{C})$ is said to be recognised in *real time.*

Real time represents for CA the most simple time complexity that is nontrivial, in the sense it is the minimal time required for the output to depend on all letters of the input. Yet, it is significantly powerful, as we do not even know whether linear time can achieve strictly more. Real time had already been evoked in [8].

3. Preliminaries

We would like to simulate a 1DIDFA on a CA as fast as possible. A computation of a general 1DFA requires a number of time steps that is linear in the size of the input word. Whereas it is rather easy for a CA to simulate such a device in linear time, there is a priori no obvious way to reduce this time bound. But we can do it in the case of DIDFA by taking the constraint of obliviousness into account. Though, before performing such a simulation, we should detail some useful features of DIDFA and CA.

[1] Notice CA are defined herein with the standard neighbourhood of radius 1, that is, such that the state of a cell at time $t+1$ depends on the states at time t of this same cell and its two nearest neighbours.

3.1. Some features of multi-head finite automata

Let $\mathcal{F} = (\Sigma, Q, \lhd, q_0, Q_\mathrm{a}, k, \delta)$ be a 1DIDFA, $n \geq 1$ be an integer and $w \in \Sigma^n$ be a word of size n. Let us look at the computation of $\mathcal{F}$ on input word w. For the multi-head is composed of k heads, it can be regarded as a device moving one point at a time in any direction within the set $\mathcal{W} = [\![0, n]\!]^k$.

As $\mathcal{F}$ is data-independent, we can separate the path P taken by the multi-head from the consecutive states of the automaton (depending on the letters of w). In other words, we can take a look at the path of the multi-head on input word a^n, for any $a \in \Sigma$; it will be the same for w. Hence, the trajectory will become periodic after at most $|Q|$ moves, until one head reaches an end-marker. Then, while the latter head does not move any longer, after another $|Q|$ moves the trajectory will become periodic again, and so on until all heads have reached the end of the input word. The key points of $\mathcal{W}$ where a head reaches the end-marker will be useful to us and denoted as finite sequence $(p_i)_{i \in [\![0,k]\!]}$, with $p_0 = (0, \dots, 0)$ and $p_k = (n, \dots, n)$.

Some notations. For convenience, we number the heads such that for all $i \in [\![0, k-1]\!]$, head i is the one that reaches the end-marker as the multi-head arrives at key point p_{i+1}. For all $i \in [\![0, k]\!]$ and all $j \in [\![0, k-1]\!]$, we denote the $(j+1)$-th coordinate of p_i by $p_{i,j}$, and if $i < k$ name $P_i \subseteq P$ the portion of trajectory that lies between p_i and p_{i+1}.

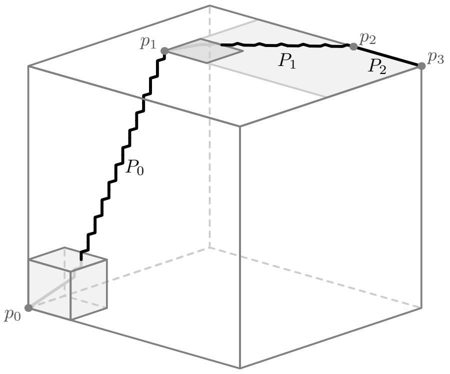


Figure 1: A representation of $\mathcal{W}$ for $k = 3$. The periodic parts of the path of the multi-head are drawn in black.

3.2. A few basic techniques on cellular automata

A given computation of a CA can be easily represented by drawing successive configurations each one above its predecessor. We thus obtain a *space-time diagram*, composed of sites, of which we only need to represent those in a non-quiescent state.

We will often have to perform several rather independent computations at the same time; this can easily be done by a 'product' automaton which works with a finite number of *layers*, each one of which supports a specific computation. Although rather independent, the layers can communicate between one another to exchange information, as any cell can see all of them.

Compression of the input word. In section 4, we will need to compress the input by some rational factor $\rho \geq 2$. This is easy to do with a CA. It consists in having the input word written on the (discrete) straight line of equation $t - 1 = (\rho - 1)(c + 1)$, where t represents the time and c a cell, as shown on fig. 2. As the concerned sites 'know' that they lie on this straight line, a computation using the compressed input word can then occur within the triangle of real time (in light grey on fig. 2).

Acceleration by a constant. For any constant $T \in \mathbb{N}$ and any CA $\mathcal{C}$, there exists a CA $\mathcal{C}'$ such that $L_{\mathrm{Id}}(\mathcal{C}') = L_{\mathrm{Id}+T}(\mathcal{C})$. In other words, to prove that a given language is CA-recognisable in real time, it suffices to exhibit a CA recognising it in time $\mathrm{Id} + T$. For more details, one can refer to [6].

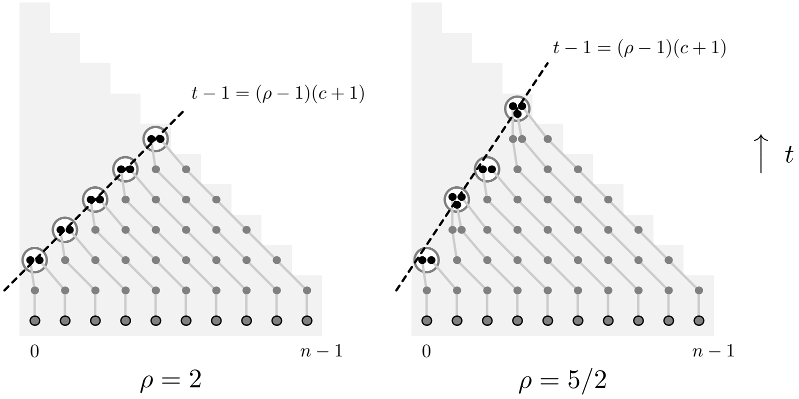


Figure 2: Schematic space-time diagrams during which input word w is compressed by rational factor ρ. Each sequence of linked dots represent a letter of w. The sites containing the compressed version of w are encircled. Notice that even though it seems the letters could be shifted one time step earlier, this first step is in fact used to mark the last letter; it is necessary because of rounding issues.

4. Simulation

Theorem 4.1. *Given $k \geq 1$, for any 1DIDFA(k) $\mathcal{F}$ recognising a language $\mathcal{L}$, there exists a CA $\mathcal{C}$ recognising $\mathcal{L}$ in real time.*

The rest of this paper will be devoted to the proof of this theorem. We assume now that we have a 1DIDFA(k) $\mathcal{F} = (\Sigma,\ Q, \lhd, q_0, Q_{\mathrm{a}}, k, \delta)$. We will define a CA $\mathcal{C} = (\Sigma, Q', \#, Q'_{\mathrm{a}}, \delta')$ such that $L_{\mathrm{Id}}(\mathcal{C}) = \mathcal{L}$. Instead of giving the full description of its state set and transition function, we will describe its behaviour on an arbitrary input word $w \in \Sigma^n$, given an integer $n \geq 1$. Within this coming description (and similarly in the whole article) the terms 'constant' and 'finite' refer to quantities that do not depend on n.

4.1. Principle

The general principle of the simulation is rather simple: instead of having k heads moving along w, we will have (at least) k copies of w shifted over a segment S of sites (of strictly increasing time steps) so that each site sees the correct letters of w. Moreover, the letters for each head will be seen in reverse order compared to what $\mathcal{F}$ does.

Each part P_i of the trajectory of the multi-head can be assimilated to a discrete straight line, with no aperiodic part. Indeed, as illustrated in fig. 3, the distance (in letters) between any point of P_i and the point of this line corresponding to same time step is bounded by some value $K = O(|Q|)$. Thus, during the execution of $\mathcal{C}$ over w, before doing anything, all cells bearing the input will gather the letters of their K nearest neighbours. This is done in time K.

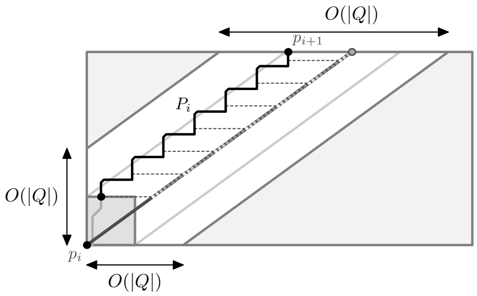


Figure 3: P_i lies within a band of width $O(|Q|)$, here drawn in white. It can hence be assimilated to a (discrete) straight line, provided a counter (within the shifting copies of w during the execution of $\mathcal{C}$) indicates for each point of this line the corresponding position within the period of P_i. Notice that although the band can broaden as i increases, this index only rises up to a constant value, so that the maximal width K remains bounded independently of the size of the input.

4.2. Key sites

We will set $S = \{(c, n-1-c+T) \ : \ c \in [\![0, n-1]\!]\}$, where T, which is to be defined (cf. subsection 4.3), is an integer greater than K that does not depend on n. The result of the execution is to appear on site $s_0 = (0, n-1+T)$. To know which speed the copies of w should be shifted at over each site of S, the latter segment should be divided into parts S_i, each one of which corresponds to part P_i of P. In other words, we want to mark some key sites $s_i = (c_i, n-1-c_i+T) \in S$ that represent key points $p_i \in P$. The main difficulty is that key cell c_i has to represent coordinate $p_{i,j}$ for any head j.

For this purpose, we observe first that for all $(i,j) \in [\![0,k]\!] \times [\![0,k-1]\!]$, since each part of P is as illustrated in fig. 3, there exists $\alpha_{i,j} \in \mathbb{Q} \cap [0,1]$ such that $|p_{i,j} - \alpha_{i,j} n| \leq K$, whatever the size n of the input. One can notice that we automatically have $\alpha_{0,j} = 0$ and $\alpha_{i,j} = 1$ for all $i > j$, and that $(\alpha_{i,j})_i$ is an increasing sequence for all j.

Then, we provisionally assume that $\alpha_{j,j} = 0 \Rightarrow j = 0$, and set key cell $c_i = \lfloor \alpha_i n \rfloor$, where $\alpha_i = \frac{1}{2} \prod_{j=i}^{k-1} \alpha_{j,j}$. The case wherein there exists some j that does not verify this hypothesis will be treated in subsection 4.6.

Now, how to mark site s_i? No trouble if $i = 0$, as c_0 is the origin. If $i > 0$, it is also feasible: it suffices to send a signal from the origin at speed $\varsigma_i = \frac{\alpha_i}{1-\alpha_i} \leq 1$ (cf. fig. 4). Note that in the definition of α_i, we have divided by 2 in case some key cells would be too far from the origin to be marked in time (in CA configurations, information cannot travel at speed of absolute value strictly greater than 1). All our computation has hence to be performed within half as much space than what S provides. In any case, the definition of α_i is based on the assumption that the copies of the input shifting over S_i are compressed versions of w.

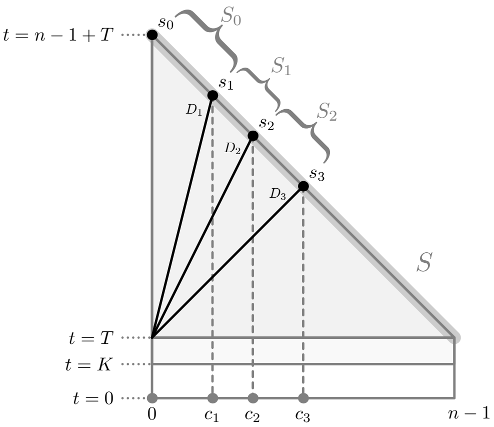


Figure 4: Schematic space-time diagram of the marking of cells c_i, for $k = 3$. Notice that $(c_i)_i = (\lfloor \alpha_i n \rfloor)_i$ is always an increasing sequence (since $\alpha_i = \alpha_{i,i}\alpha_{i+1} \leq \alpha_{i+1}$), with $c_0 = 0$ and $c_k = \lfloor \frac{n}{2} \rfloor$.

4.3. Compression of the input

For each $i \in [\![1, k]\!]$, we want to compress input word w (on a specific layer ℓ_{i-1} corresponding to head $i-1$) by factor $\frac{1}{\alpha_i}$ as illustrated in fig. 2, that is, on some straight line D_i of direction vector $(1, \frac{1}{\varsigma_i}) = (1, \frac{1}{\alpha_i} - 1)$. One can notice we are able to choose D_i such that it crosses the origin at any time $t > \lfloor \frac{1}{\varsigma_i} \rfloor$. Thus, we will make all such lines cross the origin at the same time $T \in \mathbb{N}$. As $(\frac{1}{\varsigma_i})_i$ is a decreasing sequence and as we have done some computations in time K beforehand, we set $T = K + \lfloor \frac{1}{\varsigma_1} \rfloor$. Hence, we have finally set D_i to be the line of equation $t - T = \frac{c}{\varsigma_i}$ (cf. fig. 4).

4.4. Shift of the input

Consider some head $j \in [\![0, k-1]\!]$ and an integer $i \in [\![1, j]\!]$. On layer ℓ_j, which corresponds to this head, we want to shift the compressed input at some constant speed $\varsigma_{i,j} \in]-1, \varsigma_i]$ between D_{i+1} and D_i, so that the correct letters pass over S_i. One can notice $\varsigma_{j,j} = 0$ by the definition of α_{j+1} and α_j. But this not necessarily the case when $i < j$. Indeed, $\varsigma_{i,j}$ should be defined as equal to $\frac{\beta_{i,j}}{1-\beta_{i,j}}$, with $\beta_{i,j} = \alpha_i - \alpha_{i,j}\frac{\alpha_{i+1}-\alpha_i}{\alpha_{i+1,j}-\alpha_{i,j}}$ if $\alpha_{i+1,j} - \alpha_{i,j} > 0$ and $\beta_{i,j} = \alpha_i$ otherwise. This way, $\varsigma_{i,j}$ is the speed of the signal we would use to mark cell $c_{i,j} = \lfloor \beta_{i,j} n \rfloor$ (cf. fig. 5).

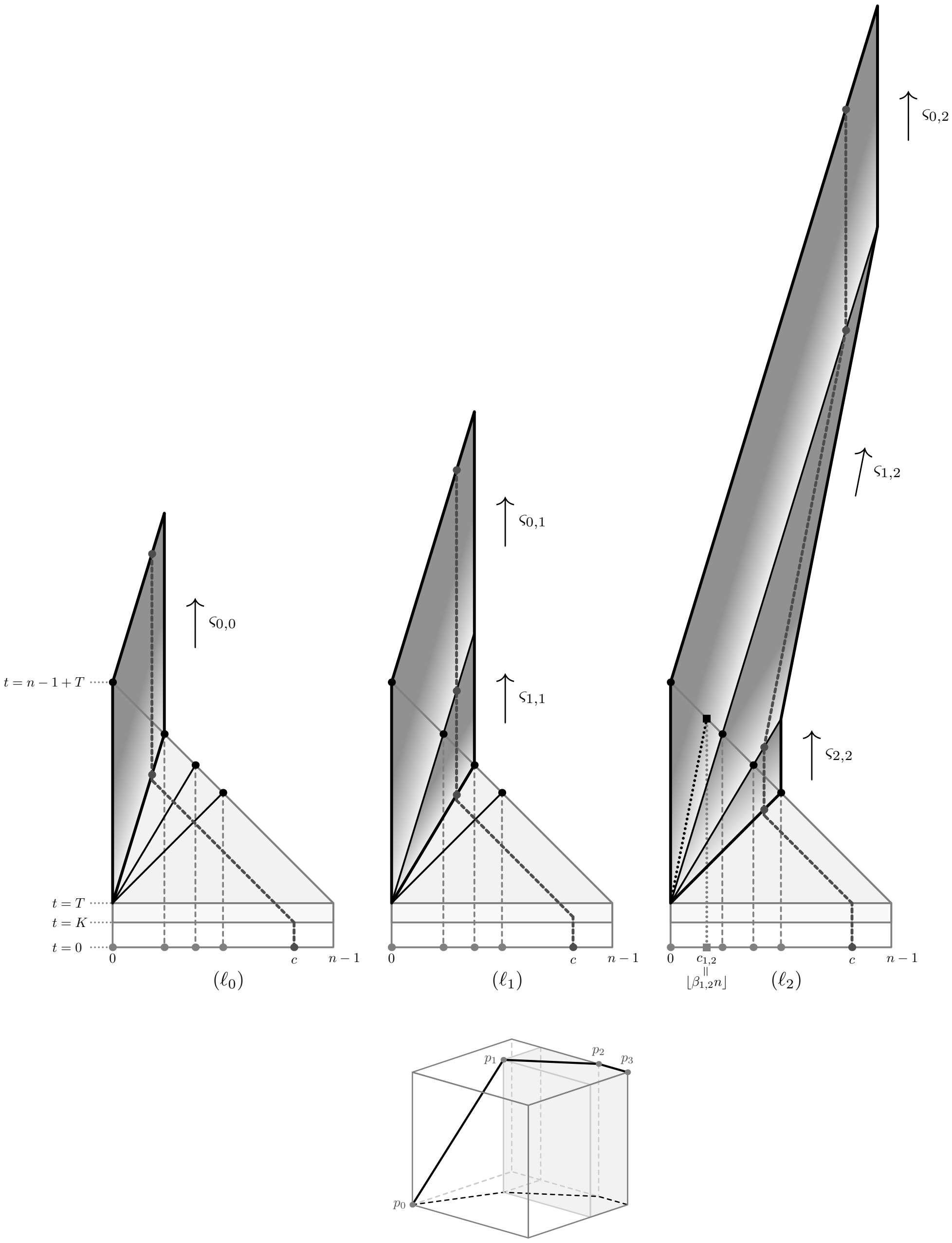


Figure 5: Different compressed copies of the input shifted over S, with the trajectory of the letter initially contained by some cell c displayed. Layer ℓ_j corresponds to head $j < k = 3$. In this example, we have $(\alpha_{1,1}, \alpha_{1,2}, \alpha_{2,2}) = (\frac{5}{8}, \frac{1}{4}, \frac{3}{4})$. Hence, $(\alpha_0, \alpha_1, \alpha_2, \alpha_3) = (0, \frac{15}{64}, \frac{3}{8}, \frac{1}{2})$ and $\beta_{1,2} = \frac{21}{128}$.

4.5. Backtracking

Now that we have ensured the correct letters are seen in reverse order for each head on each segment S_i, how do we get site s_0 to know the result of the execution of $\mathcal{F}$ over w? All we need to know is whether the final state of $\mathcal{F}$ is accepting, that is, belongs to Q_{a}.

Let p be a point of P such that $p \neq p_0$. One can observe that if we know q, the state $\mathcal{F}$ is in when its multi-head is on p, as well as the letter $l_j \in \Sigma$ each head j reads when the multi-head lies on the predecessor p' of p, then we can compute the possible states of $\mathcal{F}$ at point p'. That is, the subset Q' of Q such that for all $q' \in Q$, $\delta(q', l_0, \ldots, l_{k-1}) = (q, p - p') \Leftrightarrow q' \in Q'$. Likewise, if we know $\mathcal{F}$ is in a state of $Q'' \subseteq Q$ at point p, we can determine the subset Q' such that for all $q' \in Q$, $\delta(q', l_0, \ldots, l_{k-1}) \in Q'' \times \{p - p'\} \Leftrightarrow q' \in Q'$. We will refer to this process as *reading δ backward.*

Let then $s = (c + 1, t - 1)$ be a site of S. As the letters it sees come from compressed versions of w, it can represent a (finite) range of points of P instead of only one, depending on the part S_i it belongs to. Now suppose it contains some subset of Q for each of the successive points of P it represents. Suppose also these subsets are consistent with one another (regarded as the possible states $\mathcal{F}$ is in at each of these points). Then successor site $s' = (c, t)$ can read δ backward a finite (but sufficient) number of times to get the possible subsets of its own points.

Site s_k represents the last points of P, amongst which the very last point p_k. So, we initiate our 'reverse' computation by setting the state of s_k (on some layer ℓ on which this computation is to be held) to contain subset Q_a for point p_k and consistent ones for the predecessors it represents. By induction, every element of S will contain subsets that are consistent with Q_a on layer ℓ. In particular, s_0 will have the corresponding subset Q_0 for p_0, so that it just has to check whether $q_0 \in Q_0$ to know if w is accepted by $\mathcal{F}$. ■

4.6. Adjustments

In the preceding construction, we have put some details or particular cases aside. First, we have to mention that the whole process obviously works only for input words of size greater than some value depending on K (for all P_i to be assimilated to straight lines as in fig. 3). Nevertheless, that leaves us a finite number of words that are treated as special cases, so that the result is not affected.

Possibilities. As each P_i is not a real straight line, the next part P_{i+1} of the path depends on which point of the period of P_i the multi-head is at (that is, which state it is in over word a^n) when head i reaches the end-marker. In particular, there can be at most $|Q|$ possible values α_i, depending on n. Anyway, that makes a finite number of possible $(k-1)$-tuples $(\alpha_1, \ldots, \alpha_{k-1})$, and we can thus process all of them in parallel.

Remains to elect the right tuple at site s_0 or before. It can be done by looking at the remainder of the Euclidean division of $n - |Q|$ by some finite value $f(|Q|)$. That can be easily checked, for instance, on line D_k with a finite counter. The choice will be known at site s_k and spread toward s_0.

Aperiodic parts. It may seem we know at any site along any S_i, thanks to what precedes, which points of the period of P_i we are simulating and so, which available letters the cell has to use. This is in fact not true yet: when reaching site s_i, we have to take the aperiodic part of P_i into account, and therefore we must be able to modify the last $|Q|$ moves (that is, to adjust the choice of letters) we have simulated backward. That can be done by adding to the sites of S a finite memory of the letters seen.

Immobile heads. Suppose that, contrary to the hypothesis made in subsection 4.2, there exists some $j > 0$ such that $\alpha_{j,j} = 0$. That means that head j remains motionless until p_j and then covers the totality of the input during P_j. The trouble is that it implies for all $i \leq j$, $\alpha_i = 0$. Therefore, $s_j = s_{j-1} = \cdots = s_0$, so that a linear number of moves would have to be simulated on a single site.

A simple trick allows us to overcome this problem: for all $j \in [\![1, k-1]\!]$, we set $\alpha'_{j,j} = \alpha_{j,j}$ if $\alpha_{j,j} > 0$ and $\alpha'_{j,j} = \frac{1}{2}$ otherwise[2], and set $\alpha'_{0,0} = \alpha_{0,0} = 0$. Then, in our construction, we replace any $\alpha_{j,j}$ by $\alpha'_{j,j}$.

Finally, for each $j > 0$ verifying $\alpha_{j,j} = 0$, we still have to adjust shift speed $\varsigma_{j,j}$, which is equal to 0. All we have to do is to replace it by $\varsigma'_{j,j} = \frac{\alpha_j}{1-\alpha_j}$ (only for this j), which makes the totality of the copy of w on layer ℓ_j shift over S_j. As regards indices $i < j$, we do not need to redefine the corresponding speed $\varsigma_{i,j}$, since head j makes no more moves.

Conclusion

We have described a construction that simulates oblivious multi-head one-way finite automata on real-time cellular automata. This is better (if linear and real times are not equivalent) than what would achieve the naïve (though nontrivial) simulation of general multi-head finite automata, which would result in a linear-time CA.

In any case, this result fully exploits the obliviousness of the sequential computation. Now, it is another challenge to get a similar parallel algorithm without the constraint of data-independence.

Acknowledgement

I would like to thank G. Richard and V. Terrier for introducing me to the matter of DIDFA (which resulted in a common article about a speed-up of two-way DIDFA by CA). I would also like to thank J. Ferté for useful brainstorming sessions before the blackboard and V. Poupet for his help.

References

[1] A. J. Atrubin. A one-dimensional real-time iterative multiplier. *IEEE Transactions on Electronic Computers*, 14(1):394–399, 1965.

[2] Stephen N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machines. *IEEE Trans. Comput.*, 18(4):349–365, 1969.

[3] Karel Čulík II. Variations of the firing squad problem and applications. *Information Processing Letters*, 30(3):152–157, 1989.

[4] Markus Holzer. Multi-head finite automata: Data-independent versus data-dependent computations. *Theoretical Computer Science*, 286(1):97–116, 2002.

[5] Markus Holzer, Martin Kutrib, and Andreas Malcher. Multi-head finite automata: Characterizations, concepts and open problems. In Turlough Neary, Damien Woods, Anthony Karel Seda, and Niall Murphy, editors, *The Complexity of Simple Programs (CSP'08)*, EPTCS, pages 93–107, 2008.

[6] Jacques Mazoyer and Nicolas Reimen. A linear speed-up theorem for cellular automata. *Theoretical Computer Science*, 101(1):59–98, 1992.

[2] Notice we could have chosen any rational value strictly between 0 and 1 instead of $\frac{1}{2}$.

[7] John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, IL, USA, 1966.

[8] Alvy R. Smith III. Simple computation-universal cellular spaces. *Journal of the ACM*, 18(3):339–353, 1971.

Journées Automates Cellulaires 2010 (Turku), pp. 76-87

CONSTRUCTION OF μ-LIMIT SETS

LAURENT BOYER [1], MARTIN DELACOURT [2], AND MATHIEU SABLIK [3]

[1] LAMA, Université de Savoie

[2] Laboratoire d'Informatique Fondamentale de Marseille, Université de Provence

[3] Laboratoire d'Analyse, Topologie, Probabilités, Université de Provence

ABSTRACT. The μ-limit set of a cellular automaton is a subshift whose forbidden patterns are exactly those, whose probabilities tend to zero as time tends to infinity. In this article, for a given subshift in a large class of subshifts, we propose the construction of a cellular automaton which realizes this subshift as μ-limit set where μ is the uniform Bernoulli measure.

1. Introduction

A cellular automaton (CA) is a complex system defined by a local rule which acts synchronously and uniformly on the configuration space. These simple models have a wide variety of different dynamical behaviors. More particularly it is interesting to understand its behavior when it goes to infinity.

In the dynamical systems context, it is natural to study the limit set of a cellular automaton, it is defined as the set of configurations that can appear arbitrarily far in time. This set captures the longterm behavior of the CA and has been widely studied since the end of the 1980s. Given a cellular automaton, it is difficult to determine its limit set. Indeed it is undecidable to know if it contains only one configuration [Kar92] and more generally, every nontrivial property of limit sets is undecidable [Kar94]. Another problem is to characterize which subshift can be obtained as limit set of a cellular automaton. This was first studied in detail by Lyman Hurd [Hur87], and significant progress have been made [Maa95, FK07] but there is still no characterization. The notion of limit set can be refined if we consider the notion of attractor [Hur90a, Kur03].

However, these topological notions do not correspond to the empirical point of view where the initial configuration is chosen randomly, that is to say chosen according a measure μ. That's why the notion of μ-attractor is introduced by [Hur90b]. Like it is discussed in [KM00] with a lot of examples, this notion is not satisfactory empirically and the authors introduce the notion of μ-limit set. A μ-limit set is a subshift whose forbidden patterns are exactly those, whose probabilities tend to

Thanks to the project ANR EMC: ANR-09-BLAN-0164.

zero as time tends to infinity. This set corresponds to the configurations which are observed when a random configuration is iterated.

As for limit sets, it is difficult to determine the μ-limit set of a given cellular automaton, indeed it is already undecidable to know if it contains only one configuration [BPT06]. However, in the literature, all μ-limit sets which can be found are very simple (transitive subshifts of finite type). In this article, for every recursively enumerable family $(\Sigma_i)_{i\in\mathbb{N}}$ of subshifts generated by a generic configuration, we construct a cellular automaton which realizes $\overline{\bigcup_{i\in\mathbb{N}}\Sigma_i}$ as μ-limit set. In particular all transitive sofic subshifts can be realized. It makes a strong difference with limit sets since there are sofic subshifts, as the even subshift (subshift on alphabet $\{0,1\}$ in which all words 01^k0 with odd k are forbidden), which cannot be realized as limit set [Maa95].

To construct a cellular automaton that realizes a given subshift as μ-limit set, we first erase nearly all the information contained in a random configuration thanks to counters (section 3). Then we produce segments, which are finite areas of computation. On each segment we construct small parts of the generic configurations of many subshifts, and as time passes, segments grow larger and every word of every subshift appears often enough (section 5).

2. Definitions

2.1. Words and density

For a finite set Q called an *alphabet*, denote $Q^* = \bigcup_{n\in\mathbb{N}} Q^n$ the set of all finite words over Q. The *length* of $u = u_0u_1\dots u_{n-1}$ is $|u| = n$. We denote $Q^{\mathbb{Z}}$ the set of *configurations* over Q, which are mappings from $\mathbb{Z}$ to Q, and for $c \in Q^{\mathbb{Z}}$, we denote c_z the image of $z \in \mathbb{Z}$ by c. For $u \in Q^*$ and $0 \le i \le j \le |u|-1$ we define the *subword* $u_{[i,j]} = u_iu_{i+1}\dots u_j$; this definition can be extended to a configuration $c \in Q^{\mathbb{Z}}$ as $c_{[i,j]} = c_ic_{i+1}\dots c_j$ for $i, j \in \mathbb{Z}$ with $i \le j$. The *language* of $S \subset Q^{\mathbb{Z}}$ is defined by

$$\mathcal{L}(S) = \{u \in Q^* : \exists c \in S,\ \exists i \in \mathbb{Z} \text{ such that } u = c_{[i,i+|u|-1]}\}.$$

For every $u \in Q^*$ and $i \in \mathbb{Z}$, we define the *cylinder* $[u]_i$ as the set of configurations containing the word u in position i that is to say $[u]_i = \{c \in Q^{\mathbb{Z}} : c_{[i,i+|u|-1]} = u\}$. If the cylinder is at the position 0, we just denote it by $[u]$.

For all $u, v \in Q^*$ define $|u|_v$ the *number of occurences* of v in u as:

$$|u|_v = \mathrm{card}\{i \in [0, |u| - |v|] : u_{[i,i+|v|-1]} = v\}$$

For any two words $u, v \in Q^*$, let $d_u(v) = \frac{|u|_v}{|u|-|v|}$.

For a configuration $c \in Q^{\mathbb{Z}}$, the *density* $d_c(v)$ of a finite word v is:

$$d_c(v) = \limsup_{n\to+\infty} d_{c_{[-n,n]}}(v).$$

These definitions could be generalized, for a set of words $W \subset Q^*$, we note $|u|_W$ and $d_c(W)$.

Definition 2.1 (Normal configuration)**.** A configuration is said to be *normal* for an alphabet Q if all words of length n have the same density of apparition in the configuration.

2.2. Subshifts

We denote by σ the *shift* map $\sigma : Q^{\mathbb{Z}} \mapsto Q^{\mathbb{Z}}$ defined by $\sigma(c)_i = c_{i-1}$. A *subshift* is a closed, σ-invariant subset of $Q^{\mathbb{Z}}$. It is well known that a subshift is completely described by its language denoted $\mathcal{L}(\Sigma)$. Moreover, it is possible to define a subshift by a set of its forbidden words which do not appear in the language.

As the shift invariance is preserved, intersections and closures of unions of subshifts are still subshifts. And in particular, the union of a set $(\mathcal{L}(\Sigma_i))_i$ of languages describes the subshift that is the closure of the union of all subshifts: $\overline{\bigcup_{i\in\mathbb{N}} \Sigma_i}$.

We define some classes of subshifts. A *sofic subshift* is a subshift whose language of forbidden words is rational, i.e. given by a finite automaton. A subshift Σ is *transitive* if for all $u, v \in \mathcal{L}(\Sigma)$ there exists a word w such that $uwv \in \mathcal{L}(\Sigma)$. Let $s : Q \to Q^*$ be a primitive substitution (there exists $k \in \mathbb{N}$ such that for all $a, b \in Q$ a appears in $s^k(b)$), the *substitutive subshift* associated to s is the subshift Σ_s such that

$$\mathcal{L}(\Sigma_s) = \{u \in Q^* : \exists a \in Q \text{ and } n \in \mathbb{N} \text{ such that } u \text{ appears in } s^n(a)\}.$$

2.3. Cellular automata

Definition 2.2 (Cellular automaton)**.** A *cellular automaton (CA)* $\mathcal{A}$ is a triple $(Q_{\mathcal{A}}, r_{\mathcal{A}}, \delta_{\mathcal{A}})$ where $Q_{\mathcal{A}}$ is a finite set of states called the *alphabet*, $r_{\mathcal{A}}$ is the *radius* of the automaton, and $\delta_{\mathcal{A}} : Q_{\mathcal{A}}^{2r_{\mathcal{A}}+1} \mapsto Q_{\mathcal{A}}$ is the *local rule.*

The configurations of a cellular automaton are the configurations over $Q_{\mathcal{A}}$. A global behavior is induced and we'll note $\mathcal{A}(c)$ the image of a configuration c given by: $\forall z \in \mathbb{Z}, \mathcal{A}(c)_z = \delta_{\mathcal{A}}(c_{z-r_{\mathcal{A}}}, \dots, c_z, \dots, c_{z+r_{\mathcal{A}}})$. Studying the dynamic of $\mathcal{A}$ is studying the iterations of a configuration by the map $\mathcal{A} : Q_{\mathcal{A}}^{\mathbb{Z}} \to Q_{\mathcal{A}}^{\mathbb{Z}}$. When there is no ambiguity, we'll note Q, r and δ instead of $Q_{\mathcal{A}}$, $r_{\mathcal{A}}$ and $\delta_{\mathcal{A}}$.

2.4. μ-limit sets

Definition 2.3 (Uniform Bernoulli measure)**.** For an alphabet Q, the *uniform Bernoulli measure* μ on configurations over Q is defined by: $\forall u \in Q^*, i \in \mathbb{Z}, \mu([u]_i) = \frac{1}{|Q|^{|u|}}$.

For a CA $\mathcal{A} = (Q, r, \delta)$ and $u \in Q^*$, we denote for all $n \in \mathbb{N}$, $\mathcal{A}^n\mu([u]) = \mu\left(\mathcal{A}^{-n}([u])\right)$.

Definition 2.4 (Persistent set)**.** For a CA $\mathcal{A}$, and the uniform Bernoulli measure μ, we define the *persistent set* $L_\mu(\mathcal{A})$ with: $\forall u \in Q^*$:

$$u \notin L_\mu(\mathcal{A}) \iff \lim_{n\to\infty} \mathcal{A}^n\mu([u]_0) = 0.$$

Then the *μ-limit set* of $\mathcal{A}$ is $\Lambda_\mu(\mathcal{A}) = \left\{c \in Q^{\mathbb{Z}} : L(c) \subseteq L_\mu(\mathcal{A})\right\}$.

Remark 2.5. As this definition gives a set of forbidden finite words, we clearly see that μ-limit sets are subshifts.

Definition 2.6 (Set of predecessors)**.** We define the set of predecessors at time n of a finite word u for a CA $\mathcal{A} = (Q, r, \delta)$ as $P^n_{\mathcal{A}}(u) = \left\{v \in Q^{|u|+2rn} : \mathcal{A}^n([v]_{-rn}) \subseteq [u]_0\right\}$.

Remark 2.7. As we consider the uniform Bernoulli measure μ, $\frac{|P^n_{\mathcal{A}}(u)|}{|Q|^{|u|+2rn}} \to 0 \Leftrightarrow u \notin L_\mu(\mathcal{A})$.

Remark 2.8. The set of normal configurations has measure 1 in $Q^{\mathbb{Z}}$. Which means that a configuration that is randomly generated according to measure μ is a normal configuration.

Lemma 2.9. *Given a CA $\mathcal{A}$ and a finite word u, with μ the uniform Bernoulli measure, for any normal configuration c:*
$u \in \Lambda_\mu(\mathcal{A}) \Leftrightarrow d_{\mathcal{A}^n(c)}(u) \nrightarrow 0$ when $n \to +\infty$.

Proof. Let $n \in \mathbb{N}$, r be the radius of $\mathcal{A}$ and μ the uniform measure. We prove here that: $d_{\mathcal{A}^n(c)}(u) = \mathcal{A}^n\mu(u) = \frac{|P^n_{\mathcal{A}}(u)|}{|Q|^{|u|+2rn}}$.
The second part of the equality is obtained by definition of $\mathcal{A}^n\mu(u)$. We focus on the first part. Since any occurence of u in $\mathcal{A}^n(c)$ corresponds to an occurence of a predecessor of u in c :

$$d_{\mathcal{A}^n(c)}(u) = \limsup_{k\to+\infty} \frac{|\mathcal{A}^n(c)_{[-k,k]}|_u}{2k+1-|u|} = \limsup_{k\to+\infty} \sum_{v\in P^n_{\mathcal{A}}(u)} \frac{|c_{[-k-rn,k+rn]}|_v}{2k+2rn+1-(|u|+2rn)}.$$

And as c is normal, for any $v \in P^n_{\mathcal{A}}(u)$: $|c_{[-k-rn,k+rn]}|_v \sim_{k\to+\infty} \frac{2k+1}{|Q|^{|u|+2rn}}$.
Then:

$$d_{\mathcal{A}^n(c)}(u) = \sum_{v\in P^n_{\mathcal{A}}(u)} \limsup_{k\to+\infty} \left(\frac{1}{2k+1-|u|}\frac{2k+1}{|Q|^{|u|+2rn}}\right) = \sum_{v\in P^n_{\mathcal{A}}(u)} \frac{1}{|Q|^{|u|+2rn}} = \frac{|P^n_{\mathcal{A}}(u)|}{|Q|^{|u|+2rn}}.$$

■

Proposition 2.10. *Let $u \in L_\mu(\mathcal{A})$, there exists a word w such that $uwu \in L_\mu(\mathcal{A})$.*

Proof. Let $u \in L_\mu(\mathcal{A})$, there exists $\alpha > 0$ and an increasing sequence $(n_i)_{i\in\mathbb{N}}$ such that $\mathcal{A}^{n_i}\mu([u]) > \alpha$. Thus, for a normal configuration c, one has $d_{\mathcal{A}^{n_i}(c)}(u) > \alpha$ for all $i \in \mathbb{N}$. Let $l \in \mathbb{N}$ and $\epsilon > 0$ such that $\frac{2|u|}{2|u|+l} < \alpha - \epsilon$, we define

$$\begin{aligned} W_1 &= \{w \in Q^*_A : u \text{ is not a subword of } w \text{ and } |w| \leq l\} \text{ and} \\ W_2 &= \{w \in Q^*_A : u \text{ is not a subword of } w \text{ and } |w| > l\}. \end{aligned}$$

Consider $uW_ku = \{uwu : w \in W_i\}$ for $k \in \{1,2\}$, one has

$$d_{\mathcal{A}^{n_i}(c)}(uW_2u) = \limsup_{n\to\infty} \frac{|\mathcal{A}^{n_i}(c)_{[-n,n]}|_{uW_2u}}{2n+1} \leq \frac{2|u|}{2|u|+l}$$

since a word of uW_2u can appear at most $2|u|$ times for each pattern of length $2|u|+l$ of $\mathcal{A}^{n_i}(c)$. Moreover $d_{\mathcal{A}^{n_i}(c)}(uW_1u)+d_{\mathcal{A}^{n_i}(c)}(uW_2u) \geq d_{\mathcal{A}^{n_i}(c)}(u)$ so $d_{\mathcal{A}^{n_i}(c)}(uW_1u) \geq \epsilon$.

Since W_1 is finite, there exists a word $w \in W_1$ such that $d_{\mathcal{A}^{n_i}(c)}(uwu) \geq \epsilon$ for an infinity of $i \in \mathbb{N}$. Thus $uwu \in L_\mu(\mathcal{A})$. ■

Example 2.11. We consider here the "max" automaton $\mathcal{A}_M$. The alphabet contains only two states 0 and 1. The radius is 1. When the rule applies to three 0 (no 1), it produces a 0. In any other case, it produces a 1.

The probability to have a 0 at time t is the probability to have 0^{2t+1} on the initial configuration. Which tends to 0 when $t \to \infty$ for the uniform Bernoulli measure. So, 0 does not appear in the μ-limit set. And finally $\Lambda_\mu(\mathcal{A}_M) = \{^\infty 1^\infty\}$.

And this example gives a difference between subshifts that can be realised as limit set ($\Lambda(\mathcal{A}) = \bigcap_{i\in\mathbb{N}} \mathcal{A}^i(Q^{\mathbb{Z}})$) and subshifts that can be realised as μ-limit set.

Effectively, $\Lambda(\mathcal{A}_M) = ({}^\infty 10^*1^\infty) \bigcup ({}^\infty 0^\infty) \bigcup ({}^\infty 01^\infty) \bigcup ({}^\infty 10^\infty)$, but if we apply proposition 2.10 with the word 01, we conclude that $\Lambda(\mathcal{A}_M)$ cannot be a μ limit set.

3. Counters

In this section and the following one, we describe an automaton $\mathcal{A}_S$, which, on normal configurations, produces finite segments of size growing with time. In these segments, we will make computations described in section 5.

Before starting the computation, the automaton $\mathcal{A}_S$ has a transitory regime which erases the random configuration and generate segments between # where the computation is done. To do that, we have a special state $*$, that can only appear in the initial configuration, and which generates two counters. Between two counters, the states are initialized and when two counters intersect, they compare their respective age. If they do not have the same age, the younger deletes the older one; if they have the same age, they disappear and we put the state # in order to start the computation. The notion of counters was introduced in [DPST10] to produce equicontinuous points according to arbitrary curves.

We recall some ideas which allow to construct such automaton:

- no transition rule produces the state $*$;
- $*$ produces two couples of signals, one toward the left and another one toward the right;
- a couple of signal (called *counter*) is formed by an *inner* signal and an *outer* signal, which is faster. Their collisions are handled in the following way:
  - nothing other than an outer signal can go through another outer signal;
  - when two outer signals collide they move through each other and comparison signals are generated;
  - on each side, a signal moves at maximal speed towards the inner border of the counter, bounces on it and goes back to the point of collision;
  - the first signal to come back is the one from the youngest counter and it then moves back to the outer side of the oldest counter and deletes it;
  - the comparison signal from the older counter that arrives afterwards is deleted and will not delete the younger counter's outer border;
- between a left counter and a right counter, the configuration is initialized;
- if two counters that have the same age meet, they disappear and produce the state $\#_S$ which start the computation described in section 5
- the state $\#_S$ becomes # which delimitates segments, this state can disappear if two adjacent segments decide to merge as described in section 4, or if a counter (necessarily younger) encounters it.

The initialization of a configuration is illustrated in figure 1. The gray areas of computation begin on the left of a # produced by the meeting of two counters generated by a $*$.

Lemma 3.1. *There exists a constant K_c such that if two # are distant of k, they appeared before time $k \times K_c$.*

Proof. Consider two states # in the space time diagram separated by k cells. If # is not in the initial configuration, the only way to appear is to result from the collision of two counters coming from the left and from the right. Thus, in the initial

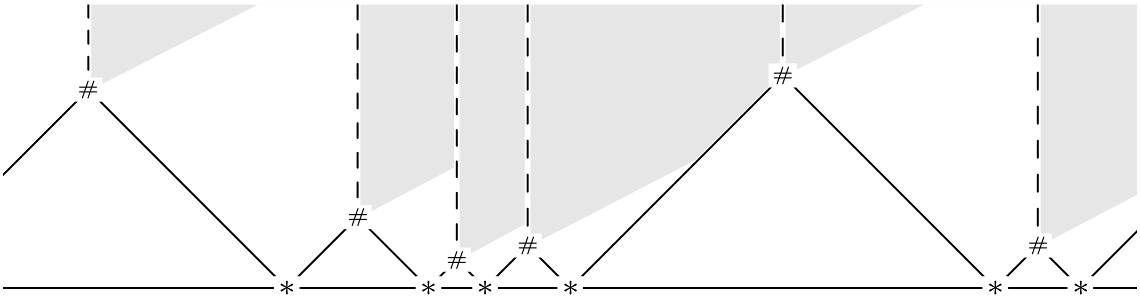


Figure 1: When two counters launched by a $*$ meet, a # is produced and a computation is launched on the right. The computation area extends until it meets the inner signal of a counter or another #.

configuration, it is necessary to have the state $*$ between the two # to create the two #. This operation take at most $k \times K_c$ where K_c is the speed of an inner signal. ■

4. Merging segments

We saw in Section 3, how a special state $*$ on the initial configuration gave birth to counters protecting everything inside them until they meet some other counter born the same way. In this section, we will describe the evolution of the automaton $\mathcal{A}_S$ after this time of initialization. When two counters of the same age meet, they disappear and a # is produced.

Definition 4.1 (Segment)**.** A *segment* u is a subword of a configuration delimited by two # and containing no # inside. So, $u \in \#(Q \setminus \{\#\})^* \#$. The *size* of a segment is the number of cells between both #.

There will be computations made inside segments, but we will describe it later. Thus, in a segment, there is a layer left for computations that remain inside the both #, and a "merging layer" that will contain signals necessary to the behavior with other segments. Every signal presented in this section will travel on this merging layer. The idea is the following: at some times, two neighbor segments will decide to merge together to form one single segment whose size will be the sum of both sizes plus one. And we will assure that each segment will eventually merge, so that no segment of finite size can still be in the μ-limit set of $\mathcal{A}_S$.

When a # is produced in automaton $\mathcal{A}_S$, it sends two signals, on its right and on its left to detect the first # on each side. If the signal catches the inside of a counter still in activity before reaching a #, it waits until the counter produces a #. Then both # have recognised each other and the segment between them is "conscious". It launches a computation inside it, and waits until it is achieved. We will assure later that this computation ends. When this is done, it will alternatively send signals to its left and to its right in order to propose successively to each neighbor to merge.

For this purpose, it computes and stores the length n of the segment as a binary representation. Then the segment puts a L mark on its left #, and waits for n^2 timesteps. If, during this time, the left side neighbor has not put a R mark on the common #, our segment erases the L mark, a signal is sent on the other side, and

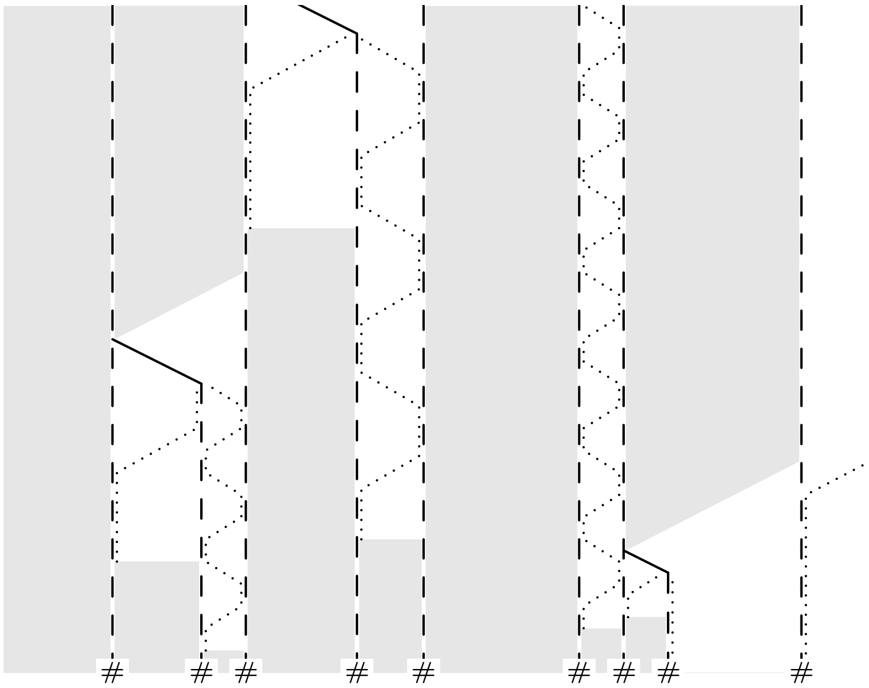


Figure 2: A # stays until two segments merge. Computation happens in gray areas, and at its end a signal (...) is sent and stays on the left of the segment, then goes to the right, stays and comes back. This cycle continues until a neighbor's signal is on the same # at the same time. Then the # is deleted and another computation is started on the left.

it puts a R mark on its right #. It waits once again n^2 timesteps before erasing the R, sending a signal to its left, and starting over. The whole cycle takes $2(n^2 + n)$ timesteps as we consider a signal at speed 1 crossing a segment of size n. We request the signal to stay n^2 timesteps because as $(n+1)^2 > n^2 + 2n$, if two segments do not have the same size, their signals eventually meet during a cycle of the smallest one. So, the only case in which two neighbor segments that try to merge do not merge, is when they have same size and are correctly synchronized. Computing and storing n, and waiting n^2 can be done with a space $\log(n)$.

This process ends when at the same time, both a L and a R mark are written on a #. When this happens, the two segments agree to merge together and they do it: the # between them is erased, and the whole activity begins again, starting with the computation inside the new segment.

The general behavior of the segments among themselves is illustrated in figure 2. We prove the following claims for automaton $\mathcal{A}_S$.

Claim 4.2. *For any two words $u, v \in Q^*$, with $|u| \neq |v|$, if the word $w = \#u\#v\#$ appears at time t in a space-time diagram of $\mathcal{A}_S$, one of the 3 # of w has disappeared at time $t + |Q|^{|w|} + 2(|w|^2 + |w|)$.*

Proof. If the word w exists at time t on a space time diagram, at time $t + |Q|^{|u|}$ (respectively $t + |Q|^{|v|}$) at most, the computation is achieved in u (resp. v). We suppose here that no # in w has disappeared, which means, u and v do not merge with any other segment outside w. So at time $t+|Q|^{|w|}$ both segments try to merge with another one. Assume $|v|>|u|$ for example, the other case is totally symmetric. Then, as $|v|^2>|u|^2+2|u|$, before the end of the cycle of v, they have put their mark simultaneously on their common # for one timestep at least. And consequently, they have merged and one # has disappeared at time $t+|Q|^{|w|}+2(|w|^2+|w|)$. ■

Claim 4.3. *If two segments of size less than $k\in\mathbb{N}$ merge together, they do it at most $|Q|^{2k}+2((2k)^2+2k)$ timesteps after being formed.*

Proof. If they don't have the same size, lemma 4.2 let us conclude. If they have the same size, their computations are achieved after $|Q|^{2k}$. And as their merging cycle takes the same time for both, if they do not merge during the first cycle, they will never merge. So if they merge, they do it before $|Q|^{2k}+2((2k)^2+2k)$. ■

Claim 4.4. *For any two words $u,v\in Q^*$, with $|u|\neq|v|$, the word $w=\#u\#v\#$ does not appear in $\Lambda_\mu(\mathcal{A}_S)$.*

Proof. We use the constant K_c from lemma 3.1. Denote
$T=|w|\times K_c+|w|\left(|Q|^{|w|}+2(|w|^2+|w|)\right)$. We prove that for $t>T$, $P^t_{\mathcal{A}_S}(w)=\emptyset$. If the two # encircling w never disappear, the dynamic inside w is not affected by the exterior. Through time, some other # possibly appeared and disappeared between them. But after time at most $|w|\times K_c$, they have all appeared. Since then, they will only disappear. There are less than $|w|-1$ excedentary # that have to disappear. Considering lemma 4.3, one disappears at least every $|Q|^{|w|}+2(|w|^2+|w|)$ timesteps. After that, the two segments of w are formed, and with lemma 4.2, one of the # of w disappear before $|Q|^{|w|}+2(|w|^2+|w|)$ new timesteps. Finally, at time T, one of the # of w has disappeared and $P^t_{\mathcal{A}_S}(w)=\emptyset$. ■

Proposition 4.5. *There is no # in the μ-limit set of $\mathcal{A}_S$.*

Proof. Assume that $\#\in L_\mu(\mathcal{A})$, by Proposition 2.10, there exits $u\in Q^*$ such that $\#u\#\in L_\mu(\mathcal{A})$, we can assume that u does not contain #. Let $k=|u|$, by Lemma 3.1, the # encircling u appeared before time $k\times K_c$. Denote $W=\{\#v\#: v\in(Q\backslash\{\#\})^k\}$ and $X_n=\{x\in Q^{\mathbb{Z}}:\mathcal{A}^k(x)_{[0,k+1]}\in W$ for all $k\in[k\times K_c,n]\}$. Since $\#u\#\in L_\mu(\mathcal{A})$, there exists $\alpha>0$ such that $\mu(X_n)>\alpha$ for an infinity of $n\in\mathbb{N}$. Moreover, as $X_{n+1}\subset X_n$, we can conclude that $\mu(X_\infty)>\alpha$ where $X_\infty=\cap_{n\in\infty}X_N$.

As μ is Bernoulli, we have $\mu(Y)>0$ where $Y=[*(Q\smallsetminus *)^{2k}*(Q\smallsetminus *)^{2k}*]_0$; moreover there exist $k_1\geq 0$ and $k_2\geq k_1+4k+1$ such that $\mu(Z)>0$ where $Z=X_\infty\cap\sigma^{-k_1}(Y)\cap\sigma^{-k_2}(X_\infty)$. For all $n\geq k\times K_c$ one has $F^n(Z)_{[0,k+1]}\subset W$, $F^n(Z)_{[k_2,k_2+k+1]}\subset W$ and $F^n(Z)_{[k_1+k,k_1+3k]}\subset F^n(Y)_{[k_1+k,k_1+3k]}$ does not contain #.

We deduce that there exists a word $w\in Q$ of length k_2-k-1 such that $w_{[k_1+k,k_1+3k]}$ does not contain # and $\#u\#w\#u\#\in L_\mu(\mathcal{A})$. However, in $\#u\#w\#u\#$ we can find two segments $\#u_1\#u_2\#$ which have different length. By Claim 4.4 we obtain a contradiction. Thus, there is no # in the μ-limit set of $\mathcal{A}_S$. ■

Finally, we prove a lemma that will be useful later.

Claim 4.6. *The density of cells outside segments generated by counters born in the initial configuration tends to* 0.

Proof. The proof is clear since such a cell needs predecessors without states $*$ on each side in the initial configuration. ■

Lemma 4.7. *Let $u \in Q^*$. If $\forall k, \forall l \geq k$, for any segment $v \in Q^l$, $d_v(u) \leq \alpha_k$ with $\alpha_k \to 0$ when $k \to \infty$, then $u \notin L_\mu(\mathcal{A}_S)$.*
Conversely, if $\forall k, \forall l \geq k$, for any segment $v \in Q^l$, $d_v(u) \geq \alpha_k$ with $\alpha_k \nrightarrow 0$ when $k \to \infty$, then $u \in L_\mu(\mathcal{A}_S)$.

Proof. Let's consider a normal configuration c. For any $k \in \mathbb{N}$, we denote

$$d_k^t = \sum_{v \in \#(Q_\mathcal{A})^l \#,\ l \leq k} l \times d_{\mathcal{A}_S^t(c)}(v)$$

the density of cells in segments of size less than k in the image at time t of c. Due to proposition 4.5, $d_k^t \to 0$ when $t \to \infty$. And due to claim 4.6, the density a^t of cells outside wellformed segments tends to 0 when $t \to \infty$.

Suppose $\forall k, \forall l \geq k, \forall v \in Q^l$ segment, $d_v(u) \leq \alpha_k$ and $\alpha_k \to 0$. Any occurence of u is either in a segment of size less than k, either in a segment of size greater than k, or out of segments. Finally, at a given time t, $d_{\mathcal{A}_S^t(c)}(u) \leq d_k^t + \alpha_k + a^t$.

As this equation holds for any k, finally, when $t \to \infty$, $d_{\mathcal{A}_S^t(c)}(u)$ has a limit which is 0. This concludes the proof of the first part of the lemma with lemma 2.9.

In the other side, suppose $\forall k, \forall l \geq k, \forall v \in Q^l$ segment, $d_v(u) \geq \alpha_k$ and $\alpha_k \nrightarrow 0$. Therefore, $d_{\mathcal{A}_S^t(c)}(u) \geq (1 - d_k^t - a^t)\alpha_k$ which does not tend to 0 when $t \to \infty$ and $k \to \infty$. Thus, $u \in L_\mu(\mathcal{A}_S)$. ∎

5. Infinite Unions

In this section we will see how to create a CA whose μ-limit set is the closure of the infinite union of a recursively enumerable family of particular subshifts.

Definition 5.1 (Generable Subshift)**.** We say that a Turing machine M *generates* a subshift $\Sigma \subseteq Q^\mathbb{Z}$ if M computes a generic configuration of Σ in the following sense:

- the tape alphabet of M contains Q;
- on an empty tape, M writes the right half of a configuration $c \in \Sigma$ such that $\limsup_{n\to\infty} \frac{|c_{[0,n]}|_u}{n+1} > 0$ if and only if $u \in \mathcal{L}(\Sigma)$; c is called a *generic configuration*;
- after a symbol of Q has been written on the tape, it is never changed.

Theorem 5.2. *Given a recursively enumerable family $(\Sigma_i)_{i\in\mathbb{N}}$ of generable subshifts, that is to say that there exists a Turing machine that enumerates a set of machines $(M_i)_i \in \mathbb{N}$ such that M_i generates the subshift Σ_i, there exists a cellular automaton $\mathcal{A}$ whose μ-limit set is exactly the subshift $\overline{\bigcup_{i\in\mathbb{N}} \Sigma_i}$.*

Proof. Let us consider a recursively enumerable family $(\Sigma_i)_{i\in\mathbb{N}}$ of generable subshifts, let us denote by M the Turing machine that enumerates the machines $(M_i)_{i\in\mathbb{N}}$ such that M_i generates the subshift Σ_i.

We now describe the behavior of such a cellular automaton $\mathcal{A}$. $\mathcal{A}$ will work as the automaton $\mathcal{A}_S$ described in Section 4: starting from a normal configuration, it will generate "counter signals" that will produce finite segments on the configuration (separated by a # symbol). We now describe the computation performed by each finite segment during the evolution of the cellular automaton.

The first thing a segment does is compute its length n and store it as a binary number. By incrementing a binary counter moving across the segment, this is easily done in space $\log(n)$. Once this is done, the segment can simulate Turing machines on its first $\log(n)$ cells (it is important to limit the computational space so that the computation states become negligible and disappear from the μ-limit set).

On the initial $\log(n)$ cells of the segment the machine M is simulated to produce the descriptions of the first k machines $(M_i)_{i<k}$, with k as big as possible for M computing on a tape of length $log(n)$. And we also request that $k \leq \log(\log(n))$. k may be 0 for short segments, but we know that as the segments grow larger, k will grow too.

The space of size $\log(n)$ is further divided into k fragments of size $\log(n)/k$. On the i-th fragment, the corresponding machine M_i is simulated to produce the word w_i beeing the begining of the generic configuration corresponding to the subshift Σ_i. The word w_i might be much smaller than $\log(n)/k$ depending on the space needed by the machine M_i to compute, but again we know that as segments grow larger, larger words will be computed.

After the k different w_i have been computed, the initial segment of length n is split into $\sqrt{n}$ fragments of length $\sqrt{n}$. Each of these fragments is filled with copies of one of the w_i in the following manner: one out of two is filled with w_1, one out of four (i.e. one out of two among the remaining fragments) is filled with w_2, one out of eight is filled with w_3 and so on. The remaining segments (if k is very small, we might run out of w_i before filling all the fragments) are filled with w_k. Fragments are separated by a symbol $\$_1 \notin Q$ and the copies of words w_i inside a given fragment are separated by a symbol $\$_2 \notin Q$.

Remark 5.3. The previous construction can be done using only $\log(n)$ cells of computation at each step (cells that are not active and that only contain a symbol from $Q \cup \{\$_1, \$_2\}$ are not counted). To fill the fragments of size $\sqrt{n}$ we only need to compute the binary expression of $\sqrt{n}$ and then advance through the segment while filling the fragment with the appropriate w_i while decreasing a counter to measure $\sqrt{n}$ cells. The important data (the words w_i and different counters) are moved through the segment so that they are always present near the location to be filled. Thus the head of the Turing machine M carries only $\log(n)$ cells used to store the w_i and to its computation. No mark of the computation remains in the other cells, even those already visited and rewritten.

When all the fragments of the segment have been filled with the w_i, the segment can erase all the remaining computation data and start the process of merging with its neighbors as described in Section 4.

When two segments merge, the whole computation is restarted but this time with a larger space. The segments are not erased immediately after a merge, but rather the new data overwrites the previous as the $\sqrt{n}$ fragments are filled.

We will prove that $L_\mu(\mathcal{A}) = \bigcup_{i\in\mathbb{N}} \mathcal{L}(\Sigma_i)$.

Claim 5.4. *The states used for computation, signals inside segments, writing fragments, $\$_1$ or $\$_2$ do not appear in $\Lambda_\mu(\mathcal{A})$.*

Proof. Here we use the lemma 4.7 for each of these states.

We use the $\log(k)$ initial cells of a segment of size k to do the computation, so the density of these cells is $\log(k)/k$, and the property is proved. The head of the Turing machine M carries at most $\log(k)$ cells for its computation or writing, thus the same argument works. The signals for the merging process are in a finite number in a segment, therefore their density in a segment tends to 0 too. The density of $\$_1$ is $\sqrt{k}/k$, and the lemma applies once again.

For the density of $\$_2$, let $\lambda > 0$, $\exists k_0 > 0$ such that the word w_i produced in a segment of size $k > k_0$ is such that $|w_i| > \lambda$ for any $i \leq \lambda$. So, for $k > k_0$, the density of $\$_2$ in a segment of size k is less than $1/\lambda$ in fragments of $S_i, i \leq \lambda$ and less than 1 in the other fragments that have themselves a density lower than $1/2^\lambda$. And thus, the density of $\$_2$ is lower than $\frac{1}{\lambda} + \frac{1}{2^\lambda}$ in segments of size $k > k_0$. Finally the density of $\$_2$ tends to 0 when $k \to \infty$. And the claim is proved. ■

Claim 5.5. *For any subshift Σ_i, $i \in \mathbb{N}$, any word $u \in \mathcal{L}(\Sigma_i)$ and any family of segments $(v_k)_k$ of size $|v_k| = k$, $d_{v_k}(u)$ does not tend to 0 when $k \to \infty$.*

Proof. As $u \in \mathcal{L}(\Sigma_i)$, its density $\alpha(u)$ in the generic configuration computed by M_i is positive. So, there exists $l_i \in \mathbb{N}$ such that any subword of this configuration contains u with density at least $\alpha(u)/2$. Let k_0 such that in any segment of size $k > k_0$, the word w_i computed has length $|w_i| > l_i$.

For any segment v_k of size $k > k_0$, there are $\log(k)$ cells occupied for computation, less than $\sqrt(k)$ cells containing a $\$_1$ and $\frac{1}{2^{i+1}}$ among the remaining cells attributed to the copies of w_i. Among these copies, a proportion $\frac{l_i - 1}{l_i}$ of the cells contain $\$_2$. $\log(k)$ additional cells can be dedicated to the head of the Turing machine M writing in the segment and a finite number K of cells can contain signals for the merging process. Finally,

$$d_{v_k}(u) \geq \left(\left(\frac{k - \log(k) - \sqrt(k)}{2^{i+1}} \right) \frac{l_i - 1}{l_i} - \log(k) - K \right) \frac{1}{k}.$$

Which does not tend to 0 when $k \to \infty$. ■

Claim 5.6. *For any subshift Σ_i, $i \in \mathbb{N}$ and any word $u \in \mathcal{L}(\Sigma_i)$, $u \in L_\mu(\mathcal{A})$.*

Proof. We clearly get the result by combining claim 5.5 and lemma 4.7. ■

Finally, the theorem is proven:

- the proposition 4.5 and the claim 5.4 assure that every state used for computation does not appear in $\Lambda_\mu(\mathcal{A})$, which means $L_\mu(\mathcal{A}) \subseteq \bigcup_{i\in\mathbb{N}} \mathcal{L}(\Sigma_i)$,
- the claim 5.6 assures that $\bigcup_{i\in\mathbb{N}} \mathcal{L}(\Sigma_i) \subseteq L_\mu(\mathcal{A})$. ■

The next proposition gives some examples of generable subshifts.

