JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

Institut für Informatik

19th International Workshop on
Cellular Automata and
Discrete Complex Systems

AUTOMATA 2013

Exploratory Papers

Jarkko Kari, Martin Kutrib, Andreas Malcher (eds.)

Giessen, Germany
September 17 – 19, 2013

IFIG Research Report 1302

Title: Proceedings 19th International Workshop on Cellular Automata and Discrete Complex Systems (AUTOMATA 2013) – Exploratory Papers

IFIG Research Report 1302

Published by: Institut für Informatik
Universität Gießen
Arndtstraße 2
35392 Gießen
Germany

Published in September 2013

Preface

AUTOMATA 2013 is the 19th International Workshop on Cellular Automata and Discrete Complex Systems and continues a series of events established in 1995. AUTOMATA is an annual workshop and aims of the workshop are:

- To establish and maintain a permanent, international, multidisciplinary forum for the collaboration of researchers in the field of Cellular Automata (CA) and Discrete Complex Systems (DCS).
- To provide a platform for presenting and discussing new ideas and results.
- To support the development of theory and applications of CA and DCS (e.g. parallel computing, physics, biology, social sciences, and others) as long as fundamental aspects and their relations are concerned.
- To identify and study within an inter- and multidisciplinary context, the important fundamental aspects, concepts, notions and problems concerning CA and DCS.

AUTOMATA 2013 was organized by the Institut für Informatik of the Universität Giessen and took place place at the campus of natural sciences. It was a three-day workshop from September 17 to September 19, 2013.
This volume contains the accepted exploratory papers of AUTOMATA 2013. We would like to thank all authors for their contributions. The accepted exploratory papers were selected by the Program Committee and we are grateful to all members of the Program Committee for their excellent work in making this selection.
We are also grateful to the additional members of the Organizing Committee consisting of Susanne Gretschel, Markus Holzer, Sebastian Jakobi, Katja Meckel, Julien Provillard, Heinz Rübeling, and Matthias Wendlandt for their support of the sessions and the accompanying events.
Finally, we are indebted to all participants for attending the workshop. We hope that this workshop will be a successful and fruitful meeting, will bear new ideas for investigations, and will bring together people for new scientific collaborations.

Giessen, September 2013

Jarkko Kari
Martin Kutrib
Andreas Malcher

Table of Contents

Program

Thuesday, September 17, 2013

08.50 – 09.00	**Opening**
09.00 – 10.00	**Invited talk:**
	Pedro de Oliveira
	Conceptual connections around density determination in cellular automata
10.00 – 10.30	**Coffee break**
10.30 – 11.00	Olivier Bouré, Nazim Fatès, Vincent Chevrier
	A robustness approach to study metastable behaviours in a lattice-gas model of swarming
11.00 – 11.30	Ramon Alonso-Sanz
	Elementary cellular automata with memory of delay type
11.30 – 11.50	Witold Bołt, Jan M. Baetens, Bernard De Baets
	Identifying CAs with evolutionary algorithms
11.50 – 14.00	**Lunch**
14.00 – 14.30	Ville Salo and Ilkka Törmä
	Commutators of bipermutive and affine cellular automata
14.30 – 15.00	Alberto Leporati and Luca Mariot
	1-Resiliency of bipermutive cellular automata rules
15.00 – 15.20	Jarkko Kari and Kuize Zhang
	Two transitive cellular automata and their strictly temporally periodic points
15.20 – 15.50	**Coffee break**

15.50 – 16.10	Véronique Terrier
	Linear acceleration for one-dimensional cellular automata
16.10 – 16.30	Thomas Worsch
	Standardizing the set of states and the neighborhood of asynchronous cellular automata
18.00	**Welcome Reception**

Wednesday, September 18, 2013

09.00 – 10.00	**Invited talk:**
	Enrico Formenti
	Exploring m-asynchronous cellular automata
10.00 – 10.30	**Coffee break**
10.30 – 11.00	Fritz von Haeseler, Hidenosuke Nishio
	On polynomial rings in information dynamics of linear CA
11.00 – 11.30	Ville Salo and Ilkka Törmä
	Color blind cellular automata
11.30 – 11.50	Jarkko Kari, Ville Salo and Ilkka Törmä
	Surjective two-neighbor cellular automata on prime alphabets
11.50 – 14.00	**Lunch**
14.00 – 15.00	**Invited tutorial:**
	Nazim Fatès
	Turing, symmetry breakings and patterns – A tutorial on stochastic cellular automata
15.00 – 15.30	**Coffee break**
15.30 – 15.50	Biswanath Sethi, Souvik Roy, Sukanta Das
	Experimental study on convergence time of elementary cellular automata under asynchronous update

15.50 – 16.10	Tetsuo Imai, Atsushi Tanaka *Analysis of discrete state space partitioned by the attractors of the dynamic network formation game model*
16.10 – 16.30	Shigeru Ninagawa, Genaro Martinez *Complexity analysis in cyclic tag system emulated by rule 110*
16.45 – 17.30	**Business meeting of IFIP WG 1.5**
17.45	**Visit of a brewery**

Thursday, September 19, 2013

10.00 – 11.00	**Invited talk:** Nazim Fatès *A guided tour of asynchronous and stochastic cellular automata – A constructive role of randomness?*
11.00 – 11.30	**Coffee break**
11.30 – 12.00	Tarek Melliti, Damien Regnault, Adrien Richard, Sylvain Sené *On the convergence of Boolean automata networks without negative cycles*
12.00 – 12.20	Pedro Montealegre, Eric Goles *Computational complexity of majority automata under different updating schemes*
12.20 – 14.30	**Lunch**
14.30 – 15.00	Sandip Karmakar, Dipanwita Roy Chowdhury *Leakage squeezing using cellular automata*

15.00 – 15.20	Nazma Naskar, Sumit Adak, Sukanta Das *Identification of non-uniform periodic boundary cellular automata having only point states*
15.20 – 15.40	Oleksandr Makarenko *Cellular automata with strong anticipation property of elements*
19.00	**Workshop Dinner**

Workshop ***AUTOMATA 2013 – Exploratory Papers***

Identifying CAs with evolutionary algorithms

Witold Bołt[1,2], Jan M. Baetens[1], and Bernard De Baets[1]

[1]KERMIT, Department of Mathematical Modelling, Statistics and Bioinformatics, Ghent University, Ghent, Belgium
[2]Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
e-mail: witold.bolt@hope.art.pl

Abstract

Cellular Automata (CA) identification from an incomplete time series of CA configurations is considered. A flexible evolutionary algorithm for identification is developed and its performance is checked for a simplified setting in which one can rely on a complete time series of configurations. Ideas for extension to the case in which some time frames are missing are presented and commented. The description is accompanied with results including a brief analysis of Elementary CAs (ECA) discoverability.

1 Introduction

Cellular Automata (CAs) present an attractive and effective modeling technique for a variety of problems. In order to use CAs in any practical modeling task, one needs to understand the underlying rules, relevant to the given phenomenon, and translate them into a CA local rule. Additionally, the state space, tessellation and neighborhood structure need to be pinned down beforehand. This narrows the application area for CAs, since there are problems for which only the initial and final states are known (*e.g.* [2,13,14]). Such problems motivate the research towards automated CA identification. Various methods have been used, including genetic algorithms [6, 11, 12, 15], genetic programming [3, 4, 10], gene expression programming [7], ant intelligence [9], machine learning [5] as well as direct search/construction approaches [1, 16–18].

Existing methods can be divided into two main groups. Firstly, methods for solving problems like the majority classification in which we only know the initial condition and desired outcome and, secondly, methods that exploit the entire time series of configurations, where it is assumed that all

configurations are known. Only limited research efforts have been devoted to problems involving an incomplete spatio–temporal image [12].

In this exploratory paper we outline a Genetic Algorithm (GA) for uncovering CA rules from a time series of CA configurations. The proposed method is flexible enough to handle cases of noisy and incomplete time series. Although direct approaches for constructing CA from data exist, they are of no use in such case, due to their requirements on completeness of the data.

The paper is organized as follows. In Section 2 we formalize the identification problem. In Section 3 we describe the search algorithm, while in Section 4 the results and performance evaluation. We conclude this paper with comments on future research goals and extensions to the algorithm.

2 Problem setting

Let $I = (I_{t,s})$ for $t \in \mathcal{T}$, $s \in \mathcal{S} = \{0, 1, \ldots, S\}$, be a binary image, *i.e.* for every t, s we have: $I_{t,s} \in \{0, 1\}$. The indices t, s refer to positions in time and space, respectively. The time domain $\mathcal{T}$ is a subset of $\{0, 1, \ldots, T\}$ that includes 0. We will also write $I_t := (I_{t,s})_{s \in \mathcal{S}}$ to denote the t-th row of the image (the t-th time step). In this paper image refers to the space–time diagram of a CA, *i.e.* bit string I_t denotes the configuration after iterating the CA t times starting from the initial condition I_0. Let $\mathcal{I} = \{I^{(j)} \mid j \in \mathcal{J}\}$ be a set of images, where for each $j \in \mathcal{J}$, we have $I^{(j)} = (I^{(j)}_{t,s})$, $t \in \mathcal{T}_j, s \in \mathcal{S}_j$. We will refer to this set as the "test set".

We consider the following identification problem: for a given test set $\mathcal{I}$, find 1D, two-state, deterministic CA $\mathscr{A}$, with a symmetric neighborhood of radius r and periodic boundary conditions, that can reproduce the images in $\mathcal{I}$. If we use A to denote the global rule (global function) of automaton $\mathscr{A}$, this question boils down to finding a function A such that:

$$\sum_{j \in \mathcal{J}} \sum_{t \in \mathcal{T}_j} \sum_{s \in \mathcal{S}_j} \left| I^{(j)}_{t,s} - A^t \left(I^{(j)}_0 \right) [s] \right| = 0, \tag{1}$$

where $A^t(I^{(j)}_0)[s]$ denotes the s-th position in the vector resulting from applying the CA rule t times starting from the initial condition $I^{(j)}_0$.

Obviously, for many test sets there is no CA that satisfies (1). For this reason it is advisable to widen the scope of the identification problem, in such a way that the goal becomes to find a CA that minimizes the error given by:

$$E(A, \mathcal{I}) := \sum_{j \in \mathcal{J}} \sum_{t \in \mathcal{T}_j} \sum_{s \in \mathcal{S}_j} \left| I^{(j)}_{t,s} - A^t \left(I^{(j)}_0 \right) [s] \right|. \tag{2}$$

As noted in the Section 1, the identification problem at stake can be solved directly, *i.e.* without relying on heuristic search techniques, when

$\mathcal{T} = \{0, \ldots, T\}$, and the test set is known to be generated by some unknown CA. This even holds in cases where the neighborhood structure is unknown [16, 17].

The goal of this research is to establish methods for solving the identification problem in more complicated cases, such as the one for which $\mathcal{T} \subsetneq \{0, 1, \ldots, T\}$ and only the ordering of elements of $\mathcal{T}$ is known, or those for which the images in the test set are noisy. Such problems justify the use of evolutionary search techniques in order to minimize (2).

In this introductory paper we validate the evolutionary search approach for simple problems, being those which could be solved directly, to show that the method produces meaningful results. For the sake of simplicity, we assume throughout the remaining sections that $\mathcal{T} = \{0, 1, \ldots, T\}$ and that the neighborhood radius is known to be $r \geq 0$. Additionally, we consider only one test image I in the test set $\mathcal{I}$, which is known to be generated by some CA with neighborhood of radius not higher than r. Such a problem is known to have at least one "perfect" solution A satisfying: $E(A, \mathcal{I}) = 0$.

3 Genetic Algorithm outline

We will use a well-established heuristic search technique, namely a GA [8], to search for solutions of our problem. To apply GA, we need to define: **(a)** the elements of the population and their representation, **(b)** a fitness function to rank individuals and **(c)** the genetic operators to evolve the populations.

(a) The populations consist of CAs encoded by the local rule's lookup table (LUT), which can be stored as a fixed-length bit string. Since the radius is known to be r, the length of such a string is $N = 2^{2r+1}$. So if A is an element of the population, then $A = (a_1, a_2, \ldots, a_N)$ and $a_i \in \{0, 1\}$. We will use fixed-size populations of C individuals.

By means of $\mathcal{P}_i$ we will denote the population at the i-th iteration of the GA. Population $\mathcal{P}_0$ is created by selecting uniformly at random, binary vectors of length N, while populations $\mathcal{P}_i$ for $i > 0$ are composed using the genetic operators defined below.

(b) Typically, the goal of a GA is to maximize a predefined fitness function. Here, the fitness function should correspond to the error function $E(A, \mathcal{I})$ defined in (2). For technical reasons it is desirable to normalize fitness values to the unit interval. The most natural choice for the fitness function, which suits those needs, would be the ratio of cells in the image that have the same states as the ones in the corresponding CA's spatio–temporal diagram to the total number of cells S, and this for each consecutive time step. Formally, such a fitness function could be expressed as:

$$\mathrm{fit}_E(A) := 1 - \frac{E(A, \mathcal{I})}{T\,S}. \tag{3}$$

At first rather surprisingly, we found this fitness function not to work, which can however be understood by acknowledging that errors propagate and amplify in time. So, even though the sum $\sum_{s\in\mathcal{S}} \left|I_{t,s} - A^t\left(I_0\right)[s]\right|$ for small t might be small, generally it grows with time, because of the errors at previous time steps. In some sense, we are accounting for the same errors multiple times.

To overcome this shortcoming a new fitness function was constructed, which considers pairs of consecutive configurations, and thereby eliminates the impact of error propagation. Its formula is given by:

$$\mathrm{fit}(A) := 1 - \frac{1}{(T-1)\,S} \sum_{t\in\mathcal{T}\setminus\{0\}} \sum_{s\in\mathcal{S}} \left|I_{t,s} - A(I_{t-1})[s]\right|. \tag{4}$$

Note that if A is a local rule of a CA, then $E(A,\mathcal{I}) = 0$ if and only if $\mathrm{fit}(A) = 1$. Hence, when a perfect solution exists, maximizing fit is equivalent to solving $E(A,\mathcal{I}) = 0$. Otherwise, it might happen that the maximum of fit doesn't match the minimum of E, but still the results obtained from maximizing fit might be meaningful. Also note that the fitness defined by (4) is closely related to formula (7) in [16], which is used in a direct search of the LUT entries.

(c) Having defined the fitness function, we now turn to the definition of the genetic operators.

Selection and reproduction – those operators are responsible for selecting individuals from population $\mathcal{P}_{i-1}$ to build population $\mathcal{P}_i$. We select individuals at random, with a selection probability weighted with the fitness value. We make $2C$ selections (to pick parents) and from each pair, we build one offspring rule that ends up in $\mathcal{P}_i$.

Cross–over – assume that R and Q are two individuals selected from $\mathcal{P}_{i-1}$ for which holds that $R = (r_k)$ and $Q = (q_k)$. The resulting rule $Z = (z_k)$ is built by randomly selecting bits from R or Q, such that $z_k \in \{r_k, q_k\}$ with a selection probability p_c for $z_k = r_k$ and $1 - p_c$ for $z_k = q_k$.

Mutation is done by inverting (flipping) one randomly selected bit in the LUT and is applied to the rule produced by cross–over. The mutation is applied with probability p_m.

In summary, the algorithm starts by randomly creating population $\mathcal{P}_0$. Then the fitness values are calculated for each of the individuals and the aforementioned genetic operators are used to build $\mathcal{P}_1$. This process is repeated until a perfect solution is found (with fitness equal to 1) or a predefined number of M iterations has been taken. In the latter case, the algorithm returns the individual with the highest fitness.