Proposition 5.7. *The following subshifts are generable:*

- *transitive sofic subshifts,*
- *substitutive subshift associated to a primitive substitution.*

Proof. As a transitive sofic subshift Σ is given by the strongly connected automaton recognizing its language. For example, we can write successively every cycle of size k for k from 1 to ∞. In this case we obtain a configuration where the density of all the words of the language of Σ is positive.

For a primitive substitution s, it is easy to generate the fix point configuration denoted $c_{[0;\infty]}$ whose all prefixes are given by $s^k(a)$ for all $k \in \mathbb{N}$ where $a \in Q$. It is well know that all words of the substitutive subshift associated appears with a positive density in $c_{[0;\infty]}$ [Fog05]. ■

6. Conclusion and perspectives

In this paper, we prove that a large class of subshifts can be realized as μ-limit sets of cellular automata. In particular, it is possible to obtain all transitive sofic subshifts, this is a profound difference with the topological case since the even shift cannot be realized as the limit set of one cellular automaton. This construction allows to control the iterations of a random configuration in view to obtain an auto-organized behavior. The construction can be adapted at least in two ways:

- to obtain the same result for a large class of measure (σ-ergodic measure of full support) modulo some technical changes
- to obtain a subshift without any word of low complexity (as suggested by V. Poupet).

Of course the main open question is in the reciprocal of the theorem, that is to say to characterize subshifts that can possibly be realized as μ-limit sets.

Acknowledgments

We are deeply grateful to Victor Poupet and Guillaume Theyssier for their ideas, and constant support to the writing of this article.

References

[BPT06] Laurent Boyer, Victor Poupet, and Guillaume Theyssier. On the Complexity of Limit Sets of Cellular Automata Associated with Probability Measures. *MFCS 2006*, LNCS 4162:190–201, 2006.

[DPST10] Martin Delacourt, Victor Poupet, Mathieu Sablik, and Guillaume Theyssier. Directional Dynamics along Arbitrary Curves in Cellular Automata. *Theoretical Computer Science*, A paraître, 2010.

[Fog05] N. Pytheas Fogg. Substitutions in Dynamics, Arithmetics and Combinatorics. V. Berthé, S. Ferenczi, C. Mauduit, A. Siegel (Eds), 2005.

[FK07] Enrico Formenti and Petr Kurka. A Search Algorithm for the Maximal Attractor of a Cellular Automaton. *STACS, 2007*, pages 356–366, 2007.

[Hur87] Lyman P. Hurd. Formal Language Characterizations of Cellular Automata Limit Sets. *Complex Systems*, 1:69–80, 1987.

[Hur90a] Mike Hurley. Attractors in cellular automata. *Ergodic Theory Dynam. Systems*, 10(1):131–140, 1990.

[Hur90b] Mike Hurley. Ergodic aspects of cellular automata. *Ergodic Theory Dynam. Systems*, 10(4):671–685, 1990.

[Kar92] Jarkko Kari. The Nilpotency Problem of One-Dimensional Cellular Automata. *SIAM J. Comput.*, 21(3):571–586, 1992.

[Kar94] Jarkko Kari. Rice's Theorem for the Limit Sets of Cellular Automata. *Theor. Comput. Sci.*, 127(2):229–254, 1994.

[KM00] Petr Kurka and Alejandro Maass. Limit sets of cellular automata associated to probability measures. *Journal of Statistical Physics*, 100(5):1031–1047, 2000.

[Kur03] Petr Kurka. *Topological and symbolic dynamics.* Société Mathématique de France, Paris, 2003.

[Maa95] Alejandro Maass. On the sofic limit sets of cellular automata. *Ergodic Theory Dynam. Systems*, 15:663–684, 1995.

A CATEGORICAL OUTLOOK ON CELLULAR AUTOMATA

SILVIO CAPOBIANCO AND TARMO UUSTALU

Institute of Cybernetics at Tallinn University of Technology, Akadeemia tee 21, 12618 Tallinn, Estonia
E-mail address: {silvio,tarmo}@cs.ioc.ee

Abstract. In programming language semantics, it has proved to be fruitful to analyze context-dependent notions of computation, e.g., dataflow computation and attribute grammars, using comonads. We explore the viability and value of similar modeling of cellular automata. We identify local behaviors of cellular automata with coKleisli maps of the exponent comonad on the category of uniform spaces and uniformly continuous functions and exploit this equivalence to conclude some standard results about cellular automata as instances of basic category-theoretic generalities. In particular, we recover Ceccherini-Silberstein and Coornaert's version of the Curtis-Hedlund theorem.

1. Introduction

Since the seminal work of Moggi [13], it has become standard in programming language semantics to analyze functions producing effects such as exceptions, input, output, interactive input-output, nondeterminism, probabilistic choice, state, continuations using monads. Specifically, effectful functions are identified with Kleisli maps of a suitable monad on the category of pure functions. Wadler [18] put this view to further use in programming methodology by extracting from it a very useful programming idiom for purely functional languages like Haskell.

The dual view of context-dependent functions as coKleisli maps of a comonad is equally useful, but less well known. Brookes and Geva [2] explained the "intensional" aspect of denotational semantics in terms of the ω-chain comonad on the category of ω-cpos. More recently, Uustalu and Vene [15, 16, 17] employed comonads to analyze dataflow computation and attribute grammars and Hasuo et al. [9] treated tree transducers.

Characteristic of many context-dependent notions of computation is shape-preserving transformation of some datastructure based on a value update rule which is local in nature and applied uniformly to every node. This is the case with dataflow computation where such a transformation is applied to a list or to a stream with a distinguished position and with attribute grammars where computation happens on a tree or a tree with a distinguished position (a "zipper"). Cellular automata are

This research was supported by the European Regional Development Fund (ERDF) through the Estonian Center of Excellence in Computer Science (EXCS) and by the Estonian Science Foundation under grant no. 6940.

similar, the datastructure being the integer line or plane or, more generally, a group. It should therefore be worthwhile to test the slogan that context-dependent computation is comonadic also on cellular automata. To a degree, this has already been done, as Piponi [14] programmed cellular automata in Haskell using the comonadic interface. However, he did not use his modeling of cellular automata to prove properties about them and also dropped the classical requirement that cellular automata rely on a finite neighborhood only.

In this paper, we study the comonadic aspect of cellular automata deeper. We identify cellular automata (more exactly their local behaviors) with coKleisli maps of the exponent monad on **Unif**, the category of uniform spaces and uniformly continuous functions, and explore whether this view can be useful. We see that it is: we can conclude some standard results about cellular automata as instances of category-theoretic generalities. In particular, we recover the Curtis-Hedlund theorem [10]—a characterization of global behaviors of cellular automata—in the version of Ceccherini-Silberstein and Coornaert [4] (this applies to general discrete alphabets rather than finite alphabets only). This theorem turns out to be an instance of the basic category-theoretic fact that the coKleisli category of a comonad is isomorphic to the full subcategory of its co-Eilenberg-Moore category given by the cofree coalgebras. We also show that the comonadic view allows one to see 2-dimensional cellular automata as 1-dimensional and treat point-dependent cellular automata.

The paper is organized as follows. Section 2 is a quick introduction to comonads while Section 3 reviews some preliminaries about topological and uniform spaces. In Section 4, we show that cellular automata local behaviors are the same as coKleisli maps of a certain comonad. In Section 5, we recover the Curtis-Hedlund theorem (in the version of Ceccherini-Silberstein and Coornaert). In Section 6, we reprove the reversibility principle. In Sections 7, 8, we discuss some further applications of the comonadic view: 2-dimensional cellular automata as 1-dimensional and point-dependent cellular automata.

The paper assumes knowledge of basic category theory (categories, functors, natural transformations, Cartesian closed categories), but is self-contained in regards to comonads. For background material on category theory and (co)monads in particular, we refer the reader to Barr and Wells [1, Ch. 1, 3]. We also assume the basics of CA as presented by Ceccherini-Silberstein and Coornaert [5, Ch. 1].

2. Comonads

Given two categories $\mathcal{C}$, $\mathcal{D}$ and a functor $L : \mathcal{D} \to \mathcal{C}$, a *right adjoint* to L is given by a functor $R : \mathcal{C} \to \mathcal{D}$ and two natural transformations $\varepsilon : LR \to \mathrm{Id}_{\mathcal{C}}$ (the *counit*) and $\eta : \mathrm{Id}_{\mathcal{D}} \to RL$ (the *unit*) such that the diagrams

$$\begin{tikzcd}
L \arrow[r, "L\eta"] \arrow[dr, equal] & LRL \arrow[d, "\varepsilon L"] & R \arrow[d, "\eta R"'] \arrow[dr, equal] & \\
& L & RLR \arrow[r, "R\varepsilon"'] & R
\end{tikzcd}$$

commute. Equivalently, a right adjoint may be given by an object mapping $R : |\mathcal{C}| \to |\mathcal{D}|$, for any object $A \in |\mathcal{C}|$, a map $\varepsilon_A : LRA \to A$, and, for any objects $A \in |\mathcal{D}|$, $B \in |\mathcal{C}|$ and map $k : LA \to B$, a map $k^{\ddagger} : A \to RB$ (the *right transpose*) such that

- for any objects $A \in |\mathcal{D}|$, $B \in |\mathcal{C}|$, and map $k : LA \to B$, $\varepsilon_B \circ Lk^\ddagger = k$,
- for any object $A \in |\mathcal{C}|$, $(\varepsilon_A)^\ddagger = \mathrm{id}_{RA}$,
- for any objects $A, B \in |\mathcal{D}|$, $C \in |\mathcal{C}|$ and maps $f : A \to B$, $k : LB \to C$, $(k \circ Lf)^\ddagger = k^\ddagger \circ f$.

The morphism mapping part of R and unit η define the right transpose of $k : LA \to B$ by $k^\ddagger =_{\mathrm{df}} Rk \circ \eta_A$. The right transpose $(-)^\ddagger$ determines the morphism mapping part of R and unit η by $Rf =_{\mathrm{df}} (f \circ \varepsilon_A)^\ddagger$, for $f : A \to B$, and $\eta_A =_{\mathrm{df}} (\mathrm{id}_{LA})^\ddagger$.

A *comonad* on a category $\mathcal{C}$ is given by a functor $D : \mathcal{C} \to \mathcal{C}$ and natural transformations $\varepsilon : D \to \mathrm{Id}_{\mathcal{C}}$ (the *counit*) and $\delta : D \to DD$ (the *comultiplication*) making the diagrams

$$\begin{tikzcd}
D \arrow[r, "\delta"] \arrow[d, "\delta"'] \arrow[rd, equal] & DD \arrow[d, "\varepsilon D"] \\
DD \arrow[r, "D\varepsilon"] & D
\end{tikzcd}
\qquad
\begin{tikzcd}
D \arrow[r, "\delta"] \arrow[d, "\delta"'] & DD \arrow[d, "\delta D"] \\
DD \arrow[r, "D\delta"] & DDD
\end{tikzcd}$$

commute. Equivalently, a comonad can be given by an object mapping $D : |\mathcal{C}| \to |\mathcal{C}|$, for any object $A \in |\mathcal{C}|$, a map $\varepsilon_A : DA \to A$, and, for any objects $A, B \in |\mathcal{C}|$ and map $k : DA \to B$, a map $k^\dagger : DA \to DB$ (the *coKleisli extension*) such that

- for any objects $A, B \in |\mathcal{C}|$ and map $k : DA \to B$, $\varepsilon_B \circ k^\dagger = k$,
- for any object A, $(\varepsilon_A)^\dagger = \mathrm{id}_{DA}$,
- for any objects $A, B, C \in |\mathcal{C}|$ and maps $k : DA \to B$, $\ell : DB \to C$, $(\ell \circ k^\dagger)^\dagger = \ell^\dagger \circ k^\dagger$.

The morphism mapping part of D and comultiplication δ define the coKleisli extension $(-)^\dagger$ by $k^\dagger =_{\mathrm{df}} Dk \circ \delta_A$. Conversely, the $(-)^\dagger$ determines the morphism mapping part of D and comultiplication δ by $Df =_{\mathrm{df}} (f \circ \varepsilon_A)^\dagger$, $\delta_A =_{\mathrm{df}} (\mathrm{id}_{DA})^\dagger$.

A functor $L : \mathcal{C} \to \mathcal{D}$ with a right adjoint (R, ε, η), defines a comonad on $\mathcal{C}$ with counit ε by $D =_{\mathrm{df}} LR$, $\delta =_{\mathrm{df}} L\eta R$, alternatively by $DA =_{\mathrm{df}} L(RA)$, $k^\dagger =_{\mathrm{df}} Lk^\ddagger$.

In the converse direction, a comonad (D, ε, δ) on $\mathcal{C}$ induces a whole category of adjunctions $(\mathcal{D}, L, R, \eta)$ that have ε as the counit and satisfy $D = LR$, $\delta = L\eta R$, called *splittings* of the comonad. This category has initial and final objects, which are known as the coKleisli and coEilenberg-Moore splittings of the comonad.

The *coKleisli category* $\mathrm{coKl}(D)$ has as objects those of $\mathcal{C}$ and as maps from A to B those from DA to B of $\mathcal{C}$. The identity jd_A on object A is defined by $\mathrm{jd}_A =_{\mathrm{df}} \varepsilon_A$. The composition $\ell \bullet k$ of maps $k : DA \to B$ and $\ell : DB \to C$ is $\ell \bullet k =_{\mathrm{df}} \ell \circ Dk \circ \delta_A = \ell \circ k^\dagger$. The functor $L : \mathrm{coKl}(D) \to \mathcal{C}$ in the coKleisli splitting is defined by $LA =_{\mathrm{df}} DA$, $Lk =_{\mathrm{df}} k^\dagger$. The right adjoint, unit and right transpose are defined by $RA =_{\mathrm{df}} A$, $Rf =_{\mathrm{df}} f \circ \varepsilon_A$, $\eta_A =_{\mathrm{df}} \mathrm{id}_{DA}$, $k^\ddagger =_{\mathrm{df}} k$.

The *coEilenberg-Moore category* $\mathrm{coEM}(D)$ has as objects coalgebras of D and as maps coalgebra maps of D. A *coalgebra* of D is given by an object $A \in |\mathcal{C}|$ and map $u : A \to DA$ (the *coalgebra structure*) making the diagrams

$$\begin{tikzcd}
A \arrow[r, "u"] \arrow[rd, equal] & DA \arrow[d, "\varepsilon_A"] \\
 & A
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "u"] \arrow[d, "u"'] & DA \arrow[d, "\delta_A"] \\
DA \arrow[r, "Du"] & D(DA)
\end{tikzcd}$$

commute. A *coalgebra map* between (A, u) and (B, v) is a map $f : A \to B$ making the diagram

$$\begin{tikzcd}
A \arrow[r, "u"] \arrow[d, "f"'] & DA \arrow[d, "Df"] \\
B \arrow[r, "v"] & DB
\end{tikzcd}$$

commute. The identity and composition are inherited from $\mathcal{C}$. The functor L : $\mathrm{coEM}(D) \to \mathcal{C}$ in the splitting of D through $\mathrm{coEM}(D)$ is the coalgebra-structure forgetful functor: $L(A, u) =_{\mathrm{df}} A$, $Lf =_{\mathrm{df}} f$. The right adjoint R is defined by $RA =_{\mathrm{df}} (DA, \delta_A)$, $Rf =_{\mathrm{df}} Df$. The unit and right transpose are defined by $\eta_{(A,u)} =_{\mathrm{df}} u$ and $k^{\ddagger} =_{\mathrm{df}} Dk \circ u$ (for $k : L(A, u) \to B$).

The functor R being the right adjoint of the forgetful functor implies that, for any B, the coalgebra $RB = (DB, \delta_B)$ is the cofree coalgebra on B, i.e., for any coalgebra (A, u), object B and map $k : A \to B$, there is a unique map $f : A \to DB$, namely $k^{\ddagger}$, such that the diagrams

$$\begin{tikzcd}
& A \arrow[r, "u"] \arrow[dl, "k"'] \arrow[d, dotted, "f"] & DA \arrow[d, dotted, "Df"] \\
B & DB \arrow[l, "\varepsilon_B"'] \arrow[r, "\delta_B"] & D(DB)
\end{tikzcd}$$

commute.

The unique splitting map between the coKleisli and coEilenberg-Moore splitting is the functor $E : \mathrm{coKl}(D) \to \mathrm{coEM}(D)$ defined by $EA =_{\mathrm{df}} (DA, \delta_A)$, $Ek =_{\mathrm{df}} Dk \circ \delta_A = k^{\dagger}$. This functor is a full embedding. The image of E is the full subcategory of $\mathrm{coEM}(D)$ given by the cofree coalgebras that is therefore isomorphic to $\mathrm{coKl}(D)$.

A simple and instructive example of a comonad and its coKleisli and coEilenberg-Moore splittings is given by the *reader* (or *product*) *comonad*. It is defined on any category $\mathcal{C}$ with finite products, but let us choose $\mathcal{C}$ to be **Set** (or **Top** or **Unif**), so we can write pointwise definitions for intuitiveness. Given some fixed object $C \in |\mathcal{C}|$, it is defined by $DA =_{\mathrm{df}} A \times C$, $Df(x, c) =_{\mathrm{df}} (f(x), c)$, $\varepsilon_A(x, c) =_{\mathrm{df}} x$, $\delta_A(x, c) =_{\mathrm{df}} ((x, c), c)$, $k^{\dagger}(x, c) =_{\mathrm{df}} (k(x, c), c)$.

The coKleisli category has as objects those of $\mathcal{C}$ and as maps from A to B those from $A \times C$ to B. The identities and composition are defined by $\mathrm{jd}(x, c) =_{\mathrm{df}} x$, $(\ell \bullet k)(x, c) =_{\mathrm{df}} \ell(k(x, c), c)$.

A coalgebra of D is given by an object A and a map $u : A \to A \times C$ satisfying the laws of a coalgebra. Let us define $(u_0(x), u_1(x)) =_{\mathrm{df}} u(x)$. The laws impose that $u_0(x) = x$ and $((u_0(x), u_1(x)), u_1(x)) = (u(u_0(x)), u_1(x))$. The first law defines u_0 and the second becomes a tautology as soon as this definition is substituted into it. Hence, a coalgebra is effectively the same as an object A with an unconstrained map $u_1 : A \to C$.

A map between D-coalgebras (A, u), (B, v) is a map $f : A \to B$ such that $(f(u_0(x)), u_1(x)) = (v_0(f(x)), v_1(f(x)))$, which boils down to $u_1(x) = v_1(f(x))$.

The coEilenberg-Moore category has thus as objects pairs of an object A and map $u_1 : A \to C$ and a map between (A, u_1), (B, v_1) is map $f : A \to B$ such that $u_1(x) = v_1(f(x))$. The cofree coalgebra on A is the pair $(A \times C, \delta_{1A})$ where $\delta_{1A}(x, c) =_{\mathrm{df}} c$.

The isomorphism between $\mathrm{coKl}(D)$ and the category of cofree D-coalgebras establishes a 1-1 correspondence between maps $k : A \times C \to B$ and maps f : $A \times C \to B \times C$ such that $\delta_{1B}(f(x, c)) = c$.

3. Exponentials, topologies, and uniformities

Given an object C in a category $\mathcal{C}$ with finite products, it is said to be *exponentiable* if the functor $(-) \times C$ has a right adjoint. This amounts to the existence, for any object A, of an object A^C (the *exponential*) and map $\mathsf{ev}_A : A^C \times A \to C$ (the *evaluation*) as well as, for any objects A, B and map $k : A \times C \to B$, a map $\mathsf{cur}(k) : A \to B^C$ (the *currying* of k) satisfying appropriate conditions. If every object of $\mathcal{C}$ is exponentiable, it is called *Cartesian closed.* Intuitively, exponentials are internalized homsets. In **Set**, every object C is exponentiable and the exponential A^C is the set of all functions from C to A.

Things are somewhat more complicated in the category **Top** of topological spaces, as the exponential A^C is to be the set of continuous functions from C to A, but it must also be given a topology. Moreover, the evaluation $\mathsf{ev}_A : A^C \times A \to C$ must be continuous and the currying $\mathsf{cur}(k) : A \to B^C$ of a continuous function must be continuous.

Not every topological space is exponentiable. Hausdorff spaces are exponentiable if and only if they are locally compact: in this case, the exponential topology on the space of continuous functions from C to A is the *compact-open topology* generated by the sets $\{f : C \to A \mid f(K) \subseteq U\}$ with K compact in C and U open in A [7, 8]. In particular, discrete spaces are exponentiable (which also follows from the discrete topology making every function from it continuous) and their compact-open topology is in fact the product topology.

That not all objects can act as exponents is also true in the category **Unif** of uniform spaces whose constituents we now define. A *uniform space* is a set A endowed with a *uniformity*, i.e., a collection $\mathcal{U}$ of binary relations on A (called *entourages*) satisfying the following properties:

(1) $\Delta \subseteq U$ for every $U \in \mathcal{U}$, where $\Delta = \{(x, x) \mid x \in A\}$ is the *diagonal.*
(2) If $U \subseteq V$ and $U \in \mathcal{U}$ then $V \in \mathcal{U}$.
(3) If $U, V \in \mathcal{U}$ then $U \cap V \in \mathcal{U}$.
(4) If $U \in \mathcal{U}$ then $U^{-1} \in \mathcal{U}$.
(5) If $U \in \mathcal{U}$ then $V^2 = \{(x, y) \mid \exists z \mid (x, z), (z, y) \in V\} \subseteq U$ for some $V \in \mathcal{U}$.

The simplest non-trivial uniformity on A is the *discrete uniformity*, made of all the supersets of the diagonal. A uniformity induces a topology as follows: $\Omega \subseteq A$ is open if and only if, for every $x \in \Omega$, there exists $U \in \mathcal{U}$ such that $\{y \in A \mid (x, y) \in U\} \subseteq \Omega$. Such topology is Hausdorff if and only if $\bigcap_{U \in \mathcal{U}} U = \Delta$. The discrete uniformity induces the discrete topology, but is not the only one that does (cf. [11, I-5]), i.e., uniform spaces may be discrete without being *uniformly discrete.*

A map $f : A \to B$ between uniform spaces is *uniformly continuous* (briefly, u.c.) if, for every entourage V on B, there is an entourage U on A such that $(f \times f)(U) \subseteq V$. Any u.c. function is continuous in the topology induced by the uniformities: the converse is true if A is compact [11, II-24] but false in general even for metric spaces. The *product uniformity* is the coarsest uniformity that makes the projections uniformly continuous: the topology induced by the product uniformity is the product topology. A product of discrete uniformities is called *prodiscrete.*

In **Unif**, uniformly discrete objects are exponentiable [11, III.19 and III.21]. Again, the reason is that every function from C is u.c. as soon as C is uniformly discrete.

4. Cellular automata as coKleisli maps

Classically, a *cellular automaton* on a monoid $(G, 1_G, \cdot)$ (the *universe*)[1] and set A (the *alphabet*)[2] is given by a finite subset N of G (the *support neighborhood*) and function $d : A^N \to A$ (the *transition rule*).

Any cellular automaton induces a *local behavior* $k : A^G \to A$ via $k(c) =_{\mathrm{df}} d(c|_N)$.

One speaks of elements A^N of finite subsets $N \subseteq G$ as *patterns* and elements of A^G as *configurations*. Transition rules work on patterns, local behaviors on configurations. Cellular automata that induce the same local behavior are considered equivalent. In this paper, we will not distinguish between equivalent cellular automata, hence we can identify cellular automata with their local behaviors.

If the alphabet A is finite, a function $k : A^G \to A$ is a local behavior (i.e., the local behavior of some cellular automaton) if and only if it is continuous for the discrete topology on A and the product topology on A^G.

In the general case (where A may be infinite), the above equivalence does not generally hold, but a refinement does. A function $k : A^G \to A$ is then a local behavior iff it is uniformly continuous for the discrete uniformity on A and product uniformity on A^G.[3] The finite case becomes an instance: if A is finite, then A^G is compact and therefore any continuous function $k : A^G \to A$ is uniformly continuous.

Based on these observations, we henceforth take it as a definition that a local behavior on a set (the alphabet) A is a uniformly continuous function $k : A^G \to A$ wrt. the prodiscrete uniformity on A^G and forget about the definition of cellular automata in terms of a support neighborhood and a transition rule.

Any local behavior k induces a *global behavior* $k^\dagger : A^G \to A^G$, a map between configurations, via $k^\dagger(c)(x) =_{\mathrm{df}} k(c \rhd x)$ where $\rhd : A^G \times G \to A^G$ (the *translation* of configurations) is defined by $(c \rhd x)(y) =_{\mathrm{df}} c(x \cdot y)$ The translation is a uniformly continuous function. It follows that the global behavior $k^\dagger$ is also uniformly continuous.

Local behaviors on a fixed universe G and fixed alphabet A form a monoid with unit jd given by $\mathrm{jd}(c) =_{\mathrm{df}} c(1_G)$ and multiplication $\bullet$ given by $\ell \bullet k =_{\mathrm{df}} \ell \circ k^\dagger$. Indeed, it is easy to see that jd is uniformly continuous and $\bullet$ preserves uniform continuity (because $(-)^\dagger$ does) and the monoid laws turn out to hold too.

We now make two small generalizations and make a richer category out of local behaviors: after all, a monoid is a category with one object. First, we do not insist that the alphabet be a discrete uniform space, it may be any uniform space. And second, we give up the idea of a fixed alphabet: we let the local behavior change the alphabet.

For a fixed monoid G (the universe), we redefine a *local behavior* between two general uniform spaces (the source and target alphabets) A and B to be a uniformly continuous function $k : A^G \to B$ where A^G is given the product uniformity.

Local behaviors now make a category that has as objects alphabets and as maps local behaviors between them. The identity on A is $\mathrm{jd}_A : A^G \to A$ given by $\mathrm{jd}_A(c) =_{\mathrm{df}} c(1_G)$ and the composition $\ell \bullet k : A^G \to C$ of two maps $k : A^G \to B$

[1] Instead of the monoid, one usually takes a group in the cellular automata literature. But we do not need inverses in this paper.

[2] The alphabet is often required to be finite. We make this assumption only where we need it.

[3] (Cf. [5, Th. 1.9.1]) Every entourage of the prodiscrete uniformity contains an entourage of the form $V_N = \{(c, e) \mid c|_N = e|_N\}$ with $N \subseteq G$ finite. If $k : A^G \to A$ is u.c. with A uniformly discrete, then $(k \times k)(V_N) \subseteq \Delta$ for some finite $N \subseteq G$: thus, $k(c)$ only depends on $c|_N$.

and $\ell : B^G \to C$ given by $\ell \bullet k =_{\mathrm{df}} \ell \circ k^\dagger$ where $k^\dagger : A^G \to B^G$ is defined by $k^\dagger(c)(x) =_{\mathrm{df}} k(c \rhd_A x)$ from $\rhd_A : A^G \times G \to A^G$ defined by $(c \rhd_A x)(y) =_{\mathrm{df}} c(x \cdot y)$. Notice that these definitions coincide exactly with those we made for the monoid of local behaviors above, except that local behaviors can now mediate between different alphabets that need not be uniformly discrete. The function jd_A is still uniformly continuous for any A and the operation $\bullet$ preserves uniform continuity.

While the generalized definition of local behaviors is more liberal than the classical one, it is conservative over it in the following sense: The local behaviors from any uniformly discrete space A back to itself are exactly the classical local behaviors on A seen as a set.

We will now recover our category of local behaviors from a categorical generality, by showing that it is a straightforward instance of the coKleisli construction for a comonad.

Any fixed monoid $(G, 1_G, \cdot)$ determines a comonad (D, ε, δ) on **Unif** (in fact, on any category where the carrier G is exponentiable, so also, e.g., on **Set** and **Top**) (the *cellular automata* or *exponent comonad*) as follows. The object mapping part of D is defined by $DA =_{\mathrm{df}} A^G$, where A^G is the G-exponential of A, i.e., the space of uniformly continuous functions from G to A equipped with the prodiscrete uniformity. The morphism mapping part is defined by $Df =_{\mathrm{df}} f^G$, i.e., $Df(c) =_{\mathrm{df}} f \circ c$. The components of the counit $\varepsilon_A : A^G \to A$ and comultiplication $\delta_A : A^G \to (A^G)^G$ are defined by $\varepsilon_A(c) =_{\mathrm{df}} c(1_G)$ and $\delta_A(c)(x) =_{\mathrm{df}} c \rhd_A x$ (so that $\delta_A(c)(x)(y) = c(x \cdot y)$); these functions are uniformly continuous. The general definition of the coKleisli extension $(-)^\dagger$ via the morphism mapping part of D and comultiplication δ tells us that $k^\dagger(c)(x) = Dk(\delta_A(c))(x) = k(\delta_A(c)(x)) = k(c \rhd_A x)$.

The laws of a comonad are proved from the monoid laws for G by the following calculations (we omit the proofs of the naturality conditions of ε and δ).

$$\varepsilon_{DA}(\delta_A(c))(x) = \delta_A(c)(1_G)(x) = c(1_G \cdot x) = c(x)$$

$$c(x) = c(x \cdot 1_G) = \delta_A(c)(x)(1_G) = \varepsilon_A(\delta_A(c)(x)) = D\varepsilon_A(\delta_A(c))(x)$$

$$\begin{aligned} \delta_{DA}(\delta_A(c))(x)(y)(z) &= \delta_A(c)(x \cdot y)(z) = c((x \cdot y) \cdot z) \\ &= c(x \cdot (y \cdot z)) = \delta_A(c)(x)(y \cdot z) = \delta_A(\delta_A(c)(x))(y)(z) = D\delta_A(\delta_A(c))(x)(y)(z) \end{aligned}$$

As we have seen, a comonad on a category always defines two canonical splittings of its underlying functor into two adjoint functors. The coKleisli splitting of our comonad D on **Unif** goes via the coKleisli category which has as objects those of **Unif** and as maps from A to B those from DA to B in **Unif**. The identity on A is $\mathrm{jd}_A =_{\mathrm{df}} \varepsilon_A$ and the composition of k and ℓ is $\ell \bullet k =_{\mathrm{df}} \ell \circ k^\dagger$. Note that these are exactly the data of the category of local behaviors that we introduced above. But this time we do not have to prove that the unital and associativity laws of the category hold. Our proof obligations went into establishing that the comonad data are well defined and the comonad laws hold.

5. Retrieving the Curtis-Hedlund theorem

Let (D, ε, δ) be the G-exponential comonad on **Unif** for a given monoid $(G, 1_G, \cdot)$, with G endowed with the discrete uniformity, as introduced in the previous section.

As we know from Section 2, $\mathrm{coKl}(D)$ is equivalent to the category of cofree D-coalgebras under a comparison functor E that sends a coKleisli map (local behavior) $k : DA \to B$ to the cofree coalgebra map $k^\dagger : (DA, \delta_A) \to (DB, \delta_B)$, which, as a map of **Unif**, we know to be the corresponding global behavior. Hence, a map $f : DA \to DB$ would be a global behavior if and only if f is a cofree coalgebra map.

Now, given an arbitrary comonad, it is usually of interest to study its general coalgebras and not only the cofree ones. We too follow this thumb rule.

By definition, objects in coEM(D) are pairs of objects A and maps $u : A \to DA$ in **Unif** satisfying

$$\begin{tikzcd} A \arrow[r, "u"] \arrow[dr, equal] & A^G \arrow[d, "\varepsilon_A"] \\ & A \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "u"] \arrow[d, "u"'] & A^G \arrow[d, "\delta_A"] \\ A^G \arrow[r, "u^G"] & (A^G)^G \end{tikzcd}$$

In our case, the first equation simply means $u(a)(1_G) = a$ while the second one simplifies to $u(a)(x \cdot y) = u(u(a)(x))(y)$. This writing, however, is cumbersome and unexplicative.

To see more, we uncurry $u : A \to A^G$ to $\otimes : A \times G \to A$, so that $a \otimes x = u(a)(x)$. Then the two equations become $a \otimes 1_G = a$, and $a \otimes (x \cdot y) = (a \otimes x) \otimes y$. Diagrammatically, this is to require commutation of

$$\begin{tikzcd} A \arrow[r, "\rho_A"] \arrow[rrd, equal] & A \times 1 \arrow[r, "A\times 1_G"] & A \times G \arrow[d, "\otimes"] \\ & & A \end{tikzcd} \qquad \begin{tikzcd} (A \times G) \times G \arrow[r, "\alpha_{A,G,G}"] \arrow[d, "\otimes\times G"'] & A \times (G \times G) \arrow[r, "A\times(\cdot)"] & A \times G \arrow[d, "\otimes"] \\ A \times G \arrow[rr, "\otimes"] & & A \end{tikzcd}$$

where ρ and α are the right unital and associative laws of the product monoidal structure. But these are precisely the laws of a *(right) action* of G on A (where A is a uniform space, so we expect an action to also be uniformly continuous).

Let us now consider coalgebra maps. A map $f : (A, u) \to (B, v)$ in coEM(D) is a map in **Unif** that commutes with u and v or (which is equivalent) with their uncurried forms $\otimes$ and $\oslash$ as shown on the left and right diagrams below, respectively:

$$\begin{tikzcd} A \arrow[r, "u"] \arrow[d, "f"'] & A^G \arrow[d, "f^G"] \\ B \arrow[r, "v"] & B^G \end{tikzcd} \qquad \begin{tikzcd} A \times G \arrow[r, "\otimes"] \arrow[d, "f\times G"'] & A \arrow[d, "f"] \\ B \times G \arrow[r, "\oslash"] & B \end{tikzcd}$$

Clearly, coalgebra maps are just action maps.

We are now ready to consider maps of cofree D-coalgebras. The uncurried form of $\delta_A : A^G \to (A^G)^G$ is $\rhd_A : A^G \times G \to A^G$. By what we have shown, a map $f : A^G \to B^G$ in **Unif** is a map between (A^G, δ_A) and (B^G, δ_B) if and only if $f(c \rhd_A x) = (f(c) \rhd_B x)$. We see that maps between the cofree coalgebras (A^G, δ_A) and (B^G, δ_B) are precisely those maps $f : A^G \to B^G$ that commute with the translation!

With our reasoning, we have reproved the version of Curtis-Hedlund theorem given by Ceccherini-Silberstein and Coornaert ([4, Th. 1.1], [5, Th. 1.9.1]): global behaviors between uniform spaces A and B are those uniformly continuous functions between the product uniformities on A^G and B^G that commute with the translation. Keep in mind that we are allowing arbitrary uniform spaces as alphabets, and speaking of global behaviors in the sense of Section 4. It is, however, immediate to specialize to the statement of C.-S. and C. by requiring A and B to be uniformly discrete. Then local behaviors are precisely the finitary functions from A^G to B.

The original Curtis-Hedlund theorem ([10], [5, Th. 1.8.1]) corresponds to the special case where A is finite and discrete. In this case, A^G is compact and any continuous function between A^G and B^G is then uniformly continuous. We conclude, in this case, that global behaviors between A to B are those continuous functions between the product topologies on A^G and B^G that commute with the translation.

We have seen that the cofree coalgebra $(B^G, \delta_B : B^G \to (B^G)^G)$ on B is the currying of the translation action $(B^G, \rhd_B : B^G \times G \to B^G)$. Because of the cofreeness on B of this coalgebra, the action must also be the cofree.

In basic terms, this means that for any action $(A, \otimes : A \times G \to A)$ and map $p : A \to B$, there is a unique action morphism f to the translation action such that $\varepsilon_B \circ f = p$, i.e., a map $f : A \to B^G$ such that the following diagrams commute:

$$\begin{tikzcd}
A \times G \arrow[r, "\otimes"] \arrow[d, dotted, "f \times G"'] & A \arrow[d, dotted, "f"] \arrow[dr, "p"] & \\
B^G \times G \arrow[r, "\rhd_B"] & B^G \arrow[r, "\varepsilon_B"] & B
\end{tikzcd}$$

This fact can of course be proved from first principles, but we learned it for free!

6. Retrieving the reversibility principle

We will now recover the reversibility principle. Let again $(G, 1_G, \cdot)$ be a monoid and (D, ε, δ) be the G-exponential comonad on **Unif** as introduced in Section 4.

Suppose we have a cofree coalgebra map $f : (DA, \delta_A) \to (DB, \delta_B)$ so that f has an inverse f^{-1} as a map of **Unif**.

A very simple diagram chase (not specific to our particular comonad; we only use that D is a functor!) shows that f^{-1} is also a cofree coalgebra map, i.e., an inverse of f in coEM(D):

$$\begin{tikzcd}
 & DA \arrow[r, "\delta_A"] \arrow[d, "f"'] & D(DA) \arrow[r, equal] \arrow[d, "Df"] & D(DA) \\
DB \arrow[r, equal] \arrow[ur, "f^{-1}"] & DB \arrow[r, "\delta_B"] & D(DB) \arrow[ur, "Df^{-1}"'] &
\end{tikzcd}$$

We have thus reproved the reversibility principle [5, Th. 1.10.1]: A global behavior $f : A^G \to B^G$ (a uniformly continuous function commuting with the translation) between uniform spaces A, B is reversible if f has a uniformly continuous inverse.

If both A and B are finite and discrete, an inverse f^{-1} of f is necessarily uniformly continuous, because, in this case, A^G and B^G are compact Hausdorff and f^{-1} is continuous. So we obtain a special case [5, Th. 1.10.2]: A global behavior $f : A^G \to B^G$ (a continuous function commuting with the translation) between uniform spaces A, B is reversible if f has an inverse.

In general, an inverse of a uniformly continuous function is not necessarily uniformly continuous. For a counterexample, see [5, Example 1.10.3].

For reversibility, it is useful, if the monoid G is actually a group. In particular, for reversibility of $\delta_A : DA \to D(DA)$, G must be a group.

7. Distributive laws and 2-dimensional cellular automata

We will now proceed to two variations on the theme of cellular automata as co-Kleisli maps—2-dimensional (classical) cellular automata and point-dependent cellular automata. In both cases we first introduce some further comonad theory relevant for our cause.

Sometimes, but not always, the composition D^1D^0 of two comonads D^0 and D^1 on the same category $\mathcal{C}$ is a comonad. It is the case, if there is a distributive law of D^1 over D^0. A *distributive law* of a comonad $(D^1, \varepsilon^1, \delta^1)$ over a comonad $(D^0, \varepsilon^0, \delta^0)$ is a natural transformation $\kappa : D^1D^0 \to D^0D^1$ making the diagrams

$$\begin{tikzcd}
D^1D^0 \arrow[rr, "\kappa"] \arrow[dr, "D^1\varepsilon^0"'] & & D^0D^1 \arrow[dl, "\varepsilon^0 D^1"] \\
& D^1 &
\end{tikzcd}
\qquad
\begin{tikzcd}
D^1D^0 \arrow[rr, "\kappa"] \arrow[d, "D^1\delta^0"'] & & D^0D^1 \arrow[d, "\delta^0 D^1"] \\
D^1D^0D^0 \arrow[r, "\kappa D^0"] & D^0D^1D^0 \arrow[r, "D^0\kappa"] & D^0D^0D^1
\end{tikzcd}$$

$$\begin{tikzcd}
D^1D^0 \arrow[rr, "\kappa"] \arrow[dr, "\varepsilon^1 D^0"'] & & D^0D^1 \arrow[dl, "D^0\varepsilon^1"] \\
& D^0 &
\end{tikzcd}
\qquad
\begin{tikzcd}
D^1D^0 \arrow[rr, "\kappa"] \arrow[d, "\delta^1 D^0"'] & & D^0D^1 \arrow[d, "D^0\delta^1"] \\
D^1D^1D^0 \arrow[r, "D^1\kappa"] & D^1D^0D^1 \arrow[r, "\kappa D^1"] & D^0D^1D^1
\end{tikzcd}$$

commute. A distributive law induces a comonad (D, ε, δ) defined by $D =_{\mathrm{df}} D^1D^0$, $\varepsilon =_{\mathrm{df}} \varepsilon^1\varepsilon^0$, $\delta =_{\mathrm{df}} D^1\kappa D^0 \circ \delta^1\delta^0$.

A distributive law also induces comonad liftings. For the lack of space, we concentrate on the coKleisli side of the picture. Given a distributive law κ of D^1 over D^0, the comonad D^0 on $\mathcal{C}$ lifts to $\mathrm{coKl}(D^1)$, i.e., induces a comonad $\bar{D}^0$ on $\mathrm{coKl}(D^1)$. This comonad is defined by: $\bar{D}^0A =_{\mathrm{df}} D^0A$, $\bar{D}^0k =_{\mathrm{df}} D^0k \circ \kappa_A : D^1D^0A \to D^0B$ (for $k : D^1A \to B$), $\varepsilon_A =_{\mathrm{df}} \varepsilon^0_A \circ \varepsilon^1_{D^0A}$, $\delta_A =_{\mathrm{df}} \delta^0_A \circ \varepsilon^1_{D^0A}$.

Via this lifting, the coKleisli category of the composite comonad D and the other ingredients of the coKleisli splitting of D can be obtained by a double coKleisli construction: We have $\mathrm{coKl}(D) = \mathrm{coKl}(\bar{D}^0)$ (equal strictly, not just isomorphic).

A simple example of a distributive law is obtained by taking D^1 to be any comonad and D^0 the product comonad defined by $D^0A =_{\mathrm{df}} A \times C$ for a fixed object $C \in |\mathcal{C}|$. The distributive law κ of D^1 over D^0 is given by $\kappa_A =_{\mathrm{df}} \langle D^1\pi_0, \varepsilon^1 \circ D^1\pi_1\rangle : D^1(A \times C) \to D^1A \times C$. It follows that the functor D defined by $DA =_{\mathrm{df}} D^1(A \times C)$ is a comonad.

Given now two monoids G_0, G_1, we can think of a map $k : (A^{G_0})^{G_1} \to B$ in **Unif** as a "2-dimensional" (2D) cellular automaton on the universes G_0, G_1 between alphabets A and B (relying on the isomorphism $A^{G_0 \times G_1} \cong (A^{G_0})^{G_1}$).

Such a cellular automaton is by definition the same thing as a "1-dimensional" (1D) cellular automaton on the universe G_1 between alphabets A^{G_0} and B. Note that we can only see 2D cellular automata as 1D in this way, if we allow source and target alphabets of a cellular automaton to differ and if we do not require them to be uniformly discrete (notice that A^{G_0} carries the prodiscrete uniformity). But this view of 2D cellular automata as 1D, although nice, suffers from a serious drawback. Since the 1D views do not have the same source alphabets as the 2D originals, they do not compose the same way.

Distributive laws come to help. Let $D^0A =_{\mathrm{df}} A^{G_0}$ and $D^1A =_{\mathrm{df}} A^{G_1}$. There is a distributive law $\kappa : D^1D^0 \to D^0D^1$ defined by $\kappa_A(c)(x_1)(x_0) = c(x_0)(x_1) : (A^{G_0})^{G_1} \to (A^{G_1})^{G_0}$. Hence, the functor $DA =_{\mathrm{df}} (A^{G_0})^{G_1}$ is a comonad, which is hardly a surprise. But there is more: We know that $\mathrm{coKl}(D) = \mathrm{coKl}(\bar{D}^0)$. Hence, a good view of $k : (A^{G_0})^{G_1} \to B$ as a 1D cellular automaton is not as a **Unif**-cellular automaton on the universe G_1 between the alphabets A^{G_0} and B, but on as a $\mathrm{coKl}(D^1)$-cellular automaton on the universe G_0 between the alphabets A and B. Then 1D views compose exactly as their 2D originals. We see that it makes sense to consider maps of categories other than **Unif**!

1D views of 2D cellular automata were of interest to Dennuzio et al. [6]

8. Comonad maps and point-dependent cellular automata

To make a category out of comonads over a fixed category $\mathcal{C}$ one needs a suitable notion of comonad maps. A *comonad map* between two comonads (D, ε, δ) and $(D', \varepsilon', \delta')$ on $\mathcal{C}$ is a natural transformation $\tau : D \to D'$ making the diagrams

$$\begin{tikzcd}
D \arrow[dr, "\varepsilon"] \arrow[dd, "\tau"'] & \\
& \mathrm{Id}_{\mathcal{C}} \\
D' \arrow[ur, "\varepsilon'"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
D \arrow[r, "\delta"] \arrow[d, "\tau"'] & DD \arrow[d, "\tau\tau"] \\
D' \arrow[r, "\delta'"] & D'D'
\end{tikzcd}$$

commute. Comonads and comonad maps on $\mathcal{C}$ form a category. The identity and composition of comonad maps is inherited from the category of natural transformations between endofunctors on $\mathcal{C}$.

A comonad map τ between (D, ε, δ) and $(D', \varepsilon', \delta')$ relates the coKleisli and coEilenberg-Moore categories between the two comonads. It defines a functor from $\mathrm{coKl}(D')$ to $\mathrm{coKl}(D)$ and a functor from $\mathrm{coEM}(D)$ to $\mathrm{coEM}(D')$.

We now introduce point-dependent cellular automata (studied under the name of non-uniform cellular automata by Cattaneo et al. [3]). For a set G, the local behavior of a point-dependent cellular automaton between uniform spaces A, B is a uniformly continuous function $k : A^G \times G \to B$. Note the added second argument compared to the definition of a classical local behavior.

It turns out that local and global behaviors of point-dependent cellular automata can be analyzed in the same way as those of classical cellular automata. In particular, their local behaviors are the same thing as coKleisli maps of a suitable comonad (D, ε, δ) on **Unif** and global behaviors are the corresponding cofree coalgebra maps!

Let us review the data of the comonad. The object mapping of D is defined by $DA =_{\mathrm{df}} A^G \times G$ and the morphism mapping by $Df =_{\mathrm{df}} f^G \times G$, i.e., $Df(c, x) =_{\mathrm{df}} (\lambda y.f(c(y)), x)$. The components of the counit and comultiplication $\varepsilon_A : A^G \times G \to A$ and $\delta_A : A^G \times G \to (A^G \times G)^G \times G$ are defined by $\varepsilon_A(c, x) =_{\mathrm{df}} c(x)$, $\delta_A(c, x) = (\lambda y.(c, y), x)$. Accordingly, the coKleisli extension $k^\dagger : A^G \times G \to B^G \times G$ of a map $k : A^G \times G \to B$ is forced to satisfy $k^\dagger(c, x) = Dk(\delta(c, x)) = Dk(\lambda y.(c, y), x) = (\lambda y.k(c, y), x)$.

When is a map $f : A^G \times G \to B^G \times G$ a global behavior? It is a global behavior iff it is a cofree coalgebra map. Not surprisingly at all, the conditions for f being a cofree coalgebra map reduce to the condition that $f(c, x) = (g(c), x)$ for some $g : A^G \to B^G$.

Assume G is endowed with a monoid structure $(1_G, \cdot)$. Let $(D', \varepsilon', \delta')$ be the comonad of classical cellular automata. The translation $\rhd$ is a comonad map from D to D'. Accordingly, any classical local behavior is also a point-dependent local behavior that simply makes no use the point information that is available.

9. Conclusions

It was not the purpose of this paper to prove deep or difficult theorems. Rather, we set out to experiment with definitions. We deem that this experiment succeeded. We were pleased to learn that, from the category-theoretic point-of-view, cellular automata are a "natural" construction with "natural" properties. Crucially, classical cellular automata are coKleisli maps of the exponential comonad on **Unif**, and it is harmless to accept alphabets with nondiscrete uniformities and variation of alphabets, once it has been decided that local behaviors are uniformly continuous functions. But other base categories can be useful too, as the example of 2-dimensional cellular automata as 1-dimensional shows.

We hope to be able to extend this work to cover more results of cellular automata theory, in particular results toward the Garden of Eden theorem.

Acknowledgments. We are grateful to Jarkko Kari and Pierre Guillon for comments.

References

[1] Barr, M. and Wells, C. (1983) *Toposes, Triples and Theories. Grundlehren der math. Wissenschaften* **278**. Springer. // Revised and corrected electronic version (2005). *Reprints in Theory and Appl. of Categ.* **12**, 1–287.

[2] Brookes, S. and Geva, S. (1992) Computational comonads and intensional semantics. In Fourman, M. P., Johnstone, P. T., and Pitts, A. M., eds., *Applications of Categories in Computer Science, London Math. Society Lect. Note Series* **177**, 1-44. Cambridge Univ. Press.

[3] Cattaneo, G., Dennunzio, A. Formenti, E., and Provillard, J. (2009) Non-uniform cellular automata. In Dediu, A. H., Ionescu, A.-M., and Martín-Vide, C., eds., *Proc. of 3rd Int. Conf. on Languages and Automata Theory and Applications, LATA 2009 (Tarragona, Apr. 2009), Lect. Notes in Comput. Sci.* **5457**, 302–313. Springer.

[4] Ceccherini-Silberstein, T. and Coornaert, M. (2008) A generalization of the Curtis-Hedlund theorem. *Theor. Comput. Sci.* **400**(1–3), 225–229.

[5] Ceccherini-Silberstein, T. and Coornaert, M. (2010) *Cellular Automata and Groups*, *Springer Monographs in Mathematics.* Springer.

[6] Dennunzio, A. and Formenti, E. Decidable properties of 2D cellular automata. (2008) In Ito, M. and Toyama, M., eds., *Proc. of 12th Int. Conf. on Developments in Language Theory, DLT 2008 (Kyoto, Sept. 2008), Lect. Notes in Comput. Sci.* **5257**, 264–275. Springer.

[7] Escardó, M. and Heckmann, R. (2001) Topologies on spaces of continuous functions. *Topol. Proc.* **26**(2), 545–564.

[8] Geroch, R. (1985) *Mathematical Physics.* University of Chicago Press.

[9] Hasuo, I., Jacobs, B., and Uustalu, T. (2007) Categorical views on computations on trees. In Arge, L., Cachin, C., Jurdzinski, T., and Tarlecki, A., eds., *Proc. of 34th Int. Coll. on Automata, Languages and Programming, ICALP 2007 (Wrocław, July 2007), Lect. Notes in Comput. Sci.* **4596**, 619–630. Springer.

[10] Hedlund, G. A. (1969) Endomorphisms and automorphisms of the shift dynamical system. *Math. Syst. Theory* **3**(4), 320–375.

[11] Isbell, J.R. (1964) *Uniform Spaces.* Am. Math. Soc.

[12] Mac Lane, S. (1997) *Categories for the Working Mathematician. Graduate Texts in Mathematics* **5**. 2nd edition. Springer.

[13] Moggi, E. (1991) Notions of computation and monads. *Inform. and Comput.* **93**(1), 55-92.

[14] Piponi, D. “sigfpe” (2006) Evaluation of cellular automata is comonadic. Entry on the author’s blog, *A Neighborhood of Infinity.* `http://blog.sigfpe.com/2006/12/evaluating-cellular-automata-is.html`

[15] Uustalu, T. and Vene, V. The essence of dataflow programming. (2006) In Horváth, Z., ed., *Revised Selected Lectures from 1st Central European Functional Programming School, CEFP 2005 (Budapest, July 2005), Lect. Notes in Comput. Sci.* **4164**, 135–167. Springer.

[16] Uustalu, T. and Vene, V. Comonadic evaluation of attribute grammars. (2006) In van Eekelen, M., ed., *Trends in Functional Programming 6*, 145–162. Intellect.

[17] Uustalu, T. and Vene, V. (2008) Comonadic notions of computation. *Electron. Notes in Theor. Comput. Sci.* **203**(5), 263–284.

[18] Wadler, P. (1992) The essence of functional programming. In *Conf. Record of 19th Ann. ACM SIGPLAN-SIGACT on Principles of Programming Languages, POPL ’92*, 1–14. ACM Press.

COMBINATORIAL SUBSTITUTIONS AND SOFIC TILINGS

THOMAS FERNIQUE AND NICOLAS OLLINGER

Laboratoire d'Informatique Fondamentale de Marseille (LIF), Aix-Marseille Université,
CNRS, 39 rue Joliot-Curie, 13013 Marseille, France
E-mail address: {Thomas.Fernique,Nicolas.Ollinger}@lif.univ-mrs.fr

Abstract. A combinatorial substitution is a map over tilings which allows to define sets of tilings with a strong hierarchical structure. In this paper, we show that such sets of tilings are sofic, that is, can be enforced by finitely many local constraints. This extends some similar previous results (Mozes'90, Goodman-Strauss'98) in a much shorter presentation.

1. Introduction

Tiling some space with geometrical shapes, or tiles, consists into covering this space with copies of the tiles. When a set of tilings can be characterized by adding finitely many local constraints on tiles so that the set of tilings corresponds exactly to the set of tilings satisfying the constraints, such a set of tilings is called sofic. Soficity corresponds to the interesting idea that the validity of a tiling can be locally proved by decorating tiles with constraints. In this paper, we contribute to a general question: what sets of tilings are sofic?

Substitutions provide a simple way to express how a set of tilings can be obtained by iteratively constructing bigger and bigger aggregates of tiles. The strong hierarchical structure of substitutions tilings permits to enforce global properties, for example aperiodicity. To prove that the set of tilings generated by a substitution is sofic, one has to encode the global hierarchical structure into local constraints on tiles. Such technique is at the root of classical papers on the undecidability of the Domino Problem [2, 9]. In the case of tilings on the square grid, Mozes [6] proved in a seminal paper that the set of tilings generated by rectangular non-deterministic substitutions satisfying a particular property is sofic. Goodman-Strauss [5] proved that such a construction method can be extended to a wide variety of geometrical substitutions. In this paper, based on ideas developed in [7], we further extend the construction to a broader class of substitutions by replacing the geometrical conditions by combinatorial conditions while decreasing the length of the presentation.

Let us sketch some definitions to state our main theorem. A sofic tiling is a valid tiling by a finite set of tiles with decorations. Combinatorial substitutions, introduced by Priebe-Frank in [8], map a tiling by tiles onto a tiling by so-called macro-tiles (finite tilings, here assumed to be connected), so that the macro-tiles of

Partially supported by the ANR projects EMC (ANR-09-BLAN-0164) and SubTile.

the latter are arranged as the tiles of the former. The limit set of a substitution is the set of complete tilings that admits preimages of any depth by the substitution. A good substitution is a substitution that is both connecting (a combinatorial condition ensuring that there is enough room for all information to flow) and consistent (a geometrical condition ensuring that the substitution can be correctly iterated). The main result obtained in this paper is:

Theorem 1.1. *The limit set of a good combinatorial substitution is sofic.*

The paper is organized as follows. Sections 2 and 3 formally define main notions, in particular sofic tilings and good combinatorial substitutions. Section 4 then presents *self-simulation*, which plays a central role in the constructive proof of Theorem 1.1, which is given in section 5. Last, section 6 concludes the paper by discussing an important parameter of this proof.

A single example illustrates definitions and results throughout the whole paper: the "Rauzy" example. It relies on the theory of *generalized substitutions* introduced in [1], a self-contained presentation of which is beyond the scope of this paper. Let us just mention that *Rauzy tilings* are digitizations of the planes of the Euclidean space with a specific given irrational normal vector. More details, as well as the results we here implicitly rely on, can be found in [3, 4]. We chose this example, not the simplest one, because it is not covered by results in [6, 5].

2. Sofic tilings

Polytopes are assumed to be homeomorphic to closed balls of $\mathbb{R}^d$ and to have finitely many faces, with the $(d-1)$-dimensional faces being called *facets*.

A *tile* T is a polytope of the Euclidean space $\mathbb{R}^d$. A *tiling* Q of a *domain* $D \subset \mathbb{R}^d$ is a covering of D by interior-disjoint tiles, with the additional condition that two tiles can intersect (if they do) only along entire faces. A tiling is said to be *finite* if its domain is bounded.

Two tiles are said to be *adjacent* if they intersect along at least one facet, and a tiling is said to be *connected* if any two of its tiles can be connected by a sequence of adjacent tiles.

A facet of a tiling is said to be *external* if it is on the boundary of the domain, *internal* otherwise. We denote by ∂Q the set of external facets of a tiling Q. If this set is empty, then the tiling is said to be *complete*: its domain is the whole $\mathbb{R}^d$.

A *decorated tile* $\mathcal{T}$ is a tile with a real map defined on its boundaries, called *decoration*. Two adjacent decorated tiles are said to *match* if their decorations are equal in any point of their intersecting facets. A *decorated tiling* $\mathcal{Q}$ is a tiling by decorated tiles which pairwise match (when adjacent). Decorations are thus local constraints on the way tiles can be arranged in a tiling.

A *tileset* τ is a set of decorated tiles. It is said to be *finite* if it contains only a finite number of tiles up to direct isometries. A decorated tiling whose tiles belong to a tileset τ is called a τ-tiling; the set of τ-tilings is denoted by Λ_τ.

One says that a tiling *can be seen as* a decorated tiling if both are equal up to decorations. One denotes by π the map which removes the decorations. One easily checks that any tiling can be seen as a τ-tiling if τ can be infinite. The interesting case is the one of finite tilesets:

Definition 2.1. A set of tilings is said to be *sofic* if it can be seen as the set of τ-tilings of some finite tileset τ.

For example, one can wonder whether the *Rauzy tilings* are sofic (Fig. 1).

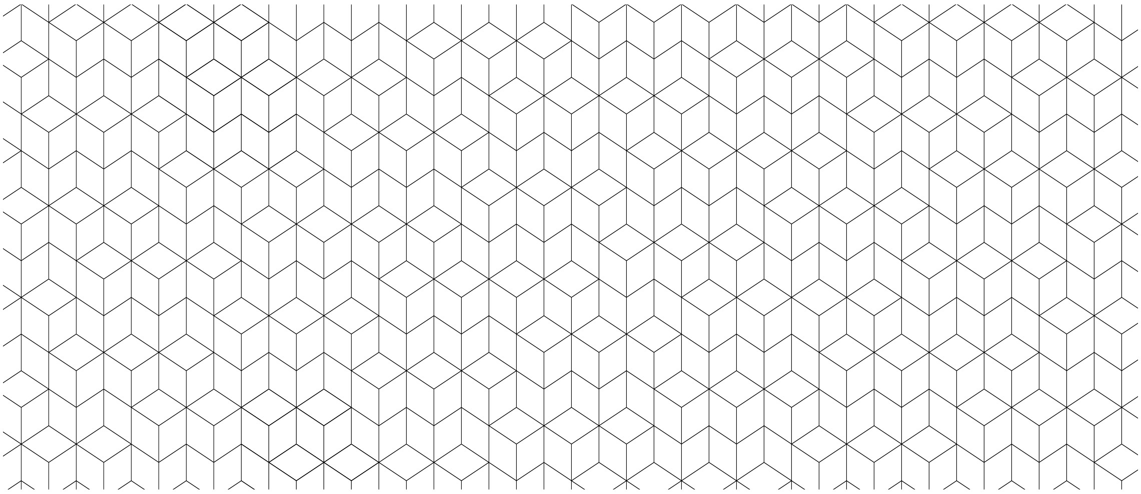


Figure 1: A Rauzy tiling (partial view). Are Rauzy tilings sofic?

3. Combinatorial substitutions

Definition 3.1. A *combinatorial substitution* is a finite set of *rules* (P, Q, γ), where P is a tile, Q is a finite connected tiling, and $\gamma : \partial P \to \partial Q$ maps distinct facets on disjoint sets of facets. The tiling Q is called a *macro-tile*, and if f is the k-th facet of P, then $\gamma(f)$ is called the k-th *macro-facet* of Q.

Fig. 2 illustrates this definition.

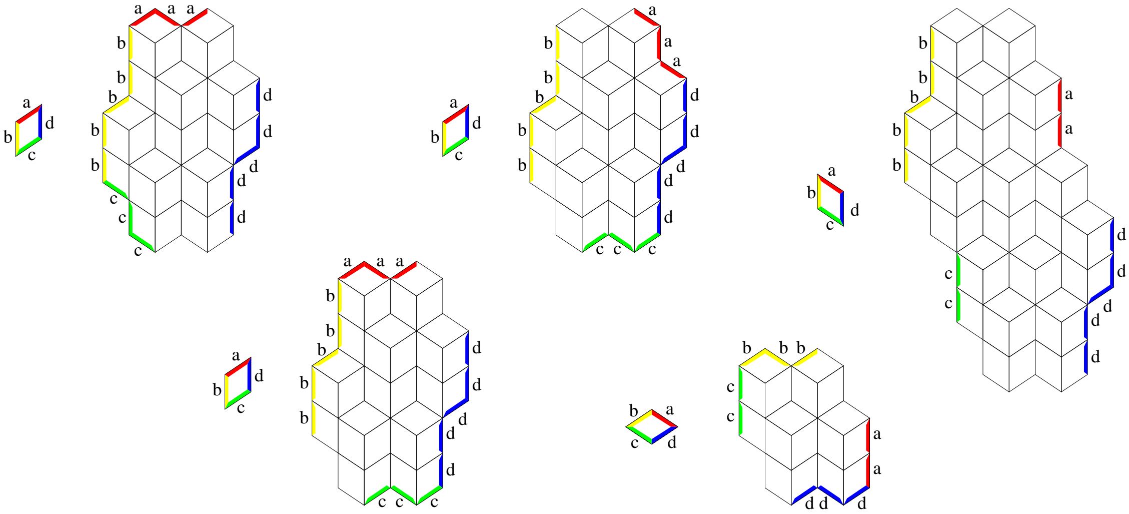


Figure 2: These five rules define the so-called Rauzy combinatorial substitution (a facet f and the corresponding macro-facet $\gamma(f)$ are similarly marked).

We call *tiling by macro-tiles* a tiling whose tiles can be partitioned into macro-tiles, with each macro-facet belonging to the intersection of exactly two macro-tiles. We can associate with each combinatorial substitution a binary relation over tilings:

Definition 3.2. Let σ be a combinatorial substitution. A tiling T by tiles of σ and a tiling T' by macro-tiles of σ are said to be *σ-related* if there is a one-to-one correspondence between the tiles of T and the macro-tiles of T' which preserves the combinatorial structure, that is, such that the a-th facet of a first tile of T matches the b-th facet of a second tile of T if and only if the a-th macro-facet of the first corresponding macro-tile of T' matches the b-th macro-facet of the second corresponding macro-tile of T'. One calls T a *preimage* of T' and T' an *image* of T.

For example, a Rauzy tiling can be uniquely seen as a tiling by Rauzy macro-tiles (Fig. 3) and has a unique preimage, which turns out to be itself a Rauzy tiling (both facts are non-trivial; they follow from results proven in [3]).

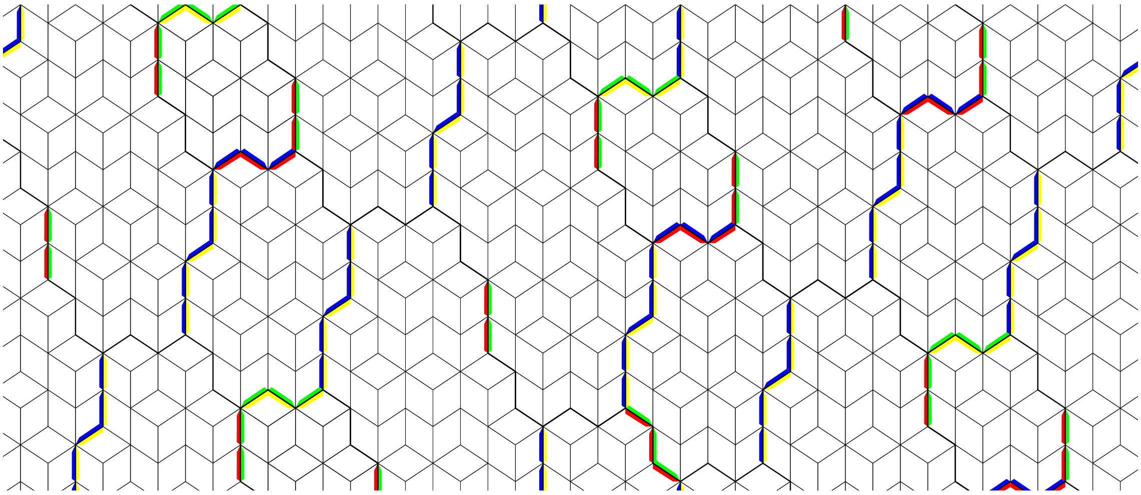


Figure 3: A Rauzy tiling can be uniquely seen as a tiling by Rauzy macro-tiles.

In particular, the relations associated with combinatorial substitutions yield a strong hierarchical structure on so-called limit-sets:

Definition 3.3. The *limit set* of a combinatorial substitution σ, denoted by Λ_σ, is the set of complete tilings which admit an infinite sequence of preimages.

For example, the limit set of the Rauzy combinatorial substitution is exactly the set of Rauzy tilings (again, this non-trivial fact follows from results proven in [3]).

Let us now turn to the *good combinatorial substitutions* to which Theorem 1.1 applies. First, a good combinatorial substitution must be *connecting*:

Definition 3.4. A combinatorial substitution σ is *connecting* if, for each rule (P, Q, γ), the dual graph[1] of Q has a subgraph N, called its *network*, such that

(1) N is a star with one branch for each macro-facet, and the leaf of the k-th branch is a tile with a facet, called k-th *port*, in the k-th macro-facet of Q;
(2) Each macro-facet has non-port facets, and removing the edges of N and its central vertex yields a connected graph which connects[2] all these facets;
(3) the center of N corresponds to a tile in the interior of Q, called *central tile*;

[1] The dual graph of a tiling is the graph whose vertices correspond to tiles of the tiling and whose edges connect vertices corresponding to adjacent tiles.

[2] One says that a subgraph *connects* a set of facets if these facets all belong to tiles which correspond to vertices of this subgraph.

(4) whenever two macro-tiles match along a port, they also match along the corresponding macro-facet.

Informally, the two first conditions ensure that the macro-facets are big enough to transfer via the network all the informations (encoded by decorations) that we need to enforce the hierarchical structure of the limit set. In particular, one easily sees that connectivity could not be achieved for combinatorial substitutions on the real line: Theorem 1.1 can apply only in dimension two or more. The third condition ensures that, by iteratively considering macro-tiles of macro-tiles (that is, when going higher and higher in the hierarchy of the limit set), we get tilings covering arbitrarily big balls. The last condition associate a port with its macro-facet (it is equivalent to the "sibling-edge-to-edge" condition of [5]).

Second, a good combinatorial substitution must be *consistent*:

Definition 3.5. A combinatorial substitution σ is said to be *consistent* if any tiling by macro-tiles of σ admits a preimage under σ.

Intuitively, consistency ensures that, if a tiling meets all the combinatorial conditions to have a preimage, then there is no geometrical obstruction to the existence of such a preimage.

Open Problem 3.6. Characterize consistent combinatorial substitutions.

For example, the Rauzy combinatorial substitution is connecting (one easily finds a suitable network for each rule, see Fig. 4 and 5) and consistent (this non-trivial fact follows from results in [4], where explicit maps defined over tilings are shown to be equivalent to such combinatorial substitutions). Theorem 1.1 thus yields that its limit set, *i.e.*, the set of Rauzy tilings, is sofic.

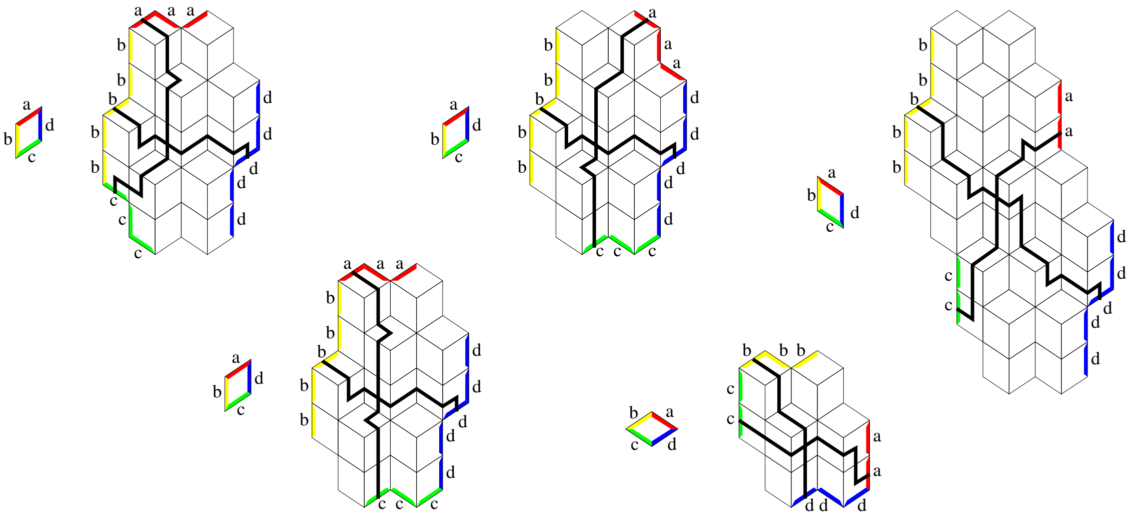


Figure 4: The Rauzy combinatorial substitution is connecting.

4. Self-simulation

Definition 4.1. Let σ be a combinatorial substitution with the rules $\{(P_i, Q_i, \gamma_i)\}_i$. A tileset τ is said to σ-*self-simulates* if there is a set of τ-tilings, called τ-macro-tiles, and a map ϕ from these τ-macro-tiles into τ such that

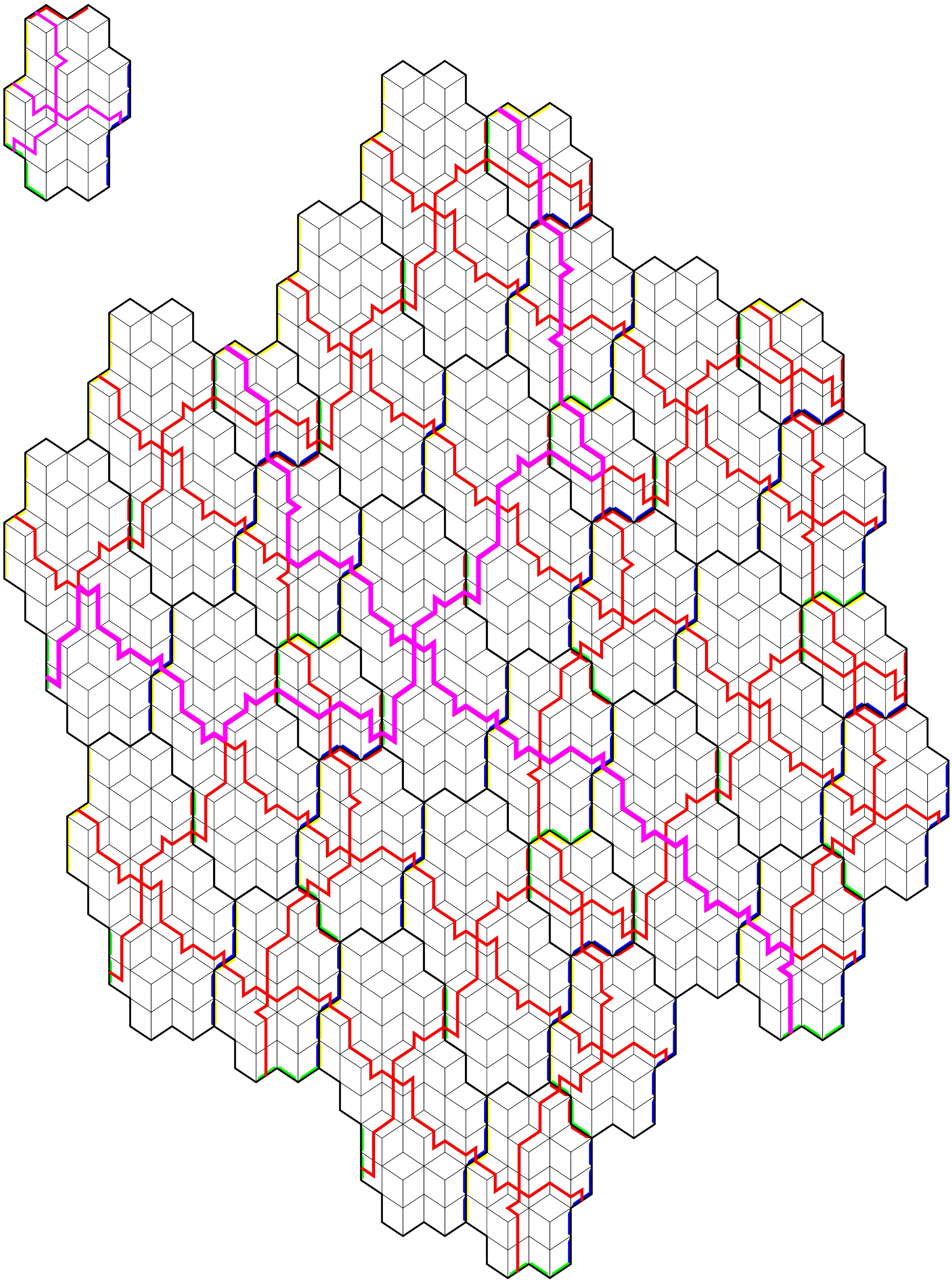


Figure 5: A Rauzy macro-tile (top-left) and an image of it under the Rauzy combinatorial substitution, that one could call Rauzy "macro-macro-tile" (center).

(1) for any τ-macro-tile $\mathcal{Q}$, there is i such that $\pi(\mathcal{Q}) = Q_i$ and $\pi(\phi(\mathcal{Q})) = P_i$;

(2) any complete τ-tiling can be seen as a tiling by τ-macro-tiles;

(3) the a-th macro-facet of a τ-macro-tile $\mathcal{Q}$ can match the b-th macro-facet of a τ-macro-tile $\mathcal{Q}'$ if and only if the a-th facet of the τ-tile $\phi(\mathcal{Q})$ can match the b-th facet of the τ-tile $\phi(\mathcal{Q}')$.

Proposition 4.2. *If a tileset σ-self-simulates for a consistent combinatorial substitution σ, then its complete tilings are, up to decorations, in the limit set of σ.*

Proof. Consider a complete τ-tiling $\mathcal{P}$. Conditions (1)–(2) ensure that removing the decorations of $\mathcal{P}$ yields a tiling by macro-tiles of σ, say P. The consistency of σ ensures that P admits a preimage under σ, say R. Let us show that R can be endowed by decorations to get a τ-tiling. Consider a tile T of R. This tile corresponds (via the one-to-one correspondence in Def. 3.2) to a macro-tile Q of P, which is itself the image under π of a τ-macro-tile $\mathcal{Q}$ of $\mathcal{P}$. We associate with T the τ-tile $\phi(\mathcal{Q})$. Condition (3) ensures that replacing tiles in R by their such associated τ-tiles yields a complete τ-tiling, say $\mathcal{R}$. We can repeat all this process, with $\mathcal{R}$ instead of $\mathcal{P}$. By induction, we get an infinite sequence of complete tilings, with each one being the preimage under σ of the previous one. Thus, $\mathcal{P} \in \Lambda_\sigma$. ■

5. Constructive proof of Theorem 1.1

Let σ be a good combinatorial substitution with rules $(P_i, Q_i, \gamma_i)_i$. We here rely on the fact that σ is connecting to construct a finite tileset τ which σ-self-simulates. The consistency of σ then ensures, via Prop. 4.2, that $\pi(\Lambda_\tau) \subseteq \Lambda_\sigma$ holds. We also show that the converse inclusion holds. This thus constructively proves Theorem 1.1.

5.1. Settings

Let $T_1, \ldots, T_n$ and $f_1, \ldots, f_m$ be numberings of, respectively, the tiles and the internal facets of all the Q_i's. Given the k-th facet of the tile T_i, $N_\sigma(i,k)$ stands either for its index if it is an internal facet, or for a special value "port", "macro-facet" or "boundary" otherwise (depending whether it is a port, a non-port facet in a macro-facet or another external facet).

Each tile of τ is a T_i endowed with a decoration which encodes on each facet a triple (f, j, g), where f and g are either facet indices or special values "port", "macro-facet" or "boundary", and j is either zero or a tile index. We call f the *macro-index*, j the *parent-index* and g the *neighbor-index*. This clearly allows only a finite number of different tiles.

We skip the technical details concerning the way these triples are encoded by decorations[3]. We assume that two decorated tiles match along a facet if and only if the same triple is encoded on both facets, and that the only direct isometry which leaves invariant the decoration of a facet is the identity, so that a decorated tile cannot trivially match with a translated or rotated copy of itself.

5.2. Decorations

The five following steps completely define a tileset τ.

1. The macro-index of the k-th facet of any decorated T_j is $N_\sigma(j,k)$. This step ensures that any complete τ-tiling can be uniquely seen as a tiling by τ-macro-tiles.

2. Consider a decorated non-central tile of Q_i. Its facets which are internal and not crossed by the network have all the same parent-index, also called parent-index of the tile, which can be any j such that $T_j = P_i$. Its facets which are external, port

[3] Recall that the decoration of a tile is a real map defined on its boundary.

excluded, have parent-index 0. This step ensure that the τ-tiles of a τ-macro-tile $\mathcal{Q}$ share a common parent-index j; the tile T_j is called the *parent-tile* of $\mathcal{Q}$.

3. Consider, in a non-central τ-tile with parent-index j, a facet which is neither a port nor crossed by the network. Its neighbor-index is either $N_\sigma(j,k)$ if it is in the k-th macro-facet, or equal to its macro-index otherwise. This step ensures that the macro-facets (ports excepted) of a τ-macro-tile are equivalent to the macro-indices of its parent tile (once decorated).

4. Consider a non-central τ-tile with parent-index j. Its facets which are either k-th port or crossed by the k-th branch of the network have all the same pair of parent/neighbor indices. This pair is either one of the pairs allowed on the k-th facet of T_j, if those are already defined in Steps 1–3 (*i.e.*, if the k-th facet of T_j is not crossed by a network), or any of the pairs defined in Steps 1–3 otherwise. This step ensures that each port of a τ-macro-tile is equivalent to the pair of parent/neighbor indices of a decorated parent-tile[4].

5. Whenever a non-central τ-tile $\mathcal{T}$ has as many facets as a central tile T_j, we define a central τ-tile $\mathcal{T}'$ by endowing each k-th facet of T_j with the pair of parent/neighbor indices on the k-th facet of $\mathcal{T}$ (the macro-indices are defined as usual, see Step 1). One says that $\mathcal{T}'$ *derives* from $\mathcal{T}$.

5.3. First inclusion

Let us show that the above defined tileset σ-self-simulates. Given a τ-macro-tile $\mathcal{Q}$ with parent-index j and central τ-tile $\mathcal{T}'$, let $\phi(\mathcal{Q})$ be the decorated tile obtained by endowing the k-th facet of T_j with the parent/neighbor indices on the k-th facet of $\mathcal{T}'$ (the macro-indices are defined as for any tile, see Step 1). One checks that ϕ is a map satisfying conditions (1)–(3) of Def. 4.1. It remains to check that $\phi(\mathcal{Q}) \in \tau$.

Let $\mathcal{T}$ be a non-central τ-tile from which derives $\mathcal{T}'$. If T_j is central, then $\phi(\mathcal{Q})$ also derives from $\mathcal{T}$, hence is in τ. Otherwise, Step 4 ensures, for each k, that the parent/neighbor indices on the k-th facet of $\mathcal{T}'$ appear on the k-th facet of a decorated T_j. In particular, this holds on facets of T_j which are not crossed by a network. Since the neighbor-indices of these facets can only[5] appear on a T_j (see Steps 1 and 3), $\mathcal{T}$ is a decorated T_j. This yields $\phi(\mathcal{Q}) = \mathcal{T}$, and thus $\phi(\mathcal{Q})$ is in τ.

Prop. 4.2 then applies and yields the first inclusion $\pi(\Lambda_\tau) \subseteq \Lambda_\sigma$.

5.4. Second inclusion

Let us extend τ in a tileset τ' as follow. For each τ-tile $\mathcal{T}$ and each subset S of its facets, we define a τ'-tile $\mathcal{T}_S$ by replacing the decorations of the facets in S by a special decoration "undefined".

Now, let P be a tiling in Λ_σ. Consider an infinite sequence $(P_n)_{n\geq 0}$ of successive preimages of P. Given $n > 0$, P_n can be seen as a τ'-tiling: it suffices to endow any facet with "undefined". Then, P_{n-1} can be seen as a tiling by τ'-macro-tiles, with

[4] With the problem that a parent-tile can be decorated in different ways, and nothing yet prevents the ports from mixing these decorations.

[5] Actually, a facet-index appears on the two tiles of a macro-tile which share the corresponding facet. We thus need, in order to completely characterize T_j, either to assume that there is at least two such facets (this is a rather mild assumption), or to endow facets with an *orientation* and to allow two facets to match if and only if they have opposite orientations.

"undefined" decorations appearing only on the network. Indeed, consider a macro-tile of T_{n-1} which corresponds (via the one-to-one correspondence in Def. 3.2) to a tile T_j in P_n: it suffices to endow its tiles as in steps 1–3 of the definition of τ, with T_j being the parent-tile, and with the decoration "undefined" on the facets crossed by the network. This can be iterated up to $P = P_0$, and the third condition of Def. 3.4 ensures that the decorations "undefined" appear only on sort of grids whose cells have bigger and bigger size.

Thus, by making n tend to infinity, one can see T as a τ'-tiling whose "undefined" decorations, if any, form either a star with k infinite branches, or a single biinfinite branch. In the first case, we can replace the central τ'-tile by any τ-tile with k facets, and then the tiles on branches by τ-tiles which carry decorations to infinity. In the second case, we can replace the τ'-tiles by τ-tiles which carry any decoration on the whole branch. In any case, we can thus see P as a τ-tiling.

We thus have the second inclusion $\Lambda_\sigma \subseteq \pi(\Lambda_\tau)$. Both inclusions prove Theorem 1.1.

6. On the number of tiles

Let us conclude this paper by discussing the size $\#\tau$ of the tileset τ defined in the previous section (although its finiteness suffices for Theorem 1.1).

Consider a good combinatorial substitution with r rules. Fix a network as in Def. 3.4 for each of these rules. Let n and m denote the total number of, respectively, tiles and internal facets of the macro-tiles of these r rules. Among these n tiles, let p denote the number of those which lie on a network.

First, the $n-p$ tiles not on the network can be decorated in at most n different ways, according to the parent-index they carry. This yields at most $N_0 = (n-p)n$ τ-tiles. Then, each of the $p-r$ non-central tiles on the network can be decorated in at most $(n-p)n+pnm$ different ways: $n-p$ parent-indices which correspond to tiles not on the network, hence allow at most n pairs of parent/neighbor indices carried on the network, and p parent-indices which correspond to tiles on the network, hence allow at most nm pairs of parent/neighbor indices carried on the network. This yields at most $N_p = (p-r)((n-p)n+pmn)$ τ-tiles. Last, the r central tiles can be decorated in at most as many different ways as there is non-central τ-tiles. Finally, this yields

$$\#\tau \leq (r+1)(N_0+N_p) \leq (r+2)p^2mn.$$

This bound is, for example, about one billion in the Rauzy case.

In order to reduce this huge number of tiles, it is worth noting that, instead of carrying a parent-index through all the tiles of a macro-tile, it suffices to carry it along a *second network* connecting the macro-facets of this macro-tile and intersecting each of the branches of its (first) network (see, *e.g.*, Fig. 6). The "control" described in Step 4 is then performed only on tiles where both networks crosses, while we simply allow any possible pairs of parent/neighbor indices to be carried through the tiles which are only on the first network. If we denote by q the total number of tiles on these new second networks and by c be the total number of crossings between second and first networks, a similar analysis yields

$$\#\tau \leq (r+1)(N_0'+N_q'+N_p'+N_c') \leq (r+2)cp(m+qn),$$

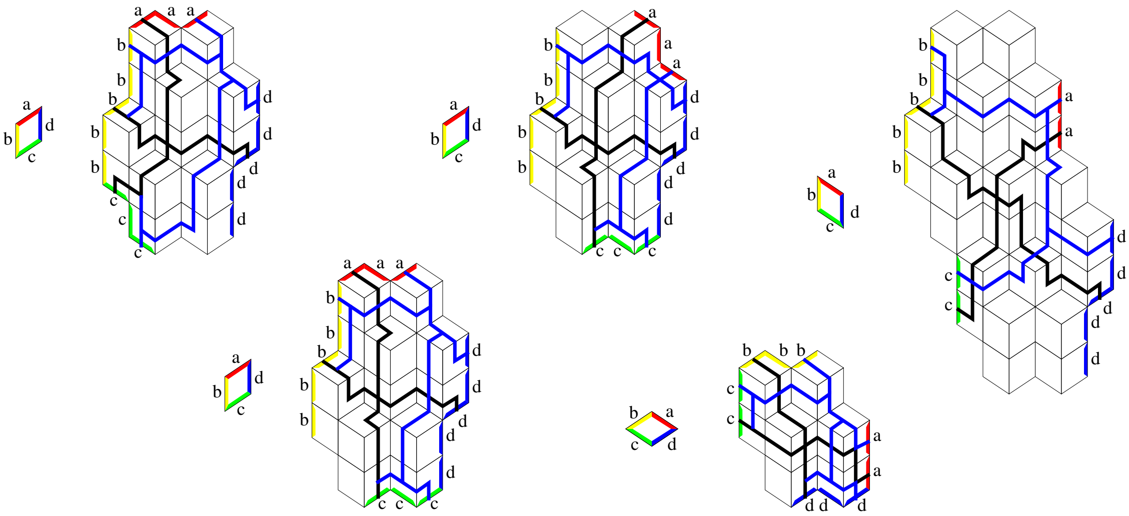


Figure 6: A second network for the Rauzy combinatorial substitution.

where:

$$
\begin{aligned}
N_0' &= n-p-q+c, & N_p' &= (p-c)(m+qn),\\
N_q' &= (q-c)n, & N_c' &= c(N_0'+N_q'+N_p').
\end{aligned}
$$

This bound is, for example, about 70 millions tiles in the Rauzy case.

This last bound is huge but generic. One can hope to dramatically decrease this bound in specific cases. Indeed, most of the τ-tiles correspond to tiles which simply carry any possible information (the tiles on networks, crossings excepted). Since these tiles all play the same role, it would be worth to replace all the tiles on a network by a single tile (one can thus hope to gain a factor pq in the above bound – this would yield about 25000 tiles in the Rauzy case). This shall however be done carefully, so that tiles still necessarily form macro-tiles. Note that it should be much easier in dimension $d \geq 3$, since the cohesion of macro-tiles can be more easily enforced without relying on tiles on networks.

Acknowledgments

We would like to thank the anonymous referees for their valuable comments. The first author also would like to thank N. Pytheas Fogg and C. Goodman-Strauss for their encouragements to write down this result, as well as for useful discussions.

References

[1] P. Arnoux and S. Ito, *Pisot substitutions and Rauzy fractals*, Bull. Belg. Math. Soc. Simon Stevin **8** (2001), no. 2, 181–207.

[2] R. Berger, *The undecidability of the domino problem*, Ph.D. thesis, Harvard University, July 1964.

[3] V. Berthé and Th. Fernique, *Brun expansions of stepped surfaces*, to appear in Disc. Math.

[4] Th. Fernique, *Local rule substitutions and stepped surfaces*, Theor. Comput. Sci. **380** (2007), no. 3, 317–329.

[5] C. Goodman-Strauss, *Matching rules and substitution tilings*, Ann. of Math. (2) **147** (1998), no. 1, 181–223.

[6] S. Mozes, *Tilings, substitution systems and dynamical systems generated by them*, J. Analyse Math. **53** (1989), 139–186.

[7] N. Ollinger, *Two-by-two substitution systems and the undecidability of the domino problem*, Proceedings of CiE'2008, LNCS, vol. 5028, Springer, 2008, pp. 476–485.

[8] N. Priebe Frank, *Detecting combinatorial hierarchy in tilings using derived voronoi tessellations*, Disc. Comput. Geom. **29** (2003), no. 3, 459–467.

[9] R. M. Robinson, *Undecidability and nonperiodicity for tilings of the plane*, Inventiones Mathematicae **12** (1971), 177–209.

Journées Automates Cellulaires 2010 (Turku), pp. 111-120

INFINITE TIME CELLULAR AUTOMATA: A REAL COMPUTATION MODEL

FABIEN GIVORS, GREGORY LAFITTE, AND NICOLAS OLLINGER

Laboratoire d'Informatique Fondamentale de Marseille (LIF), CNRS – Aix-Marseille Université, 39 rue Joliot-Curie, 13453 Marseille Cedex 13, France

Abstract. We define a new transfinite time model of computation, *infinite time cellular automata*. The model is shown to be as powerful than infinite time Turing machines, both on finite and infinite inputs; thus inheriting many of its properties. We then show how to simulate the canonical real computation model, *BSS machines*, with infinite time cellular automata in exactly ω steps.

Introduction

When the second and third authors of this paper were in their PhD years, their common advisor, Jacques Mazoyer, had encouraged them to define a computation model on real numbers using cellular automata. The second author had defined at the end of the Nineties a cellular automata generalization running into transfinite time, but it was never published and was only clumsily defined in his PhD thesis. They thought of using it as a real computation model in 2001 in Riga but never did anything about it since. This paper is a description of these ideas which had stayed for ten years in the state of scribbled notes.

Transfinite time computation models were first considered in 1989 by Hamkins and Kidder. Hamkins and Lewis later developed the model of infinite time Turing machines and its theory in [HL00]. Koepke [Koe06] defined another transfinite time model based on register machines, that was later refined by Koepke and Miller [KM08]. These models differ in their computation power, the infinite time Turing machines model being the most powerful one.

In [BSS89], Blum, Shub and Smale introduced, also in 1989, a model of computation, coined *BSS machines*, intended to describe computations over the real numbers. It can be viewed as Turing machines with tapes whose cells (or Random Access Machines with registers that) can store arbitrary real numbers and that can compute rational functions overs reals at unit cost. Their model, which is more general[1] than the presentation provided in this paper, is a canonical model of real computation. The idea of real computation is to deal with hypothetical computing

The research presented in this paper has been made possible by the support of the French ANR grants *NAFIT* (ANR-08-DEFIS-008-01) and *EMC* (ANR-09-BLAN-0164-01).

[1] It really provides a setting for computing over rather arbitrary structures.

machines using infinite-precision real numbers and to be able to prove results of computations operating on the set of real numbers. A typical example is studying the computability of the Mandelbrot set. It can be viewed as an idealized analog computer which operates on real numbers and is differential, whereas digital computers are limited to integers and are algebraic. For a survey of analog computations, the reader is referred to the survey [BC08].

In this paper, we introduce a transfinite time computation model, the *infinite time cellular automata.* The model is arguably more *natural* and *uniform* than other transfinite time models introduced, for the same reasons cellular automata are more *natural* and *uniform* than Turing machines. There is no head wandering here and there and there is no difference between states and data.

We show that the infinite time cellular automata have the same computing power than infinite time Turing machines, both on finite and infinite inputs. They thus inherit the nice properties of this latter model. We then show how to simulate the BSS machines with infinite time cellular automata in exactly ω steps. We finish by introducing another transfinite time model based on cellular automata.

1. Infinite time cellular automata

Definition 1.1. An *infinite time cellular automaton* (ITCA) A is defined by Σ, the finite set of states of A, linearly ordered by $\prec$ and with a least element $\mathbf{0}$; and $\delta : \Sigma^3 \to \Sigma$, the local rule of A, satisfying $\delta(\mathbf{0}, \mathbf{0}, \mathbf{0}) = \mathbf{0}$, so that $\mathbf{0}$ is a *quiescent* state.

A *configuration* is an element of $\Sigma^{\mathbb{Z}}$. The local rule δ induces a global rule $\Delta : \Sigma^{\mathbb{Z}} \to \Sigma^{\mathbb{Z}}$ on configurations such that $\Delta(C)_i = \delta(C_{i-1}, C_i, C_{i+1})$ for $i \in \mathbb{Z}$ and $C \in \Sigma^{\mathbb{Z}}$.

Starting from a configuration $C \in \Sigma^{\mathbb{Z}}$, the *evolution* of length $\theta \in \mathrm{Ord}$ of A is given by $(\Delta^{\alpha}(C))_{\alpha \leqslant \theta}$:

$$\begin{array}{lcl} \Delta^{\beta+1}(C) & = & \Delta(\Delta^{\beta}(C)) \\ \Delta^{\lambda}(C)_i & = & \liminf^{\prec}_{\gamma<\lambda} \Delta^{\gamma}(C)_i \text{ for all } i \in \mathbb{Z} \text{ and } \lambda \text{ limit} \end{array}$$

Definition 1.2. Let $\mathbf{h} \in \Sigma$ a particular state we will refer to as the *halting* state.

An evolution of length θ is called a *computation* if the state $\mathbf{h}$ appears in the last configuration, $\Delta^{\theta}(C)$, but not before this stage.

We settle a convention on the way we code integers or real numbers in our model. For example, we could code integers and real numbers in binary (using $\mathbf{0}$ and another state as symbols for 0 and 1) on the right cells (cells whose indices belong to $\mathbb{N}$).

Definition 1.3. Let X be a space for which a coding has been settled. (For example, $\mathbb{N}$, $\mathbb{R}$, $2^{\mathbb{N}}$ or $2^{\mathbb{Z}}$.)

A (partial) function F on X, $F : X \to X$, is said to be *infinite time computable* if there is an infinite time cellular automaton such that for each $x \in \mathit{dom}(F)$, there is a computation starting with a configuration with a coding of x that halts on a configuration with a similar coding of $F(x)$.

A set $A \subseteq X$ is *infinite time decidable* if its characteristic function is infinite time computable, and is *infinite time semi-decidable* if it is the domain of an infinite time computable function.

We use the term “semi-decidable” instead of “enumerable”, since contrarily to the classical computability concepts, in the transfinite time context, being semi-decidable is not equivalent to being the range of a computable function.

2. Properties of infinite time cellular automata

2.1. Comparisons with other infinite time models

Hamkins, Kidder and Lewis [HL00] have defined an infinite time Turing machines model and Koepke [Koe06] has defined an infinite time register machines model, that was later refined by Koepke and Miller [KM08].

The *infinite time Turing machines* (ITTM) work as a classical Turing machine with a head reading and writing on a bi-infinite tape, moving left and right in accordance with the instructions of a finite program with finitely many states. At successor stages of computation, the machine operates in exactly the classical manner. At limit stages, the machine enters a special limit state, with the head on the origin cell, and each cell of the tape taking the value of the lim inf of the values appearing in that cell before that limit stage.

The *infinite time register machines* behave like standard register machines at successor stages. At limit times, the register contents are defined using lim inf's of the previous register contents. The difficulty here is that the lim inf does not necessarily exist in that case, since a register can contain arbitrary large integers. The machines of [Koe06] crashed in such a case. Those of [KM08] continue beyond such crashes by resetting a register to 0 whenever it overflows.

The infinite time Turing machines are strictly stronger than infinite time register machines: the halting problem for infinite time register machines can be decided by an ITTM.

Theorem 2.1. *Infinite time cellular automata have the same computing power of infinite time Turing machines.*

Proof. The right to left implication goes as follows:

At successor stages, the simulation of ITTM by ITCA works the same way it does in the non-infinite-time case.

At limit stages, we have to put the configuration of the ITCA in the limit configuration simulation of the ITTM simulated. To do this, we need to be able to know that we are at a limit stage. It suffices to have two adjacent cells of the ITCA at the origin that, at successor stages, alternate between **0** and some other state such that the adjacent states are different. At the next limit stage, they will be both equal to **0**. We also have to use the same trick on the cells visited by the head to make sure that at limit stages, if the ITTM becomes stationary, we can wipe out the stationary state from our simulation tape and enter in the special limit state. The ITCA can then prepare the configuration to continue the ITTM simulation.

The left to right implication: it takes ω steps with an ITTM machine to simulate an ITCA global step (on an infinite input). It is just then a matter of determining whether the ITTM is at a limit stage or not. This is easily achieved by an ITTM since it enters a special limit state at limit stages. ■

2.2. Features of those infinite time models

Hamkins, Kidder, Lewis and Welch [HL00, Wel99, Wel00b, Wel00a] have shown many properties of infinite time Turing machines. By Theorem 2.1, infinite time cellular automata have many of these same properties. We state in the following the properties inherited by infinite time cellular automata.

Theorem 2.2. *The set of reals coding well-orders is infinite time decidable.*

The hyperarithmetic sets are those that are decidable in time less than some recursive ordinal. Every Π^1_1 set is decidable and the class of decidable sets is contained in Δ^1_2.

Definition 2.3. An ordinal α is *clockable* if there is an ITCA computation starting from the all-but-one quiescent configuration C ($C_0 \neq \mathbf{0}$ and $C_i = \mathbf{0}\ \forall i \in \mathbb{Z} \setminus \{0\}$) and that halts after exactly α steps (meaning that the α^{th} configuration, $\Delta^\alpha(C)$, is the first configuration in which the halting state $\mathbf{h}$ appears).

A real r is *writable* if it is the output of an ITCA computation. An ordinal is writable if it is coded by such a real.

There are of course only countably many clockable and writable ordinals, since there are only countably many local rules.

Theorem 2.4. *Every recursive ordinal is clockable. Even $\omega_1^{CK} + \omega$ is clockable. Beyond that, there are many intervals of non-clockable ordinals. The supremum of clockable ordinals is recursively inaccessible*[2]*. Moreover, the writable ordinals however form an initial segment of the ordinals. The supremum of the writable ordinals is the supremum of the clockable ordinals.*

One of the beautiful theorems of ITTMs that carry through to ITCAs is the Lost Melody Theorem. The real constructed in this theorem is like a lost melody that you can recognize when someones hums it to you, but which you cannot sing on your own.

Theorem 2.5 (Lost Melody Theorem)**.** *There is a real r which is recognizable ($\{r\}$ is decidable), but not writable.*

There are different ways to construct such lost melody reals. One way is to consider the supremum γ of the ordinal stages by which an ITTM computation, on an empty input, either halts or repeats. Notice that by a simple cofinality argument, we can show that all these ordinals are countable. There is a smallest ordinal $\delta \geqslant \gamma$ such that $L_{\delta+1} \models$ "δ is countable". $L_{\delta+1}$ has a canonical well-ordering, thus there is some real $r \in L_{\delta+1}$ which is least with respect to the canonical L order, such that r codes δ. γ (and thus r) are somehow a generalization of the busy beaver problem to transfinite time computations. It is then not surprising that r cannot be computable, since that would render the infinite time halting problem decidable. It is recognizable because it is possible to reconstruct the L hierarchy using an ITTM and verify that r is really the least coding of an ordinal having the properties of δ.

[2] A *recursively inaccessible* ordinal is an ordinal that is both admissible and a limit of admissibles. An ordinal α is *admissible* if the construction of the Gödel universe, L, up to a stage α, yields a model L_α of Kripke-Platek set theory. The Church-Kleene ordinal, ω_1^{CK}, is the smallest non-recursive ordinal and is the smallest admissible ordinal. For more on admissibles, see the most excellent book of Barwise [Bar75].

3. Computations on the reals

3.1. Blum-Shub-Smale model

Blum, Shub and Smale [BSS89] introduced the *BSS model*.

A simplified presentation of the BSS model goes through defining "Turing machines with real numbers". We follow Hainry's presentation in his PhD thesis [Hai06].

Definition 3.1. A *simplified BSS machine* (or shortly, BSS machine) is composed of an infinite tape and a program. The infinite tape is made of cells, each containing a real number. We denote the tape by $(x_n)_{n\in\mathbb{Z}} \in \mathbb{R}^{\mathbb{Z}}$. The program is a numbered (finite) sequence of instructions. The number of each instruction in the program sequence is seen as a state ($\in Q$). The instructions are

- *go right*: changes the tape to $(x_{n+1})_{n\in\mathbb{Z}}$;
- *go left*: changes the tape to $(x_{n-1})_{n\in\mathbb{Z}}$;
- *branch if greater than* 0: if the current cell (x_0) is greater than 0, then it branches to a specified location in the program;
- *branch if equal to* 0: if the current cell (x_0) is equal to 0, then it branches to a specified location in the program;
- *make a computation*: the current cell (x_0) is changed to be equal to the result of a computation from x_0, x_1 and possible constants ($k \in \mathbb{R}$). A computation is one of the following:
  - $x_0 \leftarrow -x_0$;
  - $x_0 \leftarrow k$;
  - $x_0 \leftarrow x_0 + x_1$;
  - $x_0 \leftarrow x_0 \times x_1$;
  - $x_0 \leftarrow k \times x_0$.

The machine starts by executing the first instruction (with the least number) of the program and continues by executing the next instruction and so on until it branches. It halts when it has no more instructions to execute.

It is possible to give a definition for a more general BSS machine on rather arbitrary structures. In this paper, we will stick with simplified BSS machines on $\mathbb{R}$.

For a lot more on the *BSS model* and computation on the real numbers, the reader is referred to the book by Blum, Cucker, Shub and Smale [BCSS98].

3.2. BSS by ω-ITCA

We show how to simulate a simplified BSS machine with an ITCA.

Theorem 3.2. *A simplified BSS machine can be simulated by an ITCA in ω steps.*

Proof. Consider a BSS machine. At each time step t of a computation, only a finite number of non-zero real numbers are defined: k constants inside the program and $l \leqslant t$ cells of the tape. For the sake of clarity, suppose that the BSS machine works on reals in the interval $[-1, 1]$[3]. Imagine that we encode these $k + l$ reals as

[3] The construction extends to $\mathbb{R}$ at the cost of non significant tricks, for example by adding just after the bit of sign the encoding of the integer part of each real on a same number of bits followed by a dot.

infinite words on the alphabet $\{0, 1\}$ encoding a bit of sign followed by their binary expansion.

Each computation instruction of the BSS machine has the nice property that it can be computed, in time ω, by a Turing machine working synchronously on the representation of its operands. For each finite initial portion of the tape, it is left untouched by such a machine after some finite time. Moreover, at the cost of some extra bookkeeping on the tape, such a computation can be achieved in a reversible way[4].

Each branch instruction of the BSS machine can be achieved in a similar way: one can choose a initial hypothesis on the branch[5] and start a computation by a Turing machine working synchronously on the representation of the operands ; either the machine eventually halts contradicting the hypothesis, or its head moves infinitely towards the end of the infinite words. For each finite initial portion of the tape, it is left untouched by such a machine after some finite time and the computation can be achieved in a reversible way.

Packing it all together, we can simulate a BSS machine if we can launch as many Turing computation threads as needed. Encode the $k + l$ reals encodings as a single infinite word and put some finite control at the beginning of it. The finite control plays the role of the head of the BSS machine, keeping the current state (instruction number). The control is responsible for launching the next instruction: each time there is enough room on its right, it starts a new thread for the current instruction. If the instruction is a move instruction, the thread simply selects the next cell, adding a new 0 real to the tuple if necessary. If the instruction is a branch instruction, the control takes the default hypothesis on the result and launches the branch thread described before. If the instruction is a computation instruction, the control launches the thread described above. At each time step, the control is pointing to a current instruction and a finite number of threads are computing on its right. Each thread works on a finite portion of tape where it should have exclusive access. Somewhere on its right is the last point where he modified the reals. When a thread wants to access a portion of the reals already modified by the next thread on its left, it enforce this thread to undo its computation to restore the reals as they were before. A cascade of undoing occurs in such a situation, each thread undoing the next thread. At some point an undone thread reaches the control and forces the control part to start again from the instruction where this thread was started, removing the thread from the computing area. As each thread eventually goes to infinity, such an inefficient compute/uncompute dance converges. The same applies when a branch thread discovers that its hypothesis was wrong: it goes back to control to take the other branch of computation, forcing threads newer than him to undo. Notice that a key point of the construction is that there is always enough room for bookkeeping: if a thread needs more space, it just can wait for more space to be available before continuing its computation, either it will eventually happen, or a backtrack will occur that can be handled.

The result of a computation is obtained easily: at time ω, if the control eventually converged to an accepting state, the control part encodes an accepting state and the encoded reals contain the result of the computation ; if the BSS machine did not converge, the control part does not encode an accepting state.

[4] Just store the non injective choices on a stack that the head pushes in front of itself.

[5] For branch if equal to zero the hypothesis is that $x_0 = 0$.

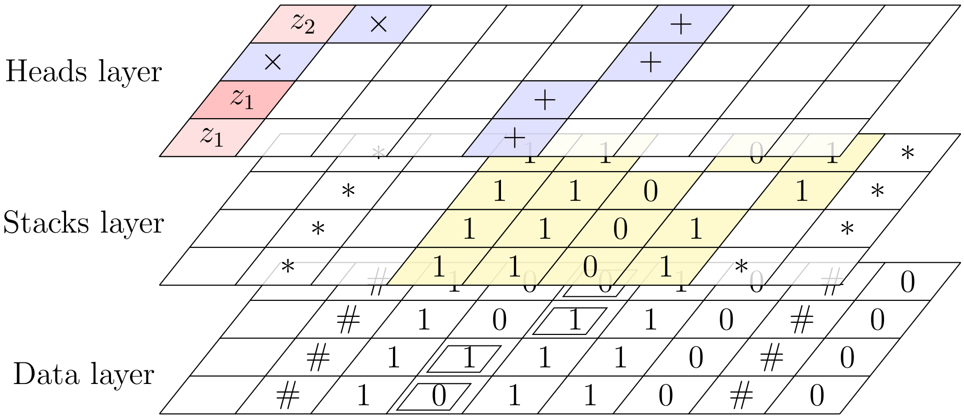


FIGURE 1. Layers encoding computation

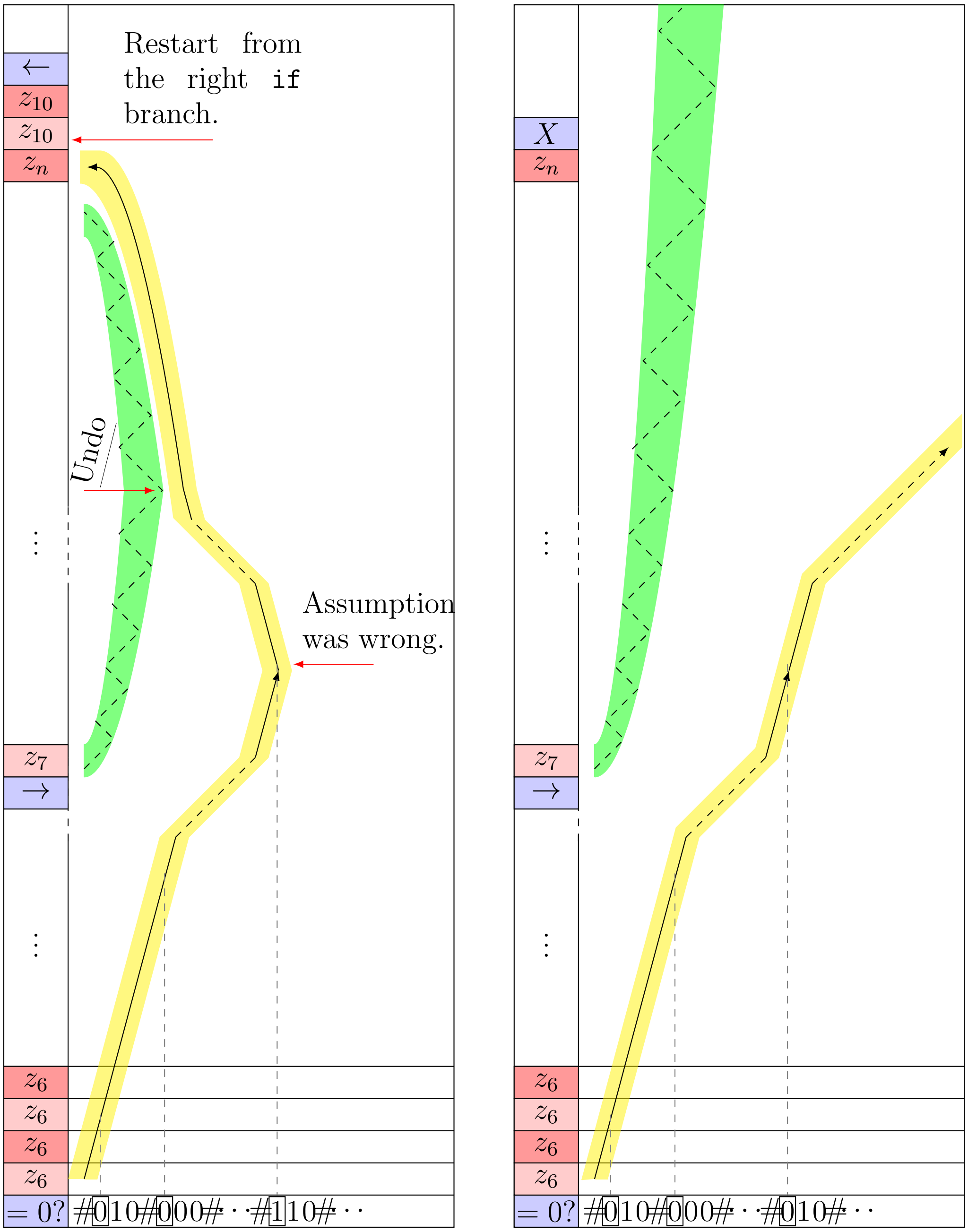
Let us now explain how the described simulation can be carried on a ITCA. Given a BSS machine, the ITCA is constructed as follows. Only a semi-infinite part of the configuration is used. Cell #0 encodes control and the cells on its right encode the reals, the computation area. The computation area is constructed in 3 layers, as depicted on figure 1. The data layer encodes the bits of real numbers and the current position of the head: it is divided into blocks of length $k+l$, separated by # border symbols, containing a bit for each real, the current position of the head being circled. The stack layer encodes the working area for the threads, where each head has its computing stack, and the boundaries of the threads: each thread delimits monotonically the area it already modified by a $*$ symbol. The head layer encodes the heads of the threads, the active parts of the ITCA.

Each instruction of the BSS machine is simulated as explained before. A move thread simply moves the circle to the previous or next bit, adding a new real if necessary, using its thread stack. A branch thread does not modify reals but checks if the hypothesis was true or false, as depicted on figure 2. A computation thread modifies the reals, making choices (for example the values of carries when adding two reals), backtracking when the choice was a bad one, as depicted for addition on figure 3a. Notice that multiplication can be significantly simplified by the fact that we can create new threads, thus it can be decomposed into additions launched by a master thread, as depicted on figure 3b.

It is important to notice that the monotonicity of the forward movement of $*$ symbols ensures that there is no concurrency problem due to neighbor threads backtracking and coming back forward in a same area: when a thread wants to access an area already explored by its follower, the follower is asked to undo its computation. To ask a neighbor to undo, a thread simply modifies its neighbor $*$ symbol to inform him.

The details of the construction use rather classical but tedious CA encoding tricks. The key argument of the proof is that an ITCA can simulate in time ω the work of an unbounded number of Turing heads, thus achieving the same quantity of work than a ITTM in time ω^2. ■

By diagonalization, it is easy to see that there are functions on the real numbers, computable by ITCAs in ω steps but that are not computable by BSS machines.



(A) Case $x_0 \neq 0$

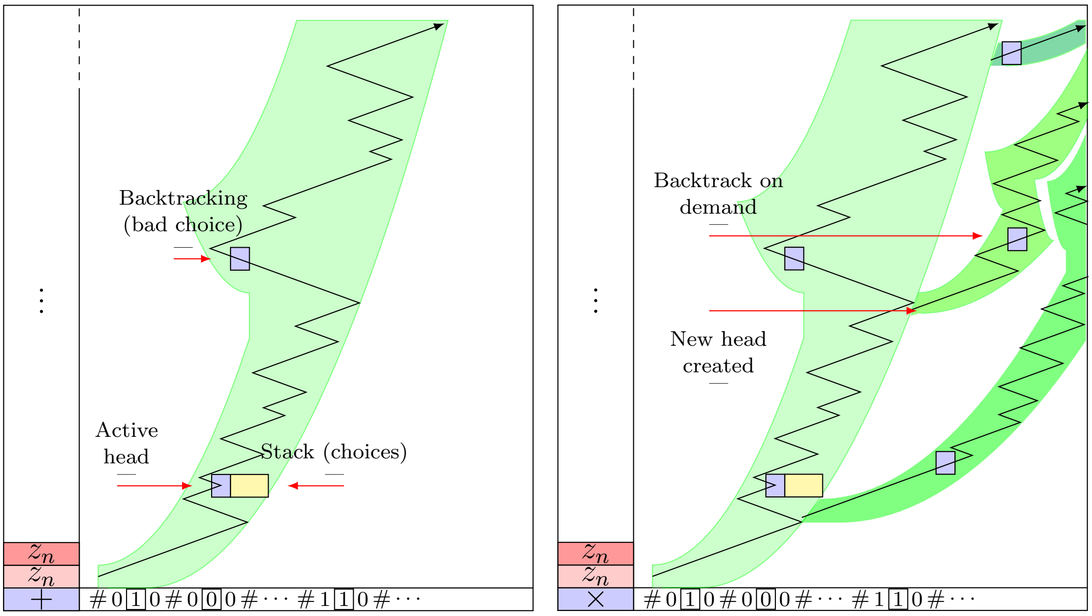
(B) Case $x_0 = 0$

FIGURE 2. *Branch if equal to 0* thread

Concluding remarks

We finish this paper by pointing in a direction that we believe to be promising.

Koepke [Koe05] has defined a transfinite time computation model based on Turing machines that has transfinite space. It can thus compute on arbitrary ordinals and sets of ordinals. Koepke and Siders [KS08] have also extended the infinite time register machines model to machines with registers containing arbitrary ordinals. These models make it possible to compute the bounded truth predicate of the constructible universe, $\{(\alpha, \varphi, \vec{x}) : \alpha \in \mathrm{Ord}, \varphi \text{ an } \in\text{-formula}, \vec{x} \in L_\alpha, L_\alpha \models \varphi(\vec{x})\}$, and



(A) Addition thread

(B) Multiplication thread

Figure 3. Computation threads

allows the following characterization of computable sets of ordinals: a set of ordinals is ordinal computable from of a finite set of ordinal parameters if and only if it an element of Gödel's constructible universe L. Ordinal computability is moreover interesting to be able to reprove some facts about L.

We propose the following definition for ordinal computations over cellular automata.

Definition 3.3. An *ordinal cellular automaton* A is defined by Σ, the finite set of states of A, linearly ordered by $\prec$ and with a least element $\mathbf{0}$; and $\delta : \Sigma^3 \to \Sigma$, the local rule of A, satisfying $\delta(\mathbf{0},\mathbf{0},\mathbf{0}) = \mathbf{0}$, so that $\mathbf{0}$ is a *quiescent* state.

A *configuration* of length ξ is an element of Σ^ξ. The local rule δ induces on configurations of length ξ a global rule $\Delta : \Sigma^\xi \to \Sigma^\xi$ such that

$$\begin{cases} \Delta(C)_0 = \delta(\liminf^{\prec}_{\gamma<\xi} C_\gamma, C_0, C_1), \\ \Delta(C)_{\beta+1} = \delta(C_\beta, C_{\beta+1}, C_{\beta+2}), \\ \Delta(C)_\lambda = \delta(\liminf^{\prec}_{\gamma<\lambda} C_\gamma, C_\lambda, C_{\lambda+1}) \quad \text{if } \lambda \text{ limit and } \lambda > 0. \end{cases}$$

Starting from a configuration $C \in \Sigma^\xi$, the *evolution* of length θ of A is given by $(\Delta^\alpha(C))_{\alpha\leqslant\theta}$:

$$\begin{array}{lcl} \Delta^{\beta+1}(C) & = & \Delta(\Delta^\beta(C)) \\ \Delta^\lambda(C)_\iota & = & \liminf^{\prec}_{\gamma<\lambda} \Delta^\gamma(C)_\iota \text{ for all } \iota < \xi \text{ and } \lambda \text{ limit} \end{array}$$

We think that it would be interesting to try to carry the results of the other ordinal machines models over to this ordinal cellular automata model. In this model, there is the limitation due to the fixed length of configurations, which is imposed by the cylindrical nature of our model. There are certainly other ways to allow information to flow from the right to the left of the configurations (and not only left to right).

References

[Bar75] Jon Barwise. *Admissible Sets and Structures: An Approach to Definability Theory*, volume 7 of *Perspectives in Mathematical Logic*. Springer Verlag, 1975.

[BC08] Olivier Bournez and Manuel L. Campagnolo. *New Computational Paradigms. Changing Conceptions of What is Computable*, chapter A Survey on Continuous Time Computations, pages 383–423. Springer-Verlag, New York, 2008.

[BCSS98] Lenore Blum, Felipe Cucker, Mike Shub, and Steve Smale. *Complexity and real computation.* Springer, 1998.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: Np-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, 1989.

[Hai06] Emmanuel Hainry. *Modèles de calcul sur les réels: résultats de comparaisons.* PhD thesis, Institut National Polytechnique de Lorraine, 2006.

[HL00] Joel David Hamkins and Andy Lewis. Infinite time turing machines. *Journal of Symbolic Logic*, 65(2):567–604, 2000.

[KM08] Peter Koepke and Russell Miller. An enhanced theory of infinite time register machines. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *Logic and Theory of Algorithms, 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008, Proceedings*, volume 5028 of *Lecture Notes in Computer Science*, pages 306–315. Springer, 2008.

[Koe05] Peter Koepke. Turing computations on ordinals. *Bulletin of Symbolic Logic*, 11:377–397, 2005.

[Koe06] Peter Koepke. Infinite time register machines. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006, Proceedings*, volume 3988 of *Lecture Notes in Computer Science*, pages 257–266. Springer, 2006.

[KS08] Peter Koepke and Ryan Siders. Register computations on ordinals. *Archive for Mathematical Logic*, 47:529–548, 2008.

[Wel99] Philip D. Welch. Minimality arguments in the infinite time turing degrees. In S. B. Cooper and J. K. Truss, editors, *Sets and Proofs, Proceedings of the ASL European Meeting, Logic Colloquium 1997*, volume 258 of *London Mathematical Society Lecture Notes in Mathematics*, pages 425–436. Cambridge University Press, April 1999.

[Wel00a] Philip D. Welch. Eventually infinite time turing machine degrees: Infinite time decidable reals. *Journal of Symbolic Logic*, 65:1193–1203, 2000.

[Wel00b] Philip D. Welch. The lengths of infinite time turing machine computations. *Bulletin of the London Mathematica Society*, 32:129–136, 2000.

Journées Automates Cellulaires 2010 (Turku), pp. 121-132

COMPUTATIONAL COMPLEXITY OF AVALANCHES IN THE KADANOFF TWO-DIMENSIONAL SANDPILE MODEL

ERIC GOLES [1] AND BRUNO MARTIN [2]

[1] Universidad Adolfo Ibañez, Av. Diagonal Las Torres 2640, Peñalolen, Santiago, Chile
E-mail address: eric.chacc@uai.cl

[2] Université de Nice Sophia-Antipolis, I3S, UMR 6070 CNRS, 2000 route des Lucioles, BP 121, F-06903 Sophia Antipolis Cedex, France
E-mail address: Bruno.Martin@unice.fr

Abstract. In this paper we prove that the *avalanche problem* for the Kadanoff sandpile model (KSPM) is P-complete for two-dimensions. Our proof is based on a reduction from the monotone circuit value problem by building logic gates and wires which work with configurations in KSPM. The proof is also related to the known *prediction problem* for sandpile which is in NC for one-dimensional sandpiles and is P-complete for dimension 3 or greater. The computational complexity of the prediction problem remains open for two-dimensional sandpiles.

1. Introduction

Predicting the behavior of discrete dynamical systems is, in general, both the “most wanted” and the hardest task. Moreover, the difficulty is still hard when considering finite phase spaces. Indeed, when the system is not solvable, numerical simulation is the only possibility to compute future states of the system.

In this paper we consider the well-known discrete dynamical system of sandpiles (SPM). Roughly speaking, its dynamics is as follows. Consider the toppling of grains of sand on a (clean) flat surface, one by one. After a while, a sandpile has formed. At this point, the simple addition of even a single grain may cause avalanches of grains to fall down along the sides of the sandpile. Then, the growth process of the sandpile starts again. Remark that this process can be naturally extended to arbitrary dimensions although for $d > 3$, the physical meaning is not clear.

The first complexity results about SPM appeared in [6, 7] where the authors proved the computation universality of SPM. For that, they modelled wires and logic gates with sandpiles configurations. Inspired by these constructions, C. Moore and M. Nilsson considered the *prediction problem* (PRED) for SPM *i.e.* the problem of computing the stable configuration (fixed point) starting from a given initial configuration of the sandpile. C. Moore and M. Nilsson proved that PRED is in NC^3 for dimension 1 and that it is P-complete for $d \geq 3$ leaving $d = 2$ as an open

Work supported by the french ANR programme “*Émergence dans les modèles de calcul*” (B.M.); by FONDECYT 1100003, BASAL-CMM-U. de Chile, Anillo ACT-88, SFI-Santa Fe, CNRS (E.G.).

problem [12]. (Recall that P-completeness plays for parallel computation a role comparable to NP-completeness for non-deterministic computation. It corresponds to problems which cannot be solved efficiently in parallel (see [9]) or, equivalently, which are *inherently sequential*, unless $\mathsf{P} = \mathsf{NC}$). Later, P.B. Miltersen improved the bound for $d = 1$ showing that PRED is in $\mathsf{LOGDCFL}$ ($\subseteq \mathsf{AC}^1$) and that it is not in $\mathsf{AC}^{1-\varepsilon}$ for any $\varepsilon > 0$ [11]. Therefore, in any case, one-dimensional sandpiles are capable of (very) elementary computations such as computing the max of n bits.

Both C. Moore and P.B. Miltersen underline that *"having a better upper-bound than P for PRED for two-dimensional sandpiles would be most interesting."*

In this paper, we address a slightly different problem: the avalanche problem (AP). Here, we start with a monotone configuration of the sandpile. We add a grain of sand to the initial pile. This eventually causes an avalanche (a sequence of topples) and we address the question of the complexity of deciding whether a certain given position –initially with no grain of sand– will receive some grains in the future. Like for the (PRED) problem, (AP) can be formulated in higher dimensions. In order to get acquainted with AP, we introduce its one-dimensional version first.

One-dimensional sandpiles can be conveniently represented by a finite sequence of integers $x_1, x_2, \ldots, \ldots, x_n$. The sandpiles are represented as a sequence of *piles* and each x_i represents the number of grains contained in pile i. In the classical SPM, a grain falls from pile i to $i+1$ if and only if the height difference $x_i - x_{i+1} \geq 2$. Kadanoff's sandpile model (KSPM) generalises SPM [10, 5] by adding a parameter p. The setting is the same except for the local rule: one grain falls to the $p-1$ adjacent piles if the difference between pile i and $i+1$ is greater than p.

Assume $x_k = 0$, for a value of k "far away" from the sandpile. The avalanche problem asks whether adding a grain at pile x_1 will cause an avalanche such that at some point in the future $x_k \geq 1$, that is to say that an avalanche is triggered and reaches the "flat" surface at the bottom.

This problem can be generalized for two-dimensional sandpiles and is related to the question addressed by C. Moore and P.B. Miltersen.

In this paper we prove that in the two-dimensional case, AP is P-complete. The proof is obtained by reduction from the Circuit Value Problem where the circuit only contains monotone gates — that is, AND's and OR's (see Section 3 for details).

We stress that our proof for the two-dimensional case needs some further hypothesis/constraints for monotonicity and determinism (see Section 3). If both properties are technical requirements for the proof's sake, monotonicity also has a physical justification. Indeed, if KSPM is used for modelling real physical sandpiles, then the image of a monotone non-increasing configuration has to be monotone non-increasing since gravity is the only force considered here. We have chosen to design the Kadanoff dynamics for $d = 2$ by considering a certain definition of the three-dimensional sandpile which does not correspond to the one of Bak's *et al.* in [1]. This hypothesis is not restrictive. It is just used for constructing the transition rules. Bak's construction was done similarly. Nevertheless, our result depends on the way the three dimensional sandpile is modelled. In our case, we have decided to formalise the sandpile as a monotone decreasing pile in three dimensions where $x_{i,j} \geq \max\{x_{i+1,j}, x_{i,j+1}\}$ (here $x_{i,j}$ denotes the sand grains initial distribution) together with Kadanoff's avalanche dynamics ruled by parameter p. The pile (i,j) can give a grain either to every pile $(i+1,j), \ldots, (i+p-1,j)$ or to every pile $(i,j+1), \ldots, (i,j+p-1)$ if the monotonicity is not violated. With such a rule and if we use the height difference for defining the monotonicity, we can define the transition rules of the dynamics for every value of the parameter p.

In the case where the value of the parameter p equals 2, we find in our definition of monotonicity something similar with Bak's SPM in two dimensions. Actually, both models are different because the definitions of the three dimensional piles differ. That is the reason why we succeed in proving the P-completeness result which remains an open problem with Bak's definition.

The paper is organized as follows. Section 2 introduces the definitions of the Kadanoff sandpile model in one dimension and presents the avalanche problem. Section 3 generalizes the Kadanoff sandpile model in two dimensions and presents the avalanche problem in two dimension, which is proved P-complete for any value of the Kadanoff parameter p. Finally, Section 4 concludes the paper and proposes further research directions.

2. Sandpiles and Kadanoff model in one dimension

A sandpile *configuration* is a distribution of sand grains over a lattice (here $\mathbb{Z}$). Each pile of the lattice is associated with an integer which represents its sand content. A finite configuration on $\mathbb{Z}$ can be identified with an ordered sequence of integers ${}^{\omega}x_1, x_2, \ldots, x_n^{\omega}$ in which x_1 (resp. x_n) is the first (resp. the last) pile such that all the piles on the left of x_1 equal x_1 (resp. all the piles on the right of x_n equal x_n). Given a configuration x, $a \in \mathbb{N}$ and $j \in \mathbb{Z}$, we use the notation ${}^{\omega}ax_j$ (resp. x_ja^{ω}) to say that $\forall i \in \mathbb{Z}$, $i < j \Rightarrow x_i = a$ (resp. $\forall i \in \mathbb{Z}$, $i > j \Rightarrow x_i = a$).

Remark that any configuration $x \equiv {}^{\omega}x_1, x_2, \ldots, x_{n-1}, x_n^{\omega}$ can be identified with its *heights differences* sequence

$$ {}^{\omega}0, h_1 = (x_1 - x_2), \ldots, h_{n-1} = (x_{n-1} - x_n), 0^{\omega} \ , $$

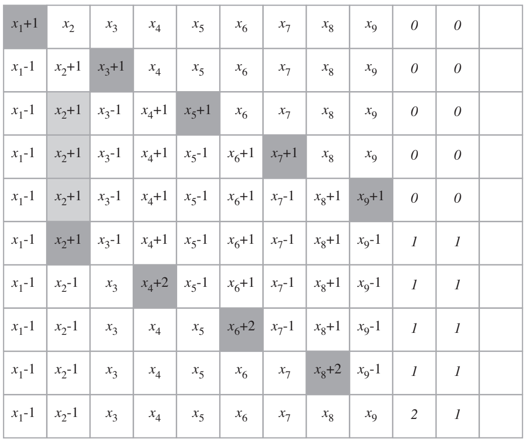
n will be referred to as the *length* of the configuration and it is denoted $|x|$. In other words, we associate the initial (infinite) configuration with a finite sequence of integers $h_1, h_2, \ldots, h_{|x|-1}$. This latter representation is more convenient and is widely used in the sequel. A configuration is *finite* if only a finite number of its heights differences sequence has non-zero sand content.