In practice, we often used a slightly modified variant of the algorithm that assures the preservation of the fittest individuals. For that purpose we use a basic elitist survival scheme: after evolving $\mathcal{P}_i$ we pick $C_E \ll C$

of the fittest individuals from $\mathcal{P}_{i-1}$ and place them at random positions in $\mathcal{P}_i$ replacing the formerly evolved individuals. In the reminder, this variant will be referred to as Algorithm E, whereas the original one will be named Algorithm NE.

Although both algorithms rely on the presumption that the neighborhood radius is known, they can be adjusted easily to detect the radius (although the resulting value won't be optimal). To achieve this, we have to add two mutation operators, namely: up–scale and down–scale mutation. The former translates the rule's LUT from radius r to $r+1$, whereas the down–scale mutation does exactly the opposite. Up–scale mutation is uniquely defined, while for its down–scale counterpart, for most of the rules, the result is not uniquely defined, and thus one of the possibilities is picked randomly. In addition to those operators, also the cross–over operator needs to be adjusted. This can be accomplished straightforwardly by taking the largest of the two individuals' radii before applying cross–over. Clearly, an upper and lower bound has to be defined for this procedure to work.

4 Performance of the GA

4.1 Test Case 1: Rule 154

The simplest test case is based on an image generated by ECA rule 154. Obviously, looking for radius 1 rules with a GA is not going to lead to results of practical interest because all of the 256 elementary rules could be evaluated in the first few GA iterations. For that reason, we looked for radius 2 rules that can evolve the image generated by rule 154. Algorithm NE was used with the following parameter values, which were tuned based upon a preliminary simulation study: $p_c = 0.5$, $p_m = 0.02$, $C = 100$, $T = 100$ and $S = 100$. The initial condition (I_0) was a random binary vector. Figure 1 depicts the fitness evolution during one exemplary GA life-cycle in which the perfect solution was found in 128 iterations. As can be seen on the plot, almost all iterations produced fitter individuals (on average), which hints that indeed the algorithm is convergent.

4.2 Discoverability of radius 1 rules

In this experiment we evaluated the 256 ECA rules using algorithm NE. Parameters of the algorithm were set to the same values as in Test Case 1. The maximum number of GA iterations was $M = 5000$. Similarly to Test Case 1, the goal was to find a radius 2 rule that can evolve the test image. The same initial condition was used for all rules. Based upon 30 runs, we measured the discoverability ratio expressed as a percentage, *i.e.* the percentage of runs in which a perfect solution was found. Additionally, we measured the average number of GA iterations needed to halt the algorithm.

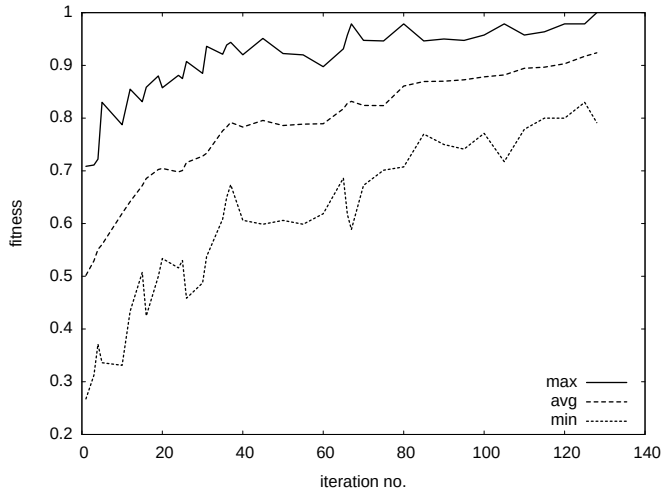


Figure 1: Evolution of the maximum, average and minimum fitness in Test Case 1.

rule	**avg. iter.**	**discoverability**
75	49.57	100%
30	53.10	100%
169	53.63	100%
165	53.80	100%
106	54.77	100%

(a)

rule	**avg. iter.**	**discoverability**
245	4997.23	3.33%
98	4993.47	3.33%
188	4987.80	3.33%
6	4982.07	3.33%
43	4981.33	3.33%

(b)

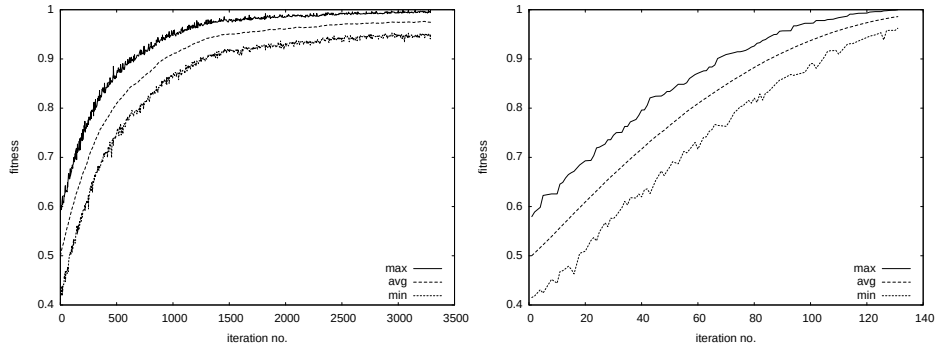
Table 1: Rule discoverability and average number of iterations for those ECA that give rise to the ten lowest (a) and highest (b) numbers of iterations.

Out of the 256 examined rules, 88 rules have zero discoverability and 42 rules have 100% discoverability. Average discoverability was found to be 31.38%. Results for the top 5 rules with both the highest and lowest number of iterations are shown in Table 1. Note that for the latter set of rules, we only consider rules that have discoverability greater than 0%, as those with zero discoverability obviously always required the maximum number of 5000 iterations.

Establishing more detailed discoverability measures and relating them to complexity measures is one of the topics of our current research.

4.3 Test Case 2: Radius 4 rule

Similar to Test Case 1, we examined an image generated by a CA rule, but in this case it was a randomly selected, radius 4 rule. Parameters for this



(a) Algorithm NE (b) Algorithm E

Figure 2: Evolution of maximum, average and minimum fitness in Test Case 2.

test case were set to: $p_c = 0.5$, $p_m = 0.02$, $C = 5000$, $T = 200$, $S = 200$ and $M = 20000$. Due to the complexity of the problem (for radius 4 the LUT bit strings contain 512 bits, compared to 32 bits in case of radius 2) the value of parameter C was increased compared to previous experiments. In this experiment we compared the performance of Algorithms E and NE (with elite size $C_E = 500$ for Algorithm E). Figure 2 shows exemplary fitness evolutions for both cases. To get a deeper understanding of differences between the two algorithms we ran this test case 30 times with both algorithms. It turned out that the average number of iterations required for Algorithm E was 132, while Algorithm NE needed 4259 iterations on average. Clearly, introducing elite preservation strongly improves the performance in this case.

5 Future work

As noted earlier the ultimate goal of the ongoing research is to infer the underlying rule if some time steps are missing in the test image. The method presented in Section 3 will be extended to support such cases. First of all, the fitness function has to be revised. Since the elements of $\mathcal{T}$ are unknown, we pick a value $t_{max} > 0$ representing the maximum time gap size, *i.e.* for all i we assume that $t_{i+1} - t_i < t_{max}$. Then, for each i we calculate one–step error values:

$$E_{i,j}(A, I) := \sum_{s \in \mathcal{S}} \left| I_{t_i,s} - R^j(I_{t_i})[s] \right|, \tag{5}$$

for $j \in \{1, 2, \ldots, t_{max}\}$ and we define the i-th step error as:

$$E_i(A, I) := \min_j E_{i,j}(A, I). \tag{6}$$

After that we can express the adjusted fitness function as:

$$\widehat{\mathrm{fit}}(A, I) := 1 - \frac{1}{(T-1)\,S} \sum_i E_i(A, I). \tag{7}$$

Note that, by optimizing the fitness function, we will get both the rule and the elements of $\mathcal{T}$, by taking: $t_{i+1} := t_i + \arg\min_j E_{i,j}(A, I)$. In case of a complete time series the fitness defined by (4) is equivalent to taking $t_{max} = 1$ in (7).

One could search for rules using the algorithm described in Section 3 and fitness function defined above, but our initial findings show that the performance of such an algorithm is relatively poor. Since the fitness function given by (7) uses all information contained in the image, further work needs to be concentrated on other parts of the algorithm. For instance, ongoing research concentrates more on flexible rule representation, new genetic operators and rules of evolving populations. Preliminary results are very promising, but still quite some effort is required to fully understand and overcome the problems in searching for rules in the incomplete image setting. Aside from the algorithm itself, some efforts are spent to a characterization of the uniqueness of solutions in different settings.

6 Summary

We briefly presented a simple, yet effective and flexible, genetic algorithm for searching CA rules matching an arbitrary test image set. We demonstrated that the presented method works for simple cases of ECAs as well as for some higher radius rules. Future research will be devoted towards applying the method to more general cases.

Acknowledgments

Witold Bołt is supported by the Foundation for Polish Science under "International PhD Projects in Intelligent Computing". Project is financed by The European Union within the Innovative Economy Operational Programme 2007–2013 and European Regional Development Fund.

References

[1] Adamatzky, A.: Identification Of Cellular Automata. Taylor & Francis Group (1994)

[2] Al-Kheder, S., Wang, J., Shan, J.: Cellular automata urban growth model calibration with genetic algorithms. In: Urban Remote Sensing Joint Event, 2007. pp. 1–5. IEEE (2007)

[3] Andre, D., Bennett III, F.H., Koza, J.R.: Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In: Proceedings of the First Annual Conference on Genetic Programming. pp. 3–11. MIT Press (1996)

[4] Bandini, S., Manzoni, S., Vanneschi, L.: Evolving robust cellular automata rules with genetic programming. In: Adamatzky, A., Alonso-Sanz, R., Lawniczak, A.T., Martínez, G.J., Morita, K., Worsch, T. (eds.) Automata. pp. 542–556. Luniver Press, Frome, UK (2008)

[5] Bull, L., Adamatzky, A.: A learning classifier system approach to the identification of cellular automata. J. Cellular Automata 2(1), 21–38 (2007)

[6] Bäck, T., Breukelaar, R., Willmes, L.: Inverse design of cellular automata by genetic algorithms: An unconventional programming paradigm. In: Banâtre, J.P., Fradet, P., Giavitto, J.L., Michel, O. (eds.) Unconventional Programming Paradigms, Lecture Notes in Computer Science, vol. 3566, pp. 161–172. Springer Berlin Heidelberg (2005)

[7] Ferreira, C.: Gene expression programming: a new adaptive algorithm for solving problems. CoRR cs.AI/0102027 (2001)

[8] Holland, J.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press (1975)

[9] Liu, X., Li, X., Liu, L., He, J., Ai, B.: A bottom-up approach to discover transition rules of cellular automata using ant intelligence. Int. J. Geogr. Inf. Sci. 22(11-12), 1247–1269 (2008)

[10] Maeda, K., Sakama, C.: Identifying cellular automata rules. J. Cellular Automata 2(1), 1–20 (2007)

[11] Mitchell, M., Crutchfield, J.P., Das, R., et al.: Evolving cellular automata with genetic algorithms: A review of recent work. In: Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA'96) (1996)

[12] Richards, F.C., Meyer, T.P., Packard, N.H.: Extracting cellular automaton rules directly from experimental data. Physica D: Nonlinear Phenomena 45(1), 189–202 (1990)

[13] Rosin, P.L.: Image processing using 3-state cellular automata. Comput. Vis. Image Underst. 114(7), 790–802 (2010)

[14] Sapin, E., Bull, L., Adamatzky, A.: Genetic approaches to search for computing patterns in cellular automata. Comp. Intell. Mag. 4(3), 20–28 (2009)

[15] Sapin, E., Bailleux, O., Chabrier, J.J.: Research of a cellular automaton simulating logic gates by evolutionary algorithms. In: Proceedings of the 6th European conference on Genetic programming. pp. 414–423. EuroGP'03, Springer-Verlag, Berlin, Heidelberg (2003)

[16] Sun, X., Rosin, P.L., Martin, R.R.: Fast rule identification and neighborhood selection for cellular automata. Trans. Sys. Man Cyber. Part B 41(3), 749–760 (2011)

[17] Yang, Y., Billings, S.A.: Neighborhood detection and rule selection from cellular automata patterns. Trans. Sys. Man Cyber. Part A 30(6), 840–847 (2000)

[18] Yang, Y., Billings, S.A.: Extracting boolean rules from ca patterns. IEEE Transactions on Systems, Man, and Cybernetics, Part B 30(4), 573–580 (2000)

Analysis of discrete state space partitioned by the attractors of the dynamic network formation game model

Tetsuo IMAI[1,2,†] and Atsushi TANAKA[2,‡]

[1]RIKEN Advanced Institute for Computational Science, Japan
[†]tetsuo.imai@riken.jp
[2]Graduate School of Science and Engineering, Yamagata University, Japan
[‡]tanaka@yamagata-u.ac.jp

Abstract

In recent years much attention are paid to complex network generation models. The dynamic network formation game model (the dynamic NWFG model) was proposed by authors as the promising candidate for generating complex networks. This is an approach for revealing what network topologies tend to be generated by multiple self-interest and distributed designers. Since the dynamic NWFG model can be treated as one of the discrete dynamics systems, it is effective to apply approaches for analyzing Boolean Networks as the method of the analysis of the state space of the dynamic NWFG model. In this article, it is shown that the analysis measures for characterizing the behaviors of random Boolean networks can also be applied to analysis of that of the dynamic NWFG model. In addition we investigate the validity of the Monte Carlo method for estimating features of state space structures of the dynamic NWFG model.

1 Introduction

The network formation game (NWFG) formulated by Jackson and Wolinsky [3] is a model of network formations by multiple agents based on the framework of non-cooperative game in the field of game theory. This is a promising approach for revealing what network topologies tend to be generated by multiple self-interest and distributed designers. With activation of research of a complex network in recent years, much attention are paid to this approach as a model of network formation. In the field of traditional NWFG, most of known results about emergent networks have been limited

to analysis of the behaviors in the case of simple cost parameters or results of small scale simulation. To reveal network topologies which are formed in heterogeneous and large scale situations, more realistic models are required. One promising approach against these limitations is the dynamic network formation game model (the dynamic NWFG model) [1]. It can obtain solutions of the previous (static) NWFG in the lesser computational ability, and it can weight multiple solutions by the occurrence probability. However, structures of state spaces partitioned by many solutions have not been revealed yet. Since this limitation reduces performance of the model for predicting properties of emergent networks formed by many selfish and distributed agents, it is important to reveal these structures of the partitioned state spaces.

Boolean Network (BN) has been analyzed through long term as one of discrete dynamical systems, and it has yielded many results about the model [6, 4]. The BN is the system which consists of the set of Boolean variables which are determined in each discrete time step depending on other Boolean variables of the system. The *Random Boolean Network (RBN)* which is also called as the *N-K model* or the *Kauffman Network*, is a particular model of the Boolean network. Triggered by the epoch-making work of Kauffman [5], the properties of various realizations of RBNs have been investigated. Attracted properties of RBNs are, the number of attractors, the length of attractor's cycle, the size of basin of attractors, transient time (the number of steps the system takes before it falls into an attractor), and properties about stability against perturbations and mutations. Since the dynamic NWFG model can be treated as one of the discrete dynamics systems, we presume that it is effective to apply approaches for analyzing BNs as the method of the analysis of the state space of the dynamic NWFG model.

The first contribution of the article is that it is shown that the analysis measures for characterizing the behaviors of RBN can also be applied to analysis of that of the dynamic NWFG model. The second is that features of the partitioned state space of the dynamic NWFG model have revealed by using approaches for the state space of BNs. It is shown by simulations that a number of attractors with diverse basin size are observed in some settings. The third is that the validity of the Monte Carlo method for estimating features of state space structures of the dynamic NWFG model is investigated.

2 Network Formation Game (NWFG)

2.1 Static Network Formation Game

The traditional NWFG was formulated by Jackson and Wolinsky [3]. In this article, we refer to their model as the *static network formation game* to contrast it with the dynamic variation described in the next subsection. It

is formulated as follows.

Let N be the set of players and n be the size of N respectively, and they can form links among any pair of players. The topology (which is same as *graph* in the graph theory) g is defined by a combination of the set of players N and the set of links $L \subset N \times N$. A link (i, j) is denoted as ij for simplicity. In this article, we consider only undirected topologies. The strategy space of player i is $S_i = 2^{N \setminus \{i\}}$, where $N \setminus \{i\} = \{1, 2, \ldots, i-1, i+1, \ldots, n\}$ and $2^{N \setminus \{i\}}$ is the power set of $N \setminus \{i\}$. If $\boldsymbol{s} \in S_1 \times \cdots \times S_n$ is the profile of strategies played, then link ij forms if and only if both $j \in s_i$ and $i \in s_j$. The outcome network $g(\boldsymbol{s})$ is represented by $g(\boldsymbol{s}) = \{ij | i \in s_j \text{ and } j \in s_i\}$. Instead of Nash equilibrium which is generally utilized as the concept of solution in the game theory, Jackson and Wolinsky [3] proposed the novel stability concept called *pairwise stability* which departs from the notion of Nash equilibrium because of specialty of the NWFG. An topology g is pairwise stable if

$$u_i(g) \geq u_i(g - ij), \quad u_j(g) \geq u_j(g - ij) \quad (\forall ij \in g)$$

and

$$u_i(g + ij) > u_i(g) \Rightarrow u_j(g + ij) < u_j(g) \quad (\forall ij \notin g)$$

where $u_i(g)$ is the payoff for player i in topology g and $g + ij$ is the topology which is added a link ij to topology g, $g - ij$ is the topology which is removed a link ij from topology g. There are two issues in the static NWFG. Firstly, it is difficult to find pairwise stable topologies because the state space is critically huge and efficient methods for finding pairwise stable topologies have not been established yet. Secondly, there are many pairwise stable topologies in general, and we can not identify important ones among these topologies. In the case that the number of players is large, these problems becomes to be especially serious.

2.2 Dynamic Network Formation Game model

The dynamic NWFG model is presented by authors as a deterministic version of introducing dynamicity to the static NWFG [1]. They adopted the concept *defeat* and *improving path* defined by Jackson and Watts, and modified their previous dynamic network formation model to that of behaving deterministically by specifying the most payoff-improving transition among possible transitions.

The model is formulated as a kind of processes of a time series of simultaneous-move game as follows. At each discrete time step t, the game in strategic form determined by the current state (same as topology) $g(t)$ is played and it determines the next state $g(t+1)$. As for the game which played at each time step,*players* are agents who intend to improve their payoffs. The *strategy* of each agent i is indicated as a vector $\mathrm{s}_i(t) = (s_{i1}(t), \ldots, s_{in}(t))$, where $s_{ij}(t) \in \{0, 1\}$. The player i independently sets $s_{ij}(t)$ according to

change of its payoff with change of link ij, $s_{ij}(t)$ is set to 1 if it is desirable to add (or maintain) the link with player j, otherwise $s_{ij}(t)$ is set to 0. s_{ii} is always equal to 0. The *payoff* of the player i is defined by following distance-based payoff function,

$$u_i(g) = \sum_{j \neq i} \delta^{d_{ij}} - \sum_{j \in \{j | ij \in g\}} c_{ij} \tag{1}$$

where $0 \leq \delta \leq 1$ indicates the decay of benefit from connected agent with increase of the distance, d_{ij} is the distance between agent i and j (the number of hops from i to j), and c_{ij} is a link cost to add or maintain the link between i and j. The *outcome* $g(t+1)$ obtained by playing the game at time step t is determined as follows. We describe about two concepts *adjacent* and *defeat*. Two topologies g and g' are *adjacent* if the g' differs only one link from g,and a state g' *defeats* an adjacent state g if either

$$g' = g - ij \text{ and } \big(u_i(g') > u_i(g) \text{ or } u_j(g') > u_j(g)\big) \tag{2}$$

or

$$\begin{aligned} &g' = g + ij \text{ and} \\ &\quad \big\{\big(u_i(g') \geq u_i(g) \text{ and } u_j(g') \geq u_j(g)\big), \\ &\quad \text{except } \big(u_i(g') = u_i(g) \text{ and } u_j(g') = u_j(g)\big)\big\}. \end{aligned} \tag{3}$$

The $g(t+1)$ is specified deterministically among states which can defeats $g(t)$ and $g(t)$ itself. For more concrete description of $g(t+1)$, two definitions $\Delta u_{ij}(t+1)$ and *acceptable link set* $L_{acceptable}(g)$ are described as follows. Firstly, $\Delta u_{ij}(t+1)$ is defined as the amount of change of i's payoff in the case that the change of link ij occurs at time step t,

$$\begin{aligned} &\Delta u_{ij}(t+1) = \\ &\begin{cases} u_i\big(g(t) + ij\big) - u_i\big(g(t)\big), & \text{if} \quad ij \notin g(t) \\ u_i\big(g(t) - ij\big) - u_i\big(g(t)\big), & \text{if} \quad ij \in g(t). \end{cases} \end{aligned} \tag{4}$$

Secondly, $L_{acceptable}(g) \subset L \subset N \times N$ is defined as the set of links which are acceptable for both of player i and player j,

$$\begin{aligned} L_{acceptable}(g) = &\{ij | ij \notin g \text{ and } g + ij \textit{ defeats } g\} \\ &\cup \{ij | ij \in g \text{ and } g - ij \textit{ defeats } g\}. \end{aligned} \tag{5}$$

The link ij which is changed at the game described as

$$ij = \underset{ij \in L_{acceptable}(g(t))}{\operatorname{argmax}} \Delta u_{ij}(t+1) \tag{6}$$

and determines the outcome $g(t+1)$, where $\underset{x \in X}{\operatorname{argmax}} f(x)$ indicates the value $x_{max} \in X$ that maximize $f(x)$. If no links satisfy this condition then $g(t+1)$ is exactly same as $g(t)$, and if there is more than one link satisfying that, the link which involved by the agent who have the youngest ID is prior than others as a matter of convenience. Note that agents decide their strategies at each time step t only to make their own payoffs of the next step $t+1$ be better off without any forecasts. The process starts from the initial state of topology $g(0)$, and it continues until the state converge at a stable state or a part of a cycle which is described as the *attractor* of the process.

It is clear from definitions that a state is pairwise stable if and only if the attractor of the process is consisted of one state. It is the most important property of the dynamic NWFG model. An attractor can be an *attractor cycle* which is consisted by a sequence of adjacent states $\{g_1, g_2, \ldots, g_K\}$ such that each defeats the previous one and $g_1 = g_K$. It is called as *improving cycle* at the context of the NWFG. Main reason of existence of attractor cycle is that agents decide their strategies without any forecasts. The condition of existence of attractor cycle is analyzed by Jackson and Watts [2]. In general, orbits of dynamic systems in continuous space can take a chaotic orbit which has infinite length of period. On the other hand, the dynamic NWFG model converges to either attractive fixed states or attractor cycles. The reason is that since the size of the whole state space is only finite, attractor cycles can also have only finite lengths.

2.3 State space of the dynamic NWFG model

In this subsection the properties of the state space G of the dynamic NWFG model are described. This is a deterministic and synchronized state transition model on discrete state and time spaces. Figure 1 shows an example of the state transitions of the dynamic NWFG model.

The size of G is the number of capable states(topologies) constructed by n agents, $|G| = 2^{\binom{n}{2}}$, therefore by increasing the node size n, $|G|$ increases rapidly. Let the set of initial states converging to an attractor be a *basin of attraction*, it is the general term of the dynamic systems, the state space is partitioned into some mutually exclusive and collectively exhaustive basins for each corresponding attractor in the model. Basins of attractions might take greatly different sizes in the cases of some conditions of payoff functions. It seems to be natural to consider the basin size of an attractor as the importance of the attractor because the basin size of the attractor is proportional to the probability of converging to the attractor in the assumption that all of initial state g_0 have the same probability.

At the last of the section, we refer the significance of applying the analysis method of BNs to analysis for the dynamic NWFG model. Firstly we point out the significance from a standpoint of the NWFG. The dynamic NWFG

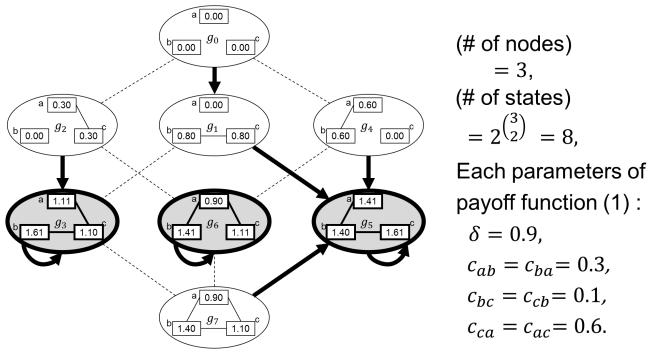
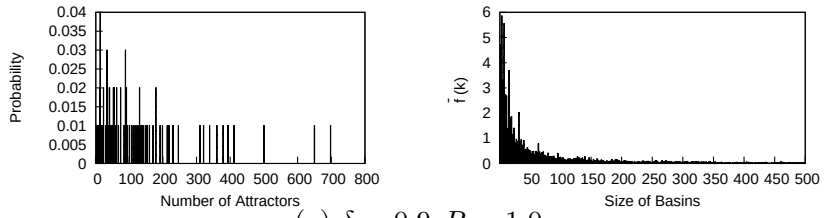


Figure 1: An example of state transitions of the dynamic NWFG model [1]. Each of two adjacent states are linked by dashed lines. Respective values of payoffs are listed in the nodes. There are arrows from a state $g(t)$ to a state $g(t+1)$ which have a relation that it stays at the state $g(t)$ at time step t and it move to the state $g(t+1)$ at next time step $t+1$. These arrows indicate the most payoff improving transition according to the equation (6). There are 3 pairwise stable states of g_3,g_5 and g_6, and no attractor cycles. The sizes of each basins of attraction are respectively 2, 5 and 1.

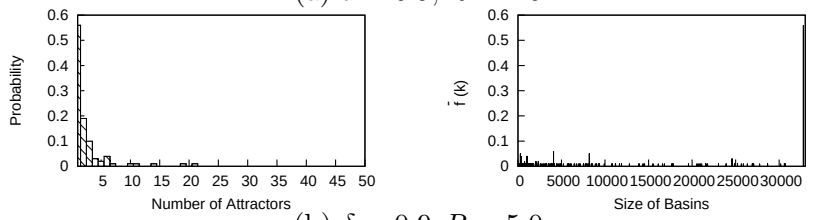
model is clearly a kind of BNs, thus methodology of discrete dynamics for analysis of RBNs is also hopeful for analysis of the dynamic NWFG model. In addition, knowledge of BNs derived by the discrete dynamics methodology might also be valid for the dynamic NWFG model. To be more precise, measures for analyzing RBN dynamics, such as the number of attractors, the size of basin of attraction, the length of cycles, transient times, etc. can also be applied for analyzing the dynamic NWFG model. Secondly we denote the significance of our contributions for the standpoint of RBN analysis. Each state are represented by the bit sequence and by the undirected topology. Thus the dynamic NWFG model (and topology transition models in general) give an additional explanation to a state which has only been a bit sequence. It might provide an additional properties of attracted states which are sets of bit sequences.

3 Investigation of the partitioned state space

As described in the previous section, the decay parameter δ and cost parameters c_{ij} specifies the state space structure. The whole state space might be partitioned into a large number of basin of attraction of multimodal size. These basins of attraction are characterized by measures used by analysis of RBNs, such as the number of basins, the size of basins of attraction, the length of cycles, etc. In this section and the next section, we perform the computer simulations for analyzing the properties of the state space



(a) $\delta = 0.9, R = 1.0$



(b) $\delta = 0.9, R = 5.0$

Figure 2: Results of the average numbers attractors and that of sizes of basins of attraction for each parameters.

constructed by the dynamic NWFG model and for the estimation of the structures. In this section, we investigate the properties of the partitioned state space for each parameter set. Since the exhaustive search in the case of the larger number of nodes becomes to be difficult, we adopt the number of nodes 6 for the exhaustive search in the realistic time, although it is of course very small. The number of states of the whole state space with n nodes is $2^{\binom{n}{2}}$, thus the size of that with 6 nodes is $2^{\binom{6}{2}} = 32768$. The decay parameter δ is set to 0.9 and the meta parameter R which cost parameters c_{ij} are sampled randomly and uniformly in the interval $(0, R]$, is set to 1.0 and 5.0. For each of all the combinations of these parameters, 100 random partition patterns are generated. Although there are no strong meanings in the value of these parameters, we believe that it will be meaningful as fundamental investigation. By simulations with these settings, frequency distributions of the following two measures the *number of basins of attraction* and the *size of basins of attraction* are investigated. The number of basins of attraction is equal to the number of attractions and to the number of cycles, and is evaluated by the average values. The size of basins of attraction is measured by investigating the average frequency distributions over 100 patterns. To be more precise, the average frequency distributions are evaluated by the value of $\bar{f}$ which is the cumulative appearance times of size k divided by the number of patterns. Figure 2 shows results of these measures for each parameter. It is observed that the numbers of attractors can take various values and distributions of the average number of attractors are similar to an exponential function with a negative exponent. Frequencies of appearance of micro size basins are increasing, in the case of R=5.0, a number of micro size basins and an extremely huge size (=32768) basin are both

observed. In the case of $R = 0.1$, extremely micro size (=1) basins appear very frequently. From these results in some cases depending on a parameter, it likely consists that the perfect prediction of basin size is difficult because of the diverse basin sizes.

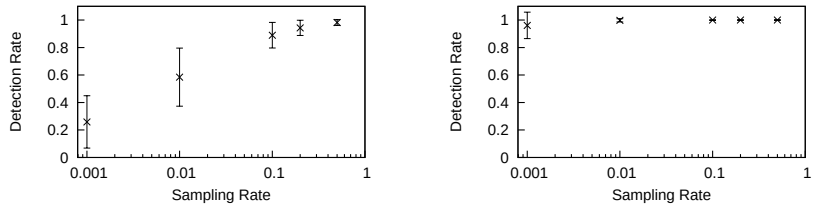
4 Evaluation of the validity of the Monte Carlo simulation

In this section, estimations of the properties of the structures by Monte Carlo simulations are performed and the efficiency of the Monte Carlo simulations is evaluated.

It is already known that the dynamic NWFG model can generate complex networks. For more detailed evaluation of networks generated by the model, it is necessary to perform simulations with 100 or more nodes and to investigate the properties of the results' topologies. However, the size of the state space constituted by such large number of nodes is very huge, because the size of state space is $2^{\binom{n}{2}}$ for the number of nodes n. Thus it is not realistic to investigate the whole state space. Moreover, it seems to be also difficult to describe the system's behaviors by approximate methods. Because processes like RBNs or the dynamics NWFG model which can be modeled as state transitions on discrete space, may converge to a different attractor with the greatly diverse properties by some perturbations.