A configuration x is *monotone* if the sequence of its heights differences is monotone *i.e.* $\forall i \in \{1, 2, \ldots, |x| - 1\}$, $h_i \geq 0$. A monotone configuration x is *stable* if the sequence of its heights differences is stable, *i.e.* $\forall i \in \{1, 2, \ldots, |x| - 1\}$, $h_i < p$ *i.e.* if the difference between any two adjacent piles is less than Kadanoff's parameter p. Let SM(n) denote the set of stable monotone configurations of the form ${}^{\omega}x_1, x_2, \ldots, x_{n-1}, x_n^{\omega}$ and of length n, for $x_i \in \mathbb{N}$.

Consider a stable monotone configuration ${}^{\omega}x_1, x_2, \ldots, x_n^{\omega}$. Adding one more sand grain, say at pile i, may cause that the pile i topples some grains to its adjacent piles. In their turn the adjacent piles receive a new grain and may also topple, and so on. This phenomenon is called an *avalanche* which ends when the system evolves to a new stable configuration.

In this paper, topplings are controlled by the *Kadanoff's parameter* $p \in \mathbb{N}, p \geq 2$ which completely determines the model and its dynamics. In KSPM(p), $p-1$ grains will fall from pile i if $h_i = (x_i - x_{i+1}) \geq p$ and the new configuration becomes

$$ {}^{\omega}x_1 \cdots (x_{i-1})(x_i - p + 1)(x_{i+1} + 1) \cdots (x_{i+p-2} + 1)(x_{i+p-1} + 1)(x_{i+p}) \cdots x_n 0^{\omega} \ . $$



x_1+1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	0	0	
x_1-1	x_2+1	x_3+1	x_4	x_5	x_6	x_7	x_8	x_9	0	0	
x_1-1	x_2+1	x_3-1	x_4+1	x_5+1	x_6	x_7	x_8	x_9	0	0	
x_1-1	x_2+1	x_3-1	x_4+1	x_5-1	x_6+1	x_7+1	x_8	x_9	0	0	
x_1-1	x_2+1	x_3-1	x_4+1	x_5-1	x_6+1	x_7-1	x_8+1	x_9+1	0	0	
x_1-1	x_2+1	x_3-1	x_4+1	x_5-1	x_6+1	x_7-1	x_8+1	x_9-1	1	1	
x_1-1	x_2-1	x_3	x_4+2	x_5-1	x_6+1	x_7-1	x_8+1	x_9-1	1	1	
x_1-1	x_2-1	x_3	x_4	x_5	x_6+2	x_7-1	x_8+1	x_9-1	1	1	
x_1-1	x_2-1	x_3	x_4	x_5	x_6	x_7	x_8+2	x_9-1	1	1	
x_1-1	x_2-1	x_3	x_4	x_5	x_6	x_7	x_8	x_9	2	1	

Figure 1: Avalanches for $p = 3$ with 9 piles. Here, $x_i + 1$ (resp. $x_i + 2$) indicates that pile i has received some grains once (resp. twice), $x_i - 1$ that pile i has given some grains according to the dynamics; a dark shaded pile indicates the toppling pile, a light shaded pile indicates a pile that could topple in the future. Times goes top-down.

In other words, the pile i distributes one grain to each of its $(p-1)$ right adjacent piles. Equivalently, if we measure the heights differences after applying the dynamics, we get $(h_{i-1}+p-1)(h_i-p)(h_{i+1})(h_{i+2})\cdots(h_{i+p-2})(h_{i+p-1}+1)$, and all remaining heights do not change. In other words, the height difference h_i gives raise to an increase of $(p-1)$ grains of sand to height h_{i-1}, a decrease of p grains to height h_i and an increase of one grain to height h_{i+p-1}.

We consider the problem of deciding whether some pile on the right of pile x_n (more precisely for x_k for $n < k \leq n+p-1$) will receive some grains according to the Kadanoff's dynamics. Since the initial configuration is stable, it is not difficult to prove that avalanches will reach at most the pile $n+p-1$ (see Fig. 1 for example).

Remark that given a configuration, several piles could topple at the same time. Therefore, at each time step, one might have to decide which pile or piles are allowed to topple. According to the update policy chosen, there might be different images of the same configuration. However, it is known [8] that for any given initial number of sand grains n, the orbit graph is a lattice and hence, for our purposes, we may only consider one decision problem to formalize AP:

Problem AP

Instance: A configuration $x \in \mathrm{SM}(n)$ and $k \in \mathbb{N}$ s.t. $n < k \leq n+p-1$.
Question: Does there exist an avalanche such that $x_k \geq 1$?

Let us consider some examples. Let $p = 3$ and consider a stable configuration whose height differences are as follows ${}^{\omega}00\underline{2}2022120000^{\omega}$. We add a single grain at x_1 (underlined in the configuration). Then, the next step should probably be ${}^{\omega}02\underline{0}21222120000^{\omega}$. In one step we see that no avalanche can be triggered, hence the answer to AP is negative. As a second example, consider the following sequence of height differences (always with $p = 3$): ${}^{\omega}0\underline{3}122122221201200^{\omega}$. There are several possibilities for avalanches from the left to the right but none of them arrives to the rightmost 0's region. So the answer to the decision problem is still negative. To get an idea of what happens for a positive instance of the problem, consider the initial configuration: ${}^{\omega}0\underline{3}12222100^{\omega}$ with $p = 3$.

The full proof of Theorem 2.1 is a bit technical and will only be sketched here.

Theorem 2.1. *AP is in* $\mathbf{NC}^1$ *for* KSPM *in dimension* 1 *and* $p > 1$.

Sketch of the proof. The first step is to prove that, in this situation, the Kadanoff's rule can only be applied once at each pile for any initial monotone stable configuration. Using this result one can see that a pile k such that $h_k = 0$ in the initial configuration and $h_k > 1$ in the final one, must have received grains from pile $k - p$. This pile, in its turn, must have received grains from $k - 2p$ and so on until a "firing" pile i with $i \in [\![1, p-1]\!]$. The height difference for all of these piles must be $p - 1$. The existence of this sequence and the values of the height differences can be checked by a parallel iterative algorithm on a PRAM in time $\mathcal{O}(\log n)$. ■

3. Sandpiles and Kadanoff model in two dimensions

There are several possibilities to extend the Kadanoff dynamics to two-dimensional sandpiles. We first generalise the definitions introduced in Section 2.

A two-dimensional sandpile *configuration* is a distribution of sand grains over the $\mathbb{N} \times \mathbb{N}$ lattice. Therefore, a configuration on $\mathbb{N} \times \mathbb{N}$ will be identified by a mapping from $\mathbb{N} \times \mathbb{N}$ into $\mathbb{N}$, giving a number of grains of sand to every position in the lattice. Thus, a configuration will be denoted by $x_{i,j}$ as $(i,j) \mapsto \mathbb{N}$. A configuration x is *monotone* if $\forall i, j \in \mathbb{N} \times \mathbb{N}$, $x_{i,j}$ is such that $x_{i,j} \geq 0$ and $x_{i,j} \geq \max\{x_{i+1,j}, x_{i,j+1}\}$. So we get a monotone sandpile, in the same sense as in [2]. Example 3.1 illustrates the case which violates the condition of monotonicity of the Kadanoff dynamics.

Example 3.1. Consider the initial configuration given in the bottom left matrix

$$\begin{tikzcd}
\begin{matrix} 0 & 1 & 0 & 0 \\ \boxed{2} & 3 & 0 & 0 \\ 8 & 4 & 2 & 2 \\ 8 & 4 & 3 & 2 \end{matrix} & \\
\begin{matrix} 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 8 & \boxed{6} & 2 & 2 \\ 8 & 4 & 3 & 2 \end{matrix} \arrow[u, "v"'] \arrow[r, "h"] & \begin{matrix} 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 8 & 4 & 3 & \boxed{3} \\ 8 & 4 & 3 & 2 \end{matrix}
\end{tikzcd}$$

Values count for the number of grains of a pile. We see that we cannot apply the Kadanoff's dynamics for a value of parameter $p = 3$ from the square boxed pile. Indeed, the resulting configurations do not remain monotone neither by applying the dynamics horizontally nor vertically (resp. $\uparrow v$ and $\xrightarrow{h}$). A pile which violates the condition has been highlighted by an oval box in the resulting configurations (it might be not unique). ■

Any configuration can be identified by the mapping of its *horizontal heights differences* (resp. vertical): $h_{\rightarrow} : (i,j) \mapsto x_{i,j} - x_{i+1,j}$ (resp. $h_{\uparrow} : (i,j) \mapsto x_{i,j} - x_{i,j+1}$). A configuration is *finite* if only a finite number of its heights differences matrices has non-zero sand content. A monotone configuration x is *horizontally stable* (resp. *vertically*) if $\forall i, j \in \mathbb{N} \times \mathbb{N}$, $h_{\rightarrow}(i,j) = x_{i,j} - x_{i+1,j} < p$ (resp. $\forall i, j \in \mathbb{N} \times \mathbb{N}$, $h_{\uparrow}(i,j) = x_{i,j} - x_{i,j+1} < p$) and is *stable* if both horizontally and vertically stable.

In other words, it is a generalisation of the Kadanoff model in one dimension, which requires the configuration to be stable if the difference between any two adjacent piles is less than the Kadanoff parameter p. To this configuration, we apply the Kadanoff dynamics for a given integer $p \geq 1$. This can be done if and only if the new configuration remains monotone. Said differently, prior its application the dynamics requires to test if the local application gives a non-negative configuration.

The Kadanoff dynamics applied to pile (i, j) for a given p consists in giving a grain of sand to any pile in the horizontal or vertical line, *i.e.* $\{(i, j + 1), ...(i, j + p - 1)\}$ or $\{(i + 1, j), ...(i + p - 1, j)\}$. Notice that when considering the dynamics defined over height differences, we work with a different lattice though isomorphic to the initial one. Fig. 2 explains how the dark pile with coordinates (i, j) with a height difference of p gives grains either horizontally (Fig. 2 left) or vertically (Fig. 2 right). Fig. 2 also depicts the relationship between the sandpiles lattice and the heights differences lattice. The local dynamics depicted by Fig. 2 will be called *Chenilles* (horizontal and vertical, respectively).

For a better understanding of the dynamics, recall that in one dimension an avalanche at pile i changes the heights of piles $i-1, i$ and $i+p-1$. In two dimensions, there are height changes on the line but also to both sides of it. The dynamics is simpler to depict than to write down formally. An example of the Kadanoff's dynamics applied horizontally (resp. vertically) is given in Fig. 3. More precisely, the Kadanoff's dynamics for a value of parameter $p = 4$ is depicted in Fig. 4. Observe that we do not need to take into account the number of grains of sand in the piles. It suffices to take the graph of the edges adjacent to each pile (depicted by thick lines) and to store the height differences. So, from now on, we will restrict ourselves to the lattice and to the dynamics defined over the height differences. In Fig. 4, we only keep the information required for applying the dynamics in the simplified view.

Example 3.2 (Obtaining Bak's)**.** In the case $p = 2$ and if we assume the real sandpile is defined as in [2] (*i.e.* $x_{i,j} \geq \max\{x_{i+1,j}, x_{i,j+1}\}$), we get the templates from Fig. 5. ■

3.1. P-completeness

Changing from dimension 1 to 2 (or greater), the statement of AP has to be adapted. Consider a finite configuration x which is non-zero for piles (i, j) with $i, j \geq 0$, stable and monotone and let Q be the sum of the height differences. Let us denote by n the maximum index of non-zero height differences along both axes. Then, SM(n) denotes the set of monotone stable configurations of the form given by a lower-triangular matrix of size $n \times n$ (a matrix where the entries above the main diagonal are zero). To generalise the avalanche problem in two dimensions, we have to find a generic position which is far enough from the initial sandpile but close enough to be attained. To get rough bounds, the following approach was followed. For the upper bound, the worst case occurs when all the grains are arranged on a single pile (with Q as a height difference) which is at an end of one of the axes -at distance n from the origin- and they fall down. For the lower bound, the pile containing the grains is at the origin and the grains fall along the main diagonal. Thus, our decision problem can be restated:

Problem AP (dimension 2)

Instance: A configuration $x \in \mathrm{SM}(n)$, $(k, \ell) \in \mathbb{N} \times \mathbb{N}$ such that $x_{k,\ell} = 0$ and $\frac{\sqrt{2}}{2}n \leq \|(k, \ell)\| \leq n + Q$ (with Q the sum of the height differences and $\|.\|$ the standard Euclidean norm).

Question: Does there exist an avalanche (obtained by using the vertical and horizontal chenilles) such that $x_{k,\ell} \geq 1$?

To prove the P-completeness of AP we proceed by reduction from the monotone circuit value problem MCVP, proved P-complete under many-one NC^1 reduction [9, Theorem 6.2.2]. Observe that the original proof [4] uses a logspace reduction but it should be noted that any logspace reduction is also a NC many-one reduction [9, page 54]. MCVP statement is: given the standard encoding of a Boolean circuit (which ensures a topological numbering of the gates) with n inputs $\{\alpha_1, ..., \alpha_n\}$, a designated output β and logic gates AND, OR we want to decide the truth of the output value β on binary input $\{\alpha_1, ..., \alpha_n\}$ [9, page 122]. Wlog., we also assume that each gate of the circuit has a fan in of two and fan out of at most two and that the gates are laid out in levels with connections only going to adjacent levels. The problem remains P-complete with these restrictions [4]. For the reduction, we have to construct, by using sandpile configurations, wires and turning the signal on the grid (Fig. 7), logic AND gates (Fig. 8 (Right)), logic OR gates (Fig. 9 (Left)), cross-overs (Fig. 8 (Left)) and signal multipliers for starting the process (Fig. 9 (Right)) and eventually doubling the output of a gate. We also need to define a way to deterministically update the network; to do this, we can apply the chenille's templates in any way such that it is spatially periodic, for instance from the left to the right and from the top to the bottom. Our main result is thus:

Theorem 3.3. AP *is P-complete for* KSPM *in dimension two and any $p \geq 2$.*

Proof. The fact that AP is in P is already known since C. Moore and M. Nilsson paper [12]. Their proof is done by proving that the total number of avalanches required to relax a sandpile is polynomial in the system's size.

For the reduction, one has to take an arbitrary instance of the MCVP variant previously defined and to build an initial configuration of a sandpile for the Kadanoff's dynamics for $p = 2$ (or greater). Thus, we have to design the following gadgets:

- a wire and how to turn the signal (Fig. 7);
- the crossing of information (Fig. 8 (Left));
- a AND gate (Fig. 8 (Right));
- a OR gate (Fig. 9 (Left));
- a signal multiplier (Fig. 9 (Right)).

Simulated gates can be made up like classical gates (up to an additive constant depending upon their size) with a fan in of two parallel wires and a fan out upperbounded by two. The sandpile circuit is built directly from the standard encoding of the instance of the MCVP variant. This construction can be done by a NC algorithm [4, 9].

The construction of each gadget is shown graphically for $p = 2$ but can be done for greater values. As an example, we give the construction for the AND gate for $p = 3$ in Fig. 6. Generalising for greater values of p is not hard though tedious and would have exceeded the number of pages. For $p = 2$, the horizontal and vertical chenilles are given in Fig. 5. Recall that the decision problem only adds a sand grain to one pile, say $(0, 0)$. To construct the entry vector to an arbitrary circuit we have to construct from the starting pile wires to simulate any variable $\alpha_i = 1$. (If $\alpha_i = 0$ nothing is done: we do not construct a wire from the initial pile. Else, there will be a wire to simulate the value 1). Also remark that the signal propagation does not require a global synchronising clock. Actually, this helps for designing the circuit since one signal arriving at a logical gate can "wait" for the second signal to arrive.

Then, by construction, positive instances of the MCVP variant are in 1:1 correspondence with positive instances of AP. ■

Remark 3.4. In the case $p = 2$, KSPM corresponds to Bak's model [1] in two dimensions with a sandpile such that $x_{i,j} \geq \max\{x_{i+1,j}, x_{i,j+1}\}$.

4. Conclusion and future work

We have proved that the avalanche problem for the KSPM model in two dimensions is P-complete with a sandpile defined as in [2] and for every value of the parameter p. Let us also point out that in the case where $p = 2$, this model corresponds to the two-dimensional Bak's model with a pile such that $x_{i,j} \geq 0$ and $x_{i,j} \geq \max\{x_{i+1,j}, x_{i,j+1}\}$. In this context, we also proved that this physical version (with a two dimension sandpile interpretation) is P-complete. It is important to notice that, by directly taking the two-dimensional Bak's tokens game (given a graph such that a vertex has a number of token greater or equal than its degree, it gives one token to each of its neighbors), its computation universality was proved in [7] by designing logical gates in non-planar graphs. Furthermore, by using the previous construction, C. Moore *et al.* proved the P-completeness of this problem for lattices of dimensions d with $d \geq 3$. But the problem remained open for two-dimensional lattices. Furthermore, it was proved in [3] that, in the latter case, it is not possible to build circuits because the information is impossible to cross. The two-dimensional Bak's operator corresponds, in our framework, to the application of the four rotations of the template (see Fig. 10). But this model is not anymore the representation of a two-dimensional sandpile as presented in [2], that is with $x_{i,j} \geq 0$ and $x_{i,j} \geq \max\{x_{i+1,j}, x_{i,j+1}\}$.

To define a reasonable two-dimensional model, consider a monotone sandpile decreasing for $i \geq 0$ and $j \geq 0$. Over this pile we define the extended Kadanoff's model as a local avalanche in the growing direction of the $i-j$ axis such that monotonicity is allowed. Certainly, one may define other local applications of Kadanoff's rule which also match with the physical sense of monotonicity. For instance, by considering the set $(i+1, j), (i+1, j+1), (i, j+1)$ as the piles to be able to receive grains from pile (i, j). In this sense it is interesting to remark that the two-dimensional sandpile defined by Bak (i.e for nearest neighbors, also called the von Neumann neihborhood, a pile gives a token to each of its four neighbors if and only if it has enough tokens) can be seen as the application of the Kadanoff rule for $p = 2$ by applying to a pile, if there are at least four tokens, the horizontal ($\rightarrow$) and the vertical ($\downarrow$) chenille simultaneously (see Fig. 10). Similarily, for an arbitrary p, one may simultaneously apply other conbinations of chenilles which, in general, allows us to get P-complete problems. For instance, when there are enough tokens, the applications of the four chenilles (*i.e.* $\leftarrow$,$\rightarrow$,$\uparrow$ and $\downarrow$) give raise to a new family of local templates called *butterflies* (because of their four wings). It is not so difficult to construct wires and circuits for butterflies. Hence, for this model, the decison problem will remain P-complete. One thing to analyze from an algebraic and complexity point of view is to classify every local rule derivated from the chenille application. Further, one may define a more general sandpile dynamics which contains both Bak's and Kadanoff's ones: i.e given an integer $p \geq 2$, we allow the application of every Kadanoff's update for $q \leq p$. That means that an active pile with more than p grains can distribute up to q grains to the adjacent piles. We are studying this dynamics and, as a first result, we observe yet that in one dimension there are several fixed points and also, given a monotone circuit with depth m and with n gates, we may simulate it on a line with this generalized rule for a given $p \geq m + n$.

For the one-dimensional avalanche problem as defined in Section 2, it can be proved that it belongs tho the class NC for $p = 2$ and that it remains in the same class when the first p piles contain more than one grain (*i.e.* that there is no hole in the pile). We are in the way to prove the same in the general case.

Acknowledgements

We thank Pr. Enrico Formenti for helpful discussions and comments while Pr. Eric Goles was visiting Nice and writing this paper.

References

[1] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality. *Phys. Rev. A*, 38(1):364–374, 1988.

[2] E. Duchi, R. Mantaci, H. Duong Phan, and D. Rossin. Bidimensional sand pile and ice pile models. *Pure Math. Appl. (PU.M.A.)*, 17(1-2):71–96, 2006.

[3] A. Gajardo and E. Goles. Crossing information in two dimensional sand piles. *Theoretical Computer Science*, 369(1-3):463–469, 2006.

[4] L.M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *ACM sigact news*, 9(2):25–29, 1977.

[5] E. Goles and M. Kiwi. Sand pile dynamics in one dimensional bounded lattice. In N. Boccara et al, editor, *Cellular Automata and Cooperative Systems*, volume 396 of *NATO-ASI*, pages 203–210. Ecole d'Hiver, Les Houches, Kluwer, 1993.

[6] E. Goles and M. Margerstern. Sand piles as a universal computer. *Journal of Modern Physics-C*, 7(2):113–122, 1996.

[7] E. Goles and M. Margerstern. Universality of the chip firing game on graphs. *Theoretical Computer Science*, 172:121–134, 1997.

[8] E. Goles Ch., M. Morvan, and Ha Duong Phan. The structure of a linear chip firing game and related models. *Theor. Comput. Sci.*, 270(1-2):827–841, 2002.

[9] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. *Limits to parallel computation.* Oxford University Press, 1995.

[10] L.P. Kadanoff, S.R. Nagel, L. Wu, and S. Zhou. Scaling and universality in avalanches. *Phys. Rev. A*, 39(12):6524–6537, 1989.

[11] P. B. Miltersen. The computational complexity of one-dimensional sandpiles. *Theory of Computing Systems*, 41:119–125, 2007.

[12] C. Moore and M. Nilsson. The computational complexity of sandpiles. *Journal of statistical physics*, 96(1-2):205–224, 1999.

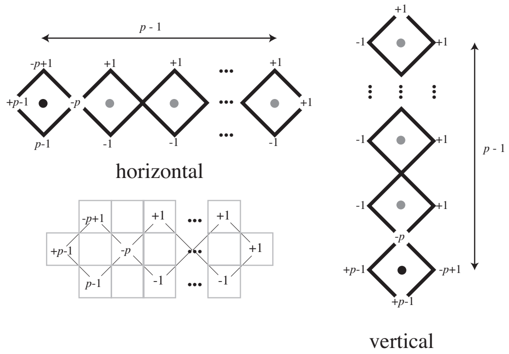


Figure 2: Horizontal resp. vertical (on the left resp. on the right) chenilles in the $\mathbb{N}\times\mathbb{N}$ lattice for arbitrary parameter $p \geq 2$. The pile (i, j) –denoted by a black bullet– gives one grain of sand to the pile $(i + p - 1, j)$ horizontally resp. one grain of sand to the pile $(i, j + p - 1)$ vertically. The figure gives the height differences of this dynamics and the change of the lattice structure between the dynamics on the grains and the corresponding dynamics on the height difference. In the sequel, we will adopt the representation on the left and on the bottom for defining the templates.

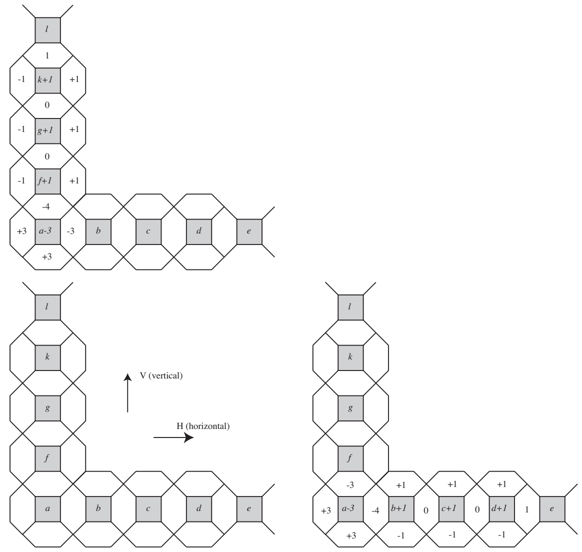


Figure 3: Horizontal and vertical chenilles for $p = 4$. Shaded squares count the number of grains on each pile and the hexagons between the squares the height difference between the corresponding two adjacent piles. The initial configuration is on the bottom-left. The Kadanoff's dynamics is applied from the shaded pile labelled a horizontally or vertically (resp. $\uparrow V$ and $\xrightarrow{H}$) to get the resulting configurations.

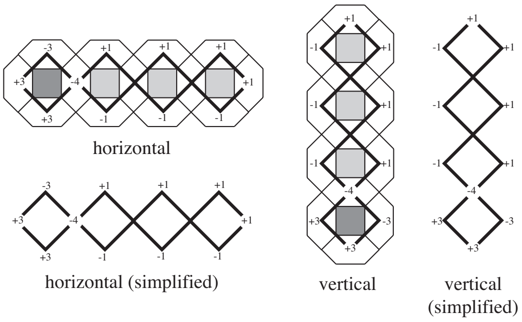


Figure 4: Horizontal and vertical chenilles for $p = 4$. The dynamics is applied to the dark-shaded pile (the leftmost one on the left part of the figure and the lowest one on the right part of the figure). The numbers express the height differences after the application of the dynamics. The two simplified views remove the number of sand grains information and only keeps the height difference information. It corresponds to a change in the lattice structure if the grains are considered or the height difference.

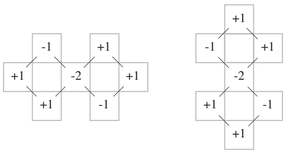


Figure 5: Templates for Bak's dynamics with $p = 2$.

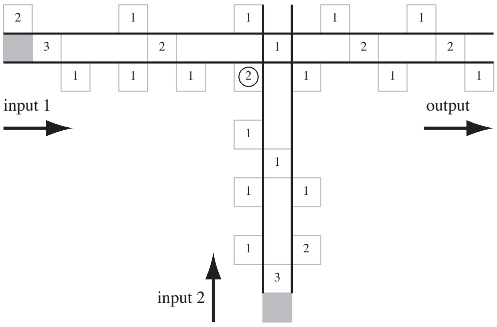


Figure 6: The logic AND gate for two inputs for $p = 3$. The circled "2" is put in order to get enough tokens for the horizontal and vertical inputs.

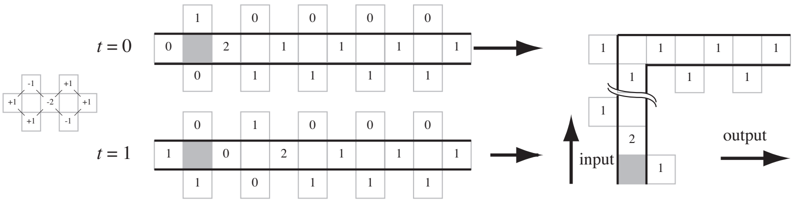


Figure 7: Information propagation in a wire for $p = 2$ at times $t = 0$ and $t = 1$ using the templates for Baks dynamics (recalled on the left of the figure) and wire for turning the information to the right.

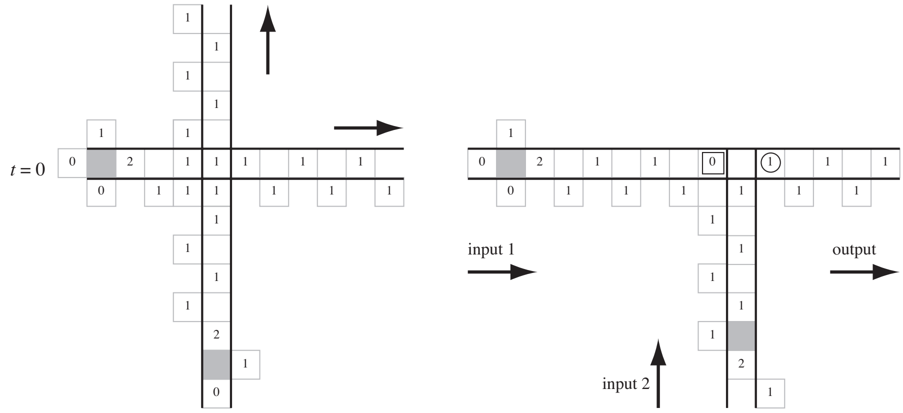


Figure 8: (Left) Crossing over two wires for $p = 2$; arrows show the directions of propagation. (Right) A logic AND gate with two inputs for $p = 2$. The upcoming "2" has to reach the horizontal "2" to change the value of the boxed "0" to "1". Then, the upcoming "2" can apply the vertical chenille template and changes the circled "1" into "2". In other words, the AND is computed by applying 3 horizontal chenilles and 4 vertical ones.

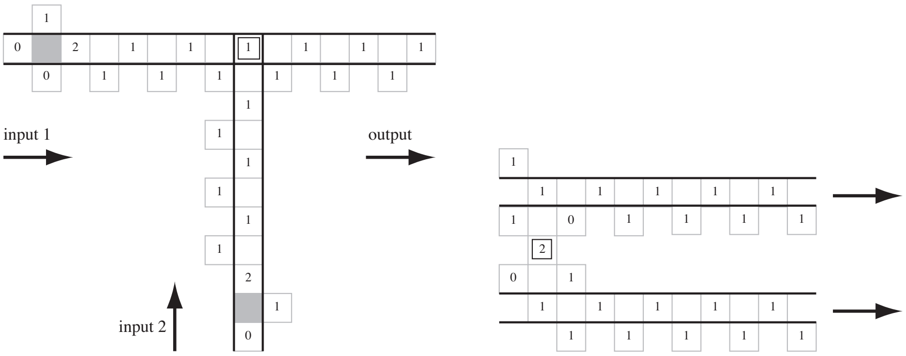


Figure 9: (Left) A logic OR gate with two inputs for $p = 2$. The boxed cell indicates the OR gate point of computation. (Right) A signal multiplier for $p = 2$. The signal starts on boxed pile with value 2 (the input) and applying the first vertical chenille ruled by the Kadanoff's dynamics multiplies the signal on both horizontal wires. Then, we use horizontal chenilles to move both signals according to the arrows.

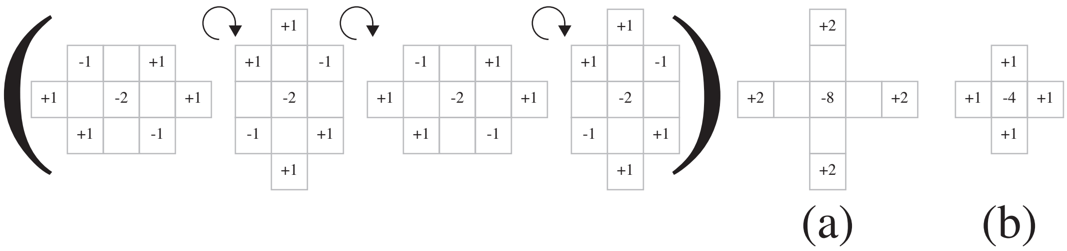


Figure 10: From Bak's to Kadanoff's operators. All the Kadanoff's operators between the brackets have been applied to get the pattern (a). The Bak's pattern (b) is obtained by eliminating the holes in (a) and by dividing the number of tokens by two.

CLANDESTINE SIMULATIONS IN CELLULAR AUTOMATA

PIERRE GUILLON [1], PIERRE-ÉTIENNE MEUNIER [2], AND GUILLAUME THEYSSIER [2]

[1] Department of Mathematics, University of Turku, 20014 Turku, Finland
E-mail address: piegui@utu.fi

[2] LAMA (CNRS, Université de Savoie), Campus Scientifique, 73376 Le Bourget-du-Lac Cedex, France
E-mail address, P.-E. Meunier: pierreetienne.meunier@univ-savoie.fr
E-mail address, G. Theyssier: guillaume.theyssier@univ-savoie.fr

Abstract. This paper studies two kinds of simulation between cellular automata: simulations based on factor and simulations based on sub-automaton. We show that these two kinds of simulation behave in two opposite ways with respect to the complexity of attractors and factor subshifts. On the one hand, the factor simulation preserves the complexity of limits sets or column factors (the simulator CA must have a higher complexity than the simulated CA). On the other hand, we show that any CA is the sub-automaton of some CA with a simple limit set (NL-recognizable) and the sub-automaton of some CA with a simple column factor (finite type). As a corollary, we get intrinsically universal CA with simple limit sets or simple column factors. Hence we are able to 'hide' the simulation power of any CA under simple dynamical indicators.

Introduction

Since the introduction of the model in the 40s, cellular automata have been studied both as dynamical systems and as a computational model. In both aspects, they can show very complex behaviors, be it through their topological dynamics [Kůr03] or through their ability to compute [vN66, Oll03]. As such, they constitute a good model to tackle one of the major question of natural computing: what kind of dynamical behavior allows to support computation, and reciprocally, what does the ability to compute imply on the dynamical behavior of a system?

In this paper, we focus on asymptotic dynamics of cellular automata (notion of limit set [ČPY89]), and on their unprecise observation (notion of column factors [Kůr97]). Intuitively, the limit set is the set of configuration that can appear arbitrarily far in the evolution of the system, and the column factor is the set of sequences of states that a cell (or group of cells) can take in a valid orbit of the system. These notions have been intensively studied in the literature as indicators of the dynamical complexity of cellular automata [Hur90, Kar94, BM97, CFG10].

Research partially supported by projects CONICYT-ANILLO ACT88 (Chile), FONDECYT 1090156 (Chile), ANR EMC NT09 555297 (France) and Academy of Finland project 131558.

Concerning limit sets of cellular automata, the initial (wrong) intuition was that a universal CA should necessarily have a non-recursive limit set (such a statement appears in [ČPY89]). Later, a Turing-complete CA with a simple limit set was constructed in [GMM93]. This result gives a first hint concerning the absence of correlation between the complexity of the limit set and the ability to handle computations. However, the construction goes through a slow simulation of two-register machines where registers are encoded in unary. Therefore, this results is *extrinsic* to the model of cellular automata and shows only that any behavior of register machines can be embedded into a cellular automaton with a simple limit set.

Here, we aim at exploring *intrinsic* versions of the same question: what can be the limit set complexity of a CA able to simulate any other CA? More precisely, we consider two flavors of simulations: one using factor (continuous projection) of a simulator CA onto the simulated CA, and the other using the local uniform injection (sub-automaton) of a simulated CA into a simulator CA. Such simulation relations were extensively studied in [DMOT10b], and the second flavor is the one giving rise to the notion of intrinsic universality [Oll02, DMOT10b]. The main point of the present paper is that factor simulations preserve limit set complexity whereas sub-automaton simulations do not. We also show that the same phenomenon appears for the complexity of column factors. Our main results are two embedding theorems showing that there exist an intrinsically universal CA with column factors of finite type and an intrinsically universal CA with an NL-recognizable limit set (that is, its limit language can be recognized by a non-deterministic Turing machine in logarithmic space). The second result solves an open problem of [The05].

1. Definitions

Let us note $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. If $i, j \in \mathbb{Z}$, we note $[\![i, j]\!]$ the interval of integers k such that $i \leq k \leq j$, $[\![i, j[\![= [\![i, j-1]\!]$, $]\!]i, j[\![= [\![i+1, j-1]\!]$ and so on. Consider a fixed finite *alphabet* A. If $x \in A^{\mathbb{Z}}$ and $[\![i, j]\!] \subset \mathbb{Z}$, then we note $x_{[\![i,j]\!]} \in A^{j-i+1}$ the *pattern* corresponding to the sequence of letters $x_i \dots x_j$ (and similarly for other kinds of intervals).

If $U \subset A^k$ for some $k \in \mathbb{N}$ and $i \in \mathbb{Z}$, the *cylinder* $[U]_i$ will denote the set of configurations $x \in A^{\mathbb{Z}}$ such that $x_{[\![i,i+k[\![} \in U$. We also note $[U] = [U]_0$. If $a \in A$, let ${}^\infty a^\infty$ be the configuration $x \in A^{\mathbb{Z}}$ such that $x_i = a$ for any $i \in \mathbb{Z}$.

A *dynamical system* is a pair (X, F) where X is a compact space and F is a continuous self-map of X. We can then study iterations F^t for any *time step* $t \in \mathbb{N}$.

Let $\mathbb{M}$ stand either for $\mathbb{N}$ or for $\mathbb{Z}$. The *shift* map σ is defined for any $z \in A^{\mathbb{M}}$ and any $i \in \mathbb{M}$ by $\sigma(x)_i = x_{i+1}$. A *subshift* is a dynamical system (Σ, σ), or simply Σ, that is a subset of $A^{\mathbb{M}}$ which is closed under the usual Tychonoff topology and invariant by the shift map. Equivalently, it is the set $\left\{ z \in A^{\mathbb{M}} \middle| \forall [\![i, j]\!] \subset \mathbb{M}, z_{[\![i,j]\!]} \notin \mathcal{F} \right\}$ of infinite words that avoid the finite patterns from some given family $\mathcal{F}$. If this family can be taken finite, then Σ is called a subshift of *finite type* (SFT). If it can be taken among words of length $k \in \mathbb{N}_+$, it has *order k*. The *language* $\mathcal{L}(\Sigma)$ of a subshift Σ is the set $\left\{ z_{[\![i,j]\!]} \middle| z \in \Sigma \text{ and } [\![i, j]\!] \subset \mathbb{M} \right\}$ of patterns appearing in the infinite words of Σ. If this language is regular, then we say that Σ is a *sofic* subshift. Equivalently thanks to the Weiss theorem [Wei73], it is obtained from an SFT by a letter-to-letter projection.

A *cellular automaton* (CA) is a dynamical system $(A^{\mathbb{Z}}, F)$ which commutes with the shift, *i.e.* $\sigma F = F\sigma$. Equivalently, thanks to the Curtis-Hedlund-Lyndon theorem, it is obtained by some *local map* $f : A^{2r+1} \to A$ for some *radius* $r \in \mathbb{N}$, *i.e.* for any $x \in A^{\mathbb{Z}}$ and $i \in \mathbb{Z}$, $F(x)_i = f(x_{[\![i-r,i+r]\!]})$. F admits a *spreading* state $0 \in A$ if it admits a local rule $f : A^{2r+1} \to A$ with $r \in \mathbb{N}$ and $f(u) = 0$ whenever there exists $i \in [\![0, 2r]\!]$ such that $u_i = 0$.

An SFT Σ is *irreducible* if for any two words $u, v \in \mathcal{L}(\Sigma)$, there exists a word w such that $uwv \in \mathcal{L}(\Sigma)$. CA can actually be applied in a very natural way to these subshifts: a *partial CA* will be a dynamical system over some irreducible SFT that commutes with the shift. It is known from [Fio00] that any injective partial CA is bijective and reversible (the inverse is also a partial CA).

1.1. Simulations

The central notion studied in this paper is that of simulation between dynamical systems and more specifically cellular automata. We will distinguish two families of simulation relations based on the following notions.

Definition 1.1 (Factors and sub-systems)**.**

- A *factor map* between two dynamical systems (X, F) and (Y, G) is a continuous onto map $\Phi : X \to Y$ such that $\Phi F = G\Phi$. In that case, we say that G is a *factor* of F.
- A *sub-system* of a dynamical system (X, F) is a dynamical system of the form (Y, G) where $Y \subseteq X$ (Y is closed) and $F(Y) \subseteq Y$.

Note that a factor map Φ between two subshifts Σ and Γ respect the Curtis-Hedlund-Lyndon theorem: there exist a *radius* $r \in \mathbb{N}$ and a *local rule* $\phi : \mathcal{L}_{2r+1}(\Sigma)$ such that $\Phi(x)_i = \phi(x_{[\![i-r,i+r]\!]})$ for any $x \in \Sigma$ and any $i \in \mathbb{Z}$. We say that the factor map between two subshifts is a *coloring* if the radius can be taken $r = 0$.

If (X, F) and (Y, G) are cellular automata, we say that (X, F) *simulates* (Y, G) *by factor* if there is a factor map Φ from (X, F) onto (Y, G) which is also a factor map from (X, σ_X) onto (Y, σ_Y). Besides, when (X, F) is a cellular automaton and Y is a full-shift included in X, then (Y, F) is a *sub-automaton* of (X, F).

These two relations (factor and sub-system) are restrictive since they don't allow entropy to increase: a factor or a sub-system of a given system has always a lower entropy. As a consequence, they don't support universality (existence of a system able to simulate any other) since it is not difficult to build systems of arbitrarily large entropy. Hence, in the literature, other ingredients were introduced to obtain richer notions of simulation: for instance, in cellular automata, operations of space and time rescaling added to the notion of sub-automaton lead to a notion of simulation supporting universality [DMOT10a, DMOT10b].

In this paper, such kind of spatio-temporal transformations are not considered explicitly for two reasons:

- our results about factor simulation (Section 2) involve properties (complexity of the subshifts) which are invariant by space and time rescaling, hence the results still hold when considering rescaling as part of the simulation relation;

- our results about sub-system simulation (Section 3) are of the form "any CA is the sub-automaton of a CA with some given property", which is actually the most general we can get, and remain true when replacing "sub-automaton" by more general notions of sub-system (such as sub-system, or simulated system in the sense of [DMOT10b]).

1.2. Complexity of limit sets and column factors

Many different points of view have been adopted to study the complexity of CA. We here use symbolic dynamics, and consider the complexity, as subshifts, of the limit set on the one hand, or the column factors on the other hand, as representing the actual complexity of the CA.

The *limit set* of the dynamical system is the nonempty closed subset $\Omega_F = \bigcap_{t\in\mathbb{N}} F^t(X)$. Its *limit system* is the maximal surjective subsystem, (Ω_F, F). It basically represent the asymptotic dynamics of the system.

With respect to CA limit sets, it is not difficult to see that the corresponding language is always corecursively enumerable (it is an *effective* subshift). However, there are known examples of non-recursive ones [Hur90]. Moreover, simple additional remarks [Taa07, Kůr03, Nas95] give the following hierarchy:
F injective $\Rightarrow F_{|\Omega_F}$ injective $\Rightarrow \Omega_F$ is an SFT $\Rightarrow F$ is stable $\Rightarrow \Omega_F$ is sofic $\Rightarrow \dots$

The *column factor* of a dynamical system $(A^{\mathbb{Z}}, F)$ upon interval $[\![i,j]\!] \subset \mathbb{Z}$ is the set $\tau_F^{[\![i,j]\!]} = T_F^{[\![i,j]\!]}(A^{\mathbb{Z}})$, where $\begin{array}{rccl} T_F^{[\![i,j]\!]}: & A^{\mathbb{Z}} & \to & (A^{[\![i,j]\!]})^{\mathbb{N}} \\ & x & \mapsto & (F^t(x)_{[\![i,j]\!]})_{t\in\mathbb{N}} \end{array}$. It is a factor subshift since $\sigma T_F^{[\![i,j]\!]} = T_F^{[\![i,j]\!]} F$. It can represent an observation of the system made by a measuring device with a finite precision (that cannot see cells which are far away). It can be seen that any factor subshift is essentially a factor of some column factor [Kůr97].

In the case of CA, shift-invariance allows to consider only the central column factors $\tau_F^{[\![-k,k]\!]}$ for radius $k \in \mathbb{N}_+$. It is known [Gil88] that these CA column factors always have a context-sensitive language; they may actually be strictly context-sensitive. In [Kůr97], strong links with topological notions are stated: finite column factors is equivalent to equicontinuity, SFT column factors imply the shadowing property which in turn imply sofic column factors.

2. Factor simulations

The two hierarchies that we have just defined, based on subshift classifications, are then very robust to factor simulations, as we can see in this section. Taking the vocabulary of order theory, we say that a class $\mathcal{C}$ of systems is *an ideal for factor simulation* if, whenever (X, F) is a factor of (Y, G), we have: $(Y, G) \in \mathcal{C} \Rightarrow (X, F) \in \mathcal{C}$.

Proposition 2.1. *If* Φ *is a factor map from* (X, F) *onto* (Y, G)*, then* $\Omega_G = \Phi(\Omega_F)$*.*

Proof. For any $t \in \mathbb{N}$, $\Phi F^t(X) = G^t(Y)$. First note that $\Phi(\bigcap_{t\in\mathbb{N}} F^t(X))$ is included in $\bigcap_{t\in\mathbb{N}} \Phi F^t(X) = \bigcap_{t\in\mathbb{N}} G^t(Y)$. Conversely, let $y \in \bigcap_{t\in\mathbb{N}} G^t(Y)$ and, for $t \in \mathbb{N}$, $X_t = \Phi^{-1}(y) \cap F^t(X)$. Note that X_t is closed (since Φ and F are continuous, and X is compact) and nonempty (since Φ is onto). By compactness, the intersection $\bigcap_{t\in\mathbb{N}} X_t = \Phi^{-1}(y) \cap \Omega_F$ is not empty, *i.e.* $y \in \Phi(\Omega_F)$. We have proven that $\Phi(\Omega_F) = \Omega_G$. ■

Moreover, we can note that $\Omega_{F^n} = \Omega_F$, since if $n \in \mathbb{N}_+$, then $F^{nt}(X) = \bigcap_{(n-1)t<j\leq nt} F^j(X)$. In the following we will no longer mention time and space rescaling, which does not essentially alter the results.

Corollary 2.2. *The class of stable systems is an ideal for factor simulation.*

Proof. If Φ is a factor map from (X, F) onto (Y, G), and $\Omega_F = F^t(X)$ for some $t \in \mathbb{N}$, then $\Omega_G = \Phi(\Omega_F) = \Phi F^t(X) = G^t(X)$. ■

We can also derive the two following results.

Proposition 2.3. *The class of CA whose limit set (resp. factor subshifts) is sofic (resp. context-free, context-sensitive, recursive) is an ideal for factor simulation.*

Proof. It is known that each of the corresponding classes of subshifts is preserved by factor maps. Proposition 2.1 states that the limit set of the simulated CA is a factor of the limit set of the simulating CA.
Moreover, note, thanks to the transitivity of the notion of factor, that a factor subshift of the simulated CA is also a factor subshift of the simulating CA. ■

The following slightly generalizes a result in [The05].

Proposition 2.4. *The class of reversible partial CA is an ideal for coloring.*

Proof. Let Φ be a factor map based on some radius-0 local rule $\phi : A \to B$, from a partial CA (Σ, F) onto another one (Γ, G), where $\Sigma \subset A^{\mathbb{Z}}$ and $\Gamma \subset B^{\mathbb{Z}}$ and there exists some partial CA $F^{-1} : \Sigma \to \Sigma$ such that FF^{-1} is the identity. By surjectivity of Φ, for any letter $b \in B$, there exists a letter, denoted abusively $\phi^{-1}(b) \in A$ such that $\phi(\phi^{-1}(b)) = b$. If Φ^{-1} represents the parallel application of ϕ^{-1} to $B^{\mathbb{Z}}$, we obtain that $\Phi\Phi^{-1}$ is the identity of Γ. We can define the map $G^{-1} = \Phi F^{-1}\Phi^{-1}$, in such a way that GG^{-1} is the identity of Γ. G^{-1} is an injective partial CA, hence it is reversible, and it is the actual inverse map of G. ■

As far as we know though, it is unknown whether the class of injective CA is still an ideal for factor simulation.

Corollary 2.5. *The class of CA which are reversible over the limit set is an ideal for coloring.*

Proof. Consider such a CA. From [Taa07], it is stable and its limit set is an irreducible SFT. Now if it is linked to some other CA by some coloring, then by Proposition 2.1, so are the two limit systems (that are partial CA), which then respect the hypotheses of Proposition 2.4. ■

From the previous, we can see that any universal CA for factor simulation (up to rescaling) must have a non-recursive limit set and strictly context-sensitive factor subshifts (and of course must not be injective), but the existence of such a CA is still open.

3. Sub-system simulations

We will see in this section that, contrary to factor simulations, sub-system simulations allow to hide the complexity of the simulated CA into the simulator CA.

3.1. Hiding the column factors

If G is a CA over alphabet C of radius $r > 0$, local rule g, then let us define the CA $\tilde{G}$ over alphabet $D = \{-1, 0, 1\} \times C$ with the same radius r as G and local rule:

$$\begin{array}{rcl} \tilde{g}: & D^{2r+1} & \to D \\ & (\varepsilon_{-r}, c_{-r}) \dots (\varepsilon_r, c_r) & \mapsto \left| \begin{array}{ll} (\varepsilon_0, c_{\varepsilon_0}) & \text{if } \varepsilon_0 \neq 0 \ ; \\ (0, g(c_{-r} \dots c_r)) & \text{otherwise.} \end{array} \right. \end{array}$$

Clearly, G is a sub-automaton of $\tilde{G}$ (up to state renaming) corresponding to the sub-alphabet $\{0\} \times C$.

Theorem 3.1. *Any cellular automaton G is a sub-automaton of some CA whose column factors are SFT of order* 2.

Proof. Let us take a CA G over alphabet C, of radius 1. Let $\tilde{G}$ be defined as above over alphabet $\tilde{C} = \{-1, 0, 1\} \times C$, $k \in \mathbb{N}_+$ and Σ its column factor of width k. Of course, Σ is included in its 2-*approximation*

$$\mathcal{A}_2(\Sigma) = \left\{ z = (z^t)_{t \in \mathbb{N}} \in (\tilde{C}^k)^{\mathbb{N}} \middle| \forall t \in \mathbb{N}, \exists x^t \in [z^t] \cap \tilde{G}^{-1}([z^{t+1}]) \right\}.$$

Let us show that they are actually equal. Let $z = (z^t)_{t \in \mathbb{N}} \in (\tilde{C}^k)^{\mathbb{N}}$ be such that for any $t \in \mathbb{N}$ there exists some configuration $x^t \in [z^t]$ with $\tilde{G}(x^t) \in [z^{t+1}]$, and x defined by:

$$x_i = \left| \begin{array}{ll} x_i^0 & \text{if } 0 \leq i < k \\ (-1, c) & \text{if } i < 0 \text{ and } x_{-1}^{-i-1} = (\varepsilon, c) \\ (1, c) & \text{if } i \geq k \text{ and } x_k^{i-k} = (\varepsilon, c) \ . \end{array} \right.$$

An inductive application of the local rule gives that for any $t \in \mathbb{N}$, we have:

$$\tilde{G}^t(x)_i = \left| \begin{array}{ll} x_i^t & \text{if } 0 \leq i < k \\ (-1, c) & \text{if } i < 0 \text{ and } x_{-1}^{t-i-1} = (\varepsilon, c) \\ (1, c) & \text{if } i \geq k \text{ and } x_k^{t+i-k} = (\varepsilon, c) \ . \end{array} \right.$$

In particular, $z = T^k_{\tilde{G}}(x) \in \Sigma$. We have proven that Σ is an SFT of order 2. If G does not have radius 1, then it is easy to widen the radius of $\tilde{G}$ (and increase the speed of the shifts) to get the same result. ■

3.2. Hiding the limit set

The main result of this section is based on the existence of a firing-squad CA with specific properties expressed by Lemma 3.2. We actually refer to the firing-squad CA defined in [Kar94], that we denote by S, and prove additional properties in Section 4. This CA admits a so-called *firing* state γ and a spreading state κ. Let r_S the radius of S, s its local rule, Q its state set, and $Q' = Q \setminus \{\kappa, \gamma\}$. Consider the set X_S of configurations having an infinite history avoiding κ and γ:

$$X_S = \left\{ y \in Q^{\mathbb{Z}} \middle| \exists (y^t)_{t \in \mathbb{N}} \in (Q^{\mathbb{Z}})^{\mathbb{N}}, y^0 = y \text{ and } \forall t \in \mathbb{N}_+, y^t \in Q' \text{ and } S(y^t) = y^{t-1} \right\} \ .$$

Lemma 3.2. *S satisfies the following:*

(1) ${}^\infty\gamma^\infty \in X_S$;

(2) $\Omega_S \cap [\gamma] \subset \{\kappa, \gamma\}^{\mathbb{Z}}$;

(3) X_S *is NL-recognizable.*

Proof. Properties (1) and (2) are proven in [Kar94]. Property (3) is given by Proposition 4.10 below. ∎

This construction allows to state the following theorem, proven at the end of the subsection.

Theorem 3.3. *Any CA is a sub-automaton of some CA whose limit set is NL-recognizable.*

By the existence of intrinsically universal CA for a simulation containing the sub-automaton relation (see for instance [Oll03]) and the transitivity of simulations, we can directly derive the following.

Corollary 3.4. *There exists an intrinsically universal CA whose limit set is NL-recognizable.*

The idea of the construction is the following: given some CA F over alphabet A, we add an extra (firing-squad) component which is able to generate any configuration of $A^{\mathbb{Z}}$ arbitrarily far in the future. The complexity of the limit set of F is thus completely flooded into the full-shift $A^{\mathbb{Z}}$. All the technical difficulty is to control the contribution of the additional component to the final limit set.

Let F be a CA of radius r_F, local rule f over alphabet A with a spreading state $0 \in A$. We define a CA $\Delta_{F,S}$ of local rule $\delta_{F,S}$ defined on alphabet $C = A \sqcup (A \times Q)$ with radius $r = \max(r_F, r_S)$ by:

$$\delta_{F,S} : c \mapsto \begin{cases} f(a_{-r_F} \dots a_{r_F}) & \text{if } c = a_{-r} \dots a_r \in A^{2r+1} \text{ ;} & (1) \\ a_0 & \text{if } c = (a_{-r}, \gamma) \dots (a_r, \gamma) \in (A \times \{\gamma\})^{2r+1} \text{ ;} & (2) \\ (a_0, s(b_{-r_S} \dots b_{r_S})) & \text{if } c = (a_{-r}, b_{-r}) \dots (a_r, b_r) \in (A \times Q')^{2r+1} \text{ ;} & (3) \\ 0 & \text{otherwise.} & (4) \end{cases}$$

Basically, this CA freezes the first component while applying the firing squad on the second component until some firing state appears, which then frees this second component and starts the application of F. When the configuration is not coherent, or when κ appears, 0 begins to spread. Clearly, F is a sub-automaton of $\Delta_{F,S}$.

The structure of the corresponding limit set will be given by the following lemmas.

Lemma 3.5. $A^{\mathbb{Z}} \subset \Omega_{\Delta_{F,S}}$.

Proof. Let $x \in A^{\mathbb{Z}}$. From Point 1 of Lemma 3.2, there is a sequence $(y^t)_{t\in\mathbb{N}}$ with $y^0 = {}^\infty\gamma^\infty$ and for any $t \in \mathbb{N}_+$, $y^t \notin \{\gamma, \kappa\}$ and $S(y^t) = y^{t-1}$. Consider now the configurations $x^t = (x_i, y_i^{t+1})_{i\in\mathbb{Z}}$ for $t \in \mathbb{N}_+$. By a quick induction on $t \in \mathbb{N}_+$, we can see that for any cell $i \in \mathbb{Z}$, only case (3) of the local rule is used, and $x^0 = \Delta_{F,S}^t(x^t)$. At time -1, since the second component of x^{-1} is ${}^\infty\gamma^\infty$, case (2) of the rule is applied in every cell, which gives $x = \Delta_{F,S}(x^{-1}) = \Delta_{F,S}^t(x^{-t})$ for any $t \in \mathbb{N}_+$. ∎

Lemma 3.6. *Let $x \in \Omega_{\Delta_{F,S}}$ and $i, j \in \mathbb{Z}$ such that $i \neq j$ and $x_i = (a_i, \gamma), x_j = (a_j, b_j) \in A \times Q$. Then $b_j \in \{\gamma, \kappa\}$.*

Proof. We can assume, by symmetry, that $i < j$, and for the sake of contradiction that $b_j \notin \{\gamma, \kappa\}$. Let $(x^t)_{t\in\mathbb{Z}}$ be a biorbit of $x = x^0$, *i.e.* a bisequence of configurations such that $\forall t \in \mathbb{Z}, \Delta_{F,S}(x^t) = x^{t+1}$. By an easy recurrence and the fact that $x_i \in A \times Q$ can only be obtained through case (3) of the rule, we can see that for any $t \in \mathbb{N}$, $x^{-t}_{[\![i-rt,i+rt]\!]}$ can be written $(a^{-t}_{i-rt}, b^{-t}_{i-rt}) \dots (a^{-t}_{i+rt}, b^{-t}_{i+rt}) \in$

$(A\times Q)^{1+2rt}$ and $s^t(b^{-t}_{i-r_St}\dots b^{-t}_{i+r_St})=b_i$; in the same way, $x^{-t}_{[\![j-rt,j+rt]\!]}$ can be written $(a^{-t}_{j-rt},b^{-t}_{j-rt})\dots(a_{j+rt},b_{j+rt})\in(A\times Q)^{1+2rt}$, and $s^t(b^{-t}_{j-r_St}\dots b^{-t}_{j+r_St})=b_j$. Then for any $t>\frac{j-i-1}{2r}$, $x^{-t}_{[\![i-2rt,j+2rt]\!]}$ is in $(A\times Q)^{j-i-1+4rt}$ and the image $s^t(x^{-t}_{[\![i-r_St,j+r_St]\!]})$ contains b_i and b_j. In other words, the cylinder $[b_iQ^{j-i-1}b_j]_i$ intersects $S^t(Q^{\mathbb{Z}})$ for any t, and by compactness intersects Ω_S, which contradicts Point 2 of Lemma 3.2. ■

If $\Sigma\subset A^{\mathbb{Z}}$ is a subshift and $0\in A$, then we consider the subshift $0\bullet\Sigma\bullet 0=\bigcup_{-\infty\le l\le m\le+\infty}\left\{x\in A^{\mathbb{Z}}\middle|\,\forall i\notin]\!]l,m[\![\,,x_i=0\text{ and }\exists y\in\Sigma,x_{]\!]l,m[\![}=y_{]\!]l,m[\![}\right\}$ of configurations or pieces of configurations of Σ surrounded by 0.

Lemma 3.7. $\Omega_{\Delta_{F,S}}\setminus A^{\mathbb{Z}}\subset 0\bullet(A\times Q)^{\mathbb{Z}}\bullet 0$.

Proof. By shift-invariance, it is sufficient to prove that $\Omega_{\Delta_{F,S}}\cap[A\times Q]_0\subset 0\bullet(A\times Q)^{\mathbb{Z}}\bullet 0$. Let us prove by induction on $n\in\mathbb{N}$ that the patterns of $(A\times Q)(A^{2rn}\setminus\{0^{2rn}\})$ are forbidden in $\Omega_{\Delta_{F,S}}$. The base case is trivial (there are no such patterns). Now suppose it is true for $n\in\mathbb{N}$, and suppose there exists a configuration $x\in[(A\times Q)0^{2rn+k}(A\setminus\{0\})]_0\cap\Omega_{\Delta_{F,S}}$ with $1\le k\le 2r$. Consider a preimage $y\in\Omega_{\Delta_{F,S}}$ of x. On the one hand, in cell 0 of y, we must have applied case (3), so $y_{[\![-r,+r]\!]}\in(A\times Q)^{2r+1}$, and this word does not involve γ. On the other hand, if we have applied case (1) in cell $2nr+k+1$ of y, then $y_{[\![(2n-1)r+k+1,(2n+1)r+k+1]\!]}\in(A\setminus\{0\})^{2r+1}$, but the space between these two neighborhoods is $(2n-1)r+k+1-r-1\le 2nr-1$, which contradicts the induction hypothesis. The other possibility was that we have applied case (2) in cell $2nr+k+1$, which involves a state γ among cells of $y_{[\![(2n-1)r+k+1,(2n+1)r+k+1]\!]}$, which contradicts Lemma 3.6. In the limit, and with a symmetric argument on the left, we obtain that all the configurations of $\Omega_{\Delta_{F,S}}\setminus A^{\mathbb{Z}}$ are in $0\bullet\Sigma\bullet 0$. ■

We shall abusively denote $A^{\mathbb{Z}}\times X_S=\left\{(a_i,s_i)_{i\in\mathbb{Z}}\in(A\times Q)^{\mathbb{Z}}\middle|\,(s_i)_{i\in\mathbb{Z}}\in X_S\right\}$.

Lemma 3.8. $\Omega_{\Delta_{F,S}}=A^{\mathbb{Z}}\cup 0\bullet(A^{\mathbb{Z}}\times X_S)\bullet 0$.

Proof. Thanks to Lemmas 3.7 and 3.5, it is enough to prove two inclusions for the configurations $x\in C^{\mathbb{Z}}$ with $l,m\in[\![-\infty,+\infty]\!]$ such that $x_{]\!]l,m[\![}\in(A\times Q)^{m-l-1}$ and for any $i\notin]\!]l,m[\![$, $x_i=0$.

First, suppose that $x\in\Omega_{\Delta_{F,S}}$, *i.e.* for any $t\in\mathbb{Z}$, there exists $x^t\in\Delta^t_{F,S}(\{x\})$. By recurrence, we can see that $x^{-t}_i\in A\times Q'$ for all $i\in]\!]l-rt,m+rt[\![$ and $t\ge 1$ since states from $A\times Q$ are only produced by case (3) of the rule. Let $w^{-t}\in Q'^{m-l+2rt+1}$ be the projection of $(x^{-t})_{]\!]l-rt,m+rt[\![}$ on its second component. Clearly, w^0 is in the language of X_S. We deduce that $x=x^0\in 0\bullet(A^{\mathbb{Z}}\times X_S)\bullet 0$.

Conversely, suppose that $x\in 0\bullet(A^{\mathbb{Z}}\times X_S)\bullet 0$, *i.e.* there is a sequence (y^t) with, for $t\ge 1$, $y^t\in Q'^{\mathbb{Z}}$ and $y^t=S(y^{t+1})$ and, for any $t\in\mathbb{N}$ and any $i\in]\!]l,m[\![$, $x=(a_i,S^t(y^t)_i)$ for some $a_i\in A$. Now take the configuration $\tilde{y}^t\in C^{\mathbb{Z}}$ such that for any $i\notin]\!]l-rt,m+rt[\![$, $\tilde{y}^t_i=0$, and for any $i\in]\!]l-rt,m+rt[\![$, $\tilde{y}^t_i=(b_i,y^t_i)$ with $b_i\in A$, and $b_i=a_i$ if $i\in]\!]l,m[\![$. By a direct recurrence, for any $j<t$ and any $i\notin]\!]l-rt+rj,m+rt-rj[\![$, we have $\Delta^j_{F,S}(\tilde{y}^j)_i=0$ and for any $i\in]\!]l-rt+rj,m+rt-rj[\![$, we have $\Delta^j_{F,S}(\tilde{y}^j)_i=(b_i,S^j(y^j)_i)$ (since $y_j\in Q'\in\mathbb{Z}$ case 3 of the definition of $\Delta_{F,S}$ applies at position i of $\tilde{y}^j$). This gives that $\Delta^t_{F,S}(y)=x$. We have proven that $\Omega_{\Delta_{F,S}}\cap 0\bullet(A\times Q)^{\mathbb{Z}}\bullet 0=0\bullet(A^{\mathbb{Z}}\times X_S)\bullet 0$. ■

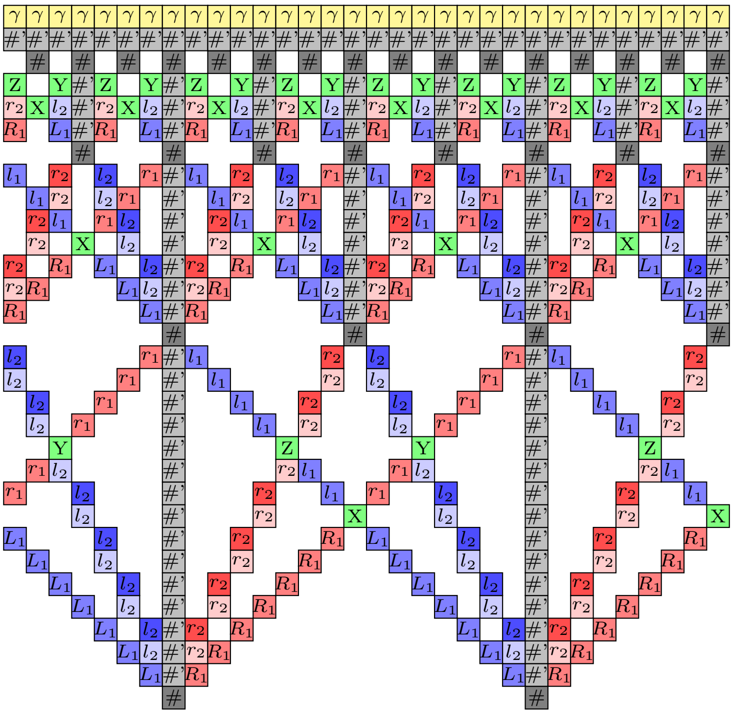


Figure 1: The 16-state firing squad of [Kar94]. Empty spaces represent the blank state.