An promising approach to tackle the problem is investigating the properties for many random samples. This is the estimation of the properties of the whole state space instead of actual investigation of the whole state space. This approach is called as the *Monte Carlo method*.

In this section, the Monte Carlo simulations are performed. In our simulation, the number of nodes is 6, thus the size of the whole state space is 32768. For two parameter sets of $\delta = 0.9, R = 1.0$ and $\delta = 0.9, R = 5.0$, 100 patterns of partitions are generated, and each property of state space is estimated by Monte Carlo simulations for 100 partition patterns. Simulations are performed for each sampling rate 0.001, 0.01, 0.1, 0.2 and 0.5. For each pattern and each sampling rate, Monte Carlo simulation is performed 1000 times. The effectiveness of estimation by Monte Carlo method is evaluated by average values for 1000 trials and the true values obtained by the strict simulation performed in the section 3. The purpose of the simulation is to evaluate the efficiency of the Monte Carlo simulation for revealing the properties of the state space constructed by the dynamic NWFG model. If it becomes clear that the simulation of small samples can fully represent the system-wide process, the property of the topologies formed by a large number of nodes can be revealed by investigating only about small samples. Instead, if the simulation of small samples cannot represent the process efficiently, investigations about results of small samples which could change by



(a) $\delta = 0.9, R = 1.0$ (b) $\delta = 0.9, R = 5.0$

Figure 3: Attractor detection rate by Monte Carlo Simulations for each sampling rate.

some perturbations might not be significant.

Figure 3 shows results of Monte Carlo simulations. It shows the value of (Number of actually detected attractors)/(Total number of attractors). In these figures, average values of each patterns are pointed by x-mark, and standard deviations are indicated by error bars. From these results, in the case that there are large number of attractors with very various sizes (shown by (a)), by Monte Carlo simulations of sampling rate only over 0.1 all of attractors can be detected certainly. It implies that features of attractors of the dynamic NWFG model might not be revealed by a small number of sampling because of the hugeness and diversity of the state space. On the other hand, in the case that diversity of the state space is not so serious (shown by (b)), a small number of Monte Carlo simulations are efficient enough.

In summary, although our investigation is very small scale simulation, it is shown that Monte Carlo simulations of a small number of sampling might not be efficient enough for investigating the features of attractors of the dynamic NWFG model. However, it seems that there is only a method of Monte Carlo simulations of a small number sampling in order to reveal features of the network formation by the dynamic NWFG model with large scale nodes. To tackle the problem, an idea might be efficient that by tracing orbits of a small number of samples and investigating the diversity of attractors to whom these orbits converge, it might be able to estimate the diversity of the size of basin of attraction. It might be able to evaluate the universality of the data obtained by sampled data. It seems to be hopeful because the diversity of partition of the state space and the rate of attractor detection have a positive correlation.

5 Conclusion

In this article, it is shown that the analysis measures for characterizing the behaviors of RBN can also be applied to the analysis of that of the dynamic NWFG model. In addition, features of the partitioned state space of the dynamic NWFG model are revealed by using approaches for the state

space of BNs. It has been shown by simulations that the diverse number of attractors with diverse basin size are observed in some settings. Thirdly we have investigated the validity of the Monte Carlo method for estimating features of state space structures of the dynamic NWFG model. It has been shown that in the case that there are large number of attractors with very various sizes, by Monte Carlo simulations of sampling rate only over 0.1 all of attractors can be detected certainly. It implies that features of attractors of the dynamic NWFG model might not be revealed by a small number of sampling because of the hugeness and diversity of the state space. On the other hand, it has also been shown that in the case that the diversity of the state space is not so serious, a small number of Monte Carlo simulations are efficient enough.

Issues in the future are as follows. The approach used for the analysis of the state space of RBN has been only used for analyzing the state space of the dynamic NWFG model. It might be important to classify the dynamic NWFG model more precisely as a kind of BN.

Acknowledgement

The computation in this work has been done using the facilities of the Institute of Statistical Mathematics in Japan. This research is partly supported by Grant-in-Aid for Scientific Research (C) (25330361).

References

[1] T. IMAI and A. TANAKA. A game theoretic model for as topology formation with the scale-free property. *IEICE Transactions on Information and Systems*, E93.D(11):3051–3058, 2010.

[2] M. O. Jackson and A. Watts. The evolution of social and economic networks. *Journal of Economic Theory*, 106(2):265–295, October 2002.

[3] M. O. Jackson and A. Wolinsky. A strategic model of social and economic networks. *Journal of Economic Theory*, 71(1):44–74, October 1996.

[4] L. Kadanoff, S. Coppersmith, and M. Aldana. Boolean dynamics with random couplings. *eprint arXiv:nlin/0204062*, Apr. 2002.

[5] S. Kauffman. *The Origins of Order: Self Organization and Selection in Evolution*. Oxford University Press, Incorporated, 1993.

[6] J. Schiff. *Cellular Automata: A Discrete View of the World*. Wiley Series in Discrete Mathematics & Optimization. Wiley, 2008.

Surjective Two-Neighbor Cellular Automata on Prime Alphabets*

Jarkko Kari[1] Ville Salo[1,2] Ilkka Törmä[1,2]

[1]University of Turku, Finland

[2]TUCS – Turku Center for Computer Science
{jkari,vosalo,iatorm}@utu.fi

Abstract

In this article, we present a simple proof for the fact that a surjective cellular automaton with neighborhood size 2 on a prime alphabet is permutive in some coordinate. We discuss the optimality of this result, and the existence of non-closing cellular automata of a given neighborhood and alphabet size.

1 Introduction

Cellular automata are topological dynamical systems that have a simple finite definition, and are discrete in both space and time. Formally, they are continuous self-maps of the space of infinite sequences over a finite alphabet. The dymamics of cellular automata can be very complex and unpredictable, and many fundamental questions about their properties are still open. In particular, the dynamics of surjective cellular automata have been extensively studied, but it is still unknown whether they always have a dense set of periodic points.

In this work-in-progress article, we study surjective cellular automata with small local neighborhoods. In particular, we show that if the neighborhood size is 2 and the size of the alphabet is a prime, the automaton is permutive in one of the coordinates. We try to optimize this result as much as possible, using computer searches and general constructions, and obtain an almost complete characterization of those alphabet-neighborhood size pairs which admit a non-closing cellular automaton.

*Research supported by the Academy of Finland Grant 131558

2 Definitions and Preliminaries

Let S be a finite set, called the *alphabet*, and denote by S^* the set of finite words over S. We endow $S^{\mathbb{Z}}$ with the product topology, making it a compact topological space. An element $x = (x_i)_{i\in\mathbb{Z}} \in S^{\mathbb{Z}}$ is called a *configuration*. For $i, j \in \mathbb{Z}$, we denote $x_{[i,j]} = x_i x_{i+1} \cdots x_j$. The *shift map* $\sigma : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$, defined by $\sigma(x)_i = x_{i+1}$, is a homeomorphism from $S^{\mathbb{Z}}$ to itself. A one-directional sequence $x \in S^{\mathbb{N}}$ (or $x \in S^{-\mathbb{N}}$) is *transitive* if every word in S^* occurs in it, and a configuration $x \in S^{\mathbb{Z}}$ is *doubly transitive* if both $x_{(-\infty,0]}$ and $x_{[0\infty)}$ are transitive.

A *cellular automaton* (CA for short) is a continuous function $f : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ that commutes with the shift: $f \circ \sigma = \sigma \circ f$. It is known [2] that a cellular automaton is given by a *local rule* $F : S^{\ell+r+1} \to S$ for some $\ell, r \in \mathbb{N}$ by $f(x)_i = F(x_{[i-\ell,i+r]})$. The numbers ℓ and r are called *left and right radii* of f, respectively. If we can take $\ell = 0$ and $r = 1$, we say f is a *radius-$\frac{1}{2}$* CA. We say f is *permutive* in a coordinate $i \in [-\ell, r]$, if fixing a word $w \in S^{\ell+r+1}$ and permuting the letter w_i also permutes the image $F(w)$. If f is permutive in the leftmost (rightmost) coordinate of its neighborhood, we say it is left (right, respectively) permutive. We say f is *left (right) closing*, if $x_{[0,\infty)} = y_{[0,\infty)}$ ($x_{(\infty,0]} = y_{(\infty,0]}$, respectively) and $f(x) = f(y)$ implies $x = y$ for all $x, y \in S^{\mathbb{Z}}$. Finally, f is *preinjective* if $f(x) = f(y)$ and $x \neq y$ imply that $x_i \neq y_i$ for infinitely many $i \in \mathbb{Z}$.

The *image graph* of a CA $f : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ with radii ℓ and r is the labeled graph with vertex set $S^{\ell+r}$, and an edge from v to w with label $s \in S$ if and only if $v_{[1,\ell+r-1]} = w_{[0,\ell+r-2]}$ and $f(v_0 w) = s$. The *determinization* of said graph is given by the standard subset construction. Note that f is surjective iff every $u \in S^*$ occurs as the label of a path in the image graph of f.

The following result is classical, and can be found, for example, in [2] or [4].

Lemma 1. *Let $f : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ be a surjective CA with left and right radii ℓ and r, respectively. Then $|f^{-1}(x)|$ is finite and bounded, and f is preinjective. If $x \in S^{\mathbb{Z}}$ is doubly transitive and $a \neq b \in f^{-1}(x)$, then a and b are also doubly transitive and $a_{[i,i+\ell+r-1]} \neq b_{[i,i+\ell+r-1]}$ for all $i \in \mathbb{Z}$. Moreover, there exists $M(f) \in \mathbb{N}$ such that $|f^{-1}(x)| = M(f)$ for every doubly transitive $x \in S^{\mathbb{Z}}$.*

3 Permutivity of Radius-$\frac{1}{2}$ Automata

For this section, we fix a surjective CA $f : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ with left and right radii ℓ and r, and denote $|S| = n$. When $x \in S^{\mathbb{N}}$, we write

$$f^{-1}(x) = \{y \in S^{\mathbb{N}} \mid \forall i \in \mathbb{N} : f(^{\infty}0.y)_{\ell+i} = x_i\},$$

and symmetrically for $y \in S^{-\mathbb{N}}$. When $x \in S^{\mathbb{N}}$ ($y \in S^{-\mathbb{N}}$) is transitive, we write $L(f, x) = |f^{-1}(x)|$ ($R(f, y) = |f^{-1}(y)|$, respectively). Denote also

$M = M(f)$.

The following results (up to but not including Proposition 1) seem classical, but we have not been able to find a reference where they are presented in this form. Closely related results can be found in sections 14 and 15 of [2]. To our knowledge, Proposition 1 is a new result.

Lemma 2. *Both L and R are functions of f, ℓ and r only, and do not depend on the chosen transitive points. Writing $L(f)$ and $R(f)$ for these functions, we have $Mn^{\ell+r} = L(f)R(f)$.*

Proof. Let $x \in S^{-\mathbb{N}}$ and $y \in S^{\mathbb{N}}$ be transitive, and consider the set of points $A = xS^{\ell+r}y$. We have $|A| = n^{\ell+r}$, and since every element of A is doubly transitive, $|f^{-1}(A)| = Mn^{\ell+r}$. On the other hand, clearly $|f^{-1}(A)| = R(f,x)L(f,y)$. In particular, L and R are functions of f, ℓ and r only. □

Lemma 3. *Writing $L(f)$ and $R(f)$ as before, the functions $f \mapsto M(f)$, $f \mapsto L'(f) = \frac{L(f)}{n^\ell}$ and $f \mapsto R'(f) = \frac{R(f)}{n^r}$ are homomorphisms from the set of surjective CA on S to $\mathbb{Q}$ (and do not depend on the choice of ℓ and r), and $M = L'R'$.*

Proof. It is clear that L' and R' do not depend on the choice of ℓ and r, as increasing ℓ effectively multiplies L by n, and similarly for r and R. We then prove that L' is a homomorphism, the case of R' being symmetric. Let $g : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ be a surjective CA with left radius ℓ'. Let $x \in S^{\mathbb{N}}$ be transitive, so that $g^{-1}(x)$ is also transitive. Taking the left radius $\ell + \ell'$ for $f \circ g$, we then have $L(f \circ g) = |f^{-1}(g^{-1}(x))| = L(f)L(g)$. It follows that L' is a homomorphism, since it does not depend on the choice of the radii. □

From now on, denote $R = R(f)$ and $L = L(f)$, since f and its radii are fixed.

Lemma 4. *The inequalities $M \leq L, R \leq n^{\ell+r}$ hold.*

Proof. Let $x \in S^{\mathbb{Z}}$ be a configuration, and denote $|f^{-1}(x)| = k$. Then there exists $i \in \mathbb{Z}$ such that already $|f^{-1}(x_{[i,\infty)})| \geq k$. The inequalities $M \leq L$ and $M \leq R$ follow. The upper bounds follow from the equality $Mn^{\ell+r} = LR$ of Lemma 2. □

Lemma 5. *Let $x \in S^{\mathbb{N}}$ be transitive. If $a \neq b \in f^{-1}(x)$, then a and b differ in their $(r+\ell)$-prefix.*

Proof. Suppose not, let $y \in S^{-\mathbb{N}}$ be transitive and $c \in f^{-1}(y)$. Then $f(c.a) = f(c.b)$ is doubly transitive, but has two distinct preimages that agree on the interval $[0, \ell + r - 1]$, a contradiction by Lemma 1. □

By the above lemma, the numbers L and R are generalizations of the left and right Welch indices of reversible cellular automata, respectively, as defined in [1] (among others).

Proposition 1. *If $n = |S|$ is prime and f is a surjective radius-$\frac{1}{2}$ CA, then f is left or right permutive.*

Proof. Because $LR = Mn$, n is prime and $L, R \leq n$, we must have either $L = n$ or $R = n$. Without loss of generality we assume the former. Now, if $x \in S^{\mathbb{N}}$ is transitive and $s \in S$, every $s' \in S$ occurs as the leftmost symbol of a preimage of sx by Lemma 5. This means that for all $s, s' \in S$, there exists $s'' \in S$ such that $f(s', s'') = s$, and thus f is right permutive. □

This proposition can be useful when trying to construct surjective cellular automata with some specific nontrivial properties, as it shows that radius-$\frac{1}{2}$ CA on prime alphabets are quite simple, and thus trivially possess or lack many interesting properties.

4 Other Neighborhoods and Alphabet Sizes

In this section, we study the case of non-prime alphabets and larger neighborhoods in the hope of generalizing Proposition 1. We proceed in the following two directions:

1. We study whether complicated surjective CA could somehow be 'decomposed' into more primitive components, preferably permutive CA.

2. With the help of computer searches, we try to enumerate the combinations of alphabet and neighborhood sizes in which all cellular automata must be permutive or closing in either direction.

In the first direction, we begin with the following definition.

Definition 1. *Let $n = k_1 k_2$, where $k_i \geq 2$. We say f is* track-reducible *if there exist two sets S_1 and S_2 with $|S_i| = k_i$, cellular automata g on S_1 and h on $S_1 \times S_2$ with $g \circ \pi_1 = \pi_1 \circ h$, and bijections $\alpha : S \to S_1 \times S_2$ and $\beta : S_1 \times S_2 \to S$ such that $f = \beta \circ h \circ \alpha$, where α and β are used as radius-0 cellular automata.*

Surjective track-reducible automata are easy to construct inductively as follows. First, let g be any surjective (perhaps track-reducible) CA in S_1. Then we can define h by choosing a coordinate $i \in \mathbb{Z}$ on the S_2-track and a neighborhood $N \subset \mathbb{Z}$ on the S_1-track, and for each $w \in S_1^N$, a CA h_w on S_2 that is left (or right) permutive in the coordinate i. Such a CA is always surjective, but if the direction of permutivity or the coordinate i is varied, it is not permutive. In the course of this section, however, we shall see that not all surjective cellular automata on non-prime alphabets are track-reducible.