Corollary 3.9. $\Omega_{\Delta_{F,S}}$ *has an NL-recognizable language.*

Proof. From Lemma 3.8 and Point 3 of Lemma 3.2, the language of the limit set is the finite boolean combination of finite concatenation of NL-recognizable languages. ■

Proof of Theorem 3.3. Let F be a CA on some alphabet A. We can artificially add some spreading state $0 \notin A$ to build a CA $\tilde{F}$ on alphabet $A \sqcup \{0\}$ which admits F as a sub-automaton. Now we have seen that $\tilde{F}$ is a sub-automaton of $\Delta_{\tilde{F},S}$. From Corollary 3.9, the corresponding limit set has an NL-recognizable language. ■

4. Analysis of a firing-squad CA

More precise proofs can be found in [GMT10].

Let S be the firing-squad CA defined in [Kar94]. It has a state set Q of size 16, including a killer state κ, radius 1 and is defined by the transitions appearing in Figure 1: precisely, any transition which is not in the space-time diagram of the figure produces the killer state κ. The complete list of transitions is given in [Kar94].

We are interested in history diagrams in X_S, *i.e.* mapping from $\mathbb{Z} \times \mathbb{N}$ to Q of the form: $(z, t) \mapsto x^t(z)$ where (x^t) is a sequence of configuration in X_S such that $S(x^{t+1}) = x^t$. We call them *valid* history diagrams.

When restricted to X_S, the behavior of S is easier to understand via a signal/collision evolving in a quiescent background. More precisely, the background is uniform and made of blank states (denoted B in the sequel) and the signals involved are:

signal	L_1	l_1	l_2	#'	r_2	r_1	R_1
speed	-1	-1	-1/2	0	1/2	1	1

The valid collisions are:

$$
\begin{aligned}
L_1 + R_1 &\to l_1 + r_1 & l_2 + r_2 &\to \# \\
l_1 + r_2 &\to l_1 + r_2 & l_1 + \#' + r_1 &\to \# \\
r_1 + l_2 &\to r_1 + l_2 & \# &\to L_1 + l_2 + \#' + r_2 + R_1
\end{aligned}
$$

Any other intersection of signal is invalid (it raises a κ state). Moreover, the last collision rule (starting from a single #) is valid only if the # is distant from any other # by at least 3 cells: if they are 1 cell away, 3 adjacent #′ are generated; if they are 2 cells away, a κ is generated 2 steps later.

To simplify proofs, we will often make reasonings over (portions of) "Euclidean" versions of history diagrams. A Euclidean history diagram is a set of labelled points and labelled (half-)lines or segments in $\mathbb{R}^2$ satisfying the following rules:

- points are only at integer coordinates ($\mathbb{Z}^2$) and labelled by #;
- (half-)lines and segments correspond to signals listed above (label and slope correspond);
- any intersection between lines or segments follow the collision rules above.

Lemma 4.1. *To each history diagram D, we can associate a valid Euclidean history diagram E such that, at any integer coordinate of E containing a point (#) or a single signal, the label gives the state of the corresponding position in D.*

This lemma allows the following proof scheme (used several times below): supposing by sake of contradiction that some word w occurs in a history diagram, we make a reasoning on the corresponding Euclidean diagram, we get a contradiction and finally deduce that no history diagram exists which contain the word w, and therefore that w is not in the language of X_S.

S satisfies points (1) and (2) of Lemma 3.2 as shown in [Kar94, Prop. 4.3]. We give below a complete characterization of the language of X_S which shows that it is NL-recognizable.

Lemma 4.2. *Consider a history diagram containing a word $w \in \#Q^*\#'$ at time t_0. Let z_1 (resp. z_2) be the cell where the first (resp. the last) letter of w occurs. We suppose in addition that the left # of w was created by a l_1 signal. Let t_1 be the first time step in the past when the cell z_1 is in state #, and t_2 be the first time step in the past when the cell z_2 is in state #. Then, both t_1 and t_2 exist.*

We denote by Σ the set $Q' \setminus \{\#, \#'\}$.

Lemma 4.3. *There is no history diagram containing a word w of the form $\#\Sigma^*\#'$, where the left # is created by r_1/l_1 signals.*

Lemma 4.4. *There is no history diagram containing a word w of the form $\#\Sigma^*\#'$, where the left # was created by a l_2/r_2 pair of signals.*

Lemma 4.5. *Any configuration from X_S with at least two # is of the following form, for some value of n: ${}^{\omega}(\#B^n)^{\omega}$*

Lemma 4.6. *Let L be the language of configurations from X_S admitting an history diagram where two # occur at some time t in the past. Then $L \in$ NL – recognizable in logarithmic space.*

Lemma 4.7. *The language of configurations from X_S which contain only one state in $\{\#, \#'\}$ is also in* NL.

Lemma 4.8. *Let L be the language of configurations from X_S with two or more $\#'$ and having a history diagram with no $\#$. Then L is regular.*

Lemma 4.9. *The language of the configurations from X_S without any $\#$ or $\#'$ is regular.*

Proposition 4.10. *The language of X_S is in* NL.

Proof. There are several cases, and the disjunction on configurations allows to express the language of X_S as a union of 'simple' NL languages.

(1) Configurations with #s or #s. We can descibe the set of these configurations by :
$$^{\omega}\{L_1, l_1, B\}\{l_2, B\}^* A\{r_2, B\}^*\{R_1, r_1, B\}^{\omega}$$
where A is one of the following (possibly infinite) configurations:
   (a) A has exactly one state in $\{\#, \#'\}$. We conclude in this case with Lemma 4.7.
   (b) A has one $\#$, and at least one other $\#$ or $\#'$. Lemmas 4.3, 4.4 and 4.5 show that the configuration satisfy the hypothesis of Lemma 4.6, which allows to conclude.
   (c) A has at least two $\#'$ but do not contain any signal. Then Lemma 4.8 conclude.
   (d) A has at least two $\#'$, along with some signal(s) between two $\#'$. Denote by c the global configuration in this case. We can simply go back a few steps in the past to find out a configuration c' of case 1b. Then we can apply Lemma 4.6 to c.

(2) Configurations without #s nor $\#'$s. This case is treated in Lemma 4.9.

Thus, since we have described above why each of the possible languages could be recognized in NL, we can just build a non-deterministic machine beginning by making a non-deterministic choice between all of these machines, then doing the computation of the chosen one. ■

Conclusion

We have thus achieved results implying that both limit set and column factors complexities are strongly linked to the factor simulation hierarchy; on the other hand, they are rather orthogonal to the sub-automaton simulation hierarchy.

Many open questions remain.

- We have obtained that universality was not forbidden by some rather strong constraints either on the limit set, or "orthogonally", on the column factors. A natural question is whether we can constrain both at the same time. The two constructions may possibly be composed together, at the price of a (yet) more difficult proof of the NL-recognizability of the limit set.
- Similarly, we believe that our results still hold when alphabets are restricted to $\{0, 1\}$ but at the price of a more technical proof.
- Is there an intrinsically universal CA with an SFT limit set? Following our construction, this raises immediately the following question: is there a firing-squad CA with an SFT limit set?

- What kind of limit system can an intrinsically universal CA have? Can it be injective?
- Is injectivity, expansivity of CA preserved by factor maps?
- Is it enough, for a CA to be a factor of another CA, that the corresponding column factors with some given width be linked by a factor map?
- Is there, for some complexity level λ, an equivalence class for the sub-automaton simulation (with space-time rescalings) of which all the elements have limit sets of complexity λ?

References

[BM97] François Blanchard and Alejandro Maass. Dynamical properties of expansive one-sided cellular automata. *Israel Journal of Mathematics*, 99:149–174, 1997.

[CFG10] Julien Cervelle, Enrico Formenti, and Pierre Guillon. Ultimate traces cellular automata. In Jean-Yves Marion, editor, 27^{th} *International Symposium on Theoretical Aspects of Computer Science (STACS'10)*, Nancy, March 2010.

[ČPY89] Karel Čulik II, Jan K. Pachl, and Sheng Yu. On the limit sets of cellular automata. *SIAM Journal on Computing*, 18(4):831–842, 1989.

[DMOT10a] Marianne Delorme, Jacques Mazoyer, Nicolas Ollinger, and Guillaume Theyssier. Bulking I: an abstract theory of bulking. HAL:hal-00451732, January 2010.

[DMOT10b] Marianne Delorme, Jacques Mazoyer, Nicolas Ollinger, and Guillaume Theyssier. Bulking II: Classifications of cellular automata. HAL:hal-00451729, January 2010.

[Fio00] Francesca Fiorenzi. The Garden of Eden theorem for sofic shifts. *Pure Mathematics and Applications*, 11(3):471–484, 2000.

[Gil88] Robert H. Gilman. Notes on cellular automata. manuscript, 1988.

[GMM93] Eric Goles, Alejandro Maass, and Servet Martínez. On the limit set of some universal cellular automata. *Theoretical Computer Science*, 110:53–78, 1993.

[GMT10] Pierre Guillon, Pierre-Étienne Meunier, and Guillaume Theyssier. Clandestine simulations in cellular automata. HAL:hal-00521624, September 2010.

[Hur90] Lyman P. Hurd. Nonrecursive cellular automata invariant sets. *Complex Systems*, 4:131–138, 1990.

[Kar94] Jarkko Kari. Rice's theorem for the limit sets of cellular automata. *Theoretical Computer Science*, 127(2):229–254, 1994.

[Kůr97] Petr Kůrka. Languages, equicontinuity and attractors in cellular automata. *Ergodic Theory & Dynamical Systems*, 17:417–433, 1997.

[Kůr03] Petr Kůrka. *Topological and symbolic dynamics*. Société Mathématique de France, 2003.

[Nas95] Masakazu Nasu. *Textile Systems for Endomorphisms and Automorphisms of the Shift*, volume 114 of *Memoirs of the American Mathematical Society*. American Mathematical Society, Providence, Rhode Island, March 1995.

[Oll02] Nicolas Ollinger. *Automates cellulaires: structures*. PhD thesis, École Normale Supérieure de Lyon, December 2002.

[Oll03] Nicolas Ollinger. The intrinsic universality problem of one-dimensional cellular automata. In Helmut Alt and Michel Habib, editors, 20^{th} *Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, volume 2607 of *Lecture Notes in Computer Science*, pages 632–641, Berlin, Germany, February 2003. Springer-Verlag.

[Taa07] Siamak Taati. Cellular automata reversible over limit set. *Journal of Cellular Automata*, 2(2):167–177, 2007.

[The05] Guillaume Theyssier. *Automates cellulaires: un modèle de complexités*. PhD thesis, École Normale Supérieure de Lyon, December 2005.

[vN66] John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[Wei73] Benjamin Weiss. Subshifts of finite type and sofic systems. *Monatshefte für Mathematik*, 77(5):462–474, 1973.

Journées Automates Cellulaires 2010 (Turku), pp. 145-155

SLOPES OF TILINGS

EMMANUEL JEANDEL AND PASCAL VANIER

Laboratoire d'Informatique Fondamentale de Marseille, CMI - 39 rue Joliot-Curie, F-13453 Marseille Cedex 13, France
E-mail address, E. Jeandel: emmanuel.jeandel@lif.univ-mrs.fr
E-mail address, P. Vanier: pascal.vanier@lif.univ-mrs.fr

Abstract. We study here slopes of periodicity of tilings. A tiling is of slope θ if it is periodic along direction θ but has no other direction of periodicity.

We characterize in this paper the set of slopes we can achieve with tilings, and prove they coincide with recursively enumerable sets of rationals.

1. Introduction

The model of tilings was introduced by Wang [14] to study fragments of the first order theory. This model is described by geometrical local properties, deciding whether a given tile can be placed on a given cell based only on its surrounding neighbours.

While the definition of tilings is deceptively simple, they exhibit complex behaviours. As an example, the most basic problem (decide if a given tiling system can tile the plane) is undecidable [2]. This is due to both a straigthforward encoding of Turing machines in tilings [3, 4, 13] and to the existence of so-called *aperiodic* tiling systems [8, 12], that can tile the plane but in no periodic way.

In this paper we explore the periodic behaviour of tiling systems. Periodic tilings have nice closure properties, in the sense that the image of a periodic point by a shift-preserving morphism (i.e. a block map) is again a periodic point. As a consequence, understanding the structure of the periodic points of a tiling system is a first step to decide when some tiling system embeds in another, or when two tilings systems are "isomorphic" (more accurately conjugate [9])

In dimension one, the question boils down to determine for a tiling system τ the set of integers n so that there is a valid tiling by τ of period (exactly) n. This question was answered succesfully: Using automata theory, a complete characterization of the set of integers we can obtain this way was obtained [9].

The question is more delicate in two dimensions. We might break it down in two parts: Given a tiling system τ,

- For which n is there a tiling of horizontal and vertical period n ?
- For which direction θ is there a tiling which is periodic only along direction θ ?

Key words and phrases: tilings, formal languages, foundations of computing.
Both authors are party supported by ANR-09-BLAN-0164.

The authors gave an answer to the first question in [6]: Sets of integers we can obtain correspond to the complexity class **NE**. We deal in this paper with the second question, characterizing the set of *slopes* we can obtain by tiling systems.

While the answer in dimension one involves finite automata theory, it turns out that the good tool to solve the problem in higher dimensions is computability theory. The undecidability of the domino problem (deciding if a tiling system tiles the plane) is indeed not an anomaly: many combinatorial aspects of tilings can only be fully comprehended by means of recursivity theory arguments [1, 5, 10].

Along these lines, we will prove here the following theorem:

Theorem 1.1. *The sets of slopes of tilings are exactly the recursively enumerable sets of rationals.*

As a consequence, one might for example build a tiling system which admits slopes arbitrary close to 0, but does not admit 0 as a slope.

This paper is organized as follows. We first give the definition of tiling systems, and an encoding of Turing machines that will be used later. Then we proceed to the proof of the theorem. The main part of this paper is a construction, for any recursively enumerable set R, of a tiling system with R as a set of slopes.

2. Definitions

2.1. Tilings

Usually when considering tiling systems, Wang rules are used. We use here a generalization that is equivalent in terms of expressivity but makes the constructions easier.

While Wang rules consider only adjacent tiles only, our rules may consider an arbitrary large (but finite) neighborhood of tiles.

A *tiling* of $\mathbb{Z}^2$ with a finite set of tiles T is a mapping $c : \mathbb{Z}^2 \to T$. A *pattern* of neighborhood $N \subseteq \mathbb{Z}^2$ is a mapping from N to T. A pattern is finite if N is finite. A *tiling system* is a pair (T, F), where F is a finite set of finite patterns. A tiling c is said to be *valid* if and only if none of the patterns of F ever appear in c. Since the number of forbidden patterns is finite, we could specify the rules by *allowed* patterns as well. We give an example of such a tiling system with the tiles of figure 1a and the forbidden patterns of figure 1b. The allowed tilings are shown in figure 2.

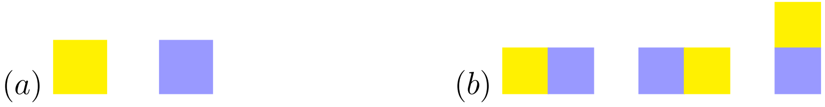


Figure 1: The set of tiles (a) and the forbidden patterns (b).

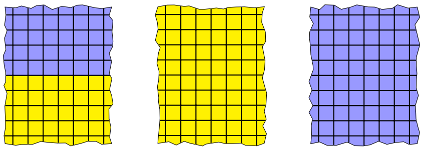


Figure 2: The only valid tilings of the system.

2.2. (a)periodicity

A tiling c is *periodic of period* $v = (v_x, v_y) \in \mathbb{Z}^2$ if for all points $x, y \in \mathbb{Z}$, $c(x,y) = c(x+v_x, y+v_y)$. The *direction* of a vector $v \neq (0,0)$ is $\theta = v_y/v_x \in \mathbb{Q} \cup \{\infty\}$ with the convention $\theta = \infty$ if $v_x = 0$.

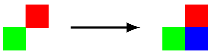
A tiling is *periodic along a direction* θ if it is periodic of period $v \neq (0,0)$ and v is of direction θ.

For a given tiling c, there are three cases:

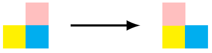
- Either c is periodic of period v, w and v, w are of different directions. In this case, the tiling c is *biperiodic*: there exists an integer $n \in \mathbb{N}$ (the period) so that $c(x,y) = c(x+n,y) = c(x,y+n)$, and as a consequence c is periodic along all directions $\theta \in \mathbb{Q} \cup \{\infty\}$
- c is periodic along one direction θ only. In this case, we will call θ the *slope* of c.
- c has no nonzero vector of periodicity. c is then called aperiodic.

The set of slopes of a tiling system τ, noted $\mathcal{S}_\tau$, is the set of the slopes of all valid tilings by τ. As an example, the first tiling in fig.2 is periodic of vector $(1,0)$ (hence of slope 0) and the two other tilings are biperiodic (hence have no slope). As a consequence, $\mathcal{S}_\tau = \{0\}$ for this example. Using rotated versions of this elementary tiling system, we can produce for each $\theta \in \mathbb{Q} \cup \{\infty\}$ a tiling system τ so that $\mathcal{S}_\tau = \{\theta\}$.

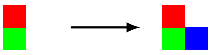
A tiling system is *aperiodic* if and only if it tiles the plane but all valid tilings are aperiodic. Such tiling systems have been shown to exist [2] and are at the core of the undecidability of the *domino problem* (decide whether a given tiling system admits a valid tiling). J. Kari [7] gave such a tiling system with an interesting property: determinism. A tiling system is *NW-deterministic* (for *North-West*) if it is given by forbidden patterns of shape ⊞ and given two tiles respectively at the north and west of a given cell, there is at most one tile that can be put in this cell so that the finite pattern is valid. The mechanism is shown below:



If we modify the forbidden patterns of this tiling system in the following way :



a tile will be forced by the one on its west and on its northwest, we will call this East-determinism :



East-determinism has the interesting property that if we set a whole column of the plane then the whole half plane on its east will be determined by it. Moreover, this tiling system is also aperiodic (the tilings are skewed versions of the original one; diagonal lines are transformed into columns).

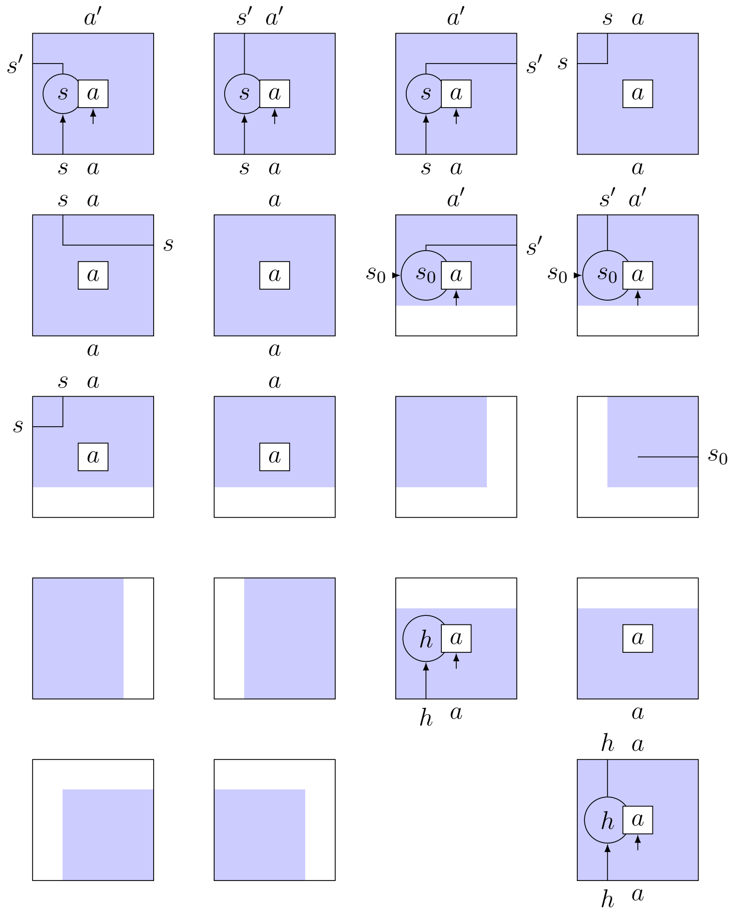


Figure 3: A tiling system, given by Wang tiles, simulating a Turing machine : the states are in the circles and the tape is in the rectangles.

2.3. Computability

The undecidability of the domino problem [2] hinted earlier also comes from a straightforward encoding of Turing machines into tilings. We provide here such an encoding for future reference.

For a given Turing machine M, consider the tiling system τ_M presented in figure 3. The tiling system is given by *Wang tiles*, i.e., we can only glue two tiles together if they coincide on their common edge. We now give some details on the picture:

- s_0 in the tiles is the initial state of the Turing machine.
- The first tile corresponds to the case where the Turing machine, given the state s and the letter a chose to go to the left and to change from s to s', writing a'. The two other tiles are similar.
- h represents a halting state. Note that the only states that can appear in the last step of a computation (before a border appears) are halting states.

This tiling system τ_M has the following property: there is an accepting path for the word u in time (less than) t using space (less than) w if and only if we can tile a rectangle of size $(w + 2) \times t$ with white borders, the first row containing the input.

3. The sets of slopes are recursively enumerable

We say that a subset S of $\mathbb{Q} \cup \{\infty\}$ is *recursively enumerable* if there exists a Turing machine M that on input $(p, q) \in \mathbb{Z}^2 \neq (0, 0)$ halts if and only if $q/p \in S$.

$$\begin{array}{lll} \theta \in S & \implies & \forall (p,q), q/p = \theta, M \text{ halts on } (p,q) \\ \theta \notin S & \implies & \forall (p,q), q/p = \theta, M \text{ does not halt on } (p,q) \end{array}$$

The exact definition is irrelevant as all reasonable definitions will give rise to the same class. An alternative interesting definition is as follows: A set S is recursively enumerable if there exists a Turing machine M so that

$$\theta \in S \iff \exists (p,q), q/p = \theta \wedge M \text{ halts on } (p,q)$$

Using a known projection technique to go down to dimension 1, we prove here:

Lemma 3.1. *For any tiling system τ, S_τ is recursively enumerable.*

Proof. We first give a procedure to decide if there is a tiling which is $(n, 0)$-periodic. Let k be an integer bigger than the size of any forbidden pattern in τ.

If w is a pattern of support $[0, n-1] \times [0, l]$ for some l, we write $w^{\mathbb{Z}}$ for the pattern of support $\mathbb{Z} \times [0, l]$ defined by $w^{\mathbb{Z}}_{i,j} = w_{(i \mod n),j}$, that is for the horizontal repetition of w.

Let V be the set of all patterns w of size $n \times k$ so that $w^{\mathbb{Z}}$ is correctly tiled. Consider this a directed graph G, where there is an edge from v to w if and only if $(v \otimes w)^{\mathbb{Z}}$ is correctly tiled, where $v \otimes w$ denotes the pattern of size $n \times 2k$ obtained by putting w above v.

It is then clear that tilings of period $(n, 0)$ correspond to biinfinite walks on this graph, so that there exists a tiling of period $(n, 0)$ if and only if there exists a cycle in the graph G. Furthermore, there exist a tiling of period $(n, 0)$ which is not biperiodic if and only if we can find two distinct cycles C_1, C_2 in the graph so that C_2 is accessible from C_1. All the construction is clearly algorithmic.

Now for a given (p, q) we use the same procedure, where w is a pattern of size $|p| \times k|q|$ and $w^{\mathbb{Z}}$ is of support $\{(i + np, j + nq), i \leq |p|, j \leq k|q|\}$ and defined by $w^{\mathbb{Z}}_{i+np,j+nq} = w_{i,j}$.

The following algorithm gives then the expected result: Starting from a given (p, q), test all (p', q') so that $q'/p' = q/p$ to see if there exists a tiling which is (p', q')-periodic but not biperiodic. ■

4. The recursively enumerable sets are sets of slopes

Lemma 4.1. *For any recursively enumerable set $R \subseteq \mathbb{Q} \cup \{\infty\}$, there exists a tiling system τ, such that $\mathcal{S}_\tau = R$.*

Proof. We use for this proof techniques similar to [6]. We will construct for each Turing machine M, corresponding to a recursively enumerable set R, a tiling system τ whose slopes are exactly the rationals θ accepted by M. We assume that M takes θ as an input under the form (p, q) in binary and that its input depends only on q/p.

We will first build a tiling system τ that has as slopes $\{\theta \in R | 0 < \theta < 1\}$. The other cases are treated in the same way and the final tiling system is the disjoint union of the tiling systems treating each case. The special cases $\theta = 0$, $\theta = \infty$, and $\theta = \pm 1$ will be shortly discussed later on.

For the particular case where $p > q > 0$ we want to enforce the fact that when a tiling of the plane has exactly one direction of periodicity, this direction of periodicity has to be accepted by the Turing machine M. The tiling τ_M will enforce the skeleton described in figure 4, where each square encodes the computation by M proving that the slope θ is accepted. For this, we need the size of the square to be arbitrarily large independently of θ, so that the computation of M has enough time to accept. This skeleton in itself could be biperiodic, we will then color the background of each square to ensure the existence of tilings with only one direction of periodicity.

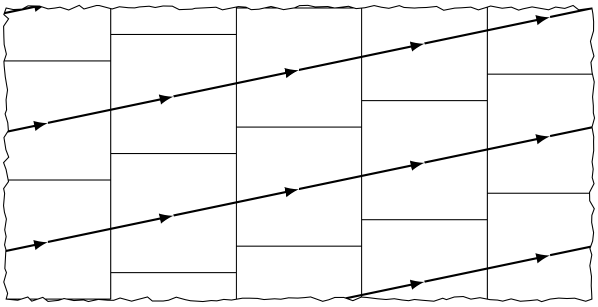


Figure 4: Skeleton of the tiling : when the tiling is periodic, the squares appear and each of them is the shifted version of its lower left neighbor. Inside the squares we will encode the Turing machine.

In order to enforce this skeleton, we will use several layers (or components), each of them having their own aim, and impose some contraints on how the layers may combine. We give here $\tau_M = C \times R \times W \times S \times P \times T_M \times A$ where :

- C will allow us to make the rows and columns,
- R to make the squares,
- W to force the periodicity vector and to write the input for the Turing machine,
- S to force the aperiodic background of the squares to be the same,
- P will reduce the size of the input,
- T_M will code the Turing machine M,
- A will allow slopes of unique periodicity to appear.

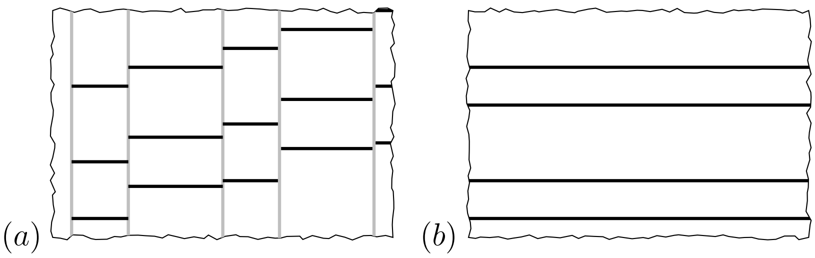


Figure 5: Valid periodic tilings are formed of columns of vertical breaking tiles (a) or of rows of horizontal breaking tiles (b). Between two columns of vertical breaking tiles there can be rows of horizontal breaking tiles.

We will now proceed to the details of the proof, by giving each component and explaining what it enforces.

Component C**:** The first component is made of an **East-deterministic** aperiodic set of tiles that we will call white tiles (the white background of figure 4), and we add two sets of tiles the horizontal breaking tiles {■} and the vertical breaking tiles {◨, ◧, ▩, □} (the horizontal and vertical lines of figure 4). The rules are simple :

- on the left of a ■ there can only be a ■ or a ◨,
- on the right of a ■ there can only be a ■ or a ◧,
- above and below a ■, there can only be a white,
- above a ◨ can only be a ▩,
- above a ▩ can only be a ◧ or a ▩,
- above a ◧ can only be a □,
- above a □ can only be a ◨ or a □.

To put it in a nutshell, it means that horizontal breaking tiles forms rows that can only be broken by vertical breaking tiles, and vertical breaking tiles can only form columns that cannot be broken.

In a periodic tiling, we cannot have a quarter of plane filled with white (aperiodic tiles). As a consequence, periodic tilings at this stage are necessarily formed by a white background broken *infinitely many times* by horizontal or vertical breaking tiles.

One more rule we add is that the rules on white tiles "jump" over the black tiles. That is to say if we remove a black row, then the white tiles have to glue themselves together correctly. The valid tilings at this stage are represented on figure 5.

Component R**:** . The next component will force the apparition of squares between two columns of vertical breaking tiles and prevent several infinite rows of horizontal breaking tiles to appear. This layer is made of the set of tiles {◐, ⬓, ◥, 🟥, 🟨, ◩, ◪}, the rules applied on this layer are given by Wang tiles. We superimpose the rules as follows :

- ◐ can only be superimposed to ▩,□,
- ⬓ can only be superimposed to ■,
- ◪ goes on ◨, and ◩ goes on ◧,
- 🟥,🟨,◥ are superimposed to the white tiles.

Figure 6 shows how this component R forces rows of black tiles to appear between two gray columns. The distance between these black rows is exactly

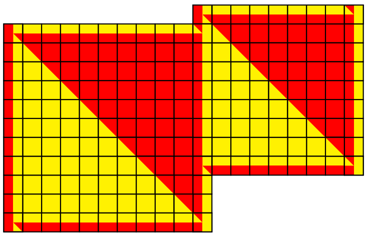


Figure 6: Component R forces squares.

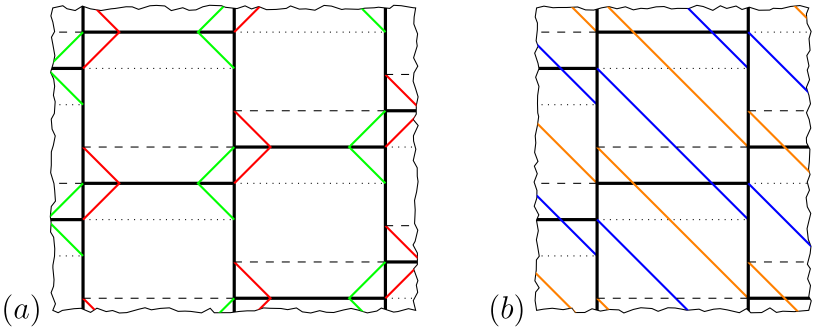


Figure 7: The dotted row (resp. dashed) corresponds to the prolongation on the right (resp. left) of the black cells. In (a) the signals sent from the extremities of the rows forming the square forces the offset between rectangles of three neighboring columns to be exactly the same for any of them. In (b) the signals sent from the extremities force the distance between columns to be identical.

the distance between the gray columns thus black rows and gray columns form squares. At this stage the valid periodic tilings cannot be formed of only rows of black tiles anymore.

Component W: What this component does is that it synchronises the offsets between squares of two neighboring columns, and forces all columns to be at equal distance of their two neighboring columns, for all of them. As a side effect, it also writes the offset between two squares (which we call q) in each square. In order to do that, what we do is that we prolongate the black rows of each column into their direct neighbors with two new layers, one for the left and one for the right. The end of the black row then sends a diagonal signal which changes its direction when it collides with the projected lines of the neighbors and its colision with the column has to coincide with the projection of the other column. Figure 7.a shows how this mechanism works. The collision of the signal sent on the right extremity of the black lines marks the end of the input q on each square. We add two other sublayers to make the white rows of same width. The first one sends a signal from the left extremity of a black line which has to meet the next column at the exact point of the extension of the square. The second one does the same for the right extremity. Figure 7.b shows these signals.

Component S: This component is meant to synchronize the aperiodic backgrounds of all the squares. In order to do that, we only need to transmit the

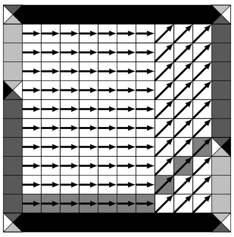


Figure 8: Tiles allowing to transmit the aperiodic background.

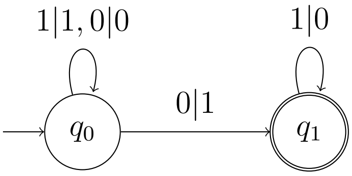


Figure 9: A transducer tranforming n in binary into $n + 1$.

first column after a vertical breaking column since our initial aperiodic tiling system is East-deterministic.

In order to do that, we take these tiles {⍈, ⧄, ◩, ▣}, with the following rules :

- on the right,above and below a ⧄ there can only be a ⧄ or a ◩.
- on the left of a ⧅ we necessarily have a ◩ and the south western neighbor of a ◩, if the tile is a white, is a ◩ or a ▣,
- the lower left white tile of a square is necessarily a ▣. The rules on ▣ is that there can only be a ▣ or a ⧄ on a white tile to its right,
- the vertical/horizontal breaking tiles have necessarily a ⍈ on them.

The tiling obtained inside a square is shown on figure 8. We add a sublayer that is a copy of the white tiles with the rules that the tiles of this component on the right of this column are identical to the white ones on component C and that this copy is transmitted to the tile pointed by the arrow. Then with the property that the black tiles continue the rules on the whites, the whole aperiodic background between two vertical breaking columns is exactly the same but shifted by the offset.

Component P**:** Now each square contains two data: its size (p) and the offset to the next square q, both in unary. We will pass them as input to the Turing machine after some transformation.

The idea is to transform the unary input (p, q) into a smaller binary one (p', q') where $\gcd(p', q')$ is not a multiple of two. Doing that is fairly easy : we first need to convert the input in binary; this can be done by the iteration of the transducer of figure 9: starting from $000 \ldots 00$ we obtain the binary representation of p (least significant bit on the rightmost part) in p iterations of the transducer. Then we strip the binary representation of p and q of their common last zeroes.

Component T_M**:** This layer implements the Turing machine M, the input has been computed by layer P. Note that the Turing machine has to halt for the tiling to be valid.

Component A: This layer is made of only two tiles, a yellow and a blue one. It will be superimposed to white tiles and to the ■ of the vertical breaking tiles of component C only. The rules are that two neighboring tiles (horizontally and vertically) have the same color. It is easy to see that the color is uniform inside a square and that it spreads to the upper right and lower left neighboring squares. Thus the squares along the direction of periodicity have the same color.

We now prove that the preceding construction works.

(1) Any slope is an accepted input of M: Let $\theta = q/p \in S_\tau$ be a slope of periodicity of τ, with $p > q > 0$ relatively prime.

By construction, the tiling has to be formed of squares of identical size with constant offset (components C, R, W). Their aperiodic background has to be the same on each column (component S), so that in fact the tiling is periodic along direction (m, n) where m and n denote respectively the width and offset of the tiling. As a consequence, the tiling is of slope $\theta = m/n = q/p \in]0; 1[$ and we have $(n, m) = 2^k k'(p, q)$ for some k, k' with k' odd.

Now the Turing Machine on each square has $(k'q, k'p)$ as an input and halts. Hence the slope $k'q/k'p$ is accepted by the machine, so $q/p \in R$, which proves $S_\tau \subseteq R \cap]0; 1[$.

(2) Any accepted input of M is a slope of some tiling: Let $\theta \in R$ be an accepted input of M with $\theta = q/p$, $p > q > 0$ and p, q relatively prime.

There exists a time t and a space s such that M accepts (p, q) in time t and space s and $s \leq t$. Take $(m, n) = 2^{\lceil \log t \rceil}(p, q) \geq (t, s)$ Now the $m \times m$ square is big enough for the computation on input (p, q) to succeed. Hence there is a tiling of period (m, n) and component A allows us to make the direction of periodicity unique by dividing the plane into two colors, half a plane yellow and half a plane blue. Hence $R \cap]0; 1[\subseteq S_\tau$.

This finishes the proof for the case $0 < \theta < 1$, i.e. $p > q > 0$.

The cases where $q > p > 0$, $-p > q > 0$, or $q > -p > 0$ are treated in a very similar way: rotating the tiling system we just constructed and changing the way the input is written on the tape (to invert the inputs, or add a minus sign) is enough. However the remaining cases ($p = \pm q, p = 0, q = 0$) need special treatment[1].

For these cases, the construction above does not work, by that we mean that just rotating it and modifying slightly the Turing machine of component T_M won't do the trick. However it is actually simpler. We now make squares facing one another, obtaining a regular grid. This requires less tiles for component C and no component W. Then according to the case, components C,S and A are modified as follows:

- for $p = q$ ($\theta = 1$), S just transmits diagonally the tiles. In component A, the color is synchronized from the top right corner to the next square at the north east. The case $p = -q$ is similar.
- for $q = 0$ ($\theta = 0$), S transmits horizontally, and the colors of component A are synchronized with the square on the right. The tiling can only be horizontally periodic if the Turing machine accepts it, this is the only way it can be periodic.

[1] As this corresponds to four specific different θs, note that we could treat them nonconstructively, adding if necessary four new tiling systems having predescribed slopes.

- for $p = 0$ ($\theta = \infty$), C has, instead of an east deterministic tileset, a north deterministic one. Components S and A are modified accordingly. The tiling can only be vertically periodic if the Turing machine accepts it and this is the only way it can be periodic. ■

5. Concluding remarks

We have shown that the sets of slopes of periodicity of tilings correspond exactly to the recursively enumerable (Σ_1^0) sets of rationals for tilings in dimension 2. Our intuition for analogous results in higher dimensions would be that the slopes of periodicity would then be characterized by Σ_2^0 sets [11], since knowing whether a tiling is periodic of vector v in dimension 3 is not decidable anymore but only Π_1^0. Hence the following conjecture:

Conjecture 5.1. The sets of slopes of tilings in dimension $d \geq 3$ are exactly the Σ_2^0 subsets of $(\mathbb{Q} \cup \{\infty\})^{d-1}$.

An analogous construction to the one detailed here should work at least for dimension 3, it would however be tedious.

References

[1] Nathalie Aubrun and Mathieu Sablik. An order on sets of tilings corresponding to an order on languages. In *STACS*, pages 99–110, 2009.

[2] Robert Berger. *The Undecidability of the Domino Problem.* Number 66 in Memoirs of the American Mathematical Society. The American Mathematical Society, 1966.

[3] J. Richard Buchi. Turing-Machines and the Entscheidungsproblem. *Math. Annalen*, 148:201–213, 1962.

[4] Gregory Chaitin. The Halting Probability via Wang Tiles. *Fundamenta Informaticae*, 86(4):429–433, 2008.

[5] Michael Hochman and Tom Meyerovitch. A characterization of the entropies of multidimensional shifts of finite type. *Annals of Mathematics*, 2008.

[6] Emmanuel Jeandel and Pascal Vanier. Periodicity in Tilings. In *Developments in Language Theory (DLT)*, 2010.

[7] Jarkko Kari. The Nilpotency Problem of One-Dimensional Cellular Automata. *SIAM Journal on Computing*, 21(3):571–586, 1992.

[8] Jarkko Kari. Recent results on aperiodic Wang tilings. In M. Gromov P. Prusinkiewicz A. Carbone, editor, *Pattern formation in biology, vision and dynamics*, pages 83–96. World Scientific, Singapore, 2000.

[9] Douglas A. Lind and Brian Marcus. *An Introduction to Symbolic Dynamics and Coding.* Cambridge University Press, New York, NY, USA, 1995.

[10] Tom Meyerovitch. Growth-type invariants for $\mathbb{Z}^d$ subshifts of finite type and classes arithmetical of real numbers. arXiv:0902.0223v1.

[11] P.G. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics.* Elsevier, 1992.

[12] Raphael M. Robinson. Undecidability and Nonperiodicity for Tilings of the Plane. *Inventiones Math.*, 12, 1971.

[13] Peter van Emde Boas. The convenience of Tilings. In *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture Notes in Pure and Applied Mathematics.* CRC, 1997.

[14] Hao Wang. Proving theorems by Pattern Recognition II. *Bell Systems technical journal*, 40:1–41, 1961.

Journées Automates Cellulaires 2010 (Turku), pp. 156-167

TRANSDUCTIONS COMPUTED BY ITERATIVE ARRAYS

MARTIN KUTRIB AND ANDREAS MALCHER

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
E-mail address: {kutrib,malcher}@informatik.uni-giessen.de

Abstract. Iterative arrays are one-dimensional arrays of interconnected interacting finite automata. The cell at the origin is equipped with a one-way read-only input tape. We consider iterative arrays as transducers. To this end, the cell at the origin is additionally equipped with a one-way write-only output tape. The families of transductions computed are classified with regard to the time allowed to compute the input and the output, respectively. In detail, the time complexities of real-time and linear-time are of particular interest, for which a proper hierarchy is shown. In the second part of the paper, iterative array transducers are compared with the conventional transducer models, namely, finite state transducers and pushdown transducers. It turns out that all deterministic variants can be simulated by iterative array transducers. Moreover, nondeterministic but unambiguous finite state transducers can be simulated as well. When considering time constraints, incomparability results to almost all families are derived.

1. Introduction

Parallel processes and cooperating systems appear almost everywhere in today's world. However, the behavior of such systems, the interaction of different components, or the predictability of the behavior in the future is far from being completely understood. Iterative arrays (IA) are a model which allows to describe massive parallel systems, since they are arrays of identical copies of deterministic finite automata. Furthermore, the single nodes, except the node at the origin, are homogeneously connected to both their immediate neighbors, and they work synchronously at discrete time steps. The distinguished cell at the origin, which is called the communication cell, is equipped with a one-way read-only input tape.

In connection with formal language recognition IA have been introduced in [5]. To recognize a formal language, every word is read symbolwise from the input tape by the communication cell. An input is accepted if the communication cell enters an accepting state during the course of its computation. The computational capacity of IA has been widely studied in the literature and a recent survey may be found in [12].

Computational models are not only interesting from the viewpoint of recognizing some input, but also from the viewpoint of transforming some input into some

2000 ACM Subject Classification: F.1.1, F.1.2, F.4.3.

Key words and phrases: iterative arrays, transductions, finite state transducers, pushdown transducers.

output. For example, a parser for a formal language should not only return the information whether or not the input word can be parsed, but also the parse tree in the positive case. The simplest model in this context is the finite state transducer which is a finite automaton with an output alphabet that assigns to each input accepted at least one output word. Transductions computed by different variants of such transducers are studied in detail in [2]. Similarly, pushdown transducers are conventional pushdown automata where each input accepted is assigned to at least one output. Deterministic and nondeterministic variants are investigated in [1]. Furthermore, characterizations of pushdown transductions as well as applications to the parsing of context-free languages are given. In [9] the model of "one-way linear iterative arrays" is introduced. In this model an input is read at one end of the array and an output is written at the other end of the array. Additionally, the number of cells is a priori defined as the length of the input, the information flow is only from left to right, and the output does not depend on the fact whether or not the input is accepted.

In this paper, we will consider IA not only as a language recognizing device but as a language transforming device. Moreover, we are interested in the comparison with the conventional transducer models. To this end, we enhance the definition slightly by adding an output alphabet to an IA and, additionally, its communication cell is equipped with a one-way write-only output tape. Since IA are deterministic devices every input accepted corresponds to exactly one output. It is known that conventional IA can accept rather complicated languages such as $\{ a^p \mid p \text{ is prime} \}$ [7] or $\{ a^{2^n} \mid n \geq 1 \}$ [4] in real time. Thus, we are interested in fast transductions of IAs as well and consider the time complexities of real-time and linear-time. Additionally, we consider the time complexities of accepting the input and computing the output separately.

The paper is organized as follows. In Section 2 we define iterative array transducers and their computed transductions with respect to the time complexities of real-time and linear-time. The relations of the families of transductions with certain time constraints are investigated in Section 3. It turns out that there exists a proper inclusion between the transductions where input and output are computed in real time and those where input and output are computed in linear time. Moreover, the transductions where the input is computed in real time and the output is computed in linear time are located properly in between both families. Section 4 is devoted to comparing iterative array transducers with finite state transducers. If the given finite state transducer is deterministic, it can be simulated by an iterative array transducer where input and output are computed in real time. This is no longer true, if the given finite state transducer is nondeterministic, but unambiguous. However, such finite state transducers can be simulated by iterative array transducers where the output is computed in linear time. Finally, we compare iterative array transducers with pushdown transducers in Section 5. The main result is that an iterative array transducer may need linear time to simulate a deterministic pushdown transduction. On the other hand, there are iterative array transductions where input and output are computed in real time which cannot be realized by any nondeterministic pushdown transducer. Thus, several incomparability results can be derived.

2. Preliminaries and Definitions

We denote the rational numbers by $\mathbb{Q}$, and the non-negative integers by $\mathbb{N}$. For the empty word we write λ, the reversal of a word w is denoted by w^R, and for the length of w we write $|w|$. The cardinality of a set M is denoted by $|M|$. We write $\subseteq$ for set inclusion, and $\subset$ for strict set inclusion.

A (one-dimensional) iterative array transducer is a linear array of finite automata, sometimes called cells, where each cell except the leftmost one is connected to its both nearest neighbors. The distinguished leftmost cell is the so-called communication cell that is connected to its neighbor to the right and to the input/output supply. For convenience we identify the cells by non-negative integers.

Initially, all cells are in the so-called quiescent state. At each time step the communication cell reads an input symbol and writes a possibly empty string of output symbols. To this end, we have different local transition functions. All cells but the communication cell change their state depending on their current state and the current states of their neighbors. The state transition and output of the communication cell depends on its current state, the current state of its neighbor, and on the current input symbol (or if the whole input has been consumed on a special end-of-input symbol). The cells work synchronously at discrete time steps.

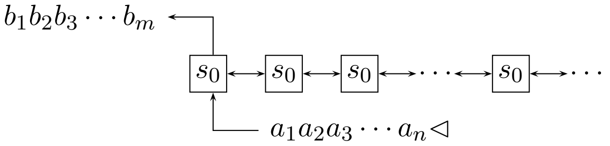


Figure 1: An iterative array transducer.

Definition 2.1. A *deterministic iterative array transducer* (IAT) is a system $\langle S, A, B, F, \lhd, s_0, \delta, \delta_0\rangle$, where

(1) S is the finite, nonempty set of *cell states*,
(2) A is the finite, nonempty set of *input symbols*,
(3) B is the finite set of *output symbols*,
(4) $F \subseteq S$ is the set of *accepting states*,
(5) $\lhd \notin A$ is the *end-of-input symbol*,
(6) $s_0 \in S$ is the *quiescent state*,
(7) $\delta : S^3 \to S$ is the *total local transition function for non-communication cells* satisfying $\delta(s_0, s_0, s_0) = s_0$,
(8) $\delta_0 : (A \cup \{\lhd\}) \times S^2 \to B^* \times S$ is the *partial local transition function for the communication cell.*

Let $\mathcal{M}$ be an IAT. A *configuration* of $\mathcal{M}$ at some time $t \geq 0$ is a description of its global state which is a triple (w_t, v_t, c_t), where $w_t \in A^*$ is the remaining input sequence, $v_t \in B^*$ is the output emitted so far, and $c_t : \mathbb{N} \to S$ is a mapping that maps the single cells to their current states. The configuration (w_0, λ, c_0) at time 0 is defined by the input word w_0, an empty output string, and the mapping $c_0(i) = s_0$, $0 \leq i$, while subsequent configurations are chosen according to the global transition function Δ. Let (w_t, v_t, c_t), $t \geq 0$, be a configuration. Then its successor configuration $(w_{t+1}, v_{t+1}, c_{t+1}) = \Delta(w_t, v_t, c_t)$ is as follows.

$$c_{t+1}(i) = \delta(c_t(i-1), c_t(i), c_t(i+1)),$$

for all $i \geq 1$. Furthermore, let $a = \lhd$, $w_{t+1} = \lambda$ if $w_t = \lambda$, and $a = a_1$, $w_{t+1} = a_2 \cdots a_n$ if $w_t = a_1 \cdots a_n$. Then, for $\delta_0(a, c_t(0), c_t(1)) = (v', s)$ we have

$$c_{t+1}(0) = s \text{ and } v_{t+1} = v_t v'.$$

Thus, the global transition function Δ is induced by δ and δ_0. The IAT halts when it enters a *halting* configuration, i.e., the transition function δ_0 is not defined for the current configuration. As usual we extend Δ to sequences of configurations and denote it by Δ^*. That is, Δ^0 is the identity, $\Delta^t = \Delta(\Delta^{t-1})$, $1 \leq t$, and $\Delta^* = \bigcup_{0 \leq t} \Delta^t$. Thus, $(w_t, v_t, c_t) \in \Delta^*(w, v, c)$ indicates that it is possible for $\mathcal{M}$ to go from the configuration (w, v, c) to the configuration (w_t, v_t, c_t) in a sequence of zero or more steps.

An input w is accepted by an IAT $\mathcal{M}$ if at some time t during the course of its computation the communication cell enters an accepting state. In most cases t will be greater than $|w|$, but it is no restriction to accept earlier. An iterative array transducer $\mathcal{M}$ *transforms* input words $w \in A^*$ into output words $v \in B^*$. For a successful transformation $\mathcal{M}$ has to accept the input, otherwise the output is not recorded:

$$\mathcal{M}(w) = v,$$

if w is accepted by $\mathcal{M}$ and $(\lambda, v, c') \in \Delta^*(w, \lambda, c_0)$, and (λ, v, c') is a halting configuration. The *transduction realized by* $\mathcal{M}$, denoted by $T(\mathcal{M})$, is the set of pairs $(w, v) \in A^* \times B^*$ such that $\mathcal{M}(w) = v$.

Let $t_i, t_o : \mathbb{N} \to \mathbb{N}$, $n + 1 \leq t_i(n) \leq t_o(n)$, be two mappings. If for all $(w, v) \in T(\mathcal{M})$, the input w is accepted after at most $t_i(|w|)$ time steps and $\mathcal{M}$ halts after at most $t_o(|w|)$ time steps, then $\mathcal{M}$ is said to be of time complexity (t_i, t_o) and we write IAT_{t_i,t_o}. The family of transductions realized by IAT_{t_i,t_o} is denoted by $\mathscr{T}(\text{IAT}_{t_i,t_o})$. If t_i, t_o are the function $n + 1$, we call it *real time* and write *rt*. Since for nontrivial computations an IAT has to read at least one end-of-input symbol, real time has to be defined as $(n+1)$-time. If $t_i(n), t_o(n)$ are of the form $r \cdot n$, for some $r \in \mathbb{Q}$, $r \geq 1$, we call it *linear time* and write *lt*.

If we build the projection on the first components of $T(\mathcal{M})$, then the iterative array transducer degenerates to an iterative acceptor (IA). The projection on the first components is denoted by $L(T(\mathcal{M}))$. In order to clarify the notation we give an example.

Example 2.2. The transduction

$$T_{expo} = \{\, (a^{2^n}, a^1ba^1ba^2ba^4b \cdots a^{2^{n-1}}b) \mid n \geq 1 \,\}$$

belongs to $\mathscr{T}(\text{IAT}_{rt,rt})$.

In [4], an iterative array has been presented that uses signals in order to construct the mapping $n \mapsto 2^n$ in real time, that is, the communication cell recognizes the time steps 2^n, $n \geq 1$. At initial time the communication cell emits a signal which moves with speed $\frac{1}{3}$ to the right. In addition, another signal is emitted which moves with maximal speed (speed 1). It bounces between the slow signal and the communication cell. One can verify that the signal passes through the communication cell exactly at the time steps 2^n, $n \geq 1$ (see Figure 2).

An $\text{IAT}_{rt,rt}$ $\mathcal{M}$ which realizes T_{expo} basically simulates the time constructor for 2^n. It reads an input symbol a at every time step and writes the output ab when it is in a distinguished state at times 2^n, and outputs a otherwise. Finally, $\mathcal{M}$ accepts and halts if and only if the end-of-input symbol appears while it is in a distinguished state. ■

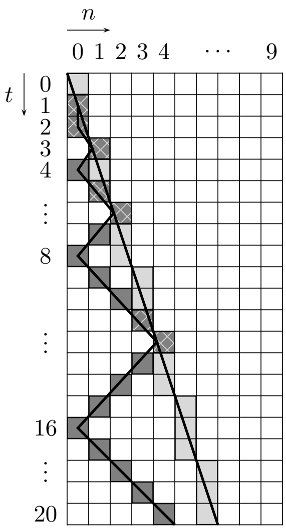


Figure 2: Space-time diagram showing signals of a time-constructor for the function $n \mapsto 2^n$.

One part of the computation of an IAT is the acceptance of the input. The previous example shows that iterative arrays can accept rather complicated languages even in real time. The language $\{ a^{2^n} \mid n \geq 1 \}$ is neither context free nor semilinear. Here, we consider iterative arrays from the computational point of view, that is, they are seen as massively parallel computing devices. So, we are mainly interested in fast computations, that is, small time complexities. In the sequel we focus on $\mathrm{IAT}_{rt,rt}$, $\mathrm{IAT}_{rt,lt}$, and $\mathrm{IAT}_{lt,lt}$. For further reading on iterative language acceptors we refer to [10, 12].

3. Computational Capacity of Iterative Array Transducers

Roughly speaking, any transduction computed by an iterative array can be divided into two tasks. One is the acceptance of the input, the other one the transformation of the input into the output. Both tasks have to end successfully in order to obtain a valid computation. In particular, this observation implies that a language, which is not accepted by any iterative array in time t_i, cannot be the projection on the first components of any transduction belonging to any class $\mathscr{T}(\mathrm{IAT}_{t_i,t_o})$.

Lemma 3.1. *The family $\mathscr{T}(IAT_{rt,lt})$ is strictly included in $\mathscr{T}(IAT_{lt,lt})$.*

Proof. The language $\{a,b\}^*\{ w \mid w \in \{a,b\}^* \wedge |w| \geq 3 \wedge w = w^R \}$ is not accepted by any real-time IA [5], but can easily be accepted by some linear-time IA. Therefore, the transduction $\{ (vw, a^{|vw|}) \mid v, w \in \{a,b\}^* \wedge |w| \geq 3 \wedge w = w^R \}$ is a witness for the assertion. ■

Since the complexity of the projections on the first components of an iterative array transduction is characterized by the power of iterative arrays when used as language acceptors, the question for the possible complexity of the projections on the second components follows immediately. It turns out that they can be arbitrarily complex within the limits of being computable.

Theorem 3.2. *Let L be an arbitrary recursively enumerable set. Then there is a transduction T belonging to $\mathscr{T}(IAT_{rt,rt})$ such that L is the projection on the second elements of T.*

Proof. Since L is recursively enumerable we may assume that it is represented by some deterministic one-tape one-head Turing machine $\mathcal{M}$ such that any accepting computation has at least three and, in general, an odd number of steps. Therefore, it is represented by an even number of configurations. Moreover, we may assume that $\mathcal{M}$ cannot print blanks, and that a configuration is halting if and only if it is accepting. A configuration of $\mathcal{M}$ can be written as a string of the form Z^*SZ^* where $z_1 \cdots z_i s z_{i+1} \cdots z_n$ is used to express that $\mathcal{M}$ is in state $s \in S$, scanning tape symbol z_{i+1}, and z_1 to z_n is the support of the tape inscription. The set of valid computations VALC($\mathcal{M}$) is now defined to be the set of words of the form $w_1\$w_3\$\cdots\$w_{2k-1}¢w_{2k}^R\$\cdots\$w_4^R\$w_2^R$, where w_i are configurations, \$ and ¢ are symbols not appearing in w_i, w_1 is an initial configuration, w_{2k} is an accepting configuration, and w_{i+1} is the successor configuration of w_i, for $1 \leq i \leq 2k$.

For some $u = w_1\$w_3\$\cdots\$w_{2k-1}¢w_{2k}^R\$\cdots\$w_4^R\$w_2^R \in \text{VALC}(\mathcal{M})$ we define $I(u)$ to be the support of the tape inscription of w_1, that is, the input word from L which is accepted by $\mathcal{M}$ with the computation represented by u. In [12, 13] it is shown that the set of valid computations is a real-time IA language.

Now, the transduction $\{\,(u,v) \mid u \in \text{VALC}(\mathcal{M}), v = I(u)\,\}$ can be realized by some $\text{IAT}_{rt,rt}$ that writes $I(u)$ while reading the input prefix w_1, and then continues to simulate an acceptor for VALC($\mathcal{M}$) whereby the empty word is written in every step. ■

Next we turn to separate the families $\mathscr{T}(\text{IAT}_{rt,rt})$ and $\mathscr{T}(\text{IAT}_{rt,lt})$. To this end, we recall the capability of iterative arrays to simulate data structures as pushdown stores, rings and queues without any loss of time. First we consider pushdown stores (stacks). The top of the stack is simulated by the communication cell. Assume without loss of generality that at most one symbol is pushed onto or popped from the stack at each time step. Then it suffices to use three additional tracks for the simulation. Let the three pushdown registers of each cell be numbered one, two, and three from top to bottom, and suppose that the third register is connected to the first register of the right neighbor. The content of the pushdown store is identified by scanning the registers in their natural ordering beginning in the communication cell, whereby empty registers are ignored. The pushdown store dynamics of the transition function is defined such that each cell prefers to have only the first two registers filled. The third register is used as a buffer (see Figure 3). Details of the construction can be found in [3, 6, 11].

Lemma 3.3. *The family $\mathscr{T}(IAT_{rt,rt})$ is strictly included in $\mathscr{T}(IAT_{rt,lt})$.*

Proof. The transduction $\{\,(u,u^R) \mid u \in \{a,b\}^*\,\}$ belongs to $\mathscr{T}(\text{IAT}_{rt,lt})$. An $\text{IAT}_{rt,lt}$ realizing the transduction simulates a pushdown store. It first reads and pushes u while the empty word is written. When the end-of-input symbol appears it accepts in real time, pops u^R from the stack and writes it with λ moves. Finally, it halts in linear time when the stack is emptied.

In contrast to the assertion assume that some $\text{IAT}_{rt,rt}$ $\mathcal{M}$ realizes the transduction. On input $u = u_1u_2\cdots u_n$ long enough $\mathcal{M}$ cannot write the first symbol of u^R before it reads the last symbol of u. Otherwise on input $u = u_1u_2\cdots u_{n-1}u_n'$, where $u_n \neq u_n'$, the same output prefix is written but $(u_1u_2\cdots u_{n-1}u_n', u_nv_{n-1}\cdots v_1)$ does not belong to the transduction, for any $v_1, v_2, \ldots, v_{n-1} \in \{a,b\}^*$. This implies, that u^R has to be written after reading the last input symbol in the single remaining step. If u has been chosen long enough, this is impossible. The contradiction shows that the transduction does not belong to $\mathscr{T}(\text{IAT}_{rt,rt})$. ■

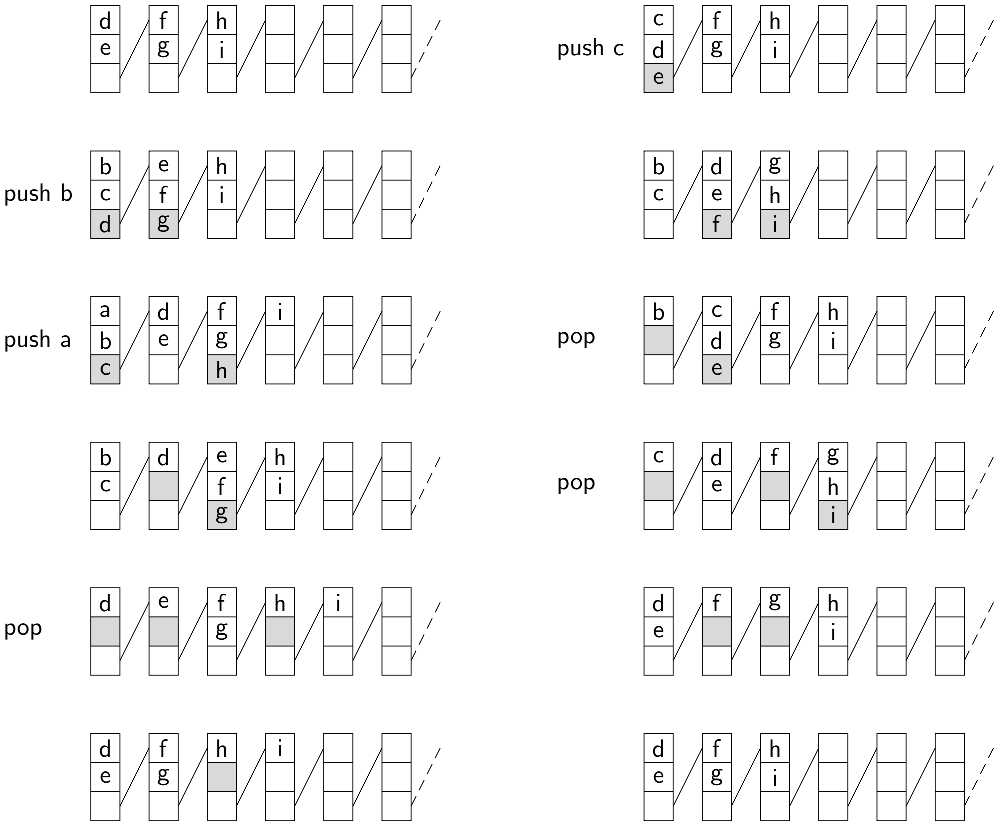


Figure 3: Principle of a pushdown store simulation. Subfigures are in row-major order.

We conclude this section with an example that utilizes the capability of an IA to simulate queues and rings without any loss of time (see [11]). The main difference between pushdown stores and rings or queues is the way how to access the data. A *ring* obeys the principle first in first out, that is, the first symbol of the stored string is read and possibly erased while, in addition, a new symbol may be added at the end of the string. So, a ring can write and erase at the same time. A *queue* is a special case of a ring. It can either write or erase a symbol, but not both at the same time.

Example 3.4. For any $k \geq 1$, the transduction $\{\, (u, u^k) \mid u \in \{a,b\}^* \,\}$ belongs to $\mathscr{T}(\mathrm{IAT}_{rt,lt})$.

An $\mathrm{IAT}_{rt,lt}$ realizing the transduction simulates a ring. It first reads and stores u where the first symbol of u in the ring is marked. When the end-of-input symbol appears the transducer accepts in real time. Then it starts to read u successively from the ring, where each symbol read is written and stored again into the ring. By the marked symbol the transducer recognizes when u has completely been processed. After the last process has been repeated k times it halts in linear time. ■

4. Comparison with and Simulation of Unambiguous Finite State Transducers

A nondeterministic finite state (rational) transducer is basically a nondeterministic finite automaton with output. At each time step the transducer reads a symbol or the empty word from the input tape in some internal state, goes nondeterministically into another state, and writes a symbol or the empty word to the output tape. So, the partial transition function δ maps from $S \times (A \cup \{\lambda\})$ into the subsets of $S \times (B \cup \{\lambda\})$. Alternatively, one could allow that longer input words may be read and longer output words may be emitted at every time step. This generalization yields the same family of transductions. Here, we use the single symbol mode which is called standard form in [15]. A nondeterministic finite state transducer $\mathcal{M}$ is said to be *single valued* (SFST) if for all $(u_1, v_1), (u_2, v_2) \in T(\mathcal{M})$ either $(u_1, v_1) = (u_2, v_2)$ or $u_1 \neq u_2$. An SFST is said to be *unambiguous* (UFST) if for all $(u, v) \in T(\mathcal{M})$ there is a unique computation transforming u into v. It has been shown in [14] that every single-valued finite state transducer can be simulated by an unambiguous one. A UFST is *deterministic* (DFST) if any computation is deterministic.

Since, in general, nondeterministic transducers can transform an input into different outputs, which is impossible for a deterministic device such as an IAT, we consider single-valued transducers only. While deterministic and nondeterministic finite automata accept the same family of languages, deterministic and nondeterministic finite state transducers have different power.

Lemma 4.1. *The family $\mathscr{T}(DFST)$ is strictly included in $\mathscr{T}(IAT_{rt,rt})$.*

Proof. The transduction of Example 2.2 does not belong to $\mathscr{T}$(DFST) since the language $\{ a^{2^n} \mid n \geq 1 \}$ is not regular. On the other hand, any DFST can effectively be converted into an equivalent DFST without λ-moves [14], which in turn can be simulated in the communication cell of an IAT$_{rt,rt}$. ∎

The next result shows that even the computational power of a massively parallel iterative array cannot compensate the presence of a little bit of nondeterminism.

Lemma 4.2. *The families $\mathscr{T}(SFST)$ and $\mathscr{T}(IAT_{rt,rt})$ are incomparable.*

Proof. As in the previous lemma the transduction of Example 2.2 does not belong to $\mathscr{T}$(SFST) since $\{ a^{2^n} \mid n \geq 1 \}$ is not regular. So, it remains to be shown that there is a transduction belonging to $\mathscr{T}$(SFST) but not to $\mathscr{T}$(IAT$_{rt,rt}$). To this end, we use $T = \{ (a^n c, a^n) \mid n \geq 1 \} \cup \{ (a^n d, b^n) \mid n \geq 1 \}$ as witness. Transduction T is realized by an SFST that guesses initially whether the last input symbol is a c or a d. Accordingly it reads the input and emits a's or b's. If the guess was correct, the input is accepted and the transduction is recorded, otherwise the input is rejected and the transduction is not recorded.

Assume that some IAT$_{rt,rt}$ $\mathcal{M}$ realizes T. On input $a^n c$, $\mathcal{M}$ cannot write the first symbol a before it reads the last input symbol. Otherwise on input $a^n d$ always a symbol a would be emitted. So, a^n has to be written after reading the last input symbol in the sole remaining step. Since n can be arbitrary long, this is impossible. The contradiction shows that the transduction does not belong to $\mathscr{T}$(IAT$_{rt,rt}$). ∎

Interestingly, if the iterative array is allowed to emit its output in linear time, the presence of a little bit of nondeterminism *can* be compensated.

Theorem 4.3. *The family $\mathscr{T}(SFST)$ is strictly included in $\mathscr{T}(IAT_{rt,lt})$.*

Proof. Let $\mathcal{M} = \langle S, A, B, F, s_0, \delta \rangle$ be an unambiguous finite state transducer. By the construction given in [14], we may assume that $\mathcal{M}$ works in real time.

The idea of the construction of an equivalent IAT$_{rt,lt}$ $\mathcal{M}'$ is as follows. In order to find out whether an input w is accepted by $\mathcal{M}$, we first consider the nondeterministic finite automaton (without output) $\mathcal{M}_{NFA} = \langle S, A, F, s_0, \delta' \rangle$ accepting $L(T(\mathcal{M}))$. It is converted into an equivalent deterministic finite automaton $\mathcal{M}_{DFA}$.

Automaton $\mathcal{M}_{DFA}$ is simulated in the communication cell of $\mathcal{M}'$ while reading the input w. Additionally, w is stored in a stack-like manner with the help of two tracks (see Figure 4). The IAT $\mathcal{M}'$ accepts if and only if the simulation of $\mathcal{M}_{DFA}$ accepts.

In order to compute the output of $\mathcal{M}$ we have to determine the accepting computation path of $\mathcal{M}_{NFA}$ on input w. As a first step, $\mathcal{M}_{NFA}$ is converted into an equivalent right linear grammar $\mathcal{G}_{NFA}$ with axiom X. The productions of $\mathcal{G}_{NFA}$ have three forms, namely,

(1) $X \to a[q']$ for all transitions $q' \in \delta'(s_0, a)$ such that $a \in A$,
(2) $[q] \to a[q']$ for all transitions $q' \in \delta'(q, a)$ such that $q \in S, a \in A$,
(3) $[q] \to a$ for all transitions $q' \in \delta'(q, a)$ such that $q \in S$, $q' \in F$, and $a \in A$.

So, each production in $\mathcal{G}_{NFA}$ corresponds to a transition rule of $\mathcal{M}_{NFA}$ and, thus, of $\mathcal{M}$. For each transition rule there is a unique output $z \in B^*$ emitted when the rule is applied.

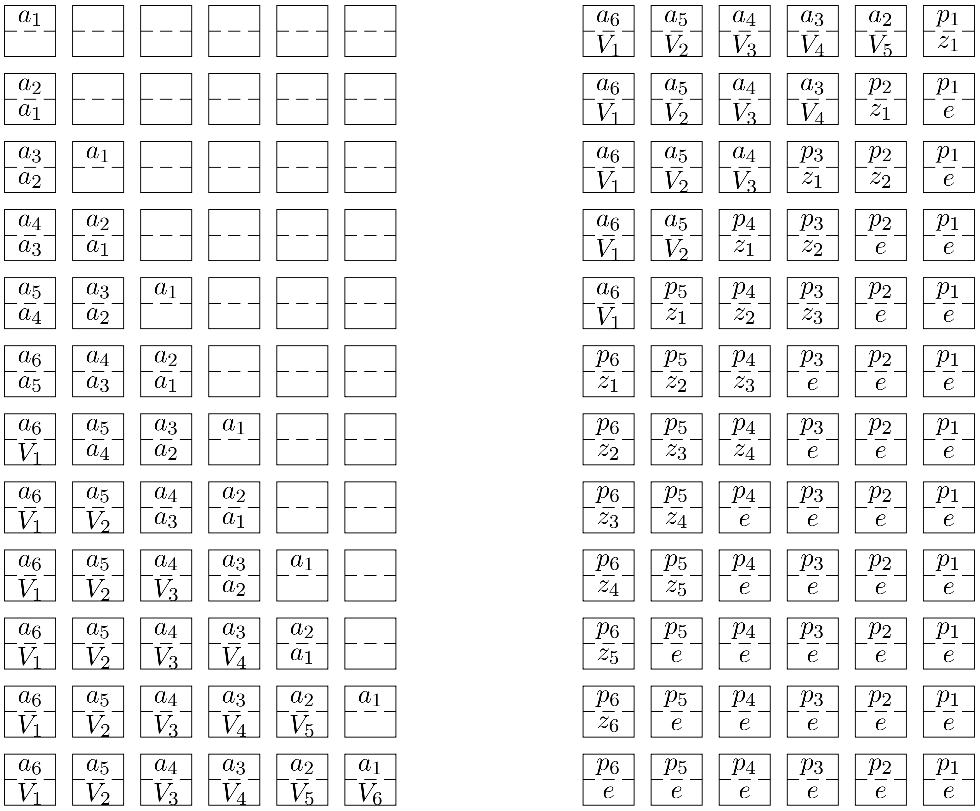

a1/– –/– –/– –/– –/– –/–
a2/a1 –/– –/– –/– –/– –/–
a3/a2 a1/– –/– –/– –/– –/–
a4/a3 a2/a1 –/– –/– –/– –/–
a5/a4 a3/a2 a1/– –/– –/– –/–
a6/a5 a4/a3 a2/a1 –/– –/– –/–
a6/V1 a5/a4 a3/a2 a1/– –/– –/–
a6/V1 a5/V2 a4/a3 a2/a1 –/– –/–
a6/V1 a5/V2 a4/V3 a3/a2 a1/– –/–
a6/V1 a5/V2 a4/V3 a3/V4 a2/a1 –/–
a6/V1 a5/V2 a4/V3 a3/V4 a2/V5 a1/–
a6/V1 a5/V2 a4/V3 a3/V4 a2/V5 a1/V6
a6/V1 a5/V2 a4/V3 a3/V4 a2/V5 p1/z1
a6/V1 a5/V2 a4/V3 a3/V4 p2/z1 p1/e
a6/V1 a5/V2 a4/V3 p3/z1 p2/z2 p1/e
a6/V1 a5/V2 p4/z1 p3/z2 p2/e p1/e
a6/V1 p5/z1 p4/z2 p3/z3 p2/e p1/e
p6/z1 p5/z2 p4/z3 p3/e p2/e p1/e
p6/z2 p5/z3 p4/z4 p3/e p2/e p1/e
p6/z3 p5/z4 p4/e p3/e p2/e p1/e
p6/z4 p5/z5 p4/e p3/e p2/e p1/e
p6/z5 p5/e p4/e p3/e p2/e p1/e
p6/z6 p5/e p4/e p3/e p2/e p1/e
p6/e p5/e p4/e p3/e p2/e p1/e

Figure 4: Schematic simulation of a single-valued finite state transducer on input $a_1a_2a_3a_4a_5a_6$. The first six cells of a simulating IAT$_{rt,lt}$ are depicted. In the first six time steps the input $a_1a_2a_3a_4a_5a_6$ is read, and in the last seven time steps the string $z_1z_2z_3z_4z_5z_6$ is emitted. The simulation of the deterministic finite automaton $\mathcal{M}_{DFA}$ in the communication cell is not depicted.

When reading the end-of-input symbol, $\mathcal{M}'$ starts to compute on an additional track all nonterminals of $\mathcal{G}_{NFA}$ from which suffixes of the input can be derived. More precisely, let $w = a_1a_2 \cdots a_n$ and, thus, in the first n cells of some track, say the first one, $a_na_{n-1} \cdots a_1$ is stored. Now, the first cell computes the set V_1 which includes all nonterminals Y for which the production $Y \to a_n$ belongs to $\mathcal{G}_{NFA}$, and stores it on a second track. Next, the second cell computes the set V_2 which includes all nonterminals Y for which the production $Y \to a_{n-1}[q']$ belongs to $\mathcal{G}_{NFA}$ and $[q']$ belongs to V_1. In general, the ith cell $(2 \leq i \leq n)$ computes the set V_i which includes all nonterminals Y for which the production $Y \to a_{n-i+1}[q']$ belongs to $\mathcal{G}_{NFA}$ and $[q']$ belongs to V_{i-1}. Since each V_i with $1 < i \leq n$ can be computed from V_{i-1}, a_{n-i+1}, and $\mathcal{G}_{NFA}$, the sets $V_1, V_2, \ldots, V_n$ can be computed in the first n cells in n time steps. It can be shown by induction that $Y \in V_i$ if and only if there is a derivation $Y \Rightarrow^* a_na_{n-1} \cdots a_{n-i+1}$ of $\mathcal{G}_{NFA}$. Therefore, we obtain that $w \in L(\mathcal{G}_{NFA})$ if and only if $X \in V_n$.

Now, we can extract the accepting computation path from the sets V_i moving from right to left and starting in the nth cell as follows. First, some production $p_1 : X \to a_1Y_1$ is chosen, where $Y_1 \in V_{n-1}$. The next productions $p_2, p_3, \ldots, p_{n-1}$ are chosen as $p_i : Y_{i-1} \to a_iY_i$ such that $Y_i \in V_{n-i}$. Finally, p_n is chosen as some production $p_n : Y_{n-1} \to a_n$. The productions are stored on a third track of $\mathcal{M}'$ and their computation takes n time steps.

Finally, the information on the output z_i of $\mathcal{M}$ which is associated with each production p_i has to be sent to the communication cell where it is emitted. To this end, after having determined p_1, the nth cell sends the information z_1 to the left. In the following time step, a signal e is sent to the left as well. The other cells i send their information z_i followed by e to the left when they receive the e from their neighbor to the right. The computation halts when the signal e arrives in the communication cell.

The overall simulation takes n time steps for reading the input, another n time steps to compute the sets V_i, and further $2n$ time steps to transmit signal e from the nth cell to the communication cell, that is, it takes linear time. ■

5. Iterative Arrays versus Pushdown Transducers

Similar as for finite state transducers, a nondeterministic pushdown transducer can be seen as nondeterministic pushdown automaton with output. So, the partial transition function δ maps from $S \times (A \cup \{\lambda\}) \times G$ into the finite subsets of $S \times B^* \times G^*$, where G denotes the stack alphabet. A nondeterministic pushdown transducer $\mathcal{M}$ is said to be *single valued* (SPDT) if for all $(u_1, v_1), (u_2, v_2) \in T(\mathcal{M})$ either $(u_1, v_1) = (u_2, v_2)$ or $u_1 \neq u_2$. An SPDT is said to be *unambiguous* (UPDT) if for all $(u, v) \in T(\mathcal{M})$ there is a unique computation transforming u into v. As opposed to finite state transducers, single-valued pushdown transducers have more computational power than unambiguous pushdown transducers. For example, the transduction $T = \{\, (a^nb^na^mb^m, a^{2m+2n}) \mid m, n \geq 1 \,\} \cup \{\, (a^nb^ma^mb^n, a^{2m+2n}) \mid m, n \geq 1 \,\}$ belongs to $\mathscr{T}(\text{SPDT})$ but not to $\mathscr{T}(\text{UPDT})$ because the projection on the first components is known to be an inherently ambiguous context-free language [8]. A UPDT is *deterministic* (DPDT) if any computation is deterministic, and it is *real-time deterministic* (DPDT$_\lambda$) if it is not allowed to move on empty input. Due to known results on the recognizability of context-free languages by different types of pushdown automata we have the proper hierarchy $\mathscr{T}(\text{DPDT}_\lambda) \subset \mathscr{T}(\text{DPDT}) \subset \mathscr{T}(\text{UPDT}) \subset \mathscr{T}(\text{SPDT})$.

We have already seen that the computational power of a massively parallel iterative array cannot compensate the presence of a little bit of nondeterminism. The same is true for the resource pushdown store equipped to a deterministic finite state device.

Lemma 5.1. *The family $\mathscr{T}(SFST)$ is incomparable with both families $\mathscr{T}(DPDT_\lambda)$ and $\mathscr{T}(DPDT)$.*

Proof. Since there are deterministic real-time context-free languages that are not regular there are transductions realized by some DPDT$_\lambda$ and DPDT but not by any SFST.

Conversely, we define the homomorphism $h : \{a, b\} \to \{a', b'\}^*$ by $h(a) = a'$, $h(b) = b'$, and consider the transduction $T = \{\, (uc, h(u)) \mid u \in \{a, b\}^* \,\} \cup \{\, (ud, u) \mid u \in \{a, b\}^* \,\}$. An SFST realizing T initially guesses whether the input ends with a c or with a d. Dependent on the guess it then computes the transduction as expected and records the transduction (accepts the input) when reading the last input symbol if and only if the guess was correct.

Now assume in contrast to the assertion that T is realized by some DPDT $\mathcal{M}$. On input ud, $\mathcal{M}$ cannot write the first output symbol a or b before it reads the last input symbol. Otherwise on input uc always a symbol a or b would be emitted. So, u has to be written after reading the last input symbol in a sequence of λ-moves. Next, a DPDT $\mathcal{M}'$ is constructed as follows. It starts to simulate $\mathcal{M}$ until a d appears in the input. Then it continues the simulation but, in addition, tries to read the symbols it emits from the input. On input c, $\mathcal{M}'$ rejects. So, $\mathcal{M}'$ realizes the transduction $\{\, (udu, u) \mid u \in \{a, b\}^* \,\}$. This is a contradiction since $L(T(\mathcal{M}'))$ is not context free. ∎

In order to draw an almost complete picture we next show the incomparability of transductions realizable by DPDT, UPDT or SPDT, and IAT$_{rt,rt}$ or IAT$_{rt,lt}$, that is, the computational power of a massively parallel iterative array cannot compensate the presence of a pushdown store and vice versa.

Lemma 5.2. *Each of the families $\mathscr{T}(DPDT)$, $\mathscr{T}(UPDT)$, and $\mathscr{T}(SPDT)$ is incomparable with both families $\mathscr{T}(IAT_{rt,rt})$ and $\mathscr{T}(IAT_{rt,lt})$.*

Proof. The incomparability follows from the incomparability of the languages accepted by real-time iterative arrays and deterministic as well as nondeterministic context-free languages [12]. ∎

However, by simulating pushdown stores as shown above IAT$_{rt,rt}$ can simulate DPDT$_\lambda$, and IAT$_{lt,lt}$ can simulate DPDT. Together with the previous incomparability results proper inclusions follow. It is known that Turing machines can simulate linear-time iterative arrays in quadratic time. On the other hand, it is not known whether every Turing machine working in quadratic time can be simulated by a linear-time iterative array. Since unambiguous context-free languages can be parsed in quadratic time using a variant of Earley's algorithm [1], it is an open problem to find out whether this approach can be applied for simulating UPDT or SPDT on linear-time iterative arrays.

Although nondeterministic devices in general have been excluded a priori, we conclude the section with an example emphasizing that there are structurally interesting "non-unary" transductions realizable by some IAT$_{rt,lt}$ but not realizable by any nondeterministic pushdown transducer.

Example 5.3. The transductions $\{\,(ucv, vcu) \mid u, v \in \{a,b\}^+\}$ and $\{\,(uc, u^Rcu) \mid u \in \{a,b\}^+\}$ belong to $\mathscr{T}(\mathrm{IAT}_{rt,lt})$, but cannot be realized by any, even nondeterministic, pushdown transducer [1].

$$\begin{tikzcd}
 & \mathscr{T}(\mathrm{IAT}_{lt,lt}) & \mathscr{T}(\mathrm{SPDT}) \arrow[l, dashed] \\
\mathscr{T}(\mathrm{IAT}_{rt,lt}) \arrow[ur] & \mathscr{T}(\mathrm{DPDT}) \arrow[u] \arrow[r] & \mathscr{T}(\mathrm{UPDT}) \arrow[u] \\
\mathscr{T}(\mathrm{IAT}_{rt,rt}) \arrow[u] & \mathscr{T}(\mathrm{DPDT}_\lambda) \arrow[u] \arrow[l] & \mathscr{T}(\mathrm{SFST}) \arrow[u] \arrow[ull] \\
 & \mathscr{T}(\mathrm{DFST}) \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 5: Summary of inclusions. Solid lines are proper inclusions, dashed lines are conjectured inclusions. All families which are not linked by a path are pairwise incomparable.

References

[1] Aho, A.V., Ullman, J.D.: The theory of parsing, translation, and compiling. Volume I: Parsing. Prentice-Hall, Englewood Cliffs (1972)

[2] Berstel, J.: Transductions and Context-Free-Languages. Teubner, Stuttgart (1979)

[3] Buchholz, Th., Kutrib, M.: Some relations between massively parallel arrays. Parallel Comput. **23** (1997) 1643–1662

[4] Choffrut, C., Čulik II, K.: On real-time cellular automata and trellis automata. Acta Inform. **21** (1984) 393–407

[5] Cole, S.N.: Real-time computation by n-dimensional iterative arrays of finite-state machines. IEEE Trans. Comput. **C-18** (1969) 349–365

[6] Čulik II, K., Yu, S.: Iterative tree automata. Theoret. Comput. Sci. **32** (1984) 227–247

[7] Fischer, P.C.: Generation of primes by a one-dimensional real-time iterative array. J. ACM **12** (1965) 388–394

[8] Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley, Reading (1978)

[9] Ibarra, O.H., Jiang, T., Wang, H.: Parallel parsing on a one-way linear array of finite-state machines. Theoret. Comput. Sci. **85** (1991) 53–74

[10] Kutrib, M.: Automata arrays and context-free languages. In Where Mathematics, Computer Science and Biology Meet. Kluwer Academic Publishers (2001) 139–148

[11] Kutrib, M.: Cellular automata – a computational point of view. In New Developments in Formal Languages and Applications. Springer (2008) 183–227

[12] Kutrib, M.: Cellular automata and language theory. In Encyclopedia of Complexity and System Science. Springer (2009) 800–823

[13] Malcher, A.: On the descriptional complexity of iterative arrays. IEICE Trans. Inf. Syst. **E87-D** (2004) 721–725

[14] Weber, A., Klemm, R.: Economy of description for single-valued transducers. Inform. Comput. **118** (1995) 327–340

[15] Yu, S.: Regular languages. In Handbook of Formal Languages. Volume 1. Springer, Berlin (1997) 41–110

Journées Automates Cellulaires 2010 (Turku), pp. 168-179

AN UPPER BOUND ON THE NUMBER OF STATES FOR A STRONGLY UNIVERSAL HYPERBOLIC CELLULAR AUTOMATON ON THE PENTAGRID

MAURICE MARGENSTERN

Université Paul Verlaine, LITA EA 3097, UFR MIM and CNRS, LORIA, Campus du Saulcy, 57045 Metz Cedex 1, France
E-mail address: margens@univ-metz.fr
URL: http://www.lita.sciences.univ-metz.fr/~margens

Abstract. In this paper, following the way opened by a previous paper deposited on *arXiv*, see[7], we give an upper bound to the number of states for a hyperbolic cellular automaton in the pentagrid. Indeed, we prove that there is a hyperbolic cellular automaton which is rotation invariant and whose halting problem is undecidable and which has 9 states.

1. Introduction

In [7], we gave a general tool to embed a 1D-cellular automaton into a whole family of tilings of the hyperbolic plane and into two tilings of the hyperbolic 3D-space.

In this paper, we try to improve the method in order to find a strongly universal cellular automaton in the pentagrid. We remind the reader that by strong universality, we mean a cellular automaton which mimics the computation of universal devices starting from a **finite** configuration. We also remind the reader that the pentagrid is the tiling $\{5, 4\}$, *i.e.* the tessellation of the hyperbolic plane based on the regular pentagon with right angles, see [2, 4]. Within the limit on the number of pages given to the author, the paper cannot be self-contained. This is why we assume that the reader is familiar with both cellular automata and their implementation in tessellations of the hyperbolic plane, and we refer him/her to the above references and to this additional one, [5], in case he/she would not be familiar with these notions.

In Section 2, we give the outline of the construction. In Section 3 we implement the preliminary structure of the implementation. In Section 4 we give a construction with 13 states. In Section 5, we reduce this number by one state, which will give the way to Section 6 where the number of states is reduced to 9 of them. Section 7 concludes the paper with indications on further work.

Key words and phrases: cellular automata, strong universality, hyperbolic spaces, tilings.

2. Scenario

In traditional literature on cellular automata, a **quiescent** state is defined as a state q such that if a cell and all its neighbours are under state q, the cell remains under state q at the next time of the clock. By analogy with the empty squares of the tape of a Turing machine which are said to contain the **blank** symbol, we shall fix a quiescent state and we shall call it the **blank**.

For cellular automata which have a blank state, an initial finite configuration is a configuration in which all cells are blank except, possibly, finitely many of them. It is plain that if the initial configuration is finite, all further configurations are also finite, even when the computation requires an infinite time: in this case, there may be no uniform bound to the size of each configuration.

Let us now turn to the idea of the construction.

The idea of [7], which embeds any 1D-cellular automaton with infinite initial configuration is very simple: it consists in embedding the 1D-structure of the considered cellular automaton into the desired tiling of the hyperbolic plane. In this construction, we simply had to devise a simple way to make the cells of the embedded structure different from the other cells of the tiling. In this way, we can easily transport the rules of the 1D-cellular automaton into those of the hyperbolic cellular automaton. The key point is that in this construction, the differentiation is made a priori: it is given in the initial configuration.

If we wish to implement a 1D-cellular automaton which starts its computation from a finite configuration, then we have to go on the embedding at the same time as the computation is going on. And so, we have to find out a simple way to construct the 1D-structure together with the computation. But this is not enough. Remember that the halting of the computation of a cellular automaton is defined by the occurrence of two consecutive identical configurations. And so, when the computation of the 1D-cellular automaton is completed, we have to stop the construction of the 1D-structure.

In [3], the propagation of the tree structure of the pentagrid is implemented in a triangular cellular automaton, and this automaton can easily be adapted to the pentagrid, also as a **rotation invariant** one. Rotation invariance means that the new state is unchanged if we perform a circular permutation on the neighbours of the cell, this cell being excepted. This was done in [5] and repeated, as an example in [6], not in the shortest way as in that context the goal was that a cell should recognize whether it is black or white with respect to the Fibonacci structure simply by looking at its neighbourhood. Here, we do not really bother of this condition so that instead of six states as it is the case in [5], three states are enough: the blank, a white one and a black one. As we have the choice for the place of the initial configuration, we can place it around the central cell which spares us the burden of initializing the propagation of the structure: it is enough to assume it is installed in the initial configuration and to continue it, this spares one state.

However, we have to stop the computation, which means that a signal has to be sent to stop the others. In order to perform this task, it is needed to slow down the propagation. Indeed, the speed of a signal is at most 1. And so, if the halting signal travels at speed 1, it can catch up previously sent signals only if these latter signals travelled at a lower pace. Now, slowing down necessarily costs states as we shall see. But, fortunately, the 1D-cellular automaton which we shall consider is also slow, so that we shall not have to slow down too much.

After that, we have to look at the way to find a 1D-cellular automaton which is strongly universal with a small number of states. We shall use the implementation of the 7×4 universal Turing Machine of Marvey Minsky which is precisely described in [1]. This gives us a 1D-, strongly universal cellular automaton with 7 states. In the rest of the paper, we denote this cellular automaton by $\mathcal{L}$.

Now, to see how many states we can obtain, we have to go into finer details of the implementation. The first step is the propagation of the 1D-structure which is a common feature of the three cellular automata which we construct in the paper.

3. Implementation of the 1D-structure

In [7], in the case of the pentagrid, we implemented the line of a 1D-cellular automaton along a line of the pentagrid. We remind the reader that such a line is any line which contains a side of a pentagon of the tiling: such a line contains the sides of infinitely many pentagons which can be gathered into two sequences of pentagons indexed by $\mathbb{Z}$, two consecutive pentagons having a side on the line, these just indicated sides being also consecutive.

Here, as we start from a finite configuration, we have at most finitely many cells along such a line and our task is to devise a way to go on this line as a continuation of the segment which already exists.

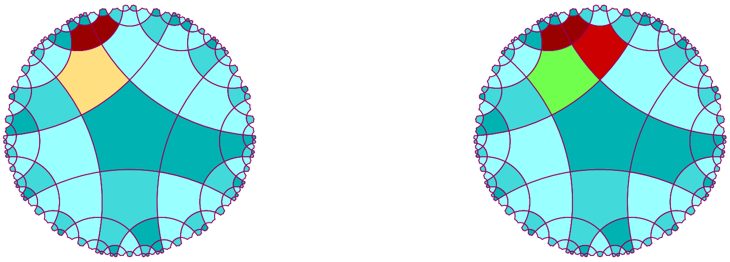


Figure 1: The first two configurations of the propagation of the $1D$-structure. Left-hand side: initial configuration, say time 0. In dark red, cells in state $\mathbf{B}$, in light yellow, the cell in state $\mathbf{W}_0$. Right-hand side: time 1. In bright red, the cell in state $\mathbf{B}_0$. In green, the cell in state $\mathbf{W}_1$. The blue cells represent the blank denoted by $\mathbf{N}$: the different hues of blue remind the tree structure of the tiling, but they represent a single state. Note that here, the central cell is not the central pentagon of the figure, it is the dark red cell in contact with the light yellow, green one in the left-, right-hand side picture respectively.

Now, the problem can be a bit simplified by the fact that $\mathcal{L}$ possesses an interesting feature. The cellular automaton $\mathcal{L}$ implements a Turing machine which is an interpreter of tag systems. This mean that we may assume that the Turing machine works on a semi-infinite tape: *i.e.* the tape has an end but it is infinite in one direction only. This is particularly interesting for our implementation: this allows us to implement a ray only, so that we can put the end of this ray around the central cell. For the propagation algorithm, we can define the first two configurations as illustrated by Figure 3.

We need six states for the propagation of the ray: $\mathbf{N}$, $\mathbf{B}_0$, $\mathbf{B}$, $\mathbf{W}_0$, $\mathbf{W}_1$ and $\mathbf{W}$. Informally, cells in $\mathbf{B}$ will follow a branch of black nodes of one of the Fibonacci trees rooted around the central cell. This branch is the leftmost branch of the chosen Fibonacci tree. Now, the cells **W** follow the rightmost branch of the next Fibonacci tree while clock-wise turning around the central cell. It is easy to remark that the pentagons of these two branches share a ray which supports one side of each one of these pentagons.

The propagation itself advances at a speed 1/2. This is suggested by the presence of the states $\mathbf{B}_0$ and **B** as well as by that of the states $\mathbf{W}_0$ and $\mathbf{W}_1$.

The mechanism is the following. A new **B** is produced by the transformation of $\mathbf{B}_0$ into **B**. Now, a new $\mathbf{B}_0$ is obtained by the continuation of both the black branch and the white one. It is created by the simultaneous occurrence of a **B** and a $\mathbf{W}_0$ around a blank cell abutting the cell at contiguous sides and in a precise order: while counter-clockwise turning around the cell, we first meet **B** and then, immediately, $\mathbf{W}_0$. The three cells, the blank, **B** and $\mathbf{W}_0$ share a common vertex. We know that there is a fourth cell. In the initial configuration it is a cell in **B**. In the other configurations, when this local configuration occurs, the fourth cell is a cell in **W**: it is a cell in $\mathbf{W}_1$ which evolved into **W**. The roles between $\mathbf{W}_0$ and $\mathbf{W}_1$ are the following. A cell in $\mathbf{W}_0$ becomes **N** at the next time if it is surrounded by cells in **N**. Otherwise, it becomes $\mathbf{W}_1$. Now, a cell in $\mathbf{W}_1$ becomes **W** if and only if it sees a neighbour in $\mathbf{B}_0$. Otherwise, it becomes **N**. Now, a cell in **N** becomes $\mathbf{W}_0$ if and only if it has one neighbour in $\mathbf{W}_1$ exactly. Otherwise, it remains **N**.

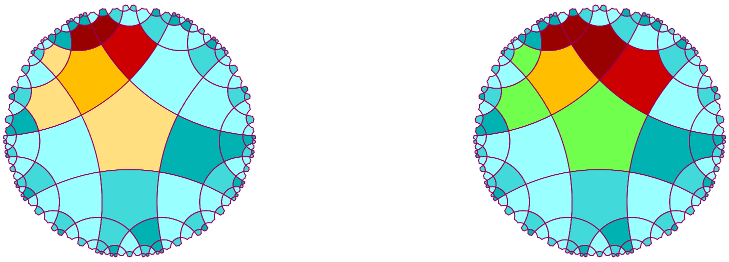


Figure 2: Left-hand side: time 2 of the propagation. Right-hand side: time 3 of the propagation. In bright yellow, state **W** which indicates a fixed cell. The blue cells have the same meaning as in Figure 3.

Now, we can see that the configuration which allows a cell in **N** to become $\mathbf{B}_0$ requires the cells in **W** or $\mathbf{W}_0$ to be one step in advance with respect to those in **B** and $\mathbf{B}_0$. This is also allowed by the progression of the cells in $\mathbf{W}_0$. The condition on $\mathbf{W}_0$ allows us to stop the progression of the $\mathbf{W}_0$'s which do not follow the ray. Whence the importance on the condition of the transformation of the $\mathbf{W}_0$'s and also of the $\mathbf{W}_1$'s which disappear unless they can see a cell in $\mathbf{B}_0$, in which case they become **W**.

We have no room here for the table of the rules. Such tables can be found in [8]. We refer the reader to this paper for them. In the present paper, the expression *the rules for* $\mathcal{A}$ refers to the tables in [8] which display the rules of the automaton $\mathcal{A}$. Note that in all the tables of [8], there are two kinds of rules. In the first group, the current state of the cell is left unchanged: this is why these rules are called **conservative**. In the second group, the current state of the cell is changed: these rules are called **propagation** rules.

Basically, there are two propagation rules: the rule `N B W0 N N N B0` and the rule `N W1 N N N N W0`. In both of them, the blank is changed into a cell which will contribute to the extension of the 1D-structure. Now, the other rules contribute to create the context required by these two propagation rules as well as the transformation of a first signal, $\mathbf{B}_0$ and $\mathbf{W}_0$ into the final one, $\mathbf{B}$ and $\mathbf{W}$, respectively.

The fact that we have three signals for the white node instead of two as the speed of progression of the $1D$-structure is $\frac{1}{2}$ is explained by the structure of the pentagrid: when a cell in $\mathbf{W}_1$ propagates the signal $\mathbf{W}_0$, this is performed upon several cells, at least two of them, while it is needed for only one of them. This is the reason of the signal $\mathbf{W}_1$ which is an intermediate step between $\mathbf{W}_0$ and $\mathbf{W}$. Now, in order to keep the speed $\frac{1}{2}$, the alternation is performed between $\mathbf{W}_1$ and $\mathbf{W}_0$.

It is now time to go to the other parts of the implementation. In these parts, we shall refer to the cells in $\mathbf{B}$ and in $\mathbf{W}$ of the just described construction as the **ray** in which the cells in $\mathbf{B}$ constitute the **track** and the cells in $\mathbf{W}$ constitute the **support** of the track.

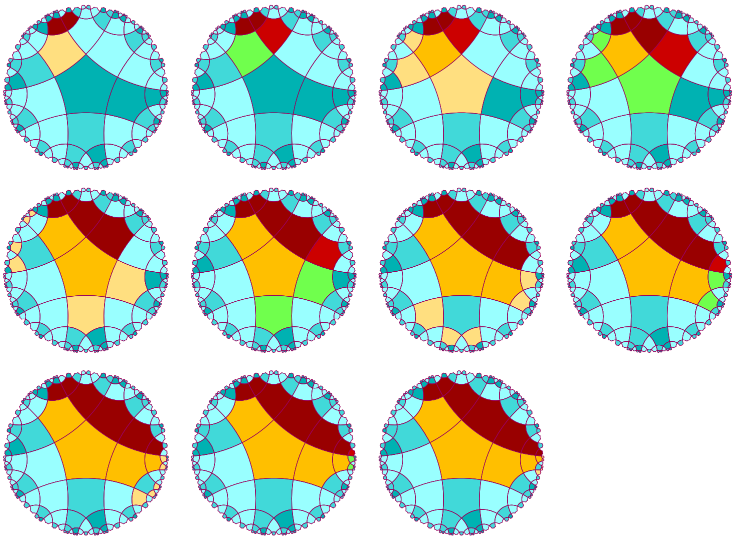


Figure 3: Illustration of the propagation from the beginning until a few steps. In bright yellow, state $\mathbf{W}$ which indicates a fixed cell. The blue cells have the same meaning as in Figure 3.

4. The 13-state cellular automaton

First of all, we remind the reader that in this paper, we shall consider deterministic cellular automata only, as $\mathcal{L}$ is itself a deterministic $1D$-cellular automaton. Let us denote by $\mathcal{A}$ the automaton which implements the computation performed by $\mathcal{L}$. We require $\mathcal{A}$ to be rotation invariant.

In the previous propagation, we consider that $\mathbf{B}$ represents the blank of $\mathcal{L}$ which has to be distinct from the blank $\mathbf{N}$ of $\mathcal{A}$. Indeed, if we give the same state for the two blanks, we shall have problems with the propagation, as can easily be seen from the scenario of Section 2.

An important point is that in the working of $\mathcal{L}$, it can be noticed from [1] that the configuration of $\mathcal{L}$, *i.e.* the smallest interval which contains the non blank cells, remains the same during at least three consecutive steps and that it may go outside by one cell, only at the fourth time. This means that the progression of the configuration of $\mathcal{L}$ is much slower than the propagation of the ray described in Section 3. As a consequence, when the signal of $\mathcal{L}$ goes outside the current configuration, the track is ready to deliver free blank cells.

Accordingly, the computation of $\mathcal{A}$ is able to perform that of $\mathcal{L}$ during the propagation stage. In fact, it is enough to consider that in the process described in section 3, $\mathbf{B}$ can be any of the states of $\mathcal{L}$ and that it must be the blank for the cell in $\mathbf{B}_0$ which is transformed into $\mathbf{B}$. This latter $\mathbf{B}$, at this moment, is the blank of $\mathcal{L}$. Indeed, the cells of the track are the single one which, except the ones which are at the ends of the track, have two neighbours which are also cells of the track. In fact, except for two exceptional cells, a rule of the track is of the form $y\ x\ \mathtt{W}\ z\ \mathtt{N}\ \mathtt{N}\ u$ where $xyz \to u$ is a rule of $\mathcal{L}$. For the exceptional cells, the origin and its neighbour of the track, the cell has three neighbours under $\mathbf{N}$, and the origin has a single neighbour on the track. All these features can easily be seen from the pictures of Figure 3, there cannot be ambiguity about these local configurations.

This means that, presently, as $\mathbf{B}$ is one of the states of $\mathcal{L}$, $\mathcal{A}$ has 12 states as $\mathcal{L}$ itself has 7 states, see [1].

Now, let us closer look at the working of $\mathcal{L}$. In [1], $\mathcal{L}$ mimics rather closely the computation of Minsky's Turing machine. In particular, there is a state T, notation of [1], which represents the position of the head of the machine. In the simulation devised in [1], the halting is performed by the disappearance of T. Our task is to change this transformation into a signal which will trigger the stage at the end of which the computation of $\mathcal{A}$ will also halt in the traditional sense of the halting of a cellular automaton starting from a finite configuration, see Section 2.

To perform this task, we replace the instruction $0Ty \to 0$ of the table of $\mathcal{L}$ in [1] by the instruction $OTy \to \mathbf{H}$, where $\mathbf{H}$ is a new state of $\mathcal{A}$. Also, we append the new instructions $\mathbf{H}y\mathbf{B} \to \mathbf{B}$, $\mathbf{B}0T \to \mathbf{B}$, where $\mathbf{B}$ is in both cases $\sqcup$, the blank of $\mathcal{L}$. Note that $\sqcup 0T \to \sqcup$ is a rule of $\mathcal{L}$, see [1].

Now, we can make a bit more precise the scenario depicted in Section 2. As just indicated, $\mathbf{H}$ appears on the track. In some sense it is far from the ends as we can put several blanks to the left of the leftmost non blank cell of the Turing tape in the initial configuration. We remind the reader that we may choose the initial configuration and we may choose it so that the initial segment of $\mathcal{L}$ outside which there are only blank cells is in middle of the initial configuration, with at least two blank cells outside this segment. We decide that $\mathbf{H}$ does not affect the cells of the track which will remain unchanged. However, we decide that a cell in $\mathbf{W}$ which sees at least one $\mathbf{H}$ among its neighbours becomes $\mathbf{H}$ itself.

In this way, the cells of the support of the track are progressively changed to $\mathbf{H}$. The corresponding rules are given in [8] and they are illustrated by Figure 4. We have just to see that we can effectively stop the generation of cells in $\mathbf{W}_0$ and in $\mathbf{W}_1$, which will to its turn stop the production of $\mathbf{B}_0$. We also have to check that no problem arises on the fixed end of the ray which represents the leftmost part of the Turing tape.

Figure 4 shows how the propagation of the ray is stopped by the arrival of the states $\mathbf{H}$. For simplicity, call signal $\mathbf{H}$, the propagation of $\mathbf{H}$ replacing the state $\mathbf{W}$ in the cells of the support of the track.

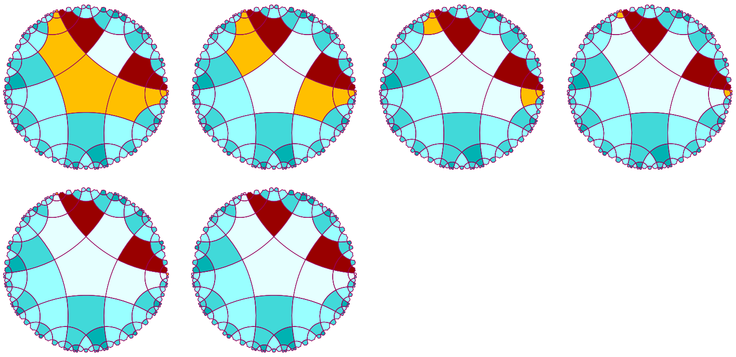


Figure 4: The propagation of the **H**-signal. Note that the back ground of blue cells is the same as in the previous figures.

First, assume that the signal **H** arrives as indicated in the figure: almost all cells of this part of the support are now in state **H**, and just a single cell in **W** remains whose next neighbour is a cell in $\mathbf{W}_1$. Necessarily, the cell in **W** is changed to **H** and the cell in $\mathbf{W}_1$ becomes **W**: the cell in $\mathbf{W}_1$ cannot see what is on another side of its neighbour in **W**. And so, at the next step, we have a cell in **W** again for which one neighbour is in **H** and two others are in $\mathbf{W}_0$. At the next time, the cells in $\mathbf{W}_0$ become $\mathbf{W}_1$ and the cell in **W** becomes **H**. Now, the cell in **H** is neighbouring a cell in $\mathbf{W}_1$. If the cell in $\mathbf{W}_1$ turns to **W** again, then the signal will never stop the propagation of the ray. And so, the cell in $\mathbf{W}_1$ turns to **H** when one of its neighbours is in **H**: this is possible as no rule with the current state $\mathbf{W}_1$ involved a neighbour in **H**. And this solves the problem: at the next time, the cells in $\mathbf{W}_0$ either have their five neighbours in **N**, or they have a neighbour in **H**. We decide that the neighbouring of **H** makes a cell in $\mathbf{W}_0$ to turn to **N** too, and this stops the process. It can be checked that the rules of [8] allow to perform this task. This was done by the computer program which also computed the data for the PostScript file producing Figures 3, 4, 4 and 4. The computer program also checked the rotation invariance of the rules.

From the figure, it is not difficult to see that the situation illustrated by Figure 4 is general: as the signal goes faster than the progression of the ray, there will always be a time when the rightmost cell in **H** will be close to the rightmost cell in **W** at a time when this cell is neighboured by cells in $\mathbf{W}_1$. Indeed, if there are two cells in **W** between the cell in **H** and the cell in $\mathbf{W}_1$, at the next time, the cell in$\mathbf{W}_1$ which is close to the track becomes **W** while the others become **N**, and the blank cells close to $\mathbf{W}_1$ become $\mathbf{W}_0$. Now, between the rightmost cell in **H** and the cell in $\mathbf{W}_0$, there are two cells in **W**. But at the next time, the cell in **W** close to **H** becomes **H** and the cell in $\mathbf{W}_0$ which is close to the border becomes $\mathbf{W}_1$. And so, at this time, we have the same configuration as the one illustrated by Figure 4. This proves that, in all cases, the signal **H** stops the progression of the ray as this was planned.

We remain with checking that the progression of the signal **H** in the other direction is stopped by the origin: this is illustrated by Figure 4. See the rules in [8].

At this point, we have proved the following result:

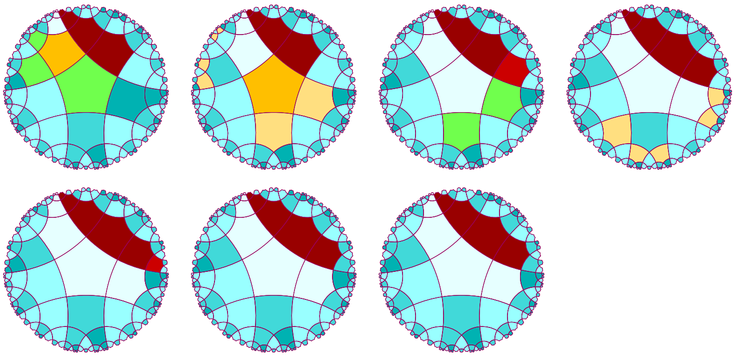


Figure 5: How the **H**-signal stops the propagation of the ray.

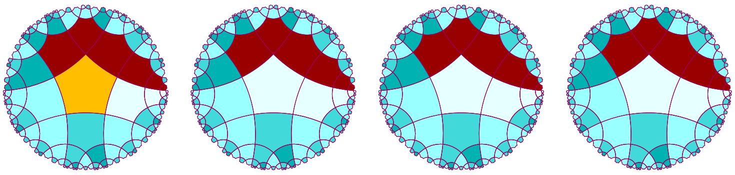


Figure 6: How the **H**-signal is stopped at the other end of the ray, near its origin.

Theorem 4.1. *There is a rotation invariant hyperbolic cellular automaton in the pentagrid which starts from finite configurations and whose halting problem is undecidable which has* 13 *states, the blank included.*

Indeed, $\mathcal{A}$ satisfies the statement of Theorem 4.1. However, the reader may wonder why we stated that the halting problem of $\mathcal{A}$ is undecidable and why we did not state that $\mathcal{A}$ is strongly universal? This is due to a strange property of Minsky's Turing machine with 7 states and 4 letters, a property which is inherited by $\mathcal{L}$ as it closely simulates this Turing machine. The problem lies in the way the Turing machine detects the halting of the simulated tag system. In fact, in this machine, when the halting production is found, the Turing machine erases its tape so that when it stops, the content of the tape can no more be red. This 'defect' of Minsky's machine was noticed and corrected by Rogozhin in [11]. Of course, 'morally' the machine is universal but rigorously, we can say no more than the statement of the theorem.

5. The 12-state cellular automaton

Now, we can show that a slight tuning of $\mathcal{A}$ allows us to obtain a cellular automaton which simulates the computation of $\mathcal{L}$ using 12 states only.

The idea is to replace the state **H** by one of the states of $\mathcal{A}$ which is not used by $\mathcal{L}$. In fact we have no choice. State **W** cannot be chosen as the support of the track consists already of cells in **W**. Using **W** would require to circumvent the support which would lead to more states. Similarly, neither $\mathbf{W}_0$ nor $\mathbf{W}_1$ can be used as they contribute to continue the propagation. For the same reason, $\mathbf{B}_0$ is rules out and, of course, **B** cannot be used: this does not give a clear signal that the computation halted. And so, the only possibility is **N**.

Let $\mathcal{B}$ be the cellular automaton in the pentagrid obtained from $\mathcal{A}$ by replacing $\mathbf{H}$ by $\mathbf{N}$ in the rules for $\mathcal{A}$.

We can easily see that under this replacement of $\mathbf{H}$ by $\mathbf{N}$ in the rules of [8], the rule obtained from `N H N N N W1 N` is in conflict with the rule we get from `N W1 N N N N W0`, as we require our cellular automaton to be rotation invariant. Of course, the solution is to cancel the rule obtained from `N H N N N W1 N`. And it works: there are no more conflicts, we just have a few repetitions with the rules for $\mathcal{A}$ where $\mathbf{H}$ does not occur.

And so, we proved the following result:

Theorem 5.1. *There is a rotation invariant hyperbolic cellular automaton in the pentagrid which starts from finite configurations and whose halting problem is undecidable which has* 12 *states exactly, the blank included.*

Note that the remark about strong universality for Theorem 4.1 also holds for this one.

Replacing $\mathbf{H}$ by $\mathbf{N}$ boils down to erase the support of the track so that, at the end of the computations, we remain with the track only. This erasing occurs at both ends of the ray. We have no room to provide the figures proving this point. These figures can be found in [8]. They also show that the erasing process stops the propagation of the ray.

Now, could we reduce again the number of states, using the same $1D$ cellular automaton? The next Section gives a positive answer to this question.

6. The 9-state cellular automaton

We have already seen that to reduce the number of states from 13 down to 12, we replaced the state $\mathbf{H}$ by the state $\mathbf{N}$. We can try to go further in this direction. In [7], we reduced the number of states for a weakly universal cellular automaton from 3 states down to 2 ones by replacing the extra state in the simulation with 3 states by a state of the embedded 2-states cellular automaton. So that here, a natural idea is to replace as many as we can states from $\mathbf{N}$, $\mathbf{W}$, $\mathbf{W}_0$, $\mathbf{W}_1$ and $\mathbf{B}_0$, by states of $\mathcal{L}$ only. If we look at the rules displayed in [1], we can notice that the symbol T has a rather empty sub-table. In particular, there is no rule assigned to TTT, so that we can decide that $TTT \to T$ is used by our new automaton, $\mathcal{C}$. Inspired by the table of the rules given in [1], we shall see that $\mathbf{W}$, $\mathbf{W}_0$ and $\mathbf{B}_0$ can be replaced by T, 0 and A respectively. It is enough to check that performing these replacements and taking the above figures, we obtain new rules which are rotation invariant and compatible.

Indeed, consider the first stage of the working of $\mathcal{B}$ which consists in propagating the structure at the same time when the computation is going on. It will be enough to ensure that a pure propagation process can be performed under the new set of states and that there is no contradiction between the new involved rules and the rules derived from the computation of $\mathcal{L}$. We also have to check that the new rules do not disturb the computation itself.

Now, this latter condition entails that we cannot replace $\mathbf{W}_1$ by a state of $\mathcal{L}$. Indeed, imagine that $\mathbf{W}_1$ is a state of $\mathcal{L}$, say α. There are occurrences of α on the track. Each cell of the track has at least two neighbours in state $\mathbf{N}$. Consider one of these neighbours. It has α as a neighbour and all the others are in $\mathbf{N}$. From Section 3, we know that in this case, the state $\mathbf{N}$ is replaced by $\mathbf{W}_0$ and this $\mathbf{W}_0$, as it is not surrounded by cells in $\mathbf{N}$ only, will become $\mathbf{W}_1$. Now, if $\mathbf{W}_1$ has a neighbour in A, which may happen in the track, this $\mathbf{W}_1$ becomes T which will disturb the computation. Even in the case it will not disturb the computation, the production of $\mathbf{W}_0$ nearby the track will be repeated periodically even when the computation has stopped. So that we cannot replace $\mathbf{W}_1$ by a state of $\mathcal{L}$. Can we replace $\mathbf{W}_0$ by a state of $\mathcal{L}$? The answer is yes: in the propagation context, a cell in $\mathbf{W}_0$ has at least four neighbours in $\mathbf{N}$. A cell of the track which is not the blank has two neighbours in $\mathbf{N}$ exactly, so that it is possible to distinguish the role of a distinguished α, depending on its neighbourhood. Moreover, a neighbour in $\mathbf{N}$ of a cell in α would remain unchanged due to the rules $\mathbf{N}$ $\mathbf{N}$ $\mathbf{N}$ $\mathbf{N}$ $\mathbf{N}$ $\mathbf{W}_0$ $\mathbf{N}$ and $\mathbf{N}$ $\mathbf{N}$ $\mathbf{N}$ $\mathbf{N}$ $\mathbf{N}$ $\mathbf{B}$ $\mathbf{N}$.

6.1. The propagation of the ray

Let us have a new inspection of Figure 3. The tables of [8] indicate the rules applied for going from one picture of the figure to the next one. As can be seen from the table, after time 7, no new rule is needed for the propagation of the ray. We can notice that these rules are obtained from those defined for the automaton $\mathcal{A}$, see [8], by replacing $\mathbf{W}$, $\mathbf{W}_0$ and $\mathbf{B}_0$ by T, 0 and A respectively.

However, we have to keep in mind that, during the propagation, the computation is going on. And so, we have to look at the cells of the track and of their neighbours in order to check that they are compatible with the rules for $\mathcal{C}$.

A rule which applies to a cell of the track is of the form

$$\mathbf{BNNB}\ T\ \mathbf{BB} \qquad\qquad (*).$$

As we know, $\mathbf{B}$ is a generic name for the states of $\mathcal{L}$. If we replace $\mathbf{B}$ by the states of $\mathcal{L}$, we get rules of the form

$$\alpha_0\mathbf{NN}\alpha_{-1}T\alpha_1\alpha_0^1 \qquad\qquad (**)$$

where $\alpha_{-1}\alpha_0\alpha_1 \rightarrow \alpha_0^1$ is a rule of $\mathcal{L}$. When $\alpha_0 \in \{B, y\}$, then the rule cannot be confused with one of $\mathcal{C}$ which we have already defined. At the times up to 5, when $\mathbf{B}$ is the current state, it is always $\sqcup$, the blank of $\mathcal{L}$. Now the rules for $\mathcal{A}$, where the current state is $\mathbf{B}$ and which are used up to time 6, either contain at least three consecutive occurrences of $\mathbf{N}$, or they contain an occurrence of A and so, they cannot be confused with a rule $(**)$ whose current state would be $\sqcup$. We have to look at all the other possibilities.

Consider the case when $\mathbf{B}$ is T: the rules for $\mathcal{A}$, where the current state is T are those of the cell 1(1) at times 3, 4 and 5, and those of cell 0 at times 5 and 6. Only one rule contains two consecutive occurrences of $\mathbf{N}$: the rule T $\mathbf{N}$ $\mathbf{N}$ T $\mathbf{B}$ $\mathbf{B}$ T. Note that the relative positions of these two occurrences of $\mathbf{N}$ and that of T are fixed. Now, when T occurs in a cell of the track, it is the single occurrence of this symbol on the track as this is the case with $\mathcal{L}$. In particular, in $(**)$, if $\alpha_0 = T$, then α_1 or α_{-1} must be $\sqcup$ and the other symbol is any one of $\mathcal{L}$ except the blank and T: see the table of the rules of $\mathcal{L}$ in [1]. Consequently, a rule of the track when the current state is T cannot be confused with the rules with T as the current state in the rules for $\mathcal{A}$.

Now, consider the case when $\mathbf{B}$ is 0 or A. We compare $(**)$ with the rules for $\mathcal{A}$ which have the same symbol as the current state. When it is A, the rules for $\mathcal{A}$ of $\mathbf{N}$ while there are only two of them in the rule $(**)$. For symbol 0, we have a similar argument: the rules $(**)$ with 0 as the current state have at least three consecutive neighbours in $\mathbf{N}$.

We remain with the case when $\mathbf{B}$ is the blank of $\mathcal{L}$. From the previous cases, we know that there is no possible confusion when the rule contains at least three consecutive occurrences of $\mathbf{N}$. Now, two rules have two consecutive occurrences of $\mathbf{N}$ exactly: the rule $\mathbf{B}\ \mathbf{N}\ \mathbf{N}\ \mathbf{B}\ T\ A\ \mathbf{B}$ and the rule $\mathbf{B}\ \mathbf{N}\ \mathbf{N}\ \mathbf{B}\ T\ \mathbf{B}\ \mathbf{B}$. These rules can be seen as rules of the form $(**)$ when $\mathbf{B}=\sqcup$. The corresponding rules are $\sqcup\ \mathbf{N}\ \mathbf{N}\ \sqcup\ T\ A\ \sqcup$ and $\sqcup\ \mathbf{N}\ \mathbf{N}\ \sqcup\ T\ \sqcup\ \sqcup$, respectively. Now, interpreted as rules induced by a rule of $\mathcal{L}$, the corresponding rules of $\mathcal{L}$ would be $\sqcup\sqcup A \rightarrow \sqcup$ and $\sqcup\sqcup\sqcup \rightarrow \sqcup$. Now, these rules are indeed present in the table of the rules of $\mathcal{L}$, see [1]. And so, at this stage, there is no confusion by replacing $\mathbf{W}$, $\mathbf{W}_0$ and $\mathbf{B}_0$ by T, 0 and A respectively.

6.2. The erasing of the support of the track

We have to look at the final stage of the process. When the halting is met, $\mathbf{N}$ is introduced onto the track and, as we know from Section 5, this starts the erasing process of the support of the track by propagation of $\mathbf{N}$ which successively replaces all occurrences of T and, at the end of the propagation of the ray, which stops the production of cells in 0.

This requires the following rules:

`0`	$T:$	$\mathbf{N}\ \mathbf{N}\ T\ \mathbf{N}\ T\ \mathbf{N}$	$T:$	$\mathbf{N}\ \mathbf{N}\ \mathbf{W}_1\ \mathbf{W}_1\ \mathbf{B}\ \mathbf{N}$
`0`	$\mathbf{N}:$	$\mathbf{N}\ \mathbf{N}\ T\ \mathbf{N}\ T\ \mathbf{N}$	$T:$	$\mathbf{N}\ \mathbf{N}\ 0\ 0\ \mathbf{B}\ \mathbf{N}$
`1(1)`	$T:$	$\mathbf{N}\ \mathbf{N}\ \mathbf{N}\ \mathbf{B}\ T\ \mathbf{N}$	$\mathbf{N}:$	$\mathbf{N}\ \mathbf{N}\ \mathbf{N}\ \mathbf{B}\ T\ \mathbf{N}$
`1(4)`	$T:$	$\mathbf{N}\ \mathbf{N}\ \mathbf{N}\ T\ \mathbf{B}\ \mathbf{N}$	$\mathbf{N}:$	$\mathbf{N}\ \mathbf{N}\ \mathbf{N}\ T\ \mathbf{B}\ \mathbf{N}$

Now, it is easy to see that none of them cannot be confused with the rule $T\ \mathbf{N}\ \mathbf{N}\ T\ \mathbf{B}\ T\ T$ nor a rule of the form $(**)$ as these latter rules have only two occurrences of $\mathbf{N}$. It can also be seen that the above rules cannot be confused with any of the other rules for $\mathcal{A}$ where the current state is T: again the number of occurrences of $\mathbf{N}$ is different. Indeed, this number is two or three in the above rules while it is at most one only in the rules for $\mathcal{A}$ except the rules $T\ \mathbf{N}\ \mathbf{N}\ T\ \mathbf{B}\ \mathbf{B}\ T$ and $T\ \mathbf{N}\ \mathbf{N}\ T\ \mathbf{B}\ T\ T$. But there can be no confusion with these latter rules either: they have T after the two occurrences of $\mathbf{N}$ while in the above rules with two occurrences of $\mathbf{N}$ exactly, which are also consecutive, there is 0 or $\mathbf{W}_1$ after the second $\mathbf{N}$.

We also have to check that the track is not disturbed by the replacement of T by $\mathbf{N}$. Indeed, this replacement has, as a consequence, that the form $(**)$ is replaced by the following one:

$$\alpha_0\mathbf{N}\mathbf{N}\alpha_{-1}\mathbf{N}\alpha_1\alpha_0^1 \qquad (***)$$

As the three occurrences of $\mathbf{N}$ in $(***)$ are not consecutive, there is no confusion with the rules for $\mathcal{A}$ where the current state is $\mathbf{B}$.

This completes the proof that $\mathcal{C}$ exactly simulates the computation of $\mathcal{L}$ with a true stopping of the cellular automaton in the case when the computation of $\mathcal{L}$ also stops. Accordingly we have proved the following result:

Theorem 6.1. *There is a rotation invariant hyperbolic cellular automaton in the pentagrid which starts from finite configurations and whose halting problem is undecidable which has* 9 *states exactly, the blank included.*

7. Conclusion

While stating Theorem 4.1, we have explained why we did not say that the cellular automaton $\mathcal{A}$ is strongly universal and the same explanation holds for the automata $\mathcal{B}$ and $\mathcal{C}$ of Theorems 5.1 and 6.1 respectively.

Can we still have a strongly universal cellular automaton with 9 states or possibly less?

One way to solve this problem would be to apply the technique of [1] to another small Turing machine. In [11] where the defect of this machine was first noticed, the author provides another Turing machine with 7 states and 4 letters which mimics any tag system of a given family, the same as for Minsky's machine, and the machine of [11] is actually universal. Another Turing machine with 7 states and 4 letters which is truly universal was later provided by R. Robinson, see [10]. It would be interesting to see whether a smaller machine, as the one devised by T. Neary and D. Woods with 6 states and 4 letters, see [9], could yield a better solution.

Accordingly, there is some work ahead, probably a tedious one if not more difficult.

Acknowledgement

The author is very much in debt to the referees for their remarks allowing him to improve the paper.

References

[1] Lindgren K. and Nordahl M.G., Universal computation in simple one-dimensional cellular automata. Complex Systems, **4**, 299–318, (1990).

[2] M. Margenstern, New Tools for Cellular Automata of the Hyperbolic Plane, *Journal of Universal Computer Science*, **6**(12), (2000), 1226–1252.

[3] M. Margenstern, Implementing Cellular Automata on the Triangular Grids of the Hyperbolic Plane for New Simulation Tools, **ASTC'2003**, (2003).

[4] M. Margenstern, Cellular Automata in Hyperbolic Spaces, Volume 1, Theory, *OCP*, Philadelphia, (2007), 422p.

[5] M. Margenstern, Cellular Automata in Hyperbolic Spaces, Volume 2, Implementation and computations, *OCP*, Philadelphia, (2008), 360p.

[6] M. Margenstern, A universal cellular automaton on the heptagrid of the hyperbolic plane with four states, *Theoretical Computer Science*, (2010), accepted.

[7] M. Margenstern, About the embedding of one dimensional cellular automata into hyperbolic cellular automata, *arXiv*:1004.1830[cs.FL], (2010), 19pp.

[8] M. Margenstern, An upper bound on the number of states for a strongly universal hyperbolic cellular automaton on the pentagrid, *arXiv*:1006.3451[cs.FL], (2010), 17pp.

[9] T. Neary, D. Woods, Four Small Universal Turing Machines, *Fundamenta Informaticae*, **91**(1), (2009), 123-144.

[10] R. Robinson, M. Minsky's small universal Turing machine, *International Journal of Mathematics*, **2**(5), (1991), 551-562.

[11] Yu. V. Rogozhin, Sem' universal'nykh mashin T'juringa. *Matematicheskie Issledovanija*, **69**, 76-90, 1982 (Seven universal Turing machines) (in Russian)

Journées Automates Cellulaires 2010 (Turku), pp. 180-190

TIME-SYMMETRIC CELLULAR AUTOMATA

ANDRÉS MOREIRA [1] AND ANAHÍ GAJARDO [2]

[1] Departamento de Informática and Centro Tecnológico de Valparaíso (CCTVal), Universidad Técnica Federico Santa María, Casilla 110-V, Valparaíso, Chile
E-mail address: amoreira@inf.utfsm.cl

[2] Departamento de Ingeniería Matemática and Centro de Investigación en Ingeniería Matemática (CI2MA), Universidad de Concepción, Casilla 160-C, Concepción, Chile

Abstract. Together with the concept of reversibility, another relevant physical notion is time-symmetry, which expresses that there is no way of distinguishing between backward and forward time directions. This notion, found in physical theories, has been neglected in the area of discrete dynamical systems. Here we formalize it in the context of cellular automata and establish some basic facts and relations. We also state some open problems that may encourage further research on the topic.

1. Introduction

An important property that may be present or not in physical or abstract dynamical systems is reversibility; consequently, it has also been an active topic of research in the context of cellular automata[8]. At least two particular reasons for this interest are often mentioned: on one hand, if CA are seen as models for massive distributed computation, then Landauer's principle suggests that we should focus on reversible cases. On the other hand, reversibility is often observed in real systems; it is therefore desirable in models of them[15]. Furthermore, a number of interesting results (like the dimension-sensitive difficulty of deciding reversibility[6]) have kept reversible CA in sight over the years.

However, there is one aspect of reversibility, as seen in real systems, which has been mostly neglected when considering cellular automata (in fact, for discrete dynamics in general): the dynamical laws governing physical reality seem to be not only reversible, but *time-symmetric*. For Newtonian mechanics, relativity or quantum mechanics, we can go back in time by applying the same dynamics, provided that we change the sense of time's arrow, through a specific transformation of phase-space. In the simplest example, Newtonian mechanics, the transformation leaves masses and positions unchanged, but reverses the sign of momenta.

Key words and phrases: time-symmetry, reversibility, universality.

This work has been supported by CONICYT FONDECYT #1090568 (A. Gajardo) and #1080592 (A. Moreira), as well as UTFSM grant 241016 (A. Moreira).