For the second direction, we begin with the radius-$\frac{1}{2}$ case, and show that here Proposition 1 is optimal.

Proposition 2. *Let S be an alphabet. Then all surjective radius-$\frac{1}{2}$ CA on S are closing in either direction iff $|S|$ is prime.*

Proof. First, if $|S|$ is prime, then by Proposition 1, every radius-$\frac{1}{2}$ CA on S is permutive in either coordinate, thus closing.

Next, let $n \geq 3$ be arbitrary. We construct a left permutive radius-$\frac{1}{2}$ CA f_n on $\{0, \ldots, n-1\}$ which is not right closing. The local rules for f_n are $f_n(a, b) = a + b \bmod 2$ and $f_n(c, b) = c$ for all $a \in \{0, 1\}$, $b \in \{0, \ldots, n-1\}$ and $c \in \{2, \ldots, n-1\}$. Now f_n is left permutive, but the point ${}^{\infty}2.0^{\infty}$ has preimages ${}^{\infty}2.0^{\infty}$ and ${}^{\infty}2.1^{\infty}$, so f_n is not right closing. Define also $g_n(a, b) = f_n(b, a)$, so that g_n is right permutive and not left closing.

Suppose $|S| = k_1 k_2$, where $k_i \geq 3$. The cellular automaton $f_{k_1} \times g_{k_2}$ on $\{0, \ldots, k_1 - 1\} \times \{0, \ldots, k_2 - 1\}$ is surjective, but not closing in either direction. Interpreting the state set as S, the claim follows for such S.

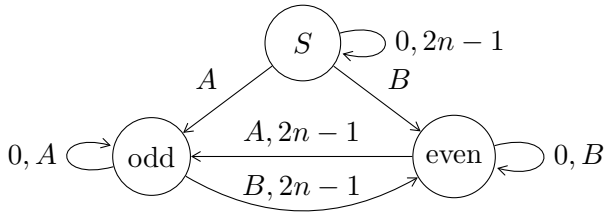
Finally, let $|S| = 2n$ for $n \in \mathbb{N}$, and assume $S = \{0, \ldots, 2n-1\}$. We construct a surjective radius-$\frac{1}{2}$ CA f on S which is not closing. First, define $A = \{1, 3, \ldots, 2n-3\}$ and $B = \{2, 4, \ldots, 2n-2\}$, so that A, B, $\{0\}$ and $\{2n-1\}$ form a partition of S. Define then f by

(a, b)	(s, s)	$(k-1, o)$	$(o, o+2k)$
$f(a, b)$	0	$k \in A$	

(a, b)	$(k-1, e)$	$(e, e+k)$	$(2n-2, o)$	$(2n-1, e)$
$f(a, b)$	$k \in B$		$2n-1$	

where $s \in S$, $o \in A \cup \{2n-1\}$ and $e \in \{0\} \cup B$ are arbitrary, and the sums are taken modulo $2n$. This is a generalization of the automaton defined in [3, Example 15].

The determinization of the image graph of f is



and it shows that f is surjective. However, the point ${}^{\infty}0.1^{\infty}$ has the three preimages ${}^{\infty}0.135\cdots$, ${}^{\infty}0.357\cdots$ and ${}^{\infty}1.357\cdots$ which show that f is neither right nor left closing. □

A simple computer search also shows that the automaton f constructed in the case $|S| = 4$ (the original example of [3]) is not track-reducible. Thus track-reducibility is not general enough to capture all radius-$\frac{1}{2}$ surjective cellular automata on composite alphabets.

For larger neighborhood sizes, we have the following.

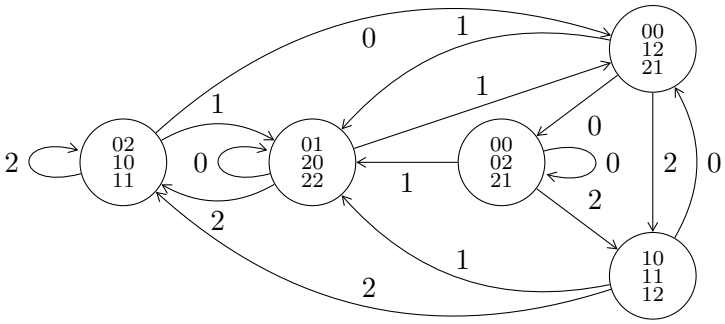
Example 1 (A non-closing surjective radius-1 CA with alphabet size at least 5)**.** *Let $n \geq 3$ and $S = \{(0,0),(0,1),(1,0),(1,1)\} \cup \{(i,i) \mid 2 \leq i < n\}$, and let f be the three-neighbor CA that simulates the automaton f_n (from the proof of Proposition 2) on the first track using the cells 0 and 1, and g_n on the second track using the cells -1 and 0. Then f is clearly surjective, but not closing in either direction.*

We show a three-state three-neighbor CA which is surjective, but not closing in either direction. This automaton was found by a computer search.

Example 2 (A non-closing surjective radius-1 CA with alphabet size 3)**.** *Consider the CA whose local rule is given by*

	00	01	02	10	11	12	20	21	22
0	0	1	0	2	2	1	1	0	1
1	0	1	2	2	2	0	1	0	1
2	1	0	2	2	2	2	0	1	0

It is surjective, since its determinized image graph contains the subgraph



Consider the points ${}^{\infty}0.1^{\infty}$ and ${}^{\infty}012.110^{\infty}$. The former has the preimages ${}^{\infty}0.(2001)^{\infty}$ and ${}^{\infty}0.(0120)^{\infty}$, while the latter has ${}^{\infty}020.20^{\infty}$ and ${}^{\infty}21.20^{\infty}$. This shows that the CA is neither left nor right closing.

Finally, we show the existence of a binary CA which is not closing in either direction. The construction is almost surely not optimal with respect to neighborhood size.

Example 3 (A non-closing surjective binary CA with neighborhood size 11)**.** *Consider the CA f on $\{0,1\}$ with neighborhood $\{-5,\dots,5\}$ defined by the rules $f(ab001cd001e) = c + e$ and $f(a001bc001de) = a + c$ for all $a,b,c,d,e \in \{0,1\}$, and $f(w) = w_5$ for all other $w \in \{0,1\}^{11}$. It is easy to check that given a configuration $x \in \{0,1\}^{\mathbb{Z}}$, we have $x_{[0,2]} = 001$ iff $f(x)_{[0,2]} = 001$.*

Consider a maximal subword w of the form $a_1b_1001a_2b_2001\cdots001a_nb_n$. A preimage for w is $a'_1b'_1001a'_2b'_2001\cdots001a'_{n-1}b'_{n-1}001a'_nb'_n$, where $a'_i, b'_1 \in$

Table 1: A table of closingness. The letter n denotes the size of the state set, while m is the size of the neighborhood. The label 'per' refers to left or right permutivity, 'clo' to left or right closingness, '?' to unknown and '-' to not necessarily closing. On each row, the leftmost '-' has been replaced by the number of the result that proves this fact.

$n \setminus m$	2	3	4	5	...	10	11
2	per	per	clo	?	...	?	Ex. 3
3	per	Ex. 2	-	-	...	-	-
4	Prop. 2	-	-	-	...	-	-
5	per	Ex. 1	-	-	...	-	-
6	Prop. 2	-	-	-	...	-	-
7	per	Ex. 1	-	-	...	-	-
8	Prop. 2	-	-	-	...	-	-
9	Prop. 2	-	-	-	...	-	-

$\{0,1\}$ are given by $a'_1 = a_1$, $b'_1 = b_1$, $a'_n = a_n$, $b'_n = b_n$, and $a'_i = a'_{i+1} + a_i \bmod 2$ and $b'_i = b'_{i-1} + b_i \bmod 2$ for all $i \in [2, n-1]$. The preimages of other patterns are given by themselves, since cells can only change their state when surrounded by the 001-patterns in the specific way. Thus f is surjective. Also, f simulates the $n = 3$ automaton from Example 1 via the substitution $(a, b) \mapsto 001ab$ for $a, b \in \{0, 1\}$ and $(2, 2) \mapsto 001000001$, and it is then easy to see that it is not closing in either direction.

Using a computer search, we have verified that all binary surjective CA with neighborhood size 4 are right or left closing. Also, all surjective elementary CA are left or right permutive (with the minimal neighborhood). The following cases are thus left open:

Question 1. *For $5 \leq m \leq 10$, does there exist a surjective binary CA of neighborhood size m which is not closing in either direction?*

References

[1] Eugen Czeizler and Jarkko Kari. A tight linear bound on the neighborhood of inverse cellular automata. In *Automata, languages and programming*, volume 3580 of *Lecture Notes in Comput. Sci.*, pages 410–420. Springer, Berlin, 2005.

[2] Gustav A. Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Math. Systems Theory*, 3:320–375, 1969.

[3] Petr Kurka. Topological dynamics of cellular automata. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 9246–9268. Springer, 2009.

[4] Douglas Lind and Brian Marcus. *An introduction to symbolic dynamics and coding*. Cambridge University Press, Cambridge, 1995.

Two transitive cellular automata and their strictly temporally periodic points *

Jarkko Kari[1], Kuize Zhang[2,1]
1. Department of Mathematics and Statistics, University of Turku, FI-20014, Finland
jkari@utu.fi
2. College of Automation, Harbin Engineering University, Harbin, 150001, PR China
zkz0017@163.com

Abstract

In this paper, we give two one-dimensional, reversible and mixing cellular automata whose sets of strictly temporally periodic (STP) points are (i) neither dense nor empty, and (ii) dense, respectively. The first example answers two questions proposed in AUTOMATA 2012: there are surjective cellular automata whose STP points are neither dense nor empty, and there are Devaney-chaotic cellular automata with STP points.

Keywords: cellular automaton, symbolic dynamics, spatially and temporally periodic configuration, Devaney-chaos

1 Introduction

A cellular automaton (CA) is a discrete dynamical system studied in computability theory, mathematics, physics, complexity science, theoretical biology, cryptology and microstructure modeling. The property of global functions, limit behavior, classification, topological structure, computational universality, reversibility and conservation law, etc, are important research issues [1].

In [2, 3], the sets of strictly temporally periodic (STP) points (configurations) in surjective CAs were studied. A configuration is called strictly temporally periodic if it is periodic in time but not periodic in space. It was proved in [2] that the set of STP points is residual for an equicontinuous surjective CA, dense for an almost equicontinuous surjective CA, and empty for a positively expansive CA, respectively. It was also shown that in the class of additive surjective CAs, the set of STP points can be either dense or empty, the latter happens if and only if the CA is topologically transitive. That is to say, for surjective CAs belonging to a certain class, the size of the set of STP points is inversely related to be the dynamical complexity of the class. Then a question was asked whether there exists

*Research supported by the Academy of Finland Grant 131558

a surjective CA whose set of STP points is neither empty nor dense. It was also asked whether one can restate the Devaney-chaos in terms of STP points, that is, whether chaotic CAs are precisely those surjective CAs that have no STP points. The investigation was continued in [3] where it was proved that the set of STP points has strictly positive measure if and only if the CA is equicontinuous.

In this paper, we answer questions asked in [2] by providing two one-dimensional (1-D), reversible and Devaney-chaotic CAs whose sets of STP points are (i) neither dense nor empty, and (ii) dense, respectively.

2 Notations

We only consider 1-D CAs. A 1-D CA is a discrete-time dynamical system defined as

$$A = (\mathbb{Z}, S, N, f_A), \tag{1}$$

where $\mathbb{Z}$ denotes the set of integers (cells); S is a finite set that has at least two elements, called the state set; $N = (n_1, \ldots, n_m)$ is a nonempty finite ordered subset of $\mathbb{Z}$ consisting of m distinct integers, called the neighborhood; $f_A : S^m \to S$ is a mapping, called the local rule. A configuration is a mapping $c : \mathbb{Z} \to S$. Let $S^{\mathbb{Z}}$ denote the set of all configurations over S. For $i < j$ we denote by $c[i \ldots j]$ the word $c(i)c(i+1)\ldots c(j) \in S^{j-i+1}$.

The function $F_A : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ defined by

$$F_A(c)(i) = f_A(c(i+n_1), \ldots, c(i+n_m)) \text{ for all } i \in \mathbb{Z} \tag{2}$$

is called the global function of CA A. Any initial configuration evolves under iterations of the global function (2).

A metric d on $S^{\mathbb{Z}}$ is defined for $x, y \in S^{\mathbb{Z}}$ by

$$d(x,y) = \begin{cases} 2^{-\min\{|i| | x(i) \neq y(i)\}}, & \text{if } x \neq y, \\ 0, & \text{if } x = y. \end{cases} \tag{3}$$

The metric defines a topology on $S^{\mathbb{Z}}$ that is compact, totally disconnected and perfect. Any word $w \in S^+ = \cup_{i=1}^{\infty} S^i$ and $n \in \mathbb{Z}$ defines the cylinder

$$[w]_n = \{c \in S^{\mathbb{Z}} | c[n \ldots n + |w| - 1] = w\}$$

of all configurations with word w starting in position n. The cylinders are open and closed (clopen), and are well known to form a basis of the topology.

A translation $\tau_i : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ is a function defined by an integer i: for any configuration $c \in S^{\mathbb{Z}}$, for any integer $j \in \mathbb{Z}$,

$$\tau_i(c)(j) = c(j+i).$$

$\tau_1 := \sigma$ is called shift. Translation τ_i is non-trivial if $i \neq 0$.

It is well known that a function $G : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ is a global function of a CA (a CA function) if and only if G is continuous and commutes with the shift [4]. A CA A is said to be injective (surjective), if F_A is injective (surjective, respectively). Every injective CA is automatically surjective and the inverse function F_A^{-1} is also a CA function. For this reason injective CAs are also called reversible.

Let $q \in S$. A configuration c is said to be q-finite (q-uniform), if only a finite number of cells are not in state q (if each cell is in state q, respectively). A configuration c is said to be spatially periodic if $\tau(c) = c$ for some non-trivial translation τ. The set of spatially periodic configurations is dense in $S^{\mathbb{Z}}$. For a CA A, a configuration c is said to be temporally periodic, if $c = F_A^p(c)$ for some positive integer p. For every reversible CA, each spatially periodic configuration is temporally periodic. Then the set of temporally periodic configurations of every reversible CA is dense in $S^{\mathbb{Z}}$. A configuration c is said to be strictly temporally periodic (STP) for CA A, if it is temporally periodic for CA A but not spatially periodic.

A CA A is (topologically) transitive if for any nonempty open subsets U and V of $S^{\mathbb{Z}}$, there exists a nonnegative integer p such that $F_A^p(U) \cap V \neq \emptyset$. A CA A is called (topologically) mixing if for any nonempty open subsets U and V of $S^{\mathbb{Z}}$, there exists a nonnegative integer p such that $F_A^q(U) \cap V \neq \emptyset$ for all $q \geq p$. By the definitions, mixing CAs are transitive. As cylinders are a basis of the topology, the transitivity of CA A can be equivalently defined using words: A CA is transitive if and only if for all $l \geq 1$ and for all words $u, v \in S^l$, there exists configuration $c \in [u]_0$ such that $F_A^p(c) \in [v]_0$ for some nonnegative integer p. The property of being mixing can be characterized analogously using words.

A CA A is said to be sensitive to initial conditions if there exists a positive number ϵ such that for any configuration c and for any positive number δ, there exist $y \in B_\delta(x)$ and a positive integer p such that $d(F_A^p(y), F_A^p(x)) > \epsilon$. Every transitive CA is known to be sensitive [5]. A CA A is said to be chaotic in the sense of Devaney (Devaney-chaotic) [6] if

(i) it is transitive,

(ii) the set of temporally periodic configurations is dense in $S^{\mathbb{Z}}$, and

(iii) it is sensitive to initial conditions.

For general discrete dynamical systems, (i) and (ii) imply (iii) [7]. For cellular automata, (i) implies (iii) [5]. For reversible CAs, (ii) always holds so a reversible CA is Devaney-chaotic if and only if it is transitive. It is a well-known open problem whether (ii) holds for all surjective CA, or even whether (i) implies (ii).

3 Our examples

3.1 A 1-D mixing, reversible CA whose set of STP points is neither empty nor dense

Consider first the 1-D CA $A = (\mathbb{Z}, S, N, f_A)$ where S consists of the 5 states shown in Fig. 1, $N = (-1, 0, 1)$, and the local rule is defined as follows: for any configuration $c \in S^{\mathbb{Z}}$, and for any cell $i \in \mathbb{Z}$,

- $F_A(c)(i)$ is in state WALL if and only if $c(i)$ is in state WALL ,
- $F_A(c)(i)$ has a left arrow if and only if $c(i)$ has a right arrow and $c(i+1)$ is in state WALL , or $c(i)$ is not in state WALL and $c(i+1)$ has a left arrow,
- $F_A(c)(i)$ has a right arrow if and only if $c(i)$ has a left arrow and $c(i-1)$ is in state WALL , or $c(i)$ is not in state WALL and $c(i-1)$ has a right arrow.

Figure 1: The states of the CA F_A in Section 3.1, where the first state is called BLANK , and the second state is called WALL . The last two states contain a right arrow, and the third and the last states contain a left arrow.

The rule is very simple: a WALL remains stationary and is a 1-blocking word. Arrows are signals that bounce from the walls. From the local rule, one easily sees that in any space-time diagram the arrows can move backward uniquely as time decreases. That is, this CA is reversible. Since an r-blocking word exists, where $r = 1$ is the neighborhood radius of the CA, the CA is not sensitive, so it is almost equicontinuous [8].

Denote the shift CA with state set S by σ. Next we prove that CA $F_A \circ \sigma := F$ is a mixing and reversible CA whose set of STP configurations is neither empty nor dense in $S^{\mathbb{Z}}$. In F the stationary walls have become left moving signals, while the right (left) arrows are stationary signals (signals of speed 2 to the left, respectively). See Fig. 2 for an example of a space-time diagram.

First, the CA F is easily seen to be mixing. In fact, the composition of any surjective, almost equicontinuous CA with a non-trivial translation is mixing (Theorem 2 in [9]).

Next we observe that the set of STP configurations of CA F is not empty. Indeed, any configuration containing only BLANK states and right arrows (which are stationary signals in F) is a fixed point of F. There are uncountably many choices of such configurations that are not spatially periodic.

Finally we show that the set of STP configurations are not dense by proving that any temporally periodic configuration that contains a cell in state WALL is spatially periodic. Consider configuration $e \in S^{\mathbb{Z}}$ that contains a WALL such that

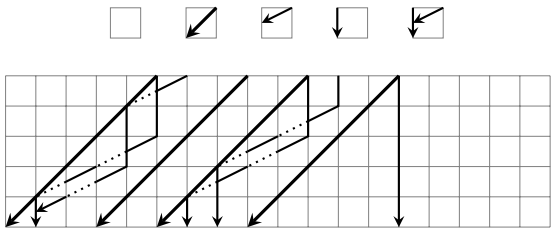


Figure 2: The top five squares are the states of the CA F in Section 3.1. Note that these states are essentially the same as those in Fig. 1, respectively. The bottom grid shows a subblock of a space-time diagram of the CA F in Section 3.1. The first line denotes the initial configuration, where all cells except for the middle ten are in state BLANK . The dotted segments are not part of the states, but are included to indicate how the left arrows become signals of speed 2 to the left.

$F^t(e) = e$ for some $t > 0$. We might as well assume that $e(0)$ is in state WALL . The WALL moves left with speed one so, due to temporal period t, we have $e(tk) =$ WALL for all $k \in \mathbb{Z}$. Now consider the non-shifted original CA A: let us show that e is temporally periodic for F_A also. For all integers k, the arrows in $e[tk+1 \dots tk+t-1]$ will always stay inside the segment $[tk+1, tk+2, \dots, tk+t-1]$ bordered by states WALL when configuration e evolves according to F_A. Hence their pattern repeats periodically: for each k there exists $1 \leq s \leq 5^{t-1}$ such that $F_A^{js}(e)[tk+1 \dots tk+t-1] = e[tk+1 \dots tk+t-1]$ for all $j \in \mathbb{Z}$. Denote $T = (5^{t-1})!$. For all integers k we have $F_A^T(e)[tk+1 \dots tk+t-1] = e[tk+1 \dots tk+t-1]$, that is, $F_A^T(e) = e$.

We have shown that e is temporally periodic for $F = F_A \circ \sigma$ and F_A. It follows that e is spatially periodic:

$$e = F^{tT}(e) = \sigma^{tT}(F_A^{tT}(e)) = \sigma^{tT}(e).$$

3.2 A 1-D mixing, reversible CA whose set of STP points is dense

The example above shows that there is a chaotic CA whose set of STP points is not empty. One could still wonder whether Devaney-chaotic CAs would always have non-dense sets of STP points. The example of this section shows that this is not the case.

Figure 3: The states of the CA in Section 3.2, where the first state is called BLANK . The arrows are colored red and green.

Consider CA $A = (\mathbb{Z}, S, N, f_A)$ where S consists of the 9 states shown in Fig. 3, $N = (-1, 0, 1)$, and the local rule is defined as follows: in the space-time

diagram, each corner is exactly one of the 9 figures shown in Fig. 4. In other words, there are left and right moving signals, colored red and green, that bounce upon collisions.

Figure 4: The local rule of the CA in Section 3.2.

From the local rule, one easily sees that in any space-time diagram the arrows can move backward uniquely as time decreases. Hence this CA is reversible. One easily sees the following properties hold for any BLANK -finite configuration c:

1. the order of the colors of the arrows remains the same at all times (but the directions may change),
2. the number of left (right) arrows remains unchanged over time,
3. for all positive integers m holds: for all sufficiently large positive integers p we have $F_A^p(c)(i)$ is in state BLANK for all $|i| \leq m$, $F_A^p(c)(i)$ is either in state BLANK or only contains a left arrow for all $i < -m$, $F_A^p(c)(i)$ is either in state BLANK or only contains a right arrow for all $i > m$ (see Fig. 5.),
4. for all positive integers m holds: for all sufficiently large positive integers p we have $F_A^{-p}(c)(i)$ is in state BLANK for all $|i| \leq m$, $F_A^{-p}(c)(i)$ is either in state BLANK or only contains a right arrow for all $i < -m$, $F_A^{-p}(c)(i)$ is either in state BLANK or only contains a left arrow for all $i > m$.

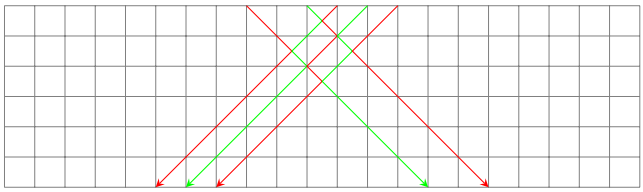


Figure 5: A subblock of a space-time diagram of the CA in Section 3.2. The first line denotes the initial configuration, where all cells except for the middle five are in state BLANK . After a finite time, all left arrows are on the left of all the right arrows.

Next we prove that the CA A is mixing, and hence Devaney-chaotic. Choose arbitrarily two words $u_1, u_2 \in S^l$, where $l \geq 1$. Denote the numbers of left arrows, right arrows of u_1, the numbers of left arrows, right arrows of u_2 by l_1, r_1, l_2, r_2, respectively. Let $w_2 = s_1 s_2 \dots s_m \in \{\text{RED}, \text{GREEN}\}^m$ be the sequence of colors of the arrows in u_2 from left to right, where $m = l_2 + r_2$, and let $w_1 \in \{\text{RED}, \text{GREEN}\}^{l_1+r_1}$ be the analogous sequence of colors in u_1.

Construct first the configuration $c_1 \in S^{\mathbb{Z}}$ such that $c_1[0 \dots l-1] = u_1$, and $c_1[-(l_1+l_2) \cdots -1]$ is a sequence of l_1+l_2 right arrows (of any colors). All $c_1(i)$ for i outside of interval $[-(l_1+l_2), \dots, l-1]$ are in state BLANK . Next construct the configuration $c_2 \in S^{\mathbb{Z}}$ such that $c_2[0 \dots l-1] = u_2$, and for all $i < 0$ or $i > l-1$, $c_2(i)$ is in state BLANK . By properties 3 and 4 above, for all sufficiently large positive integers p all $F_A^p(c_1)(i)$ are in state BLANK for $i \in [0, \dots, l-1]$, and all $F_A^{-p}(c_2)(i)$ are in state BLANK for $i \in [-(l_1+l_2), \dots, l-1]$. Consider arbitrary such p, and note that for all $i \in \mathbb{Z}$ either $c_1(i)$ or $F_A^{-p}(c_2)(i)$ is BLANK . Merge c_1 and $F_A^{-p}(c_2)$ into a single configuration $c \in S^{\mathbb{Z}}$ as follows:

$$c(i) = \begin{cases} c_1(i), & \text{if } i \in [-(l_1+l_2), \dots, l-1], \\ F_A^{-p}(c_2)(i), & \text{otherwise.} \end{cases}$$

Ignoring the colors of the arrows, we have that c and $F_A^p(c)$ have in domain $[0, \dots, l-1]$ the patterns of arrows of u_1 and u_2, respectively.

On the left of cell 0, c has exactly $l_1+l_2+r_2$ right arrows and no left arrows. Configuration $F_A^p(c)$ has on the left of 0 exactly l_1 left arrows and no right arrows. Let us recolor the arrows in c so that the leftmost l_1 arrows get arbitrary colors, the following l_2+r_2 colors read word w_2 and the colors of the next l_1+r_1 arrows (which are in positions $[0, \dots, l-1]$) read word w_1. The last l_2 arrows can again be colored arbitrarily. We then have that $c \in [u_1]_0$. By property 1 the sequence of colors is the same in $F_A^p(c)$, which means that $F_A^p(c) \in [u_2]_0$. See Fig. 6 for an illustration.

Since p was an arbitrary sufficiently large number we get that CA A is mixing.

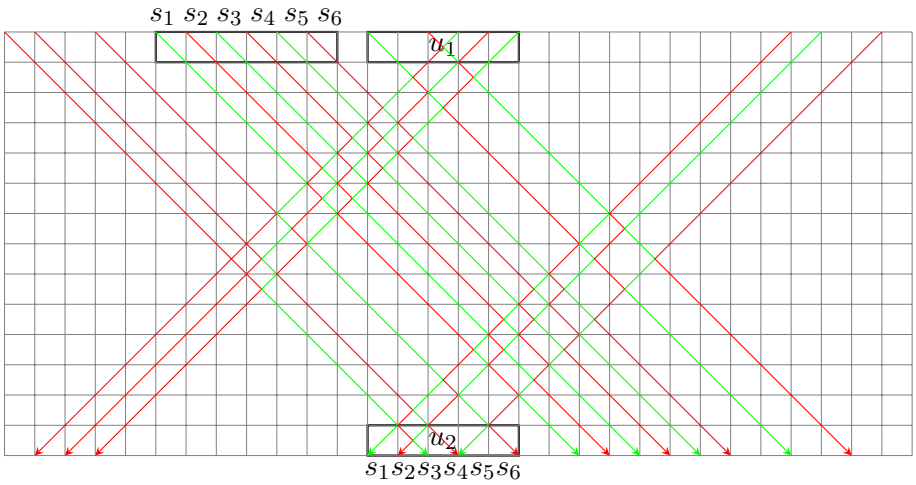


Figure 6: Transitivity of cylinder $[u_1]_0$ to cylinder $[u_2]_0$ in the CA of Section 3.2. The first line denotes the initial configuration, where all cells except for the middle thirty are in state BLANK .

At last, we prove the set of STP configurations of CA A is dense in $S^{\mathbb{Z}}$. That is to say, for any word $u \in S^l$, where $l \geq 1$, we can construct an STP configuration that contains u as a subpattern.

For any word $u \in S^l$ that is not BLANK -uniform, where $l \geq 1$, we construct configuration c as follows: $c[0 \dots l-1] = u$; $c[l \dots 2l-1] = u^R$, where u^R is the mirror image of word u. For all nonzero k, we set $c[2kl \dots 2kl + 2l - 1] = c[0 \dots 2l-1]$, so the configuration is spatially periodic with period $2l$. Word u and its mirror image u^R are alternatingly repeated in c. Next we recolor c by replacing all green arrows to the left of cell 0 by red ones, and by replacing all red arrows to the right of cell $2l-1$ by green ones. This guarantees that the coloring is not spatially periodic, while the original pattern u remains starting in position 0.

It is now enough to show that c is temporally periodic. If we ignore colors, the configuration is spatially periodic and hence also temporally periodic. By symmetry, for all integers k, every arrow of c in the region $[2kl, \dots, 2kl + 2l - 1]$ will always stay inside it when configuration c evolves. Hence configuration c is STP (e.g., see Fig. 7).

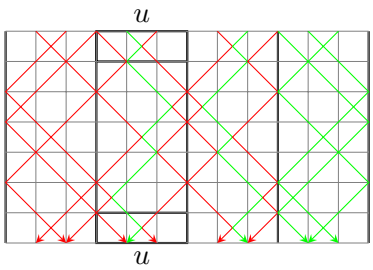


Figure 7: A subblock of the space-time diagram starting at an STP configuration in the CA of Section 3.2. The first line denotes the middle part of the initial configuration.

Acknowledgments.

The second author is supported by the China Scholarship Council.

References

[1] J. Kari, Theory of cellular automata: a survey, *Theoret. Comput. Sci.*, 334: 3–33, 2005.

[2] A. Dennunzio, P. Di Lena, L. Margara. Strictly temporally periodic points in cellular automata, *18th International Workshop on Cellular Automata and Discrete Complex Systems*, 225–235, 2012.

[3] A. Dennunzio, P. Di Lena, E. Formenti, L. Margara. Periodic orbits and dynamical complexity in Cellular Automata, *Fundamenta Informaticae*, to appear.

[4] G. Hedlund, Endomorphisms and automorphisms of shift dynamical systems, *Math. Systems Theory*, 3: 320–375, 1969.

[5] B. Codenotti, L. Margara, Transitive cellular automata are sensitive, *Amer. Math. Monthly*, 103: 58–62, 1996.

[6] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems*, Addison-Wesley, Redwood City, CA, 1989.

[7] J. Banks, J. Brooks, G. Cairns, G. Davis, P. Stacey, On the Devaneys definition of chaos, *Amer. Math. Monthly*, 99: 332–334, 1992.

[8] P. Kůrka, Languages, equicontinuity and attractors in cellular automata, *Ergodic Theory Dynam. Systems*, 17: 417–433, 1997.

[9] L. Acerbi, A. Dennunzio, E. Formenti, Conservation of some dynamical properties for operations on cellular automata, *Theoret. Comput. Sci.*, 410: 3685–3693, 2009.