In the most general sense, we say that a dynamical system (X, T) is time-symmetric if there exists a reversible $R : X \to X$ such that $R \circ T \circ R^{-1} = T^{-1}$ [10](notice that this applies to systems with discrete or continuous time). However, time-symmetries observed in physical systems follow usually a more restricted definition, in which $R^{-1} = R$, and therefore R is an *involution* on X. This is a natural restriction, which follows whenever there is no way to distinguish where the arrow of time is heading. Apparent irreversibility (Loschmidt's paradox) comes only from macroscopic (*i.e.*, coarse-grained) differences in entropy.

Here we will discuss some basic facts about time-symmetric cellular automata, defined as those CA F for which there exists an involution H (which is a CA itself) such that

$$F^{-1} = H \circ F \circ H \tag{1.1}$$

Requiring H to be a CA is somewhat arbitrary, since for other systems the time-reversing transformation is not necessarily of the same nature as the dynamics (in fact, the physical theories discussed above are continuous in time). The reason for this restriction is that we expect reversibility (including the particular case of time-symmetry) to be a *local* property. Even if we do not address the case when H is not a CA, it may be an interesting direction for future studies.

CA are usually defined over a full shift $S^{\mathbb{Z}}$, but they can also be studied over (stable) subshifts. We remark that in this case, for time-symmetry to apply, the subshift must be stable for both F and H. This may cause some problems, since subsystems of a time-symmetric CA cannot be assumed to be time-symmetric too, even if they are stable for F.

2. Some motivating examples

Not only our models of physical reality turn out to exhibit time-symmetry; it is also found in some well known reversible discrete dynamical system. We show in this section how it applies to two 2D system, Margolus' billiard and Langton's ant. As a technical note, notice that in both cases the system is not originally described as a cellular automaton in the strict sense; therefore, we describe for each of them a CA that contains them as particular case for a subshift of valid configurations. This should -in principle- be followed by an extension of the rule to the full shift, and such that the system remains time-symmetric; however, doing that is not really required, since we want to show the time-symmetry of the original system; we hence restrict ourselves to the valid subshift.

Margolus' billiard. A well known example of time-symmetric CA is the Billiard ball model of Margolus [12]. It is not a proper CA, but rather a so-called *partitioned CA*, where the space $\mathbb{Z}^2$ is partitioned in 2×2 blocks of cells in two different ways (see Figure 1(a)). A transformation is applied to each block of each partition alternately. It is easy to see that such an automaton is reversible if and only if its local transformation is one-to-one. The rule used by Margolus is shown in Figure 1(b). It tries to emulate balls that move in straight lines, colliding elastically with each other or with static obstacles. The importance of this model comes from its Turing-universality, proved in [12] by computing reversible Fredkin gates [3].

We can express Margolus' system in terms of a CA with alphabet $\{white, black\} \times \{\nearrow, \searrow, \swarrow, \nwarrow\}$ and Moore neighbourhood. Here the first layer (the white/black component) represents the states of the original Margolus model, and the other represents the current partition, along with the relative position of the cell within its current block. This layer must be initialized in an appropriate way in order to work correctly (see Figure 1)(c).

Notice that reversing the arrows makes the partition flip to the alternative one. At each time step, each cell computes its next white/black state by applying Margolus' rule to the quadrant indicated by its arrow, and then reverses its arrow. Each of this actions -the first on the first layer, the second on the second- is an involution. Furthermore, if at one time step we omit any of them, further iterations will make the automaton evolve back in time.

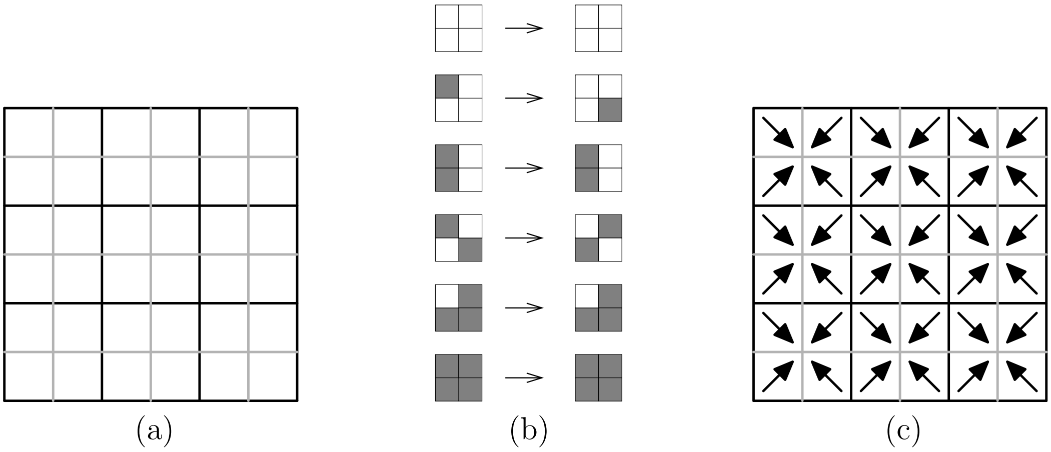


Figure 1: (a) The two partitions of the Margolus model are shown, one with solid lines and the other with dashed lines. (b) The Billiard Ball Model is defined through a permutation over 2 × 2 blocks of cells. (c) The current partition is obtained by grouping the four cells that point to the same point; reversing the arrows gives the alternate partition.

Langton's ant. Langton's ant was introduced in [11] together with several models emulating different life properties. It was also defined in physics as a model for particles presenting self correlated trajectories [1]. The model can be seen as a Turing machine working on a 2-dimensional tape. Its internal state is an arrow that represents its last movement direction. At each step, the ant turns to the left or to the right depending on the cell color (*white* or *black*), it flips this color and moves one cell forward (see Figure 2(a)). Besides being Turing-universal [4], its celebrity is due mostly to its particular behavior over finite initial configurations. Simulations show that it always falls eventually into a repetitive movement -of period 104- that makes it propagate unboundedly (see Figure 2(b)); this assertion has not been proved, and appears to be very difficult despite the simplicity of the transition rule.

Langton's ant can also be described in terms of a CA with Moore neighborhood and state cell $\{head, tail, empty\} \times \{white, black\}$. We represent the arrow through two adjacent cells, one in state *head* and the other in state *tail*. The cell in state *tail* always becomes *empty*, while the cell in state *head* always becomes *tail* and flip its color. Cells adjacent to a *head* can decide to become *head* themselves by looking at the tail position and the color of the head cell. The system simulates Langton's ant only if it starts with only one ant.

Here again, we can define the involution consisting in exchanging *tails* and *heads*. This immediately makes the ant come back to the cell it just had left, which it finds in the color opposite to the one it had found before, causing the ant to turn in the opposite direction, which in turn makes it again go to a previously visited cell, and so on: the ant will forever retrace (and undo) its past trajectory.

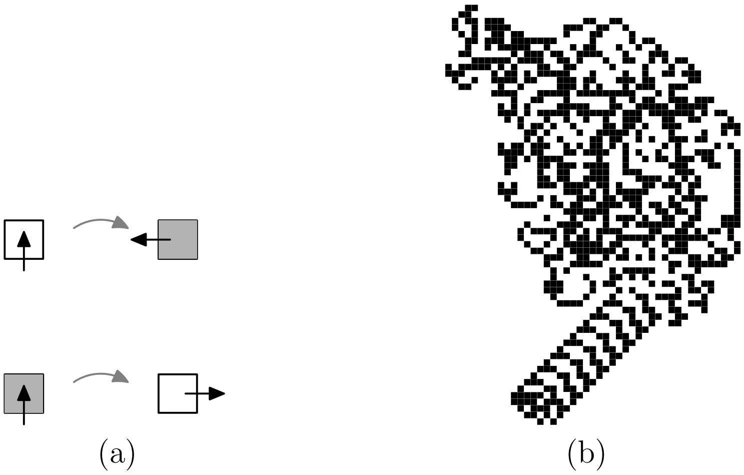


Figure 2: (a) Langton's ant rule. (b) Space configuration at iteration 10,837 after starting with every cell in *white* color.

3. Basic results

Proposition 3.1. *Let F be a CA. Then the following are equivalent:*

(1) *F is time-symmetric.*
(2) *There exists an involution H such that $(F \circ H)$ is an involution.*
(3) *F is the composition of two involutions.*

Proof. (1) $\Longrightarrow$ (2)
Let F and H be the CA satisfying 1.1. Then

$$(F \circ H)^2 \;=\; F \circ H \circ F \circ H \;=\; F \circ F^{-1} \;=\; id$$

(2) $\Longrightarrow$ (3)
Take H from (2) and let $G = F \circ H$ which is an involution. We have

$$F \;=\; F \circ id \;=\; F \circ (H \circ H) \;=\; (F \circ H) \circ H \;=\; G \circ H$$

(3) $\Longrightarrow$ (1)
Let G and H be involutions such that $F = G \circ H$. Then

$$F^{-1} \;=\; (G \circ H)^{-1} \;=\; H^{-1} \circ G^{-1} \;=\; H \circ G \;=\; H \circ G \circ H \circ H \;=\; H \circ F \circ H$$

■

Remarks 3.2. The following additional facts are noteworthy:

(1) If F is time-symmetric, then so is its inverse F^{-1}. Moreover, if $F = G \circ H$ is a decomposition into involutions, then $F^{-1} = H \circ G$ is a decomposition for the inverse. If H was the involution verifying 1.1, then G plays that role for F^{-1}.
(2) For any $i \in \mathbb{Z}$, F^i is also time-symmetric.
(3) The identity is a (trivial) involution; from there and the third condition we have that any involution is trivially time-symmetric.
(4) Not every reversible CA is time-symmetric. For example, σ (the shift): if for some H, $(\sigma \circ H) \circ (\sigma \circ H) = id$, since any CA commutes with the shift, we would have $\sigma^2 = id$, which is a contradiction.

The following diagram commutes:

$$\begin{tikzcd}
X \arrow[r, shift left, "h"] \arrow[d, shift right, "F"'] & X \arrow[l, shift left, "h"] \arrow[d, shift right, "F"'] \\
X \arrow[r, shift left, "h"] \arrow[u, shift right, "F^{-1}"'] & X \arrow[l, shift left, "h"] \arrow[u, shift right, "F^{-1}"']
\end{tikzcd}$$

Moreover, if we use $F = G \circ H$ to decompose the dynamics into the alternate applications of the involutions, so that successive configurations are computed as $c'_t = H(c_t)$, $c_{t+1} = G(c'_t)$, we get a dynamics $c_0, c'_0, c_1, c'_1, \ldots$, where both F and F^{-1} are being iterated: $c_{t+1} = F(c_t)$ and $c'_{t+1} = F^{-1}(c'_t)$. This curious situation is represented in Figure 3.

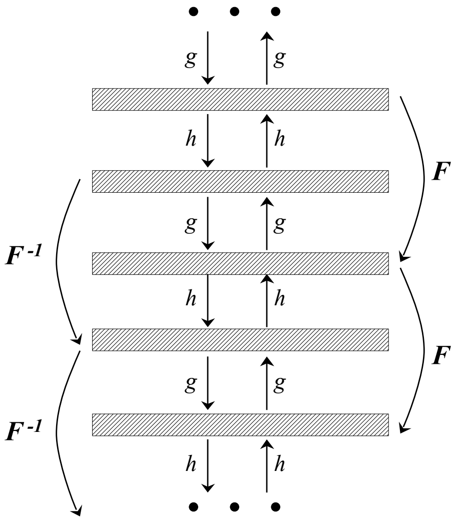


Figure 3: The decomposition of time-symmetric CA into alternating involutions creates a situation where both F and its inverse can be read from the space-time diagram as time moves forward (or backward).

4. Involutions

Involutions are quite infrequent in the space of CA. For example, the only elementary CA of period two are the identity and its negation.

It is easy to decide whether a given CA is an involution or not: we just have to compute its square and compare it to the identity. Nevertheless, if we want to enumerate the set of involutions, this procedure is very slow. A constructive characterization or a practical set of strong necessary conditions is still missing.

Meanwhile, it may be useful to consider restricted families of CA. For instance, if we restrict ourselves to additive CA, we get an alternative characterization in terms of coefficients. If we consider an additive CA of radius r defined on $(\mathbb{Z}_m)^{\mathbb{Z}}$ by the local rule

$$h(x_{-r},\ldots,x_0,\ldots,x_r) \;=\; \sum_{i=-r}^{r} a_i x_i$$

then by applying the rule twice and grouping the terms it is easily seen that h is an involution if and only if

$$a_0 + 2\sum_{i=1}^{r} a_i a_{-i} = 1 \quad \text{and} \quad \sum_{i=-r+\max\{j,0\}}^{r+\min\{j,0\}} a_i a_{j-i} = 0\,, \forall j \neq 0$$

For instance, for $m = 4$ and $r = 1$, we get $2a_{-1}a_0 = 2a_0a_1 = a_{-1}^2 = a_1^2 = 0$ and $a_0^2 + 2a_{-1}a_1 = 1$. Putting $a_{-1} = a_1 = 2$ we obtain all the zeroes, and with $a_0 = 1$ or $a_0 = 3$ we have the last condition. An example of an additive involution is thus

$$h(x_{-1},x_0,x_1) \;=\; (2(x_{-1}+x_1)+3x_0) \mod 4 \qquad (4.1)$$

Another well studied family of CAs are permutative ones. Unfortunately, we do not have a characterization there. A necessary condition is given by the following fact:

Proposition 4.1. *Given an involution h, the following two assertions are equivalent:*

- *h is left-permutative.*
- *h is one-way to the right*[1]*.*

Proof. If h is left-permutative of left radius l, h^2 is also left-permutative of left radius $2l$, but the left radius of h^2 is 0, then l is 0. Conversely, if h is oneway to the right and it is not permutative, there exists $x_1 ... x_n$ and y_1 such that $h(x_1, ..., x_n) = h(y_1, x_2, .., x_n)$. Taking any extension z of $x_1..x_n$ to $\mathbb{Z}$, and $z'_i = z_i$ for every $i \neq 1$ and $z'_1 = y_1$, we have that $h(z)_{[1,\infty[} = h(z')_{[1,\infty[}$. Thus, $z'_1 = h^2(z)_1 = h^2(z')_1 = z_1$, which is a contradiction. ■

Thus the involution in (4.1), which is clearly two-way, is not permutative. An example of a permutative involution of radius r is:

$$h(x_{-r},..x_0) = \begin{cases} 1 & \text{if } x_{-r} = 0 \wedge x_0 = 2 \\ 2 & \text{if } x_{-r} = 0 \wedge x_0 = 1 \\ x_0 & \text{otherwise} \end{cases} \qquad (4.2)$$

This kind of construction is the simplest one, with permutations which are transpositions and which do not affect the states that regulate their application. More

[1] The neighbourhood is a finite subset N of $\mathbb{N}_0$.

complicated examples may have associated permutations which are not transpositions.

Example (4.2), incidentally, shows how an involution can have an arbitrarily large neighbourhood. This long-distance dependence may be lost when it is composed with another involution, making the determination of time-symmetry non-trivial. For instance, the involution (4.2) yields the permutation (12) of radius 0 when it is composed with

$$g(x_{-r},..x_0) = \begin{cases} 1 & \text{if } x_{-r} \neq 0 \wedge x_0 = 2 \\ 2 & \text{if } x_{-r} \neq 0 \wedge x_0 = 1 \\ x_0 & \text{otherwise} \end{cases}$$

5. Diversity in the class of time-symmetric CA

One simple example of time-symmetric automata is given by the following.

Proposition 5.1. *Every reversible CA of radius 0 is time-symmetric.*

Proof. Let f be the local rule of a reversible CA of radius 0, and let S be its set of states. Suppose first that $f : S \to S$ is a cyclic permutation and, without loss of generality, that $S = \{0,..,n-1\}$ and $f(i) = i+1 \mod n$. Let us define the involution $h(i) = n-i-1$. Consider $g = h \circ f$; for $i < n-1$, $g(i) = h(f(i)) = h(i+1) = n-(i+1)-1 = n-i-2$, while otherwise $g(n-1) = h(f(n-1)) = n-1$. Thus g is an involution: $g^2(n-1) = g(n-1) = n-1$, and for other i, $g^2(i) = g(n-i-2) = n-(n-i-2)-2 = i$. Since $f = h \circ h \circ f$=$h \circ g$ is the composition of two involutions, it is time-symmetric.

If f decomposes into more than one cycle, we define h and g as before over each of them, obtaining again a decomposition into involutions. ■

It is important to notice here the preservation of time-symmetry under conjugacy.

Proposition 5.2. *If F is conjugated to T and T is time-symmetric, then F is also time-symmetric.*

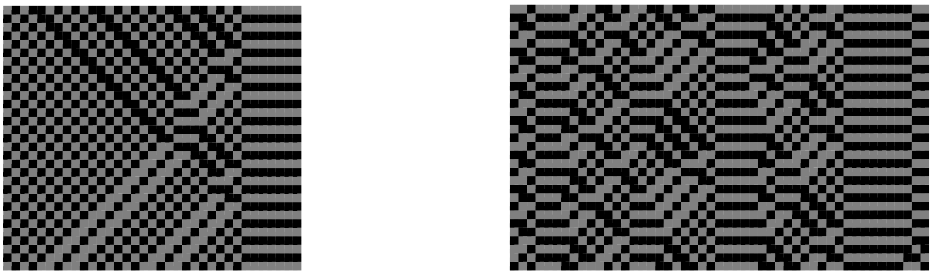
Proof. From time-symmetry, there is an involution H such that $T^{-1} = H \circ T \circ H$. From conjugacy, there is a bijective, continuous, shift-commuting ϕ such that $T = \phi \circ F \circ \phi^{-1}$. Then we have (removing the composition symbol, for clarity) that

$$F^{-1} = \phi^{-1}T^{-1}\phi = \phi^{-1}HTH\phi = \phi^{-1}H\phi F\phi^{-1}H\phi = GFG$$

and $G = \phi^{-1}H\phi$ is cleary an involution, making F time-symmetric. ■

Periodic CA of radius $r > 0$ behave almost like a CA with radius 0, in the sense that information cannot travel “very far”; this makes them nearly time-symmetric, because they are conjugated to a subshift of a radius 0 CA. To see this, let F be a p-periodic CA with states S and define $\varphi : S^{\mathbb{Z}} \to (S^p)^{\mathbb{Z}}$ as $\varphi(x)_i = (x_i, F(x)_i, .., F^{p-1}(x)_i)$. This φ is continuous and injective, and the induced CA F' in $(S^p)^{\mathbb{Z}}$ has radius 0 and period p; its local rule is $f'(a_0, a_1, .., a_{p-1}) = (a_1, a_2, .., a_{p-1}, a_0)$. From Proposition 5.1 we see that F' is time-symmetric; moreover, $\varphi(S^{\mathbb{Z}})$ is F' invariant. However, $\varphi(S^{\mathbb{Z}})$ is not invariant for the involution defined in the proof, and therefore we cannot conclude that F is time-symmetric.

When we consider the group of reversible CA with the composition operation, involutions correspond to elements of order two. These objects were already considered by Hedlund *et al* in [5], where the last theorem shows that the composition of two involutions (i.e., time-symmetric CA) can have infinite order. The example that proves this theorem is defined on alphabet $\{0,1\}$ and consists in the composition of α, the negation of the identity, and β, a CA that negates x_i if and only if $x_{i-1}x_{i+1}x_{i+2} = 101$. Figure 4 shows simulations of this CA; the Hedlund's proof exhibited a configuration with infinite orbit consisting in one traveling signal over a periodic background like the one appearing in Figure 4(a).



(a) (b)

Figure 4: Simulations of the time-symmetric CA defined by Hedlund *et al* on periodic boundary conditions. (a) Two traveling signals. (b) A simulation over a random initial configuration.

This already suggests a variety of dynamical behaviors within the class. But the examples given in Section 2 are even more interesting, as they correspond to Turing-complete systems. The following results shows that, indeed, the whole range of reversible dynamical behaviors can be observed in time-symmetric CA.

Proposition 5.3. *Let F be a 1D reversible CA. Then there exists a 1D CA $\tilde{F}$ which is time-symmetric and simulates F in real time.*

Proof. Let f be the local rule of F and denote with f^{-1} the local rule of its inverse F^{-1}; let ℓ and r be large enough so that $N = \{-\ell, \dots, r\}$ contains the neighbourhoods of both f and f^{-1}; finally, let S be the set of states. We define the CA $\tilde{F}$ with neighbourhood N and states S^2, through the local rule

$$\tilde{f}\left((x_{-\ell}, y_{-\ell}), \dots, (x_r, y_r)\right) \;=\; (f(x_{-\ell}, \dots, x_r), f^{-1}(y_{-\ell}, \dots, y_r))$$

$\tilde{F}$ simulates F in real time: to project the space-time diagram of $\tilde{F}$ into that of F, we just discard the second component of the ordered pairs. By discarding the first component instead, we note that $\tilde{F}$ simulates F^{-1} as well.

Let H be the involution given by the radius 0 local rule $h(x,y) = (y,x)$. Abusing notation, denote configurations $c \in (S^2)^{\mathbb{Z}}$ as pairs $(x,y) \in (S^{\mathbb{Z}})^2$. Then we have

$$\tilde{F} \circ H(x,y) \;=\; \tilde{F}(y,x) \;=\; (F(y), F^{-1}(x))$$

and

$$(\tilde{F} \circ H)^2(x,y) \;=\; \tilde{F} \circ H(F(y), F^{-1}(x)) \;=\; \tilde{F}(F^{-1}(x), F(y)) \;=\; (x,y)$$

and thus $\tilde{F}$ is time-symmetric. ■

Cellular automata are said to be intrinsically universal if they are able to simulate any other CA. The details vary according to the accepted notion of *simulation*, from which there is a variety. Delorme *et al* [2] have recently reviewed and completed the study of three of these, *surjective, injective* and *mixed* simulation, and shown that for every pair of CA F and G, $F \times G$ simulates both F and G in all three senses.

Corollary 5.4. *There exist time-symmetric CA which are intrinsically universal within the class of reversible CA.*

Proof. This follows from the previous results and comment, and from the existence of reversible intrinsically universal CA (see for example [14]). ∎

Notice that reversible CA cannot simulate arbitrary CA: intrinsic universality is therefore limited to the reversible class, and time-symmetric CA are as general as reversible CA can get. Turing-universality is not limited by reversibility (information can be "swept away" to preserve it and maintain reversibility) and hence is implied by reversible intrinsic universality.

Not every reversible CA is time-symmetric; a simple example is the shift σ, which commutes with every CA and therefore cannot satisfy equation (1.1) for any involution H. But in general, it is not easy to prove non-time-symmetry. An interesting theory which may provide better tools for doing this, and possibly for characterizing time-symmetry, is the one developed by Kari in [7]. We will not reproduce here his construction, but one important fact is the following: he introduces a morphism h_- from the set of reversible CA with the composition $(Aut(A), \circ)$ into the set of rational numbers with the multiplication $(\mathbb{Q}, \cdot)$: $h_-(f \circ g) = h_-(f)h_-(g)$ for all reversible f, g. Clearly, involutions and every periodic CA are in the kernel of h_-. Moreover, since time-symmetric CA are compositions of involutions, they are in this kernel as well. We do not presently know whether they are identical to the whole kernel or not.

Kari proves that reversible CA which are not in this kernel are compositions of some element of the kernel with a *partial shift* which is easily computed from the value of h_-; in turn, every element of the kernel can be written as a composition of two block permutations (akin to Margolus rule), and thus he expresses reversible CA in an explicitly reversible way. Our motivation here is different, but the approach is promising and the connection should be explored. As a first conclusion, we obtain that every CA in the kernel of h_- is a composition of two time-symmetric CAs, and hence is also the composition of four involutions.

6. Conclusions

We believe that this note just scratches the surface of the topic of time-symmetry in cellular automata; their rich internal structure and the connection to physical models suggests that much more can be done with them.

On the other hand, as shown by the examples in Section 2, time-symmetric CA are actually quite familiar to CA researchers, and have appeared in different contexts. Some cases are very explicit, like the automata constructed in the proof of undecidability of periodicity [9], which actually include an "arrow of time" toggle. Moreover, there are ways of constructing CA rules that make the construction of time-symmetric CA straightforward. For instance, Margolus' billiard is an example of a block automata, *i.e.*, a system which is a composition of two functions applied to independent blocks of the configuration. By incorporating the current function and block to be applied into the configuration, a block automaton can always be expressed as a CA. Defining an involution that toggles the current block is a good idea to prove time-symmetry, but it only works if both block functions are also involutions. What must be stressed is that this is only a *sufficient* condition; the system may be time-symmetric by means of an entirely different involution.

Likewise, partitioned CA (in the sense of Morita [13]) can easily give birth to time-symmetric CA. In that case, cells are partitioned into sub-cells, one for each neighbours; iteration proceeds by the alternation between an exchange step, where cells exchange the contents of the sub-cells associated to each other, and a step which applies a block transformation on the cell. This scheme was succesfully used to construct reversible CA (all we need is a reversible block transformation), and can produce time-symmetric CA as well if the block transformation is chosen as an involution: the exchange step already is one. Again, what we want to stress is that this is a sufficient condition: we could have a partitioned CA which is time-symmetric while having a non-involutive block transformation, if the decomposition happens to be another one.

There are several interesting questions that should probably be addressed next, and have appeared along this text:

- Is there a constructive characterization of CA involutions that can make their enumeration practical? Right now the only way we have to find the involutions is to test all CA exhaustively; some trivial necessary conditions can be used to reduce the search, but they are not enough to make it efficient.
- Is time-symmetry a decidable property? Since the definition calls for the existence of an involution that verifies a condition, a bound on the necessary neighbourhood for the involution would be enough to ensure decidability.
- Do time-symmetric CA correspond to the kernel of Kari's h_- morphism?

We conjecture a positive answer for these three questions, at least in dimension 1; Kari's result on undecidability of reversibility in dimension 2 [6] suggests that answers here may be dimension-sensitive too. The answers to the questions may be related to each other. For instance, a better understanding of the structure of involutions may be useful for bounding the required neighbourhood and thus deciding time-symmetry. On the other hand, since h_- is easily computed, a positive answer to the third question would imply a positive answer to the second as well.

Notice that if the answer to the third question is positive, then time-symmetric CA would be closed under composition. This is by no means obvious, and in fact it is a further interesting open question.

A further direction for future work may be the study of time-symmetry in other discrete dynamical systems. In each case an important issue is to precise what kind of involution is to be applied. Generally speaking, what we need is an involutive and hopefully local transformation of the system's configuration. That transformation may not be, in general, an object of the same kind as the dynamics itself: that was the case for CA because of the special nature of CA, which transform the whole configuration in discrete time too, and will be the case for automata networks in general. In other cases, like for instance Turing machines, it is not only difficult (the composition of two Turing machines moves the head two steps, and is no longer a Turing machine unless we extend the definitions) but also not expected; rather, for

Turing machines, the involution would likely be a transformation on the tape (a CA involution?) along with a change in the current state of the machine. Finally, the locality of the time-reversing involution is not completely granted either: even in CA, it would be interesting to see what happens if that requirement is removed.

Acknowledgment

Andrés Moreira thanks Gaetan Richard for useful remarks on the characterization of time-symmetry, and Nicolas Ollinger and the Laboratoire d'Informatique Fondamentale de Marseille for the visit during which that discussion took place.

References

[1] E. G. D. Cohen. New types of diffusion in lattice gas cellular automata. In M. Mareschal and B. Holian, editors, *Microscopic Simulations of Complex Hydrodynamic Phenomena.* Plenum Press, 1992.

[2] M. Delorme, J. Mazoyer, N. Ollinger, and G. Theyssier. Bulking II: Classifications of cellular automata. 2010.

[3] E. Fredkin and T Toffoli. Conservative logic. *Int. J. of Theoret. Phys.*, 21(3/4):219–253, 1982.

[4] A. Gajardo, A. Moreira, and E. Goles. Complexity of Langton's ant. *Discrete Applied Mathematics*, 117(1-3):41 – 50, 2002.

[5] G. A. Hedlund. Endomorphisms and automorphisms of the shift dynamical systems. *Mathematical Systems Theory*, 3(4):320–375, 1969.

[6] J. Kari. Reversibility of 2d cellular automata is undecidable. *Physica D: Nonlinear Phenomena*, 45(1-3):379 – 385, 1990.

[7] J Kari. Representation of reversible cellular automata with block permutations. *Mathematical Systems Theory*, 29(1):pp 47–61, 1996.

[8] J. Kari. Reversible cellular automata. In Clelia De Felice and Antonio Restivo, editors, *Developments in Language Theory*, volume 3572 of *Lecture Notes in Computer Science*, pages 57–68. Springer Berlin / Heidelberg, 2005.

[9] J. Kari and N. Ollinger. Periodicity and immortality in reversible computing. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008*, volume 5162 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2008.

[10] J. S. W. Lamb and J. A. G. Roberts. Time-reversal symmetry in dynamical systems: A survey. *Physica D: Nonlinear Phenomena*, 112:1 – 39, 1998.

[11] C. G. Langton. Studying artificial life with cellular automata. *Physica D*, 22:120–149, 1986.

[12] N. Margolus. *Physics and Computation.* PhD thesis, M. I. T., Cambridge, Mass., U.S.A., 1987.

[13] K Morita and M Harao. Computation universality of one-dimensional reversible (injective) cellular automata. *The Trans. of the IEICE*, E72-E(6):758–762, 1989.

[14] N. Ollinger. Universalities in cellular automata: a (short) survey. In B. Durand, editor, *Symposium on Cellular Automata - Journes Automates Cellulaires (JAC'2008)*, pages 102–118, Moscow, 2008. MCCME Publishing House.

[15] T. Toffoli and N. H. Margolus. Invertible cellular automata: A review. *Physica D: Nonlinear Phenomena*, 45(1-3):229 – 253, 1990.

Journées Automates Cellulaires 2010 (Turku), pp. 191-202

YET ANOTHER APERIODIC TILE SET

VICTOR POUPET

LIF Marseille
E-mail address: victor.poupet@lif.univ-mrs.fr

ABSTRACT. We present here an elementary construction of an aperiodic tile set. Although there already exist dozens of examples of aperiodic tile sets we believe this construction introduces an approach that is different enough to be interesting and that the whole construction and the proof of aperiodicity are hopefully simpler than most existing techniques.

Aperiodic tile sets have been widely studied since their introduction in 1962 by Hao Wang [7]. It was initially conjectured by Wang that it was impossible to enforce the aperiodicity of a coloring of the discrete plane $\mathbb{Z}^2$ with a finite set of local constraints (there either was a valid periodic coloring or none at all). This would imply that it was decidable whether there existed a valid coloring of the plane for a given set of local rules. This last problem was introduced as the *domino problem*, and eventually proved undecidable by Robert Berger in 1964 [1, 2]. In doing so, Berger produced the first known aperiodic tile set: a set of local rules that admitted valid colorings of the plane, none of them periodic. Berger's proof was later made significantly simpler by Raphael M. Robinson in 1971 [12] who created the set of Wang tiles now commonly known as the *Robinson tile set*.

Since then many other aperiodic tile sets have been found, not only on the discrete plane [3, 4, 8, 9], but also on the continuous plane [5, 11].

In this article we will describe yet another construction of an aperiodic tile set. Although the resulting tile set will produce tilings quite similar to those of Robinson's (an infinite hierarchical structure of embedded squares) the local constraints will be presented in a (hopefully) more natural way: we will start from simple geometrical figures and organize them step by step by adding new rules progressively.

1. Tilings

There are many different ways to define tilings of the discrete plane $\mathbb{Z}^2$. The historical definition as presented by Wang is that of unit square tiles with colored edges (nowadays called *Wang tiles*). The domino problem is to decide whether one can arrange copies of a given set of such tiles on the plane so that the adjacent sides of two neighbor tiles have the same color.

Key words and phrases: tiling, Wang tile, tile set, aperiodic, domino problem.
The author is partly supported by ANR-09-BLAN-0164.

Although the description of the problem with Wang tiles is extremely simple, it is not the easiest way to deal with tilings. A more modern approach consists in defining a tile set as a set of local constraints in the form of a finite set of forbidden patterns. A tiling of the plane according to such a tile set is a coloring of the plane such that no forbidden pattern appears. Both definitions are known to be equivalent.

1.1. Patterns and Configurations

Definition 1.1 (Configuration)**.** Given a finite set of symbols Σ, a Σ-*configuration* is a mapping $\mathfrak{C} : \mathbb{Z}^2 \to \Sigma$ that associates a symbol of Σ to each element of $\mathbb{Z}^2$ (elements of the plane $\mathbb{Z}^2$ will be referred to as *cells*).

Definition 1.2 (Pattern)**.** Given a finite set of symbols Σ, a Σ-*pattern* is a mapping $\mathcal{P} : D_\mathcal{P} \to \Sigma$ from a finite subset of cells $D_\mathcal{P} \subset \mathbb{Z}^2$ to Σ.

Definition 1.3 (Tile Set)**.** A *tile set* is a couple $\tau = (\Sigma, \mathcal{F})$ where Σ is a finite set of symbols and $\mathcal{F}$ is a finite set of patterns called *forbidden patterns.*

Definition 1.4 (Tilings)**.** Given a finite set of symbols Σ, we say that a Σ-pattern $\mathcal{P} : D_\mathcal{P} \to \Sigma$ *appears* in a Σ-configuration $\mathfrak{C}$ if there exists a vector $v \in \mathbb{Z}^2$ such that

$$\forall x \in D_\mathcal{P}, \mathcal{P}(x) = \mathfrak{C}(x + v)$$

A Σ-configuration $\mathfrak{C}$ is said to be *valid* for a tile set $\tau = (\Sigma, \mathcal{F})$ if it contains none of the patterns in $\mathcal{F}$. A tiling of the plane by a tile set τ is a valid configuration for τ.

1.2. Periodicity

Definition 1.5 (Periodicity)**.** A configuration $\mathfrak{C}$ is said to be *periodic* if there exists a non-zero vector $v \in \mathbb{Z}^2$ such that $\mathfrak{C}$ is invariant by a translation of v (v is a vector of periodicity of $\mathfrak{C}$):

$$\forall x \in \mathbb{Z}^2, \mathfrak{C}(x) = \mathfrak{C}(x + v)$$

A configuration is said to be *bi-periodic* if it has two independent vectors of periodicity.

Definition 1.6 (Aperiodicity)**.** A tile set is said to be *aperiodic* if it admits at least one tiling of the plane but admits no periodic tiling.

The two following propositions will be of use later. The first one is folklore and its proof will be omitted.

Proposition 1.7. *If a tile set admits a valid periodic tiling of the plane it admits a valid bi-periodic tiling of the plane.*

Remark 1.8. The contraposition of Proposition 1.7 states that if a tile set cannot tile the plane bi-periodically it cannot tile it periodically either. We will use this in our construction of an aperiodic tile set as it is easier to prove that there exist no bi-periodic valid configuration.

Proposition 1.9. *If a configuration is bi-periodic it has both a vertical and a horizontal vector of periodicity.*

Proof. If (x, y) and (x', y') are two independent vectors of periodicity of a configuration then $x'.(x, y) - x.(x', y') = (0, x'y - xy')$ and $y'.(x, y) - y.(x', y') = (xy' - x'y, 0)$ also are. ∎

2. Construction of an Aperiodic Tile Set

2.1. General Overview

The aperiodic tile set that we are going to describe is based on the following simple observation: if a picture contains arbitrarily large squares such that none of these squares intersect each other, the picture cannot be periodic. Indeed, because squares do not intersect a translation vector that leaves the picture unchanged must be larger than the side of every square for if a square is translated less than the length of its sides it intersects its original position.

What we will do now is design a set of local constraints that only accepts pictures on the discrete plane that contain arbitrarily large non-intersecting squares.

These "pictures" will contain lines of different sorts made of horizontal, vertical and diagonal segments. To be consistent with the previous definitions of configurations, tile sets and tilings we should be describing configurations as symbols on the cells of $\mathbb{Z}^2$. However it will be much easier to explain (and understand) the construction by describing geometrical shapes.

This means that when we will say something like "blue lines are made of horizontal and vertical segments, have no extremities and cannot cross" what this really means is that we have symbols representing blue lines going through cells vertically, horizontally and changing directions (for example entering from the top side and exiting from the right side). Once represented, it is easy to enforce the stated properties with a set of forbidden patterns. In this case the forbidden patterns are those where a blue line is interrupted because it exits a cell from one side but does not enter its neighbor from the corresponding side. The fact that blue lines cannot cross is simply enforced by having no symbol corresponding to a crossing on a cell (no symbol corresponds to a blue line going through a cell both vertically and horizontally).

Our construction will consist in two main types of lines that we will call "blue lines" and "arms". Blue lines will be made to draw non intersecting squares while arms will be used to control the size of squares and connect them together to build a structure that enables us to prove the existence of arbitrarily large squares.

2.2. Blue Lines

Blue lines are made of vertical and horizontal segments. They have no extremities (only infinite or closed paths) and cannot cross or overlap. They are oriented (they have an *inner* and an *outer* side) and can only change their direction by turning towards the inside.

All of these rules are local conditions and can therefore be enforced by a tile set.

Because blue lines cannot cross and can only turn towards the inside, finite blue lines can only be rectangles. For the same reasons, infinite blue lines can only be of three kinds, each corresponding to a degenerate rectangle with some bi-infinite or semi-infinite sides (see Figure 1).

2.3. Blue Squares

We now want to make sure that only squares are valid. To do so, the usual method is to draw a diagonal line from the upper-left and lower-right angles of every

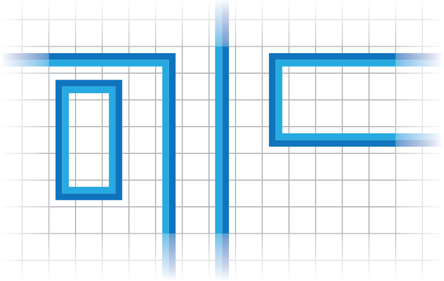


Figure 1: Possible blue paths. The infinite paths can have 0, 1 or 2 angles.

blue rectangle. This diagonal line is oriented towards the inside of the rectangle and is not allowed to meet a blue line other than the angles from which it starts. If the rectangle is a square, the two diagonal lines merge into one but if it is not a square the diagonals will reach a side of the rectangle, which is forbidden.

This however only works if there are no smaller squares inside larger ones. Because we need some small blue squares to lie on the diagonal of larger ones, we will have to allow the diagonal line to "go around" a square, but only from one corner to the other as shown in Figure 2.

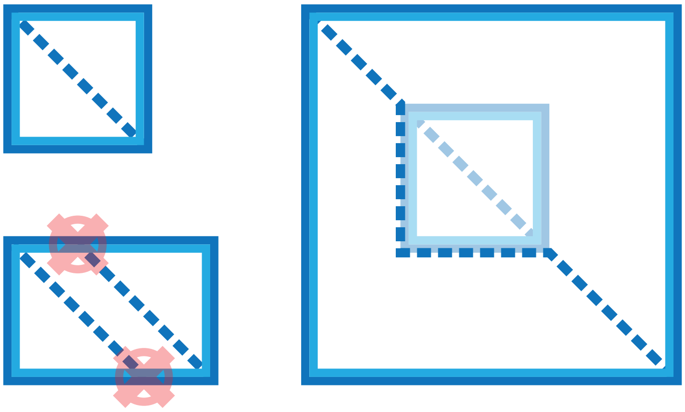


Figure 2: Diagonal line used to ensure all closed blue paths are squares

We can show inductively that all closed blue paths are squares:

- if a path has no other blue path inside, the diagonal line goes straight from its upper-left to its lower-right corners, it is therefore a square;
- if all blue paths inside a larger one are squares, the diagonal line only goes around squares and hence it remains on the real diagonal of the larger one, the large one is a square too.

Remark 2.1. A small blue square inside a larger one can only be either perfectly aligned with the latter's diagonal or far enough from it so that it does not intersect it.

2.4. Infinite Paths

Lemma 2.2. *The only possible infinite blue paths in a valid bi-periodic configuration are infinite straight lines (no angle).*

Proof. According to the basic rules of blue lines, infinite blue paths can be of three different kinds (illustrated by Figure 1): they can have zero, one or two angles.

However, because of the diagonal line that starts from the upper left and lower right angles of any blue line, infinite paths with two angles cannot be valid (see Figure 3).

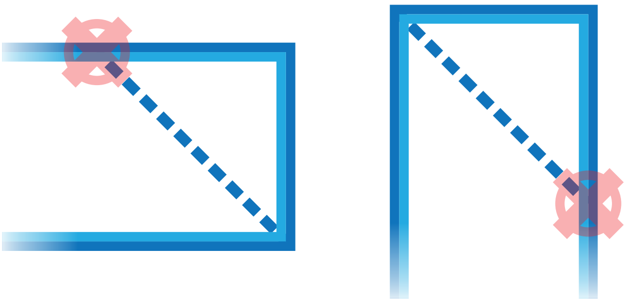


Figure 3: Two-angled infinite paths cannot be valid.

Moreover, no valid bi-periodic configuration can contain an infinite path having only one angle. Indeed, such a configuration must be both horizontally and vertically periodic (Proposition 1.7) and any finite horizontal or vertical translation of the infinite angle would intersect it.

The only remaining case of infinite blue path is that of bi-infinite vertical or horizontal straight lines (with no angle). ■

2.5. Arms

Blue squares alone are not sufficient to ensure the aperiodicity of the tile set. What we will do now is organize them into groups in such a way that for every blue square of finite size we can prove the existence of a larger finite blue square. In order to group them, we extend vertical and horizontal lines from every corner of a blue square towards the exterior (see Figure 4). These new lines are called *arms*.

The basic properties of arms can be described by the following rules:

- Arms are horizontal or vertical continuous straight lines. They do not turn.
- Arms and blue lines cannot overlap.
- Arms are allowed to cross other perpendicular arms.
- The extremities of an arm must be angles of blue paths (some extremities might not exist if the arm is semi or bi-infinite).
- The orientations of two blue squares connected by an arm must match: an arm cannot connect the upper (resp. right) side of a square to the lower (resp. left) side of another.
- There can be at most one point on an arm where it crosses a blue line.

The last rule is the key to most of the properties that we will need later. It might appear as a non-local constraint as it is formulated as a global condition on the arm but it can be enforced locally by orienting the arms from their extremities as shown in Figure 5: blue lines are only allowed to cross an arm where the two opposite orientations meet (which needs not be the middle of the arm).

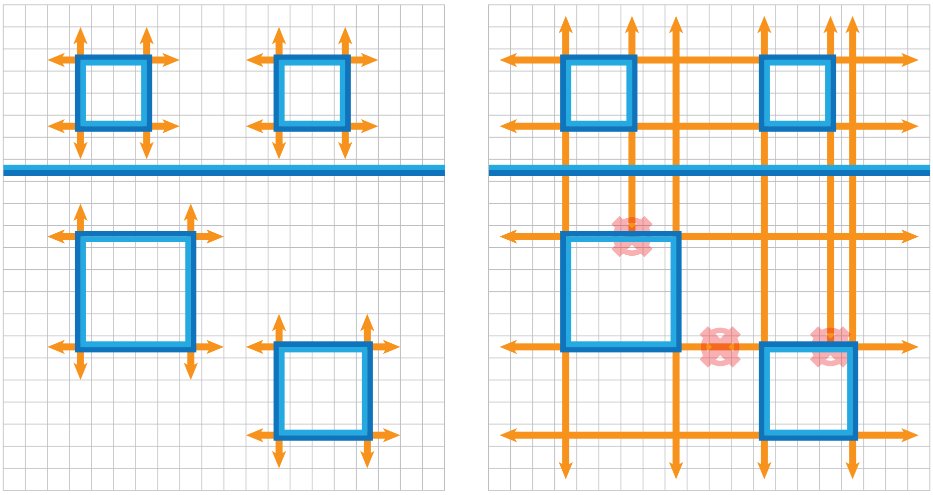


Figure 4: Arms extend from every angle of a blue path towards the exterior. In the right part there are three errors: the middle one is an orientation error (down side connected to an up side) while the two others are arms that cross more than one blue line.

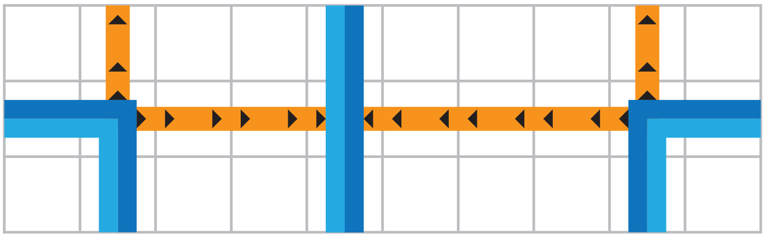


Figure 5: Orientation on the arms to enforce locally the fact that an arm can cross at most one blue line. The blue line can only cross where the orientations meet.

Two blue squares are said to be *neighbors* if they are connected by an arm.

2.6. Size Matching

Lemma 2.3. *In a valid bi-periodic configuration, if two blue squares are neighbors they are of equal size.*

Proof. By contradiction, let us assume there exists a valid bi-periodic configuration having two connected squares of different size. Let us consider one of the smallest squares so connected to a larger square. In order to describe the situation, we will consider that the two squares are connected horizontally by an arm joining their lower sides (as shown in Figure 6).

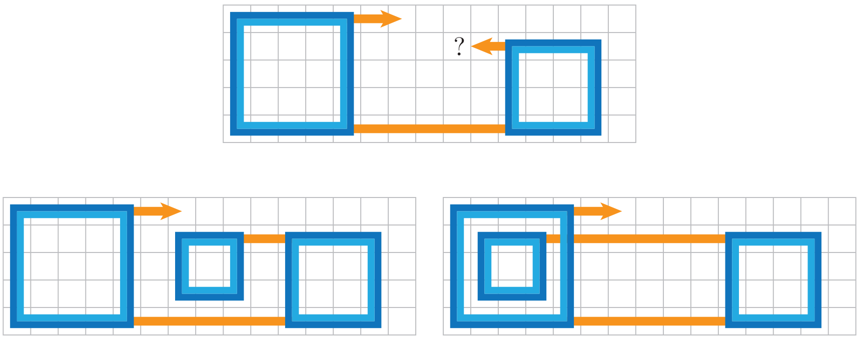


Figure 6: Why arms cannot connect squares of different sizes.

There has to be an arm that starts from the upper side of the smaller square and goes towards the larger. Because this arm is not allowed to cross two blue lines, it cannot go entirely through the larger square. Thus it must be connected to the upper side of another blue square, either before entering the larger square or inside it (both cases are illustrated by Figure 6). In both cases, this third square at the other extremity of the arm must be smaller than the initial small square :

- if the third square is outside of the larger one, its vertical sides cannot cross the arm connecting the two initially considered squares for this would mean this arm is crossed by two blue lines;
- if the third square is inside the larger one, it cannot cross the side of the larger square;

This contradicts the fact that the initial square was chosen as being one of the smallest squares connected to a square of different size. ■

Lemma 2.4. *In a valid bi-periodic configuration, every finite blue square has exactly four neighbors, one in each direction, and it is connected to each of its neighbors by two arms.*

Proof. Because all connected squares have the same size, if two squares are connected by an arm, they are also connected by a second arm. Since every finite blue square has eight arms, it is connected to at most four neighbors.

Moreover there can be no semi-infinite horizontal or vertical line in a configuration that is both vertically and horizontally periodic (the line would have to be bi-infinite) hence every arm connected to a square is connected to another one. Every square must then have at least four neighbors, one in each direction. ■

2.7. Groups

It is now time to add a communication between the different finite blue squares in order to organize them in groups in such a way that for each group of neighbor squares there exists a larger finite square associated with this group, in turn leading to the proof that there exist arbitrarily large finite squares.

To do this, we add two coordinates $(x, y) \in (\mathbb{Z}/3\mathbb{Z})^2$ to every blue line, and the arms only allow a connection that corresponds to a correct arrangement of squares:

the right neighbor of a square (x, y) must have coordinates $(x+1, y)$ and its up neighbor must have coordinates $(x, y+1)$ (all additions are performed modulo 3).

To realize this with a tile set we use different sorts of blue lines for each possible set of coordinates (9 possibilities), and different sorts of arms depending on the coordinates of the squares they connect. Obviously we require that the coordinates of a blue line are constant along the line (which is a local condition) and that the coordinates of an arm (the coordinates of the squares it connects) are also constant along one arm.

The structure is then enforced by the arms at their extremities: a horizontal arm whose coordinates are $((x, y), (x+1, y))$ must be connected to a square of coordinates (x, y) by its left extremity and to a square of coordinates $(x+1, y)$ by its right extremity, and similarly with vertical arms of coordinates $((x, y), (x, y+1))$.

Lemma 2.5. *In a valid bi-periodic configuration, if there exists a blue square then there exists a blue square of the same size with coordinates* $(1, 1)$.

Proof. This is a straightforward consequence of Lemmas 2.3 and 2.4 and the coordinates system. All squares have neighbors in all directions and all neighbors have the same size. Because the first (resp. second) coordinate is incremented by 1 modulo 3 each time we consider the right (resp. up) neighbor, we eventually find a square of coordinates $(1, 1)$. ∎

The construction is now nearing its end. All we need to do is ensure that in any possible bi-periodic configuration, for every finite blue square there exists another blue square that is larger. The easy way to prove that a square is larger than another is to have the large one contain the other. Because for every square there is a $(1, 1)$ square of the same size, it is enough to make every $(1, 1)$ square be inside a larger one.

To do so, we slightly change the arms connecting $(1, 1)$ squares to their neighbors. Instead of being allowed to cross at most one blue line, these arms are *required* to cross exactly one blue line. Moreover the inner side of the crossing blue line must be towards the $(1, 1)$ square. This is easy to do with local constraints by requiring a blue line to cross such an arm where the opposite orientations meet (as explained in sub-section 2.5 and illustrated by Figure 5). We can now prove the following lemma:

Lemma 2.6. *In a valid bi-periodic configuration, every finite* $(1, 1)$ *blue square is contained in a larger finite blue square.*

Proof. Consider a finite $(1, 1)$ blue square. By lemma 2.4 it has both an up and a right neighbor. The arms that connect it to these neighbors are each crossed by a blue line, with its inside turned towards the $(1, 1)$ square. The situation is illustrated in Figure 7 (a). The two blue lines that cross the arms must be connected:

- if the vertical one turns before the position of the horizontal one, it will have to cross the arm that is already crossed by the horizontal portion of blue line (b) or turn once more and move a second time though the arm it has already crossed once;
- if the vertical blue line goes further up than the position of the horizontal one, the horizontal one must turn before and cross one of the two arms that have already been crossed (c).

The two blue lines that cross the arms are two sides of the same blue path (d). This blue path contains the $(1, 1)$ square and is therefore larger. ∎

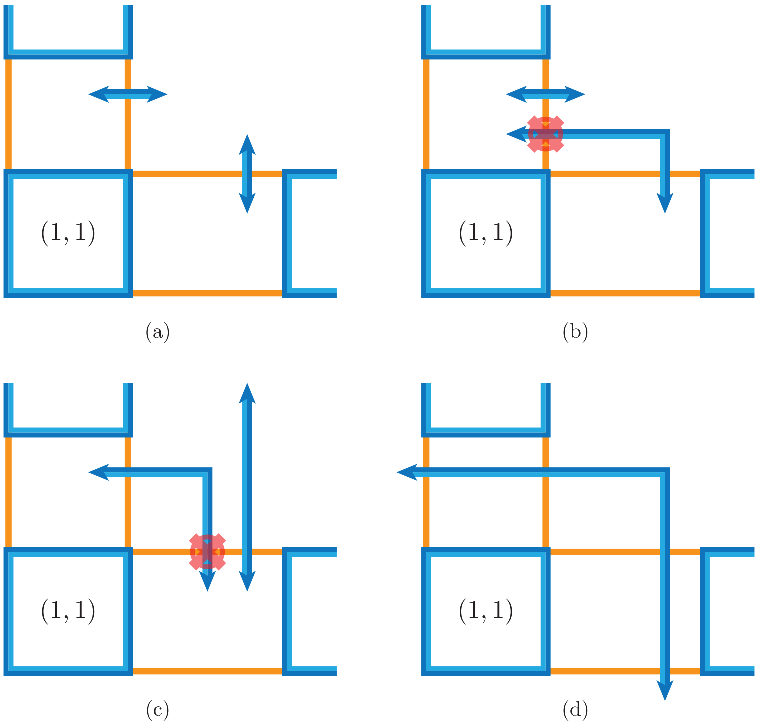


Figure 7: The two blue lines that cross the right and top arms of a $(1, 1)$ square are connected.

2.8. Aperiodicity

All we need to do now is make sure there is a blue square somewhere in any valid configuration. The simplest way to do this locally is to forbid large patterns that have no blue angle. In our specific case, patterns of size 2 are sufficient so we add this last rule to our tiling constraints: every 2×2 pattern must contain a blue angle.

We can now prove the key proposition of the construction:

Proposition 2.7. *There exists no valid periodic configuration.*

Proof. By Proposition 1.7, we need only show that there exists no valid bi-periodic configuration. As a consequence of the last rule any valid configuration has a blue angle. By Lemma 2.2 this angle is part of a finite blue square. Finally, by Lemmas 2.5 and 2.6 for every finite blue square in a valid bi-periodic configuration there exists a larger finite blue square. This means that any such configuration contains arbitrarily large non-intersecting blue squares, which contradicts its periodicity. ■

2.9. Valid Configuration

We still need to show that there exists at least one valid configuration, for the tile set would otherwise be of very limited interest. We will now show that the configuration illustrated by Figure 8 is valid.

This configuration is very regular and has a simple structure. It contains squares of size 3^k for every $k \in \mathbb{N}$. For every k, the squares of size 3^k are arranged regularly, each being at a distance 2.3^k from its neighbors. They are then considered in groups of 3×3 and there is a square of size 3^{k+1} that has the same center as the central

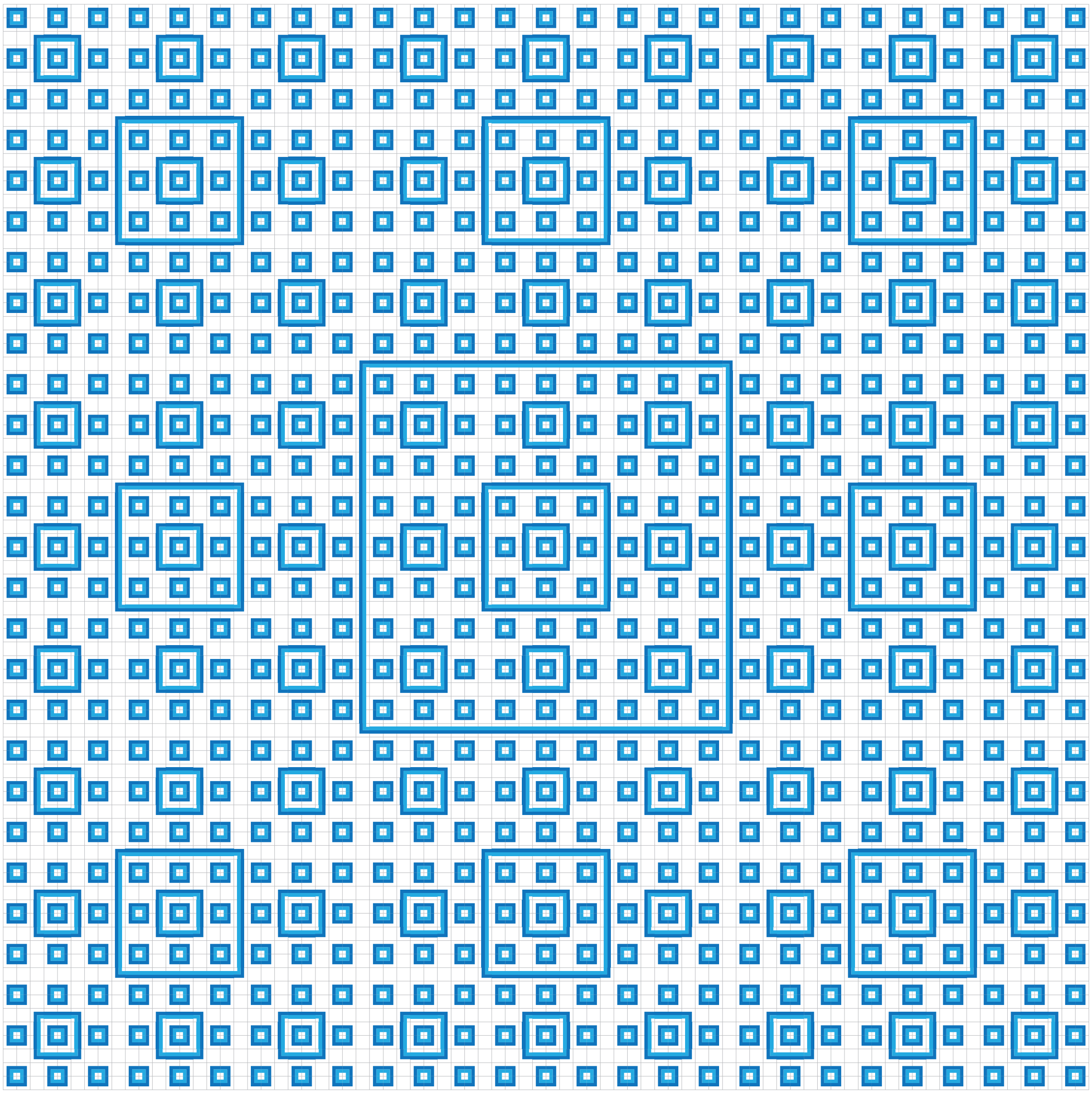


Figure 8: A valid configuration

square of size 3^k of each group. The central square in each 3×3 group has coordinates $(1,1)$ while the others have the matching coordinates (the arms are not represented in the Figure for better clarity).

This configuration satisfies all the rules of the tile set:

- blue paths are all finite blue squares;
- because squares are so regularly arranged, smaller squares that intersect the diagonal of a larger one are perfectly aligned with this diagonal;
- arms connect squares of the same size, and every square has four neighbors;
- the squares of size 3^{k+1} contain the squares of size 3^k that have coordinates $(1,1)$ but none of their neighbors so they cross all the required arms.

Two things must still be justified. The first is that the arrangement of squares that has been described can fill an infinite configuration and more precisely that there is always room for the larger squares without overlapping the previously existing lines. This can be proved by observing that between two consecutive columns (resp. rows) of squares of size 3^k, if we ignore all the larger squares, there is an empty column (resp. row) of cells (cells on which there is no blue line). This property can be proved inductively since it is true for the squares of size 1 and at each step, the squares of size 3^{k+1} occupy two out of three empty columns and rows, leaving exactly one empty column or row between neighboring squares. The construction can therefore be continued indefinitely.

Lastly we must verify that in the valid configuration no arm is crossed by more than one blue line. This fact is closely related to the previous point: the sides of squares of length 3^k lie on rows and columns that were not crossed by smaller squares (the empty rows and columns previously discussed). No smaller square can therefore cross arms connecting two squares of side 3^k. Finally, between two neighboring squares there is exactly one empty column or row on which a larger square could have its side and hence at most one blue line can cross an arm.

3. Conclusion

What we have described is a set of local rules (all rules concern neighboring cells and can be described with 2×2 forbidden patterns) that admits infinite valid configurations but none of these are periodic. Although the local rules remain simple and the number of geometric structures used is quite limited (blue lines, arms and diagonals), the number of symbols necessary to represent them on the cells is very large. Because each cell can contain different combinations of lines and that said lines must be different depending on the information they hold (orientation, coordinates in a group of squares, number of blue lines crossed by an arm, etc.) tens of thousands of different symbols are used.

In order to keep the construction as simple as possible we have only proved that the tile set was aperiodic but it is not sufficient to prove the undecidability of the domino problem as it is. The construction can be strengthened however by forcing blue squares of length one to be regularly arranged as they are in the configuration described in Subsection 2.9. It is then possible to show inductively that the larger squares are also regularly arranged by observing the empty columns and rows. By doing so one can then embed partial space-time diagrams of a Turing Machine in the free space of each blue square as it is done in Robinson's construction (see Figure 9). If the halting state of the Turing machine is not included in the tile set, large valid space-time diagrams of the Turing machine cannot appear in a tiling. The produced tile set can hence tile the plane if and only if the Turing machine does not halt, which proves the undecidability of the domino problem.

The structure of the valid tilings can also be easily altered. Groups could be larger and their inner structure can be more complex. For instance it would be possible to mimic the behavior of recursive geometric constructions such as the space-filling curves of Peano [10] or Hilbert [6] to enforce their structure with a tile set.

References

[1] Robert Berger. *The Undecidability of the Domino Problem.* PhD thesis, Harvard University, 1964.

[2] Robert Berger. *The Undecidability of the Domino Problem.* Number 66 in Memoirs of the American Mathematical Society. The American Mathematical Society, 1966.

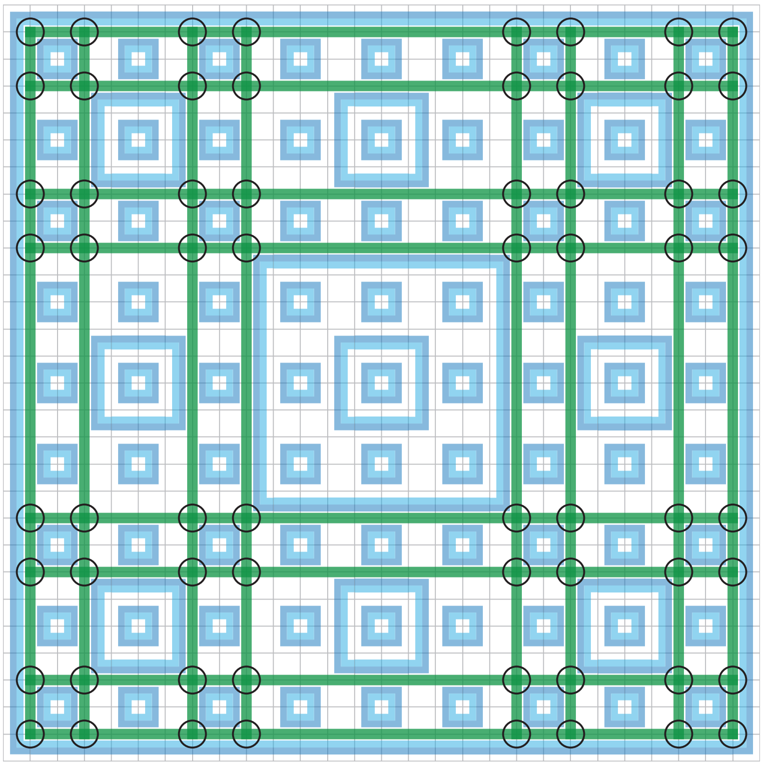


Figure 9: Computation area in a square of size 27. Only the intersections correspond to cells of the space-time diagram of the Turing machine, the horizontal and vertical lines are used to transmit the data. In this example the square can compute 8 × 8 cells of the space-time diagram.

[3] Bruno Durand, Leonid A. Levin, and Alexander Shen. Local rules and global order, or aperiodic tilings. *Mathematical Intelligencer*, 27(1):64–68, 2004.

[4] Bruno Durand, Alexander Shen, and Andrei Romashchenko. Fixed Point and Aperiodic Tilings . Technical Report TR08-030, ECCC, 2008.

[5] Chaim Goodman-Strauss. A Small Aperiodic Set of Planar Tiles. *Europ. J. Combinatorics*, 20:375–384, 1999.

[6] David Hilbert. Ueber die stetige abbildung einer line auf ein flchenstck. *Mathematische Annalen*, 38:459–460, 1891. 10.1007/BF01199431.

[7] A.S. Kahr, Edward F. Moore, and Hao Wang. Entscheidungsproblem reduced to the ∀∃∀ case. *Proceedings of the National Academy of Sciences of the United States of America*, 48(3):365–377, March 1962.