Cellular Automata with Strong Anticipation Property of Elements

Alexander Makarenko

Institute for Applied System Analysis
at National Technical University of Ukraine,
37 Prospect Peremogy, 03056, Kyiv-36, Ukraine
makalex@i.com.ua, makalex51@gmail.com

Abstract

It is considered the general formulations and properties of cellular automata with strong anticipatory property of elements. Some examples are discused including game of 'Life' with anticipation. Multivalued behavior of solutions of such CA is discribed and their possible consequences for CA theory are proposed.

Keywords. Cellular automata, anticipation, multivalued behavior, computation theory.

1 Introduction

The history of cellular automata (CA) concept has more then 50 years development origin formally from the works of S. Ulam and J. von Neumann (see the reviews at [1]-[3]). During the development of CA concept the formalization of so called classical CA had been proposed. Conditionally the classical CA is defined on regular lattice of cells; each cell has finite number of states and transition roles for changing the states exist. The next in time state of cell depends on cell's states from sum neighborhood of cell.

Recently the number of investigations as theoretical as applied had been proposed with classical CA (see for example papers in J. of Cellular Automata, Proceeding of ACRI, CiE, AUTOMATICA conferences).

At the same time many extending of CA concept have been proposed. Here we remember only some of them: quantum automata; CA with special properties (for example non-deterministic); CA with extended spaces for cell's states (complex numbers, p-adic, 'grossones') [3]-[6]. Also we should remark CA with the memory [7] and hierarchical CA [8]. Usually new modifications of CA follow from the new applications of cellular automata. It is evident that further development of CA also will be related to the new fields of investigations.

So in given paper we propose the new class of CA, namely the cellular automata with anticipatory property. Roughly speaking the anticipatory property consists in dependence of transitions to next states on presumable future states of the system. This property is some counterpart to memory property. One of the most prospective ideas is strong anticipation introduced be Daniel Dubois from Liege (Belgium) [9, 10]. Remark that strong anticipation is necessary in large socio-economical systems investigation [11].

The structure of the paper is the next. At first we describe the system with strong anticipation. Then we describe some examples of CA with anticipatory property and of their solutions behavior. Finally we propose some further prospects for investigations.

2 Cellular automata with anticipation

2.1 Classical cellular automata description

Here we give the formal description of cellular automata with anticipation. First of all we follow the papers on CA (see for example [3, 4, 7]). We pose here the description one-dimensional CA from [7, p.2] but the description can be modified to many-dimensional case.

'One-dimensional CA is represented by an array of *cells* x_i where $i \in \Sigma$ (integer set) and each x takes a value from a finite alphabet Σ. Thus, a sequence of cells $\{x_i\}$ of finite length n represents a string or *global configuration* c on Σ. This way, the set of finite configurations will be represented as Σ^n. An evolution is represented by a sequence of configurations $\{c^t\}$ given by the mapping $\Phi\colon \Sigma^n \to \Sigma^n$; thus their global relation is following

$$\Phi(c^t) \to c^{t+1} \tag{1}$$

where t time step and every global state of are c is defined by a sequence of cell states. Also the cell states in configuration c^t are updated at the next configuration c^{t+1} simultaneously by a local function φ as follows'

$$\varphi(x_{i-r}^t, \ldots, x_i^t, \ldots, x_{i+r}^t) \to x_i^{t+1} \tag{2}$$

Also for further comparing and discussion we show the description of CA with memory from [7, p.3]:

'CA with *memory* extends standard framework of CA by allowing every cell x_i to remember some period of its previous evolution. Thus to implement a memory we design a memory function ϕ, as follows:

$$\phi(x_i^{t-\tau}, \ldots, x_i^{t-1}, x_i^t) \to s_i \tag{3}$$

Such that $\tau < t$ determines the degree of memory backwards and each cell $s_i \in \Sigma$ is a state function of the series of the states of the cell x_i with

memory up to time-step. To execute the evolution we apply the original rule as follows:

$$\varphi(\ldots, s_{i-1}^t, s_i^t, s_{i+1}^t, \ldots) \to x_i^{t+1} \tag{4}$$

In CA with memory, while the mapping φ remains unaltered, historic memory of all past iterations is retained by featuring each cell as a summary of its past states from ϕ.'

2.2 Strong anticipation property

The term 'anticipation' had been introduced in active using in biology and applied mathematics by Robert Rosen [12]. So this name is well known for common researchers. Less known is contribution by Daniel Dubois to strong anticipation.

Since the beginning of 90-th in the works by D.Dubois – see [9, 10, 13] the idea of strong anticipation had been introduced: "Definition of an incursive discrete strong anticipatory system ...: an incursive discrete system is a system which computes its current state at time t, as a function of its states at past times $\ldots, t-3, t-2, t-1$, present time,t, and even its states at future times $t+1, t+2, t+3, \ldots$

$$x(t+1) = A(\ldots, x(t-2), x(t-1), x(t), x(t+1), x(t+2), \ldots, p) \tag{5}$$

where the variable x at future times $t+1, t+2, t+3, \ldots$ is computed in using the equation itself.

Definition of an incursive discrete weak anticipatory system: an incursive discrete system is a system which computes its current state at time$t$, as a function of its states at past times $\ldots, t-3, t-2, t-1$, present time t , and even its predicted states at future times $t+1, t+2, t+3, \ldots$

$$x(t+1) = A(\ldots, x(t-2), x(t-1), x(t), x^*(t+1), x^*(t+2), \ldots, p) \tag{6}$$

where the variable x^* at future times $t+1, t+2, t+3, \ldots$ are computed in using the predictive model of the system" [10, p. 447].

Thus as the further research problem in the field of CA it should be considered system with strong anticipation.

The equations (5), (6) are rather new mathematical objects and their full considerations are the tasks for further investigations.

Remark those systems with anticipation (especially strong anticipation) early have been investigated in many research fields: biology, computer science, social sciences, physics, neurophysiology etc. (see Journal of Computing Anticipatory Systems, Proceedings of Conferences CASYS). Because of importance of such systems below we describe one of the tools for their investigations – CA with strong anticipation.

2.3 Cellular Automata with Strong Anticipation

The key idea is to introduce strong anticipation into CA construction. Of course many ways exist for implementation such idea. First of all we will describe one of the simplest. Then we will discuss the results of some computer experiments with such CA.

For such goal we will suppose that states of the cells of CA can depend on future (virtual) states of cells. Then the modified rules for CA in one of possible modifications have the form:

$$\phi(\bar{x}_i^{t-\tau}, \ldots, \bar{x}_i^{t-1}, \bar{x}_i^t, \bar{x}_i^{t+1}, \ldots, \bar{x}_i^{t+k}) \to \bar{s}_i^{t+k} \tag{7}$$

$$\bar{\varphi}(\ldots, \bar{s}_{i-1}^{t+k}, \bar{s}_i^{t+k}, \bar{s}_{i+1}^{t+k}, \ldots) \to (\bar{x}_i^{t+1}, \bar{x}_i^{t+2}, \ldots, \bar{x}_i^{t+k}) \tag{8}$$

where k (integer) is horizon of anticipation. But because of presumable properties of such solutions (multi-valuedness, see examples and discussion below) the values of variables in (7), (8) have different interpretations. This is connected with the fact that equation (7), (8) are nonlinear equations on the values of cell's states in future moments of time. Such equations can have one solution, no solutions and many solutions in dependence on parameters and the current and past states of the cells). So in equations (7), (8) $\bar{x}_i^{t+j}$ represent all presumable values of cell i at moment $t+j$ and $\bar{s}_i^{t+k}$ represent all pre-history (multi-valued) for cell i till the moment$t+k$. Remark that in some particular cases (one – valued solutions of (7), (8)) the results correspond to such in equations (3), (4).

Further we for simplicity describe the system of such CA without memory and only with one-step anticipation $k = 1$. The general forms of such equations in this case are:

$$\phi(\bar{x}_i^t, \bar{x}_i^{t+1}) \to \bar{s}_i^{t+1} \tag{9}$$

$$\varphi(\ldots, \bar{s}_{i-1}^{t+1}, \bar{s}_i^{t+1}, \bar{s}_{i+1}^{t+1}, \ldots) \to \bar{x}_i^{t+1} \tag{10}$$

The main peculiarity of solutions of (9), (10) is presumable multi-validness of solutions and existing of many branches of solutions. This implies also the existence of many configurations in CA at the same moment of time. Remark that this follows to existing of new possibilities in solutions and interpretations of already existing and new originating research problems.

3 Game 'Life' with strong anticipation.

In this subsection we propose very short description of computer experiments with one of simple CA with strong anticipation – game 'Life_A'. More details are described in our publications [14, 15].

Let's consider rules driving the simple classical "Life" [1, 2, 14, 15] with two possible states of the cells marked as '1' and '0'. To formalize the

transitions roles, we have introduced two auxiliary functions f_0 and f_1 for transition from such states. Their values may be represented by different ways. Here we give below representation by the table form.

Table 1: Auxiliary functions $f_0(x)$ and $f_1(x)$.

x	0	1	2	3	4	5	6	7	8
$f_0(x)$	0	0	0	1	0	0	0	0	0
$f_1(x)$	0	0	1	1	0	0	0	0	0

Suppose that out of the interval [0;8] both these functions are zero-valued. Moreover, if the argument is not an integer value, it is rounded to the nearest integer.

Under the introduced notions next state function for k-th cell will be as follows:

$$\begin{cases} f_0(S_k), C_k = 0 \\ f_1(S_k), C_k = 1 \end{cases}, F_k \in \{0,1\}, \tag{11}$$

where S_k – number of (non-zero) neighbors of k-th cell, $C_k \in \{0,1\}$(usually 0 interprets as if k-th cell is dead, 1 - otherwise).

Thus, the evolution of an automaton with N cells may be denoted as:

$$C_k^{t+1} = F_k^t = F(S_k^t), k = 1...N, \tag{12}$$

where t – discrete time.

3.1 Roles with anticipation

To endow a model with anticipation property one has to design a next step function that depends on the next (expected, virtual) cell states [14, 15]. Many possibilities exist to implement it. We took in [14, 15] as the illustrative example for CA with anticipation the case when the next state function of classical "Life" defined as:

$$F_k^t = F(S_k^t) \tag{13}$$

had been replaced by:

$$F_k^t = F((1-\alpha) \cdot S_k^t + \alpha \cdot S_k^{t+1}), \alpha \in [0,1] \tag{14}$$

or

$$F_k^t = F(S_k^t + \alpha \cdot S_k^{t+1}), \alpha \in \mathbb{R} \tag{15}$$

Such rules correspond to 'integral' accounting the future virtual states by counting only number of presumably occupied cells in the neighborhood of the cell at future moment of time. Further, these two variants of a next

state function will be referred to as weighted and additive, correspondingly. Cellular automata driven by either of these functions were implemented as the computer program.

Below we pose for illustration of presumable behavior of such game 'Life_A' with anticipation only one picture. The figure below corresponds to the case with 16 cells with periodic boundary conditions.

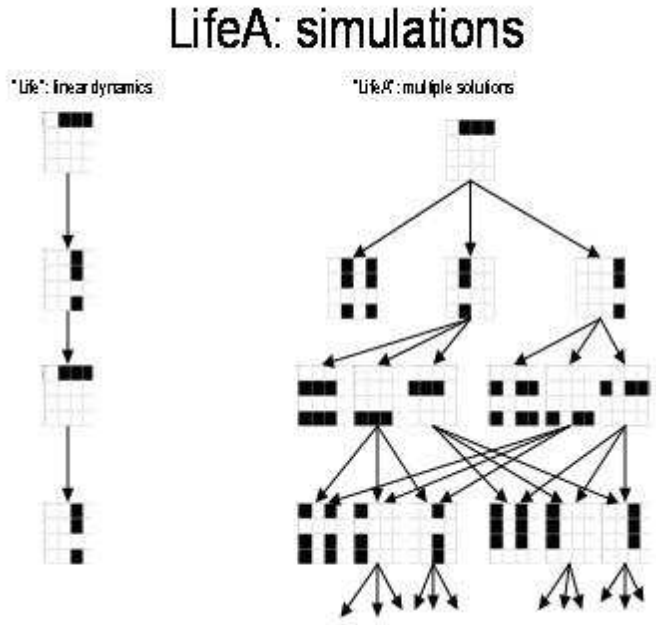


Figure 1: Comparison the solutions of classical game 'Life' (left side) and of game 'Life_A' with strong anticipation (right side).

Only 3 discrete time steps are represented at Fig. 1 and at right side of the figure we see possible branching of the solution, existing of many values of cellular state and multiplicity of configurations (more examples see at [14, 15]). So we can anticipate that such kind of behavior is typical to CA with strong anticipation.

4 Discussions and further research problems

As we hope described results open new possibilities for receiving interesting results as in CA behavior as in interpretations of presumable solutions. Here we describe only few of such possibilities (more we suppose to describe in separate publications.

1. At first we remember the new possibilities in considering of non-deterministic CA (and moreover usual automata). Non-deterministic au-

tomata allow few transition ways from one state to others [4, 16, 17]. Usually it is supposed that such structure is only theoretical and in reality only one of the ways is used in each transition. CA with anticipation opens the natural possibility for considering of the systems with many different ways in parallel. Accepting possibilities of physical realization of strong anticipatory systems it may be accepted existence of CA with many branches. Also such systems are interesting as multi-valued dynamical systems.

2. In proposed paper we have considered only the case of finite alphabet for indexing the cell's states. But previous investigations of dynamical systems with strong anticipation show the possibilities of existing the solutions with infinite numbers of solution branches [18]. This allows introducing CA with infinite number of cell' states (or at least infinite alphabet for CA).

3. The generalizations from point 1 and 2 at this subsection and analysis of automata and CA theories origin follows to presumable considering of some aspects of computation theory (see [14, 15, 19]). The short list of topics may be the next: computability; Turing machines; automata and languages; recursive functions theory; models of computation; new possibilities for computations with accounting possible branching.

Conclusion

Thus in given paper we describe first results on cellular automata with anticipatory property. This results show the new possibilities in theory and applications of such objects.

Acknowledgements

The author would like to give the appreciation for D. Krusinsky for computer realizations of models and for D. Krushinski and B. Goldengorin for some discussions.

References

[1] Wolfram S. New kind of science. Wolfram Media Inc., USA, (2002).

[2] Illiachinski A., Cellular Automata. A Discrete Universe. World Scientific Publishing, Singapoure, (2001).

[3] Kari J. Theory of cellular automata: A survey. Theoretical Computer Science, 334, 3-33 (2005)

[4] Kutrib M. Non-deterministic cellular automata and languages. Int. Journal of General Systems. 41, 555 – 568. (2012)

[5] Sergeyev Ya.D., Garro A. Single-tape and multi-tape Turing machines through the lens of the Grossone methodology, Journal of Supercomputing, 65(2), 645-663. (2013)

[6] D'Alotto . Cellular Automata Using Infinite Computations, Applied Mathematics and Computation, 218(16), 8077-8082. (2012).

[7] Martinez G.J., Adamatzky A., Alonso-Sanz R. Complex dynamics of elementary cellular automata emerging from chaotic rules. Int. Journal of bifurcation and chaos, 22, 1250023, (2012).

[8] Bandini S., Mauri G. Multilayered cellular automata. Theoretical Computer Science, 217, 99-113, (1999).

[9] Dubois D., Generation of fractals from incursive automata, digital diffusion and wave equation systems. BioSystems, 43, 97-114, (1997).

[10] Dubois D., Incursive and hyperincursive systems, fractal machine and anticipatory ence. Published by the American Institute of Physics, AIP Conference Proceedings 573, 437 – 451, (2001).

[11] Makarenko A., Anticipating in modeling of large social systems - neuronets with internal structure and multivaluedness. International .Journal of .Computing Anticipatory Systems, 13, 77 - 92. (2002).