[8] Jarkko Kari. The Tiling Problem Revisited . In *Machines, Computations, and Universality (MCU)*, number 4664 in Lecture Notes in Computer Science, pages 72–79, 2007.

[9] Nicolas Ollinger. Two-by-two substitution systems and the undecidability of the domino problem. In *CiE '08: Proceedings of the 4th conference on Computability in Europe*, pages 476–485, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36:157–160, 1890. 10.1007/BF01199438.

[11] Roger Penrose. Pentaplexity: A class of Non-Periodic Tilings of the Plane. *Eureka*, 39, 1978.

[12] Raphael M. Robinson. Undecidability and Nonperiodicity for Tilings of the Plane. *Inventiones Mathematicae*, 12(3), 1971.

Journées Automates Cellulaires 2010 (Turku), pp. 203-213

DECOMPOSITION COMPLEXITY

ALEXANDER SHEN

LIF Marseille, CNRS & University Aix–Marseille; on leave from IITP RAS, Moscow
E-mail address: Alexander.Shen@lif.univ-mrs.fr
URL: http://www.lif.univ-mrs.fr/~ashen

Abstract. We consider a problem of decomposition of a ternary function into a composition of binary ones from the viewpoint of communication complexity and algorithmic information theory as well as some applications to cellular automata.

1. Introduction

The 13th Hilbert problem asks whether all functions can be represented as compositions of binary functions. This question can be understood in different ways. Initially Hilbert was interested in a specific function (roots of a polynomial as function of its coefficients). Kolmogorov and Arnold (see [5]) gave kind of a positive answer for continuous functions proving that any continuous function of several real arguments can be represented as a composition of continuous unary functions and addition (a binary function). On the other hand, for differentiable functions negative answer was obtained by Vituschkin. Later Kolmogorov interpreted this result in terms of information theory (see [4]): the decomposition is impossible since we have "much more" ternary functions than compositions of binary ones. In a discrete setting this information-theoretic argument was used by Hansen, Lachish and Miltersen ([3]. We consider similar questions in a (slightly) different setting.

Let us start with a simple decomposition problem. An input (say, a binary string) is divided into three parts x, y and z. We want to represent $T(x, y, z)$ (for some function T) as a composition of three binary functions:

$$T(x, y, z) = t(a(x, y), b(y, z)).$$

In other words, we want to compute $T(x, y, z)$ under the following restrictions: node A gets x and y and computes some function $a(x, y)$; node B gets y and z and computes some function $b(y, z)$; finally, the output node T gets $a(x, y)$ and $b(y, z)$ and should compute $T(x, y, z)$.

The two upper channels have limited capacity; the question is how much capacity is needed to make such a decomposition possible. If a- and b-channels are wide enough, we may transmit all the available information, i.e., let $a(x, y) = \langle x, y\rangle$ and

Key words and phrases: decomposition complexity, 13th Hilbert problem, cellular automata.

Author is grateful to V.V. Podolskii, A.E. Romashchenko, and ESCAPE team in general for useful discussions, and to the reviewers for very helpful comments. The paper was supported in part by NAFIT ANR -08-EMER-008-01 and RFBR 09-01-00709-a grants.

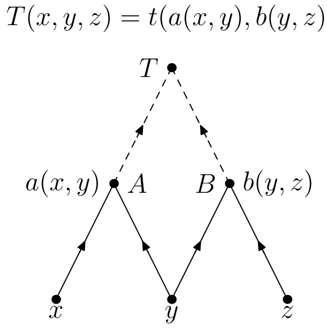


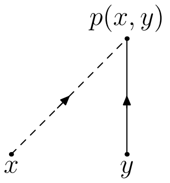
Figure 1: Information transmission for the decomposition.

$b(y,z) = \langle y,z \rangle$. Even better, we can split y in an arbitrary proportion and send one part with x and the other one with z.

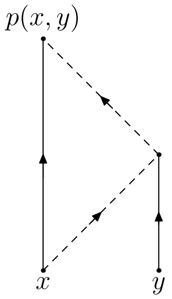
Is it possible to use less capacity? The answer evidently depends on the function T. If, say, $T(x,y,z)$ is `xor` of all bits in x, y and z, one bit for a- and b-values is enough. However, for other functions T it is not the case, as we see below.

In the sequel we prove different lower bounds for the necessary capacity of two upper channels in different settings; then we consider related questions in the framework of multi-source algorithmic information theory [7]).

Before going into details, let us note that the definition of communication complexity can be reformulated in similar terms: one-round communication complexity corresponds to the network



(dotted line indicates channel of limited capacity) while two-rounds communication complexity corresponds to the network



etc. Another related setting that appears in communication complexity theory: three inputs x, y, z are distributed between three participants; one knows x and y, the other knows y and z, the third one knows x and z; all three participants send their messages to the fourth one who should compute $T(x,y,z)$ based on their messages (see [6]).

One can naturally define communication complexity for other networks (we select some channels and count the bits that go through these channels).

2. Decomposition complexity

Now let us give formal definitions. Let $T = T(x, y, z)$ be a function defined on $\mathbb{B}^p \times \mathbb{B}^q \times \mathbb{B}^r$ (here $\mathbb{B}^k$ is the set of k-bit binary strings) whose values belong to some set M. We say that *decomposition complexity* of T does not exceed n if there exist $u + v \leqslant n$ and functions $a\colon \mathbb{B}^p \times \mathbb{B}^q \to \mathbb{B}^u$, $b\colon \mathbb{B}^q \times \mathbb{B}^r \to \mathbb{B}^v$ and $t\colon \mathbb{B}^u \times \mathbb{B}^v \to M$ such that

$$T(x, y, z) = t(a(x, y), b(y, z))$$

for all $x \in \mathbb{B}^p$, $y \in \mathbb{B}^q$, $z \in \mathbb{B}^r$. (As in communication complexity, we take into account the total number of bits transmitted via both restricted links. More detailed analysis could consider u and v separately.)

2.1. General upper and lower bounds

Since the logarithm of the image cardinality is an evident lower bound for decomposition complexity, it is natural to consider *predicates* T (so this lower bound is trivial). This makes our setting different from [3] where all the arguments and values have the same size. However, the same simple counting argument can be used to provide worst-case lower bounds for arbitrary functions.

Theorem 2.1. (Upper bounds) *Complexity of any function does not exceed $n = p + q + r$; complexity of any predicate does not exceed $2^r + r$ as well as $2^p + p$.*

(Lower bound) *If p and r are not too small (at least $\log n + O(1)$), then there exists a predicate with decomposition complexity $n - O(1)$.*

The second statement shows that the upper bounds provided by the first one are rather tight.

Proof. (Upper bounds) For the first bound one can let, say, $a(x, y) = \langle x, y\rangle$ and $b(y, z) = z$. (One can also split y between a and b in an arbitrary proportion.)

For the second bound: for each x, y the predicate $T_{x,y}$

$$z \mapsto T_{x,y}(z) = T(x, y, z)$$

can be encoded by 2^r bits, so we let $a(x, y) = T_{x,y}$ and $b(z) = z$ and get decomposition complexity at most $2^r + r$. The bound $2^p + p$ is obtained in a symmetric way.

(Lower bound) We can use a standard counting argument (in the same way as in [3]; they consider functions, not predicates, but this does not matter much.) Let us count how many possibilities we have for a predicate with decomposition complexity m or less. Choosing such a predicate, we first have to choose numbers u and v such that $u + v \leqslant m$. Without loss of generality we may assume that $u + v = m$ (adding dummy bits). First, let us count (for fixed u and v) all the decompositions where a has u-bit values and b has v-bit values. We have $(2^u)^{2^{p+q}}$ possible a's, $(2^v)^{2^{q+r}}$ possible b's and $2^{2^{u+v}}$ possible t's, i.e.,

$$2^{u2^{p+q}} \cdot 2^{v2^{q+r}} \cdot 2^{2^{u+v}} = 2^{u2^{p+q}+v2^{q+r}+2^{u+v}} \leqslant 2^{(u+v)2^{p+q}+(u+v)2^{q+r}+2^{u+v}}$$

possibilities (for fixed u, v). In total we get at most

$$m2^{m2^{p+q}+m2^{q+r}+2^m}$$

predicates of decomposition complexity m or less (the factor m appears since there are at most m decompositions of m into a sum of positive integers u and v). Therefore, if all 2^{2^n} predicates $\mathbb{B}^p \times \mathbb{B}^q \times \mathbb{B}^r \to \mathbb{B}$ have decomposition complexity at most m, then

$$m2^{m2^{p+q}+m2^{q+r}+2^m} \geqslant 2^{2^n}$$

or

$$\log m + m2^{p+q} + m2^{q+r} + 2^m \geqslant 2^n$$

At least one of the terms in the left-hand side should be $\Omega(2^n)$, therefore either $m \geqslant n - O(1)$ [if $2^m = \Omega(2^n)$], or $\log m \geqslant r - O(1)$ [if $m2^{p+q} \geqslant \Omega(2^n) = \Omega(2^{p+q+r})$], or $\log m \geqslant p - O(1)$ [if $m2^{q+r} \geqslant \Omega(2^n) = \Omega(2^{p+q+r})$]. ■

2.2. Bounds for explicit predicates

As with circuit complexity, an interesting question is to provide a lower bound for an explicit function; it is usually much harder than proving the existence results. The following statement provides a lower bound for a simple function.

Consider the predicate $T\colon \mathbb{B}^k \times \mathbb{B}^{2^{2k}} \times \mathbb{B}^k \to \mathbb{B}$ defined as follows:

$$T(x, y, z) = y(x, z)$$

where $y \in \mathbb{B}^{2^{2k}}$ is treated as a function $\mathbb{B}^k \times \mathbb{B}^k \to \mathbb{B}$.

Theorem 2.2. *The decomposition complexity of T is at least 2^k.*

(Note that this lower bound almost matches the second upper bound of Theorem 2.1, which is $k + 2^k$.)

Proof. Assume that some decomposition of T is given:

$$T(x, y, z) = t(a(x, y), b(y, z)),$$

where $a(x, y)$ and $b(y, z)$ consist of u and v bits respectively. Then every $y : \mathbb{B}^k \times \mathbb{B}^k \to \mathbb{B}$ determines two functions $a_y\colon \mathbb{B}^k \to \mathbb{B}^u$ and $b_y\colon \mathbb{B}^k \to \mathbb{B}^v$ obtained from a and b by fixing y. Knowing these two functions (and t) one should be able to reconstruct $T(x, y, z)$ for all x and z, since

$$T(x, y, z) = t(a_y(x), b_y(z)),$$

i.e., to reconstruct y. Therefore, the number of possible pairs $\langle a_y, b_y \rangle$, which is at most

$$2^{u2^k} \cdot 2^{v2^k},$$

is at least the number of all y's, i.e. $2^{2^{2k}}$. So we get

$$(u + v)2^k \geqslant 2^{2k},$$

or $u + v \geqslant 2^k$, therefore the decomposition complexity of T is at least 2^k. ■

Remarks.

1. In this way we get a lower bound $\Omega(\sqrt{n})$ (where n is the total input size) for the case when x and z are of size about $\frac{1}{2}\log n$. In this case this lower bound matches the upper bound of Theorem 2.1, as we have noted.

2. Here is another example where upper and lower bounds match. If the predicate $t(x, y, z)$ is defined as $x = z$, we need to transmit x and z completely (see [6] or use the pigeon-hole principle). So there is a trivial (and tight) linear lower bound if we let x and z be long (of $\Theta(n)$) size.

3. It would be interesting to get a linear bound for an explicit function in an intermediate case when x and z are short compared to y (preferable even of logarithmic size) but not as short as in Theorem 2.2 (so a non-constructive lower bound applies). Such a lower bound would mean that $a(x, y)$ or $b(y, z)$ has to retain a significant part of information in y. Intuitive explanation for this necessity could be: "since we do not know z when computing $a(x, y)$, we do not know which part of y-information is relevant and need to retain a significant fraction of y". Note that for the function T defined above this is not the case: not knowing z, we still know x so only one row (xth row) in the matrix y is relevant.

The natural candidate is the function $T' \colon \mathbb{B}^k \times \mathbb{B}^{2^k} \times \mathbb{B}^k \to \mathbb{B}$ defined by $T'(x, y, z) = y(x \oplus z)$. Here y is considered as a vector $\mathbb{B}^k \to \mathbb{B}$, not matrix, and $x \oplus z$ denotes bitwise XOR of two k-bit strings x and z. The size of x and z is about $\log n$ (where n is the total input size), and for these input sizes the worst-case lower bound is indeed linear. One could think that this lower bound could be obtained for T': "when computing $a(x, y)$ we do not know z, and $x \oplus z$ could be any bit string of length k, so all the information in y is relevant". However, this intuition is false, and there exists a sublinear upper bound $O(n^{0.92})$, see [1] or [6], p. 95.[1] (This upper bound should be compared to the $\Omega(\sqrt{n})$ lower bound obtained by reduction to T: in the special case when the left half of x and the right half of z contain only zeros, we get T out of T'.)

Question: what happens if we replace $x \oplus z$ by $x + z \bmod 2^k$ in the definition of T'? It seems that the upper bound argument does not work any more.

[1] This upper bound is obtained as follows. Let us consider y as a Boolean function of k Boolean variables; $y \colon (u_1, \ldots, u_k) \mapsto y(u_1, \ldots, u_k)$. Such a Boolean function can be represented as a multilinear polynomial of degree k over the 2-element field $\mathbb{F}_2$. This polynomial $y(u_1, \ldots, u_k)$ has 2^k bit coefficients and is known when $a(x, y)$ or $b(y, z)$ are computed. Let us separate terms of "high" and "low" degree in this polynomial:

$$y(u_1, \ldots) = y_{\text{low}}(u_1, \ldots) + y_{\text{high}}(u_1, \ldots),$$

taking $\frac{2}{3}k$ as the threshold between "low" and "high". The polynomial y_{high} is included in a (or b) as is, just by listing all its coefficients. (We have about $2^{H(\frac{2}{3})k} \approx n^{0.92}$ of them, where H is Shannon entropy function.) For y_{low} we use the following trick. Consider $y(X_1 \oplus Z_1, \ldots, X_k \oplus Z_k)$ as a polynomial $\tilde{y}$ of $2k$ variables $X_1, \ldots, X_k, Z_1, \ldots, Z_k \in \mathbb{F}_2$. Its degree is at most $\frac{2}{3}k$, and each monomial includes at most $\frac{2}{3}k$ variables. So we can split $\tilde{y}$ again:

$$\tilde{y}(X_1, \ldots, Z_1, \ldots) = \tilde{y}_{x\text{-low}}(X_1, \ldots, Z_1, \ldots) + \tilde{y}_{z\text{-low}}(X_1, \ldots, Z_1, \ldots);$$

here the first term has small X-degree (Z-variables are treated as constants), and the second term has small Z-degree. Here "small" means "at most $\frac{1}{3}k$". All this could be done in both nodes (while computing a and b), since y is known there; X_i and Z_i are just variables. Now we include in $a(x, y)$ the coefficients of the polynomial $(Z_1, \ldots, Z_k) \mapsto \tilde{y}_{z\text{-low}}(x_1, \ldots, x_k, Z_1, \ldots, Z_k)$, and do the symmetric thing for $b(y, z)$. Both polynomial have degree at most $\frac{1}{3}k$, so we again need only $O(n^{0.92})$ bits to specify them.

3. Probabilistic decomposition

As in communication complexity theory, we may consider also probabilistic and distributional versions of decomposition complexity. In the probabilistic version we consider random variables instead of binary functions a, b, t (with shared random bits or independent random bits). In the distributional version we look for a decomposition that is Hamming-close to a given function.

It turns out that the lower bounds mentioned above are robust in that sense and remain valid for distributional (and therefore probabilistic) decomposition complexity almost unchanged.

Let ε be a positive number less than 1/2. We are interested in a minimum decomposition complexity of a function that ε-approximates a given one (coincides with it with probability at least $1-\varepsilon$ with respect to uniform distribution on inputs). For $\varepsilon \geqslant \frac{1}{2}$ this question is trivial (either 0 or 1 constant provide the required approximation). So we assume that some $\varepsilon < \frac{1}{2}$ is fixed (the $O()$-constants in the statements will depend on it).

A standard argument shows that lower bounds established for distributional decomposition complexity remain true for probabilistic complexity (where a, b, t use random bits and for every input x, y, z the random variable $t(a(x, y), b(y, z))$ should coincide with a given function with probability at least $1-\varepsilon$). So we may consider only the distributional complexity.

Theorem 3.1. **(1)** *Let $n = p + q + r$ and $p, r \geqslant \log n + O(1)$. Then there exists a predicate $T\colon \mathbb{B}^p \times \mathbb{B}^q \times \mathbb{B}^r \to \mathbb{B}$ such that decomposition complexity of any its ε-approximation is at least $n - O(1)$.*

(2) *For the predicate T used in Theorem 2.2 we get the lower bound $\Omega(2^k)$ (in the same setting).*

Proof. **1**. Assume this is not the case. We repeat the same counting argument as in Theorem 2.1. Now we have to count not only the predicates that have decomposition complexity at most m, but also their ε-approximations. The volume of an ε-ball in $\mathbb{B}^{2^n}$ is about $2^{H(\varepsilon)2^n}$, so the number of the centers of the balls that cover the entire space is at least $2^{(1-H(\varepsilon))2^n}$. So after taking the logarithms we get a constant factor $(1 - H(\varepsilon))$, and the lower bound for m remains $n - O(1)$.

2. If the computation is correct for $1-\varepsilon$ fraction of all triples (x, y, z), then there exist $\varepsilon' < \frac{1}{2}$ and $\varepsilon'' > 0$ such that for at least ε''-fraction of all y the computation is correct with probability at least $1 - \varepsilon'$ (with respect to uniform distribution on x and z). This means that ε'-balls around functions $(x, z) \mapsto t(a_y(x), b_y(z))$ cover at least ε''-fraction of all functions y. (See the proof of Theorem 2.2.) Again this gives us a constant factor before 2^{2k}, but here we do not take the logarithm second time, so we get $u + v \geqslant \Omega(2^k)$, not $2^k - O(1)$. ∎

4. Applications to cellular automata

An (one-dimensional) cellular automata is a linear array of cells. Each of the cells can be in some state from a finite set S of states (the same for all cells). At each step all the cells update their state; new state of a cell is some fixed function of its old state and the states of its two neighbors. All the updates are made synchronously.

Using a cellular automaton to compute a predicate, we assume that there are two special states 0 and 1 and a neutral state that is stable (if a cell and both its neighbors are in the neutral state, then the cell remains neutral). To compute $P(x)$ for a n-bit string x, we assemble n cells and put them into states that correspond to x; the rest of the (biinfinite) cell array is in a neutral state.

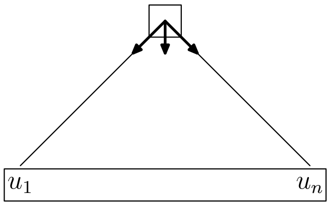
Then we start the computation; the answer should appear in some predefined cell (see below about the choice of this cell).

There is a natural non-uniform version of cellular automata: we assume that in each vertex of the time-space diagram an arbitrary ternary transition function (different for different vertices) is used. Then the only restriction is caused by the limited capacity of links: we require that inputs/outputs of all functions (in all vertices) belong to some fixed set S.

In this non-uniform setting a predicate P on binary strings is considered as a family of Boolean functions P_n (where P_n is a restriction of P onto n-bit strings) and for each P_n we measure the minimal size of a set S needed to compute P_n in a non-uniform way described above. If this size is an unbounded function of n, we conclude that predicate P is not computable by a cellular automaton. (In classical complexity theory we use the same approach when we try to prove that some predicate is not in P since it needs superpolynomial circuits in a non-uniform setting.)

As usual, getting lower bounds for nonuniform models is difficult, but it turns out that decomposition complexity can be used if the cellular automaton is required to produce the answer as soon as possible.

Since each cell gets information only from itself and its two neighbors, the first occasion to use all n input bits happens around time $n/2$ in the middle of the string:



Now we assume that the output of a cellular automaton is produced at this place (both in uniform and non-uniform model). (This is a very strong version of real-time computation by cellular automata; we could call it “as soon as possible”-computation.)

The next theorem observes that non-uniformly computable family of predicates is transformed into a function with small decomposition complexity if we split the input string in three parts.

Theorem 4.1. *Let* $T_k\colon \mathbb{B}^{k+f(k)+k} = \mathbb{B}^k \times \mathbb{B}^{f(k)} \times \mathbb{B}^k \to \mathbb{B}$ *be a family of predicates that is non-uniformly computable in this sense. Then the decomposition complexity of* T_k *is* $O(k)$*, and the constant in* O*-notation is the logarithm of the number of states.*

Proof. Consider Figure 2 where the (nonuniform) computation is presented (we use bigger units for time direction to make the picture more clear).

Let us look at the contents of the line of length $2k$ located k steps before the end of the computation. The left half is $a(x, y)$, the right half is $b(y, z)$ and the function t is computed by the upper part of the circuit. It is easy to see that $a(x, y)$ indeed depends only on x and y since information about z has not arrived yet; for

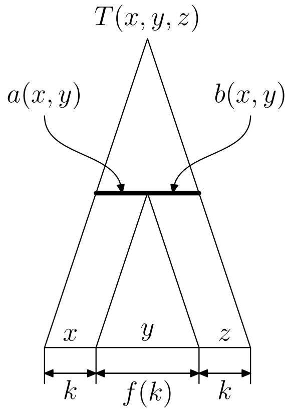


Figure 2: Automaton run and its decomposition.

the same reason $b(y,z)$ depends only on y and z. The bit size of $a(x,y)$ and $b(y,z)$ is $k\log\#S$. ∎

Corollary 4.2. *The predicate T from Theorem 2.2 cannot be computed in this model.*

This predicate splits a string of length $k+2^{2k}+k$ into three pieces x,y,z of length k, 2^{2k} and k respectively, and then computes $y(x,z)$. Note that this can be done by a cellular automaton in linear time. Indeed, we combine the string x and z into a $2k$-binary string; then we move this string across the middle part of input subtracting one at each step and waiting until our counter decreases to zero; then we know where the output bit should be read. So we get the following result:

Theorem 4.3. *There exists a linear-time computable predicate that is not computable "as soon as possible" even in a non-uniform model.*

Remark. This result and the intuition behind the proof are not new (see the paper of V. Terrier [8]; see also [2]). However, the explicit use of decomposition complexity helps to formalize the intuition behind the proof. It also allows us to show (in a similar way) that this predicate cannot be computed not only "as soon as possible", but even after $o(\sqrt{n})$ steps after this moment (which seems to be an improvement).

Another improvement that we get for free is that we cannot even ε-approximate this predicate in the "as soon as possible" model.

Question: There could be other ways to get lower bounds for non-uniform automata (=triangle circuits). Of course, there is a counting lower bound, but this does not give any explicit function. Are there some other tools?

5. Algorithmic Information Theory

Now we can consider the Kolmogorov complexity version of the same decomposition problem. Let us start with some informal comments. Assume that we have four binary strings x,y,z,t such that $\mathrm{K}(t|x,y,z)$ is small (we write $\mathrm{K}(t|x,y,z)\approx 0$, not specifying exactly how small should it be). Here $\mathrm{K}(\alpha|\beta)$ stands for conditional complexity of α when β is known, i.e., for the minimal length of a program that transforms β to α. (Hence our requirement says that there is a short program that produces t given x,y,z.)

We are looking for strings a and b such that $\mathrm{K}(a|x,y)\approx 0$, $\mathrm{K}(b|y,z)\approx 0$, and $\mathrm{K}(t|a,b)\approx 0$. Such a and b always exist, since we may let $a=\langle x,y\rangle$ and $b=\langle y,z\rangle$ (again, y can also be split between a and b). However, the situation changes if we restrict the complexities of a and b (or their lengths, this does not matter, since each string can be replaced by its shortest description). As we shall see, sometimes we need a and b of total complexity close to $\mathrm{K}(x)+\mathrm{K}(y)+\mathrm{K}(z)$ even if t has much smaller complexity. (Note that now we cannot restrict ourselves to one-bit strings t for evident reasons.)

To be specific, let us agree that all the strings x,y,z,t have the same length n; we look for strings a and b of length m, and “small” conditional complexity means that complexity is less than some c.

Theorem 5.1. *If* $3c<n-O(1)$ *and* $2m+c<3n-O(1)$*, there exist strings* x,y,z,t *of length* n *such that* $K(t|x,y,z)=O(\log n)$*, but there are no strings* a,b *of length* m *such that*

$$K(a|x,y)<c,\qquad K(b|y,z)<c,\qquad K(t|a,b)<c.$$

For example, this is true if $c=O(\log n)$ and m is $1.5n-O(\log n)$ (note that for $m=1.5n$ we can split y into two halves and combine the first half with x, and the second half with y).

Proof. Consider the following algorithm. Given n, we generate (in parallel for all $x,y\in\mathbb{B}^n$) the lists of those m-bit strings who have conditional complexity (with respect to x and y) less than c (one list for each pair x,y). Also we generate (in parallel for all strings a and b of length m) the lists of those strings t who have complexity less than c given a and b (one list for each pair a,b). At every step of enumeration we imagine that these lists are final and construct a quadruple x,y,z,t that satisfies the statement of the theorem. It is done as follows: we take a “fresh” triple x,y,z (that was not used on the previous steps of the construction), take all strings a that are in the list for x,y, take all strings b that are in the list for y,z, and take all strings t that are in the lists for those as and bs. Then we choose some t that does not appear in all these lists.

Such a t exists since we have at most 2^c strings a (for given x and y), and at most 2^c strings b (for given y and z). For every of 2^{2c} pairs (a,b) there are at most 2^c strings t, so in total at most 2^{3c} values of t are unsuitable, and we can choose a suitable one.

We also need to ensure that there are enough “fresh” pairs for all the steps of the construction. The new elements in the first series of lists may appear at most $2^n\times 2^n\times 2^c$ times (we have at most $2^n\times 2^n$ pairs (x,y) and at most 2^c values of a for each pair). Then we have $2^m\times 2^m\times 2^c$ events for the second series of lists. On the other hand, we have 2^{3n} triples (x,y,z), so we need the inequality

$$2^{2n+c}+2^{2m+c}<2^{3n},$$

which is guaranteed by our assumptions.

To run this process, it is enough to know n, so for every x,y,z,t generated by this algorithm we have $K(t|x,y,z)=O(\log n)$. (For given x,y,z only one t may appear since we take a fresh triple each time.) ■

This result can be improved:

Theorem 5.2. *Assume that $3c < n - O(1)$ and $m \leqslant 1.5n - O(\log n)$. We can effectively construct for every n a total function $T : \mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{B}^n$ such that for random* (*= incompressible*) *triple x, y, z and $t = T(x, y, z)$ the strings a and b of length m that provide a decomposition* (*as defined above*) *do not exist.*

The improvement is two-fold: first, we have a total function T (instead of a partial one provided by the previous construction); second, we claim that all random triples have the required property (instead of mere existence of such a triple).

Proof. Let us first deal with the first improvement. Consider multi-valued functions $A, B \colon \mathbb{B}^n \times \mathbb{B}^n \to \mathcal{P}(\mathbb{B}^m)$ that map every pair of n-bit strings into a 2^c-element set of m-bit strings. Consider also multi-valued function $F \colon \mathbb{B}^m \times \mathbb{B}^m \to \mathcal{P}(\mathbb{B}^n)$ whose values are 2^c-element sets of n-bit strings. We say that A, B, F *cover* a total function $T : \mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{B}^n$ if for every $x, y, z \in \mathbb{B}^n$ there exist strings $a, b \in \mathbb{B}^m$ such that $a \in A(x, y)$, $b \in B(y, z)$, and $T(x, y, z) \in F(a, b)$.

Let us prove first the following combinatorial statement: *there exists a function T that is not covered by any triple of functions A, B, F.* This can be shown by a counting argument similar to the proof of Theorem 2.1. Indeed, let us compute the probability of the event "random function T is covered by some fixed A, B, F". This event is the intersection of independent events (for each triple x, y, z). For given x, y, z there are 2^c possible as, 2^c possible bs, and 2^c possible elements in $F(a, b)$ for each a and b, i.e., 2^{3c} possibilities altogether. Since $3c < n - O(1)$, each of the independent events has probability less than $\frac{1}{2}$, and their intersection has probability less than $2^{-2^{3n}}$.

This probability then should be multiplied by the number of triples A, B, F. For A and B we have at most $(2^m)^{2^n \times 2^n \times 2^c}$ possibilities, for F we have at most $(2^n)^{2^m \times 2^m \times 2^c}$ possibilities. So the existence of a function T not covered by any triple is guaranteed if

$$2^{m2^{2n+c}} \times 2^{m2^{2n+c}} \times 2^{n2^{2m+c}} \times 2^{-2^{3n}} < 1,$$

i.e.,

$$m2^{2n+c} + m2^{2n+c} + n2^{2m+c} < 2^{3n},$$

and this inequality follows from the assumptions.

The property "T can be covered by some triple A, B, F" can be computably tested by an exhaustive search over all triples A, B, F. So we can (for every n) computably find the first (in some order) function T that does not have this property. For these T there are some x, y, z that do not allow decomposition. Indeed, we can choose A so that $A(x, y)$ contains all strings a of length m such that $K(a|x, y) < c$, etc.

However, we promised more: we need to show not only the existence of x, y, z but that all incompressible triples (this means that $K(x, y, z) \geqslant 3n - O(1)$) have the required property. This is done in two steps. First, we show than (for some F that computably depends on n) most triples do not allow decomposition. Then we note that one can enumerate triples that allow decomposition, so they can be encoded by their ordinal number in the enumeration and therefore are compressible.

To make this plan work, we need to consider other property of function T. Now we say that T is covered by A, B, F if at least $2^{-O(1)}$-fraction of all triples (x, y, z) admit a and b. The probability of this event should now be estimated by Chernoff inequality (we guarantee first that the probability of each individual event is, say, twice smaller than the threshold), and we get a bound of the same type, with $\Omega(2^{3n})$ instead of 2^{3n}, which is enough. ∎

In fact, this argument provides a decomposition complexity bound similar to Theorem 2.1, but now the functions a, b and t are multi-valued and we can choose any of their values to obtain $t(x, y, z)$.

Remarks and questions

1. Similar results can be obtained for more binary operations in the decomposition. Imagine that we have some strings x, y, z, t of length n such that $K(t|x, y, z)$ is small and want to construct some "intermediate" strings $u_1, \ldots, u_s$ such that in the sequence

$$x, y, z, u_1, u_2, \ldots, u_s, t$$

every string, starting from u_1, is conditionally simple with respect to some *pair* of its predecessors. We can use our technique to show that this is not possible if all u_i have length close to n and the number s is not large.

2. As before, it would be nice to get lower bounds for some explicit function $T(x, y, z)$ (even a non-optimal lower bound, like in Theorem 2.2) for the algorithmic information theory version of decomposition problem.

3. Many results of multi-source algorithmic information theory have some counterparts in classical information theory. Can we find some statement that corresponds to the lower bound for decomposition complexity?

4. Is it possible to use the techniques of [3] to get some bounds for explicit functions in algorithmic information theory setting?

References

[1] L. Babai, P. Kimmel, Satyanarayana V. Lokam: Simultaneous messages vs communication, *12th Annual Symposium on Theoretical Aspects of Computer Science* (STACS'95), Munich, Lecture Notes in Computer Science, v. 900, 1995, Springer-Verlag, p. 361–372.

[2] C. Choffrut and K. Culik II, On Real-Time Cellular Automata and Trellis Automata, *Acta Informatica*, **21**, 393–407 (1984).

[3] Hansen, K.A., Lachish, O., Miltersen P.B., Hilbert's thirteenth problem and circuit complexity. ISAAC 2009, p. 153–162.

[4] Колмогоров А.Н., Тихомиров В.М., ε-энтропия и ε-ёмкость множеств в функциональных пространствах. *Успехи математических наук*, **14** (2), p. 3–86.

[5] Колмогоров А. Н., О представлении непрерывных функций нескольких переменных в виде суперпозиций непрерывных функций одного переменного и сложения. *Доклады Академии наук СССР*, **114**(5), 953–956 (1957)

[6] Eyal Kushilevitz, Noam Nisan, *Communication complexity*, Cambridge University Press, 1997.

[7] Shen A., Multisource information theory, *Theory and Applications of Models of Computation*, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, 3959 (2006), p. 327-338.

[8] Véronique Terrier, Language not recognizable in real time by one-way cellular automata. *Theoretical Computer Science*, **156**(1–2), 281–287 (1996).

Journées Automates Cellulaires 2010 (Turku), pp. 214-225

REAL-TIME SORTING OF BINARY NUMBERS ON ONE-DIMENSIONAL CA

THOMAS WORSCH [1] AND HIDENOSUKE NISHIO [2]

[1] KIT (Karlsruhe Institute of Technology)
E-mail address: worsch@kit.edu

[2] Kyoto University
E-mail address: yra05762@nifty.com

Abstract. A new fast (real time) sorter of binary numbers by one-dimensional cellular automata is proposed. It sorts a list of n numbers represented by k-bits each in exactly nk steps. This is only one step more than a lower bound.

1. Introduction

Sorting is one of the most fundamental subjects of computer science and many sorting algorithms including sorting arrays and networks can for example be found in volume 3 of Knuth's TAOCP [2]. However, for cellular automata there are only a few papers on this important topic. It should be pointed out that the algorithm described by one of the authors in an earlier paper [3] has running $3nk$ (despite the title starting with the words "real time").

We are not aware of any speedup techniques which would allow to turn this CA or any other solving the problem into one running in real-time, i. e. exactly the number of steps which is the length of the input. In the present paper we propose a sorting algorithm of binary numbers and its implementation on one-dimensional CA with nearest neighbors, which sorts n numbers of k bits each in exactly nk steps.

Sequential comparison based sorting algorithms need time $\Omega(n \log n)$ where n is the number of elements to be sorted and it is assumed that each comparison can be done in constant time independent of the size of the elements. The latter assumption is also usually made for parallel sorting algorithms. On linear arrays odd-even transposition sort needs exactly n steps. But there the additional assumption is made that from the beginning each processor knows the parity of its own address (assuming those are e. g. 1 to n). Of course on parallel models from the second machine class [1] like PRAM there are algorithms running in poly-logarithmic (or even logarithmic) time. But for these models one has to assume non-local communication in constant time when embedded into Euclidean space [4].

The rest of the paper is organized as follows. In Section 2 we precisely state the problem and the results obtained in this paper. In Section 3 we give a short proof for the lower bound of the sorting problem. The main aspects of our algorithm are presented in Section 4. The first version does not achieve a running time which matches the lower bound. For that two modifications are needed which are described in Section 5.

2. Statement of problem and results

We are considering one-dimensional CA with von Neumann neighborhood of radius 1 and assume that the reader is familiar with these concepts. Since we will not define our CA on such a low level there is no need to introduce any related formalism.

We also assume that the reader is familiar with the firing squad synchronization problem [5]. If a block of k cells needs to be synchronized and there are generals at both ends, then synchronization can be achieved in exactly k steps.

The inputs for our CA are provided as finite words with all surrounding cells in a quiescent state. Those cells will never be used during computations.

The inputs which have to be processed by our CA are n numbers of equal length k, with the most and least significant bits marked as such.

Problem 2.1. The input alphabet is $A = \{0, 1, \langle 0, \langle 1, 0|, 1|\}$.

Each input w that has to be processed properly is of the form

$$w = w_1 \cdots w_n = \langle x_1 y_1 z_1| \cdots \langle x_n y_n z_n|$$

for some $n \geq 1$ and $k \geq 2$ where all $x_i, z_i \in \{0, 1\}$ and all $y_i \in \{0, 1\}^{k-2}$. Each w_i is the binary representation of a non-negative integer (also denoted w_i) with most significant bit x_i and least significant bit z_i.

For every such input after a finite number of steps a stable configuration of the form $w_{\sigma(1)} \cdots w_{\sigma(n)}$ has to be reached where σ is a permutation of the numbers $1, \ldots, n$ such that $w_{\sigma(i)} \leq w_{\sigma(i+1)}$ holds for all $1 \leq i < n$.

We note that it would have been sufficient to mark either the most or the least significant bits, because the other end of each number can then always be identified by looking at neighbor cells.

The above problem statement also excludes the case of 1-bit inputs. For those the "traffic rule" 184 can be easily extended to do sorting, taking into account quiescent neighbors. The resulting CA works as follows: A cell in state 1 (0) becomes 0 (1) if its right (left) neighbor is 0, otherwise it keeps its current state.

In the following we always call a sequence of cells which initially stores one input number w_i the *block* i (or simply a block).

It is clear that it can be necessary to move a number from block 1 to block n. This immediately gives a lower bound of $(n-1)k$ steps for the sorting time. We shall see, that one can do slightly better:

Theorem 2.2. *Every CA solving Problem 2.1 needs at least time* $nk - 1$.

Until now the fastest sorting algorithms known needed time cnk for some constant $c > 1$ with no obvious possibility to speed up the computation to run in nk steps. The main contribution of the present paper therefore is the following:

Theorem 2.3. *There is a CA (which does not depend on* n *or* k*) with von Neumann neighborhood of radius* 1 *solving the Sorting Problem 2.1 in exactly* nk *steps.*

3. Lower bound on sorting time

First consider the input $w = w_1 w_2 \cdots w_n$ where $w_1 = \langle 01^{k-2}1|$ and $w_2 = \cdots = w_n = \langle 10^{k-2}0|$. Clearly this input sequence is already sorted and the rightmost bit of the output is a 0.

If on the other hand we flip the leftmost bit of the first block only and consider the input

$$w_1' \cdot w_2 \cdots w_n = \langle 11^{k-2}1| \cdot \langle 10^{k-2}0| \cdots \langle 10^{k-2}0|$$

then the correct sorted output is

$$w_2 \cdots w_n \cdot w_1' = \langle 10^{k-2}0| \cdots \langle 10^{k-2}0| \cdot \langle 11^{k-2}1|$$

That is, by changing only the leftmost bit of the input the rightmost bit of the output must change. Hence no CA correctly solving the sorting problem can be faster then the distance between leftmost and rightmost bit which is $nk - 1$.

This proves Theorem 2.2.

4. The base sorting algorithm

The goal of this section is to prove a weakened version of Theorem 2.3.

Lemma 4.1. *There is a CA (which does not depend on n or k) with the von Neumann neighborhood of radius* 1 *solving the sorting Problem 2.1 in exactly $k + nk$ steps.*

Before going into details and explaining some aspects of the CA on the cell level, we describe the main idea on the level of numbers. In particular we will employ a well-known simple algorithm for parallel sorting.

4.1. Odd-even transposition sort

Throughout this section, one can assume that N is an even number; this is the case needed for the CA below.

Assume that N numbers $a_1, a_2, \ldots, a_N$ are given, arranged in an array of processors. In addition each processor (or each number) has a direction (indicated by arrows below). In each step each pair of adjacent processors whose arrows point to each other exchange their numbers. Both processors compare the two numbers; the left one keeps the smaller number, the right one the larger number, and both change their direction to the other neighbor. Thus one step can for example look like this:

t	$\overleftarrow{a_1}$	$\overrightarrow{a_2}$	$\overleftarrow{a_3}$	$\overrightarrow{a_4}$	$\overleftarrow{a_5}$	$\overrightarrow{a_6}$	......	$\overleftarrow{a_{N-1}}$	$\overrightarrow{a_N}$
$t+1$	$\overrightarrow{b_1}$	$\overleftarrow{b_2}$	$\overrightarrow{b_3}$	$\overleftarrow{b_4}$	$\overrightarrow{b_5}$	$\overleftarrow{b_6}$	......	$\overrightarrow{b_{N-1}}$	$\overleftarrow{b_N}$

$$\text{where } b_i = \begin{cases} \min(a_i, a_{i+1}) & \text{if } a_i \text{ points to the right} \\ \max(a_{i-1}, a_i) & \text{if } a_i \text{ points to the left} \end{cases}.$$

A missing neighboring number to the left is treated as if it were $-\infty$ and a missing neighboring number to the right is treated as if it were ∞.

It is known that odd-even transposition sort always produces the correct result after exactly N steps (see e.g.[2]).

4.2. Outline of the base algorithm

The algorithm which will be the basis for the improved construction in Section 5 will simply work as follows. Given an input of the form

w_1	w_2	w_3	w_4	......	w_{n-1}	w_n

first each number is copied and the two copies get opposite directions assigned. This will take k steps in the CA. Instead of storing two copies side by side they are stored in parallel:

$\overleftarrow{w_1}$	$\overleftarrow{w_2}$	$\overleftarrow{w_3}$	$\overleftarrow{w_4}$	......	$\overleftarrow{w_{n-1}}$	$\overleftarrow{w_n}$
$\overrightarrow{w_1}$	$\overrightarrow{w_2}$	$\overrightarrow{w_3}$	$\overrightarrow{w_4}$	......	$\overrightarrow{w_{n-1}}$	$\overrightarrow{w_n}$

These are now treated as $N = 2n$ numbers that are sorted using odd-even transposition sort. In the end one would get

$\overleftarrow{w_{\sigma(1)}}$	$\overleftarrow{w_{\sigma(2)}}$	$\overleftarrow{w_{\sigma(3)}}$	$\overleftarrow{w_{\sigma(4)}}$	......	$\overleftarrow{w_{\sigma(n-1)}}$	$\overleftarrow{w_{\sigma(n)}}$
$\overrightarrow{w_{\sigma(1)}}$	$\overrightarrow{w_{\sigma(2)}}$	$\overrightarrow{w_{\sigma(3)}}$	$\overrightarrow{w_{\sigma(4)}}$	......	$\overrightarrow{w_{\sigma(n-1)}}$	$\overrightarrow{w_{\sigma(n)}}$

where σ again denotes the "sorting permutation" as in Problem 2.1.

During the last sorting step the lower parts and the arrows are deleted, and the required output is obtained.

It will become clear in the next subsection why it is actually useful to first copy each number and then seemingly spend twice as much time for the $N = 2n$ sorting steps. It will be shown that each such step can be implemented in the CA in $k/2$ steps. Hence the total running time of the CA for this base version of the algorithm will be $k + nk$ steps.

4.3. Outline of the CA for the base algorithm

In this section we will describe how the base sorting algorithm can be implemented on a CA. It will need $n+1$ phases each of which needs exactly k steps. First comes a setup phase followed by phases $1, \ldots, n$. In order to avoid more complicated descriptions, throughout the rest of the paper we assume that k is even.

If k is odd the middle cell of a block plays the role of the two middle cells one would have for the (even) case $k+1$. In that case synchronization of a block using generals at both ends needs at least time $2\frac{k+1}{2} - 2 = k - 1$ and hence is possible in time k (as is the case for even k).

Algorithm 4.2 (Setup phase)**.**

(1) During the setup phase the following tasks are carried out in each block:

- By sending a signal from the left and the right end of the block the two middle cells are found. In each resulting *sub-block* the leftmost and the rightmost cell are marked as such. They are called L and R respectively.
- Using an additional register the mirror image of the input number is computed. Below we call the register holding the original value *left* and the registers with the mirrored value *right*.
  The numbers in the *left* registers will play the role of the $\overleftarrow{w_i}$ and the numbers in the *right* registers the role of the $\overrightarrow{w_i}$ used in the previous subsection.
- Using synchronization, the preliminary phase is stopped after k steps.

(2) The leftmost and the rightmost cell of the whole input are set up as generals. Starting with the first step of phase 1 an algorithm is started to synchronize all nk cells after nk steps.

A concrete example is shown in Figure 1 for two 6-bit numbers. The borders between sub-blocks are shown as double vertical lines. It can be noted that we also use markers for the most and least significant bits of the mirrored numbers.

	L		R	L		R	L		R	L		R
left	$\langle 1$	0	0	0	0	$0\vert$	$\langle 0$	1	1	1	1	$1\vert$
right	$\vert 0$	0	0	0	0	$1\rangle$	$\vert 1$	1	1	1	1	$0\rangle$

Figure 1: An example configuration for two 6-bit numbers after the setup phase.

In addition to the registers *left* and *right* the L- and R-cells of each sub-block will make use of a register *comp* which will hold a (preliminary) comparison result (see also Figure 2). Each *comp* register can hold one of the values $=$, $<$ or $>$. Their use is described in Algorithm 4.4 below.

The core idea is the following:

C1. Bits are shifted in the *left* and *right* registers in the corresponding directions.

C2. Whenever the most significant bits of two numbers arrive in a pair of adjacent $R\|L$-cells, the numbers are compared sequentially bit by bit. The smaller number will be directed to move to the left and the larger one will be directed to move to the right.

Part C1 basically means that numbers are unconditionally shifted everywhere except at $R\|L$ pairs. Since those are located at distance $k/2$ and numbers have length k this might look suspicious at first sight, because in general a number simultaneously gets compared at two such pairs. It will become clear later why this does not pose any problems. Ignoring it for the moment, C1 is easy to implement:

Algorithm 4.3 (Implementation of C1)**.**

(1) The cells which are *not* an R- or L-cell have a very simple behavior.
- The *left* register gets its content from the *left* register of the right neighbor.
- The *right* register gets its content from the *right* register of the left neighbor.

(2) Analogously the *left* register of an L-cell gets its content from the *left* register of the right neighbor and the *right* register of an R-cell gets its content from the *right* register of the left neighbor.

(3) The same holds for the *left* register of a R-cell and the *right* register of a L-cell if the *comp* register of that cell has value $=$.

First of all, this part is needed during the first sub-phase of phase 1 when the $R\|L$ pairs in the middle of a block still have no meaning. As will be seen this requirement is also consistent with the rules for later sub-phases.

Each of the phases 1, …, n is subdivided into two sub-phases of $k/2$ steps each.

We will now describe how the comparison of numbers is done. For this we use $R.\mathit{left}$ to denote the *left* register of the R-cell and similarly for the other cases. It will be seen that all information needed to update $R.\mathit{comp}$ are also available in the neighboring L-cell, so that the invariant $R.\mathit{comp} = L.\mathit{comp}$ can be maintained. Hence it suffices to describe the case of $R.\mathit{comp}$:

Algorithm 4.4 (Implementation of C2)**.** If the two bits in *R.right* and *L.left* are most significant ones, then the new value of *R.comp* is determined as follows:

$$\mathit{R.comp} \text{ becomes } \begin{cases} < & \text{if } \mathit{R.right} < \mathit{L.left} \\ = & \text{if } \mathit{R.right} = \mathit{L.left} \\ > & \text{if } \mathit{R.right} > \mathit{L.left} \end{cases} \tag{4.1}$$

If the two bits to be compared are not most significant ones, the new value of *R.comp* is determined as follows:

- If *R.comp* already has value $<$ or $>$ it is not changed.
- If *R.comp* has old value $=$ its new value is determined according to rule 4.1 above.

It remains to define how the new value for *R.left* is computed. That is most easily described as depending on the just defined *new* value of *R.comp.*:

$$\mathit{R.left} \text{ becomes } \begin{cases} \mathit{R.right} & \text{if } \mathit{R.comp} \text{ is now } < \\ \mathit{R.right} & \text{if } \mathit{R.comp} \text{ is now } = \\ \mathit{L.left} & \text{if } \mathit{R.comp} \text{ is now } > \end{cases}$$

Dually the new value for *L.right* depends on the *new* value of *L.comp* (remember that always *R.comp* = *L.comp*):

$$\mathit{L.right} \text{ becomes } \begin{cases} \mathit{L.left} & \text{if } \mathit{L.comp} \text{ is now } < \\ \mathit{L.left} & \text{if } \mathit{L.comp} \text{ is now } = \\ \mathit{R.right} & \text{if } \mathit{L.comp} \text{ is now } > \end{cases}$$

Figure 2 shows the relevant parts of computations for the comparison of two numbers. In the left part of the figure initially the larger number is on the left, in the right part the smaller number is on the left. After the comparison in both cases the smaller number is on the left. We remind the reader that at the left resp. right end of the complete input a missing number is treated as $-\infty$ resp. ∞. Hence a number arriving at a border is simply reflected.

The following picture may be helpful: When the smaller number comes from the left and the larger from the right, then from the first (most significant) bit both numbers are reflected at the border between the *R*- and the *L*-cell. When the larger number comes from the left and the smaller from the right, then from the first (most significant) bit both numbers pass through the border. This is a correct picture even if the numbers have identical higher order bits, and hence in the beginning no cell knows which is the present case. Readers are encouraged to check Figure 2 again.

At least from a formal point of view it is now straightforward to put the pieces together. An even better intuition of how the algorithm works may arise in the subsequent Subsection 4.4 when the correctness of the algorithm will be shown.

				R	L			
left					⟨0	0	1	1
right	1	0	1	0⟩				
comp								
left				⟨0	0	1	1	
right		1	0	1	0⟩			
comp				=	=			
left			⟨0	0	1	1		
right			1	0	1	0⟩		
comp				>	>			
left		⟨0	0	1	1			
right				1	0	1	0⟩	
comp				>	>			
left	⟨0	0	1	1				
right					1	0	1	0⟩
comp				>	>			

				R	L			
left					⟨0	1	0	1
right	1	1	0	0⟩				
comp								
left				⟨0	1	0	1	
right		1	1	0	0⟩			
comp				=	=			
left			⟨0	0	0	1		
right			1	1	1	0⟩		
comp				<	<			
left		⟨0	0	1	1			
right				1	0	1	0⟩	
comp				<	<			
left	⟨0	0	1	1				
right					1	0	1	0⟩
comp				<	<			

Figure 2: Two comparisons of 4 bits. On the left hand side the number coming from the left (in the *right* registers) is larger, on the right hand side the number coming from the right (in the *left* registers). If the complete numbers were longer, the comparisons would continue analogously.

Algorithm 4.5.

- First the setup phase is done as described in Algorithm 4.2. This phase takes k steps. It is stopped at the correct time using a synchronization algorithm in each block separately. Both ends of each block have to act as generals in order to achieve the required synchronization time.
- Once all cells are synchronized they will work as described in Algorithms 4.3 and 4.4: All numbers are shifted to the left or right, and whenever two most significant bits meet, the sequential comparison of the two numbers is started. The smaller number is sent to the left and the larger to the right.
- This is repeated until the synchronization started immediately after the setup phase fires all nk cells after nk steps.

  It will be shown that at that point in time the *left* registers contain the sorted numbers.

4.4. Correctness of the base sorting algorithm

The correctness of algorithm 4.5 is essentially due to the correctness of odd-even transposition sort. This is basically a proof by induction. The main parts are stated in the following Lemma.

Lemma 4.6. *Let $\mathcal{X} = |xX\rangle$ and $\mathcal{Y} = \langle Yy|$ be two numbers with the $k/2$ higher order bits denoted by capital letters and the $k/2$ lower order bits denoted by small letters. Similarly let $\mathcal{A} = \langle Aa|$ be the minimum of $\mathcal{X}$ and $\mathcal{Y}$ and $\mathcal{B} = |bB\rangle$ be the maximum. In other words one basic compare-and-exchange step of odd-even transposition sort transforms the pair $\mathcal{X}, \mathcal{Y}$ into the pair $\mathcal{A}, \mathcal{B}$.*

If the most significant bits of $X\rangle$ and $\langle Y$ meet in an $R\|L$-pair (with the other higher order bits following) and if after $k/2$ steps the lower order bits $|x$ and $y|$ arrive in the correct order, then during the first $k/2$ steps the higher order bits of $\langle A$ and $B\rangle$ will be produced moving to the right directions, followed by the lower order bits $a|$ and $|b$ afterwards.

This is basically a restatement of the construction from Algorithm 4.4.

Since the configuration produced by the setup phase corresponds to the initial configuration for the odd-even transposition sort, and since the preconditions of the if-statement in Lemma 4.6 are met, an induction teaches that in particular the higher order bits of each number after t phases are in the sub-block corresponding to its position in odd-even transposition sort after t sorting steps.

Corollary 4.7. *For all input sequences $w_1, \ldots, w_n$ Algorithm 4.5 does sort the numbers as required in Problem 2.1.*

Proof. Since odd-even transposition sort does sort N numbers in N sorting steps, it immediately follows from Lemma 4.6 that at the end of phase n of Algorithm 4.5 the higher order bits of each of the n input numbers and of its n copies are in the correct blocks. That is, there are the same higher order bits of a number w_i in each block twice, once in the *left* registers and once in the *right* registers.

Furthermore it is clear that the most significant bit of the number stored in the *left* (resp. *right*) registers is in the L-cell (resp. R-cell) of the *block* (not sub-block). This implies that the lower order bits are in the same block:

	L	R
left	$\langle k/2$ higher order bits of w_i	$k/2$ lower order bits of $w_i|$
right	$|k/2$ lower order bits of w_i	$k/2$ higher order bits of $w_i\rangle$

Therefore the *left* registers of the full block hold the correct value. ■

5. A sorting algorithm matching the lower bound

We will save k steps of the running time of Algorithm 4.5 by starting with some comparisons not after k but already after $k/2$ steps and stopping the odd-even transposition sort $k/2$ steps earlier. A detailed description will be given in Section 5.1. The resulting algorithm still computes the correct output *except* for the rightmost block. This will be fixed in Section 5.2.

5.1. Speeding up the algorithm

We describe the fast algorithm as three changes to Algorithm 4.5.

The first change is simple: Since we want to have the result after nk steps instead of $k + nk$, the synchronization of all nk cells is not started after the setup phase, but in the very first step.

The second change concerns the computation of the mirror of a bit string as required by Algorithm 4.2. It can be implemented by shifting the original to the left, and letting the L-cell of the block act as reflector sending bits back to the right in the *right* cells. This means that after $k/2$ steps the lower order bits of an input number have arrived in the *left* registers of the left sub-block and the higher order bits are its *right* registers. It is useful if the shift to the left is not done using a temporary register but *left*. In Figure 3 the resulting process is shown for two adjacent 6-bit numbers. It can be seen that due to the simultaneous shift to the left, already after $k/2$ steps for the first time most significant bits meet. (This also determines the border of the sub-blocks.) Thus comparisons can be started $k/2$ steps earlier. The rightmost block now needs some special attention. Analogously to the other blocks we assume that symbols representing the "number ∞" are shifted to the left from the rightmost cell. This is also depicted in Figure 3.

This completes the second change to Algorithm 4.5.

left	$\langle a_6$	a_5	a_4	a_3	a_2	$a_1\vert$	$\langle b_6$	b_5	b_4	b_3	b_2	$b_1\vert$
right												

left	a_5	a_4	a_3	a_2	$a_1\vert$	$\langle b_6$	b_5	b_4	b_3	b_2	$b_1\vert$	∞
right	$a_6\rangle$						$b_6\rangle$					

left	a_4	a_3	a_2	$a_1\vert$	$\langle b_6$	b_5	b_4	b_3	b_2	$b_1\vert$	∞	∞
right	a_5	$a_6\rangle$					b_5	$b_6\rangle$				

left	a_3	a_2	$a_1\vert$	$\langle b_6$	b_5	b_4	b_3	b_2	$b_1\vert$	∞	∞	∞
right	a_4	a_5	$a_6\rangle$				b_4	b_5	$b_6\rangle$			

Figure 3: Computing the mirror of two 6-bit numbers. Already after 3 steps most significant bits meet at the border between sub-blocks.

The third change is the most complicated. For the shifts to the right *two* registers are used, *right* and *right2*. The additional register *right2* is empty almost everywhere. After each sub-phase there are only two adjacent sub-blocks in which *right2* stores a number:

(1) The initialization takes place in the leftmost block: During the first $k/2$ steps the reflected bits are shifted to the right in *right* and *right2*. And this is done *only* during the first $k/2$ steps, but not afterwards.

(2) Then it happens for the first time that three most significant bits meet, two coming from the left and one coming from the right.

In such a case the comparisons are done as follows:

- The largest of the three numbers is shifted to the right in *right2*.
- The other two numbers are compared and shifted as described in Algorithm 4.4.

A consequence of these rules is, that after k steps the *right2* registers are used in the sub-blocks of block 1, after $2k$ steps in the sub-blocks of block 2, etc. and after nk steps in the sub-blocks of block n, and nowhere else.

Why do these three changes lead to a result after nk steps where in all blocks except the rightmost one the *left* registers contain the correct numbers? That the rightmost block can still be wrong can be seen in examples.

First of all, reflecting w_1 twice make sure that there are really $2n$ numbers which are sorted, each w_i twice. Therefore in the end each block will again contain twice the same number. This would in general not be the case if w_1 would be reflected only once, and the argument below would fail.

Since we are still using odd-even transposition sort, it is clear that after $k/2+nk$ steps the correct results are obtained everywhere, but with the most significant bits in the middle of the blocks. Thus the result would look like

	L	R
left	$k/2$ lower order bits of $w_i\vert$	$\langle k/2$ higher order bits of w_i
right	$k/2$ higher order bits of $w_i\rangle$	$\vert k/2$ lower order bits of w_i

How can such a left sub-block arise? Without loss of generality assume that the input numbers are pairwise different. Then in the left neighboring full block a smaller number (or $-\infty$) is present. Hence the lower left half must have been reflected and going back $k/2$ steps, the bits must have been in the upper part. Analogously the upper right half must have been in the lower part. Except in the rightmost block (where the *right2* registers are in use) there is no other possibility than that the lower order bits are in the remaining registers:

	L	R
left	$\langle k/2$ higher order bits of w_i	$k/2$ lower order bits of $w_i\vert$
right	$\vert k/2$ lower order bits of w_i	$k/2$ higher order bits of $w_i\rangle$

Thus the *left* registers hold the desired result.

In the rightmost block it can happen that lower order bits are not stored in the *left* registers of the right sub-block but in the *right2* registers of the left sub-block.

5.2. Determining the rightmost output block

In order to produce the largest number in the rightmost output block we use a separate algorithm which has to be run in parallel to the one described above. It will have finished after nk steps. Remember that the input is a word

$$w = w_1 \cdots w_n = \langle x_1 y_1 z_1| \cdots \langle x_n y_n z_n|$$

and the task is to have the maximum of the w_i be stored in the rightmost block in the end. This can be achieved as follows.

Algorithm 5.1.

(1) During the k steps of the setup phase a signal is sent from the right end of the input until it reaches the most significant bit of w_n, marking all cells as belonging to the last block.

When the last block is synchronized after k steps, all cells have received the information and know that they have an additional task.

(2) From the very first step *all* cells shift their input to the right using an additional register.

Hence after $1 \cdot k$ steps the number w_{n-1} reaches the last block, after $2 \cdot k$ steps number w_{n-2} reaches the last block, etc. and after $(n-1)k$ steps number w_1 reaches the last block.

(3) The cells in the rightmost block use two additional registers for storing numbers; call them *max* and *next*. Register *max* is initialized in the very first step with w_n, register *next* is marked as not holding a value. Whenever the rightmost block ends a phase, in register *next* the number is stored that has arrived from the left in *right* because of the shifting.

(4) If at the beginning of a phase register *next* has a valid number the rightmost block computes $\mathit{max} \leftarrow \max(\mathit{max}, \mathit{next})$. For this a signal is sent from left to right, that is from the most significant bit to the least significant bit, comparing *next* and *max*. (While this comparison takes place the next number is already arriving in *right*.)

As long as the same bit value is found in both registers nothing is changed and the signal moves one cell to the right.

As soon as at some position for the first time different bit values are found, the following happens:

- If *next* has a 1 bit, but *max* has a 0 bit, *max* is smaller than *next* and this and all remaining bits are copied from *next* to *max*.
- If *next* has a 0 bit, but *max* has a 1 bit, *max* is larger than *next* and the signal is simply killed leaving *max* unchanged.

It is straightforward to verify by induction that for $1 \leq i < n$ after phase i one has

$$\mathit{max} = \max\{w_{n-j} \mid 0 \leq j < i\}$$

and hence in the end $\mathit{max} = \max\{w_i \mid 1 \leq i \leq n\}$ as required. Since w_1 is copied to *next* after $(n-1)k$ steps the final correct value is stored in *max* k steps later, i.e. after nk steps as required.

Taking together the changes to the base Algorithm 4.5 described in Section 5.1 and the additional algorithm just described one gets a proof of Theorem 2.3.

6. Conclusion

We have shown the sorting of n numbers with k bits can be achieved in (almost) real-time. Thus the situation is very similar to the firing squad synchronization problem: There is an algorithm which has — in our case except for one step — a running time matching a lower bound.

Clearly, the number of states per cell required by our algorithm is finite but large, at least when compared to algorithms e. g. for the synchronization problem. We do not know how much the set of states can be reduced.

The authors gratefully acknowledge a number of suggestions by the referees for improving the presentation of the algorithms.

References

[1] Peter van Emde Boas. Machine Models and Simulations. *Handbook of Theoretical Computer Science, Volume A*, 1–66, Elsevier, 1990.

[2] Donald Knuth. *The Art of Computer Programming; Volume 3, 2nd ed.*, Addison-Wesley, 1998.

[3] Hidenosuke Nishio. Real time sorting of binary numbers by 1-dimensional cellular automaton. In *Proceedings of the International Symposium on Uniformly Structured Automata and Logic, Tokyo, Japan, August 21-23, 1975, IEEE Catalog Number 75 CH1052-OC*, pages 153–162, 1975.

[4] Amir R. Schorr. Physical parallel devices are not much faster than sequential ones. *Information Processing Letters*, 17(2):103–106, 1983.

[5] Hiroshi Umeo, Masaya Hisaoka, and Takashi Sogabe. A Survey on Optimum-Time Firing Squad Synchronization Algorithms for One-Dimensional Cellular Automata. *Int. Journal of Unconventional Computing*, 1(4):403–426, 2005.

Turku Centre for Computer Science
TUCS Lecture Notes

1. **Ralph-Johan Back och Joakim von Wright**, Matematik med lite logik: Strukturerade härledningar i gymnasiematematiken
2. **Ralph-Johan Back och Joakim von Wright**, Matematik med lite logik: En kort kurs i talteori
3. **Ralph-Johan Back och Joakim von Wright**, Matematik med lite logik: Studentexamen i lång matematik, våren 2003
4. **Ralph-Johan Back och Joakim von Wright**, Matematiikkaa logiikan avulla: Rakenteiset päättelyketjut lukiomatematiikassa
5. **Ralph-Johan Back och Joakim von Wright**, Matematiikkaa logiikan avulla: Lyhyt lukuteorian kurssi
6. **Ralph-Johan Back och Joakim von Wright**, Matematiikkaa logiikan avulla: Pitkän matematiikan ylioppilaskoe, kevät 2003
7. **Ralph-Johan Back och Joakim von Wright**, Matematik med lite logik: Introduktion till strukturerade härledningar
8. **Ralph-Johan Back och Joakim von Wright**, Matematik med lite logik: Logik för strukturerade härledningar
9. **Ralph-Johan Back och Joakim von Wright**, Matematik med lite logik: Strukturerade härledningar som allmänt bevisformat
10. **Ralph-Johan Back och Joakim von Wright**, Matematiikkaa logiikan avulla: Johdatus rakenteisiin päättelyketjuihin
11. **Ralph-Johan Back och Joakim von Wright**, Matematiikkaa logiikan avulla: Logiikka ja rakenteiset päättelyketjut
12. **Ralph-Johan Back och Joakim von Wright**, Matematiikkaa logiikan avulla: Rakenteiset päättelyketjut yleisenä todistusmuotona
13. **Jarkko Kari (Editor)**, Proceedings of JAC 2010 – Journées Automates Cellulaires

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku

- Department of Information Technology
- Department of Mathematics

Åbo Akademi University

- Department of Information Technologies

Turku School of Economics

- Institute of Information Systems Sciences

ISBN 978-952-12-2504-8
ISSN 1797-8831

Jarkko Kari

Proceedings of JAC 2010 – Journées Automates Cellulaires