[12] Rosen R., Anticipatory Systems. Pergamon Press. (1985).

[13] Dubois D., Introduction to Computing Anticipatory Systems. International Journal of Computing Anticipatory Systems, 2, 3-14. (1998).

[14] Makarenko A., Goldengorin B., Krushinsky D., Game 'Life' with Anticipatory Property. Proceed. Int. Conf. ACRI'08, Eds. Umeo, H. et al, LNCS 5191, (2008). pp. 77-82.

[15] Krushinskiy D., Makarenko A., Cellular Automata with Anticipation. Examples and Presumable Applications. *AIP Conf. Proc.*, vol.1303, ed. D.M.Dubois, USA, (2010). pp. 246-254.

[16] Cooper S. Barry Computability Theory. Chapman Hall/CRC (2003)

[17] Sipser M. Inrtoduction to the theory of computation. Second edition. Thomson Course Technology, USA (2006).

[18] Makarenko A., Stashenko A. Some two- steps discrete-time anticipatory models with 'boiling' multivaluedness. AIP Conference Proceedings, vol.839, ed. Daniel M. Dubois, USA, (2006). pp.265-272.

[19] Makarenko A. Cellular automata and some problems in concepts reconstruction. Herald of Chmelnickiy State University (Ukraine), 2, 248-251, (2007). (in Russian).

J. Kari, M. Kutrib, A. Malcher (eds.):
Workshop on *Cellular Automata and Discrete Complex Systems* 2013 57

Computational complexity of majority automata under different updating schemes

Pedro Montealegre * Eric Goles †

August 17, 2013

Abstract

The majority automata is the one where the satiate at each site is decided by a majority rule of the sites in its neighborhood. We study the computational complexity of predict the changes in the state of a given vertex from an initial configuration, and how this complexity is affected over changes in the updating scheme. A block sequential updating scheme is the one where the vertex set is partitioned, and every partition is updated synchronously, but sequentially with respect to the others. We show that for sequential and synchronous updating schemes, the problem is P-Complete, but for any block sequential updating scheme the problem is NP-Hard. We conjecture that this problem is $PSPACE$-Complete.

Keywords: *Majority automata, Computational Complexity, Updating Scheme, P-Complete, NP-Hard*

1 Introduction

Let $G = (V, E)$ be a simple undirected graph, where $V = \{1, ..., n\}$ is the set of vertices, E the set of edges. An *Automata Network* is a triple $\mathcal{A} = (G, \{0, 1\}, (f_i : i \in V))$, where, $\{0, 1\}$ is the set of states and $f_i : \{0, 1\}^{|V|} \to \{0, 1\}$ is the transition function associated to the vertex i. The set $\{0, 1\}^{|V|}$ is called the set of *configurations*, and the automaton's global transition function $F : \{0, 1\}^{|V|} \to \{0, 1\}^{|V|}$ is constructed from the local functions $((f_i : i \in V))$. In this paper we consider the *majority functions*, i.e.:

$$f_i(x) = \begin{cases} 1 & \text{if } \sum_{j \in N(i)} x_i > \frac{|N(i)|}{2} \\ 0 & \text{if } \sum_{j \in N(i)} x_i \leq \frac{|N(i)|}{2} \end{cases}$$

*LIFO, Université d'Orléans, 45067 Orléans Cedex 2, France. Email: pedro.montealegre@etu.univ-orleans.fr

†Facultad de Ciencias y Tecnología, Universidad Adolfo Ibáñez, Santiago, Chile. Email: eric.chacc@uai.cl

where $N(i)$ is the set of neighbors of vertex i. We say that vertices in state 1 are *active* while vertices in state 0 are *passive*. An automata network with this rule is called a *majority automata*.

An *updating scheme* of the automata $\mathcal{A}$ is a function $\phi : V \to \{1 \dots |V|\}$ such that if u and v are vertices and $\phi(u) < \phi(v)$ then the state of u is updated before v, and if $\phi(u) = \phi(v)$ then nodes u and v are update at the same time. When $\phi(v) = 1$ for every vertex v, i.e. all vertices are updated at the same time, we have the *synchronous* updating scheme. When $\phi(v) = \sigma_v$, where σ is a permutation of the set of vertices, we have a *sequential* updating scheme. A *block sequential* updating scheme is the one where the vertex set is partitioned into several subsets, such that into the same set every vertex is updated at the same time, and different subsets are updated sequentially in some order.

The *trajectory* of a configuration x for an updating scheme ϕ is the set $T(x) = \{x(t) : t \geq 0\}$ where $x(0) = x$ and $x(t+1) = (f_i(x(t))^\phi$, where $(y)^\phi$ denotes the update of the local functions under the scheme ϕ. We say that the trajectory of x *enters in a limit cycle of period* p if for some t the set $\{x(t), x(t+1), \dots, x(t+p-1)\}$ has $p-1$ different values, and $x(t+p) = x(t)$; in other words if $|T(x(t))| = p$. A cycle of period 1 is a *fixed point*. Naturally, since the graph has a finite number of vertices, there are at most $2^{|V|}$ different configurations and then, for every configuration x in any graph, local rule and updating scheme, every trajectory reaches a limit cycle. We call that limit cycle the *steady state* given by x.

We define the transient length of an initial configuration x under the updating scheme ϕ, $\tau_\phi(x)$ as the number of steps to reach the steady state, and we denote the transient length of the automaton $\mathcal{A}$ under the updating scheme ϕ as the the greatest of these values:

$$\tau_\phi(\mathcal{A}) = \max\{\tau_\phi(x) : x \in \{0,1\}^{|V|}\}.$$

We define the *complexity of the majority automata* as the computational complexity of the following decision problem of predict changes on a vertex, given an initial configuration:

MAJORITY: Consider the majority automaton $\mathcal{A} = (G, \{0,1\}, (f_i : i \in V))$, $x \in \{0,1\}^{|V|}$ a configuration of G, $v \in V$ a vertex initially passive ($x_v = 0$), and ϕ an updating scheme of G . Does there exists $T > 0$ such that $x_v(T) = 1$ when the updating scheme ϕ is applied?

In the following sections, we will study how does the complexity of the majority automata changes when we apply different updating schemes. In Section 2 we will show that for both sequential and synchronous updating schemes, this problem is P-Complete. After that, in Section 3, we will show that for block sequential updating schemes the problem is unlikely to be in

P, by showing that block of size two are enough to prove NP-Hardness. Finally we will give some conclusions.

2 Complexity of majority automata with synchronous and sequential updating schemes

In this section, we will show that the problem **MAJORITY** for threshold automata is in P for the synchronous and any sequential updating schemes. Clearly for any configuration x, $F(x)$ can be computed in polynomial time, just evaluating the local functions in the order of the vertices given by the updating scheme. The idea then is to show that the number of steps that one should simulate to decide **MAJORITY** is polynomial in the size of the graph.

Before showing the theorem, we define $\overline{A}$ as the automata over the same graph that $\mathcal{A}$, but with local function:

$$f_i(x) = \begin{cases} 1 & \text{if} \quad \sum_{j\in N(i)} x_i > \theta_i \\ 0 & \text{if} \quad \sum_{j\in N(i)} x_i \leq \theta_i \end{cases}$$

where $\theta_i = \frac{|N(i)|}{2}$ if $|N(i)|$ is odd, and $\theta_i = \frac{|N(i)|+1}{2}$ if $|N(i)|$ is even. Clearly the dynamics of $\mathcal{A}$ and $\overline{\mathcal{A}}$ are the same, but with $\overline{\mathcal{A}}$ we can ensure that the threshold $\sum_{j\in N(i)} x_j \neq \theta_i$, for every $i \in \{1, \ldots, n\}$, which is a condition that we will need for the next theorem.

Theorem 1. *For both the synchronous and sequential updating schemes, the problem* **MAJORTY** *is in P.*

Proof. We will use the results obtained in [1], [2]. Let $\mathcal{A} = (G, \{0,1\}, (f_i : i \in V))$ be a the majority automata, and let $\eta : \mathbb{R} \to \{0,1\}$ be the threshold function $\eta(x) = \begin{cases} 1 & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases}$. We can characterize any threshold automata using the following function: $f_i(x) = \eta\left(\sum_{j\in N(i)} x_j - \theta_i\right)$ where $\theta_i = \frac{|N(i)|}{2}$ if $|N(i)|$ is odd, and $\theta_i = \frac{|N(i)|+1}{2}$ if $|N(i)|$ is even. If $\overline{\eta} : \mathbb{R}^{|V|} \to \{0,1\}^{|V|}$ is the multidimensional threshold function: $\overline{\eta}((x_i)_{i\in V}) = (\eta(x_i))_{i\in V}$, then the synchronous updating scheme can be characterized as $F(x) = \overline{\eta}(Ax - b)$ where $A = (a_{ij})$ is the adjacency matrix of G and b is a vector of size $|V|$ with $b_i = \theta_i$.

Using this characterization, consider $T(x)$ be a trajectory of the majority automata with initial condition x, and define the following functional for $t \geq 1$:

$$\begin{aligned} E(x(t)) \quad = \quad & \sum_{i\in V}(2b_i - \sum_{j\in V} a_{ij})(2x_i(t) + 2x_i(t-1) - 2) \\ & -(2x_i(t) - 1)\sum_{j\in V} a_{ij}(2x_j(t-1) - 1) \end{aligned}$$

then if $\Delta_t E = E(x(t)) - E(x(t-1))$ we have that

$$\Delta_t E = -4 \sum_{i \in V} (x_i(t) - x_i(t-2))(\sum_{j \in V} a_{ij} x_j(t-1) - b_i) \leq 0$$

and then E is a strictly decreasing function. Hence, if $\{x(t) : t \in [t_1, t_2]\}$ is a cycle, then $E(x(t))$ is necessarily constant for any $t \in [t_1, t_2]$. Actually, since for any $x \in \{0,1\}^{|V|}$ and $i \in V$ we have $\sum_{j \in V} a_{ij} x_u \neq b_i$, then the steady states are only fixed points and/or cycles of length two, i.e. $t_2 = t_1$ or $t_2 = t_1 + 2$.

Then we have that for any initial configuration, the dynamics necessarily enters in a cycle of length at most 2. Remember that $\tau_\phi(\mathcal{A})$ is the transient length of $\mathcal{A}$ with updating scheme ϕ, this is, the maximum number of steps required to enter into a steady state. Also by a result in [1], [2] we have that $\tau_\phi(\mathcal{A}) \leq |V|^3$. This tells us that in a number of steps that is polynomial in the size of the input, the trajectory enters into a cycle of length 2. Since any iteration can be simulated in polynomial time, **MAJORITY** is in P for the synchronous updating scheme.

For any sequential updating we use the same approach. But in this case the decreasing functional is given by:

$$E(x) = -\frac{1}{2} \sum_{i \in V} \sum_{j \in V} a_{ij} x_i x_j + \sum_{i \in V} b_i x_i$$

then, following [1], [2], we prove the dynamics converges always to fixed points in at most $O(|V|^3)$ steps, so in polynomial time. □

Once that we have proved that the problem **MAJORITY** is in P for synchronous and sequential updating schemes, a straight forward question is what else can we say for this updating schemes, this is, determine if the problem is in some subclass of P, like NC, or it is P-Complete. In this context, in [3], [4] is shown that for the synchronous updating scheme the problem **MAJORITY** is P-Complete. The proof is done reducing to **MAJORITY** the *Monotone circuit value problem*, which is a well known P-Complete problem. This hardness results can easily be adapted to sequential updating schemes, and then we have the following theorem:

Theorem 2. *For the majority automata and both the synchronous and sequential updating schemes, the problem* **MAJORITY** *is in P-Complete.*

In the next section, we will show that for block sequential updating schemes the problem is unlikely to be in P, by showing that blocks of size two are enough to prove NP-Hardness.

3 Block sequential updating scheme

In the proof of Theorem 1, the strategy to obtain the membership in P was to demonstrate that in a polynomial number of steps the automata enters into a steady state, and the steady state is small (cycle of length at most 2). Since to simulate a step of the dynamics can be done in polynomial time (just calculating the new state of every vertex), and we have to simulate just a polynomial number of steps, then we can decide in polynomial time.

But the fact that **MAJORITY** belongs to P for the synchronous and asynchronous updates seems not easy to generalize to other updating schemes. Actually, as we prove in next theorem, very simple updating schemes allows super-polynomial limit cycles and transient lengths, which implies that we can not (is very unlikely to be possible to) bound the number of steps by a polynomial by using the energy functional approach.

Theorem 3. *There is a block sequential update scheme in a majority automata, such that each block has cardinality* 2 *and the limit cycle has a exponential length.*

Proof. A *ladder* of length k (Figures 1 and 2) is a graph with $2(k+1)$ vertices, such that can be updated in some scheme in order to obtain a limit cycle of length k. One can connect two ladders as shown in Figure 3, and then obtain that the limit cycle is the last common multiple (lcm) of the lengths of the two ladders.

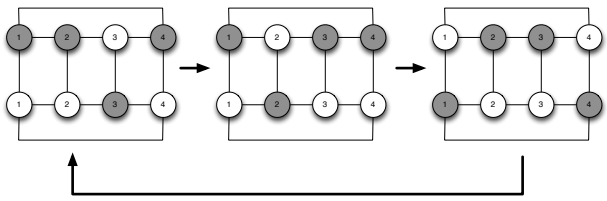


Figure 1: A *ladder* of length 3. Gray vertices represent active ones, and vertex numbers represent its value for the updating scheme. Notice that we obtain a limit cycle of length 3.

To obtain a lower bound of the periods, we follow the arguments developed in [5]. Let m a positive integer, and let $l = \pi(m)$ the number of primes not exceeding m. Let G the graph obtained from $\pi(m)$ ladders of sizes $p_1, p_2 \ldots, p_{\pi(m)}$, where $\{p_1, p_2 \ldots, p_{\pi(m)}\}$ the first $\pi(m)$ primes. We have then that

$$V(G) \leq \sum_{i=1}^{\pi(m)} 2(p_i + 1) \leq 2\pi(m)(m+1) \tag{1}$$

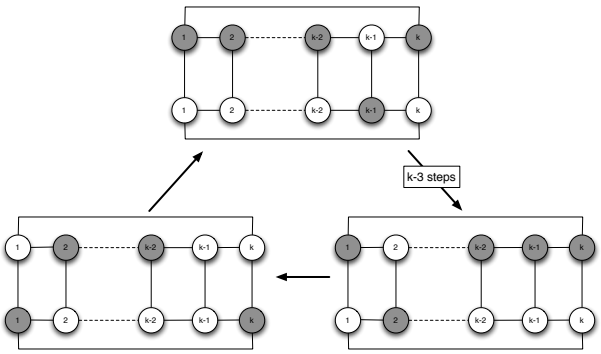


Figure 2: A *ladder* of length $k-1$.

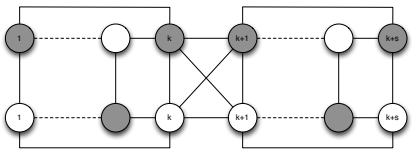


Figure 3: A *ladder* of length $k-1$ united with a ladder of length $s-1$. The limit cycle of this graph is $lcm(k-1, s-1)$.

and

$$lcm(p_1, \ldots, p_{\pi(m)}) = \prod_{i=1}^{\pi(m)} p_i = e^{\theta(m)} \tag{2}$$

where $\theta(m) = \sum_{i=1}^{\pi(m)} \log(pi)$. From the Prime Number Theorem [6] we know that $\pi(m) = \Theta(m/\log(m))$, furthermore in [6] is shown that $\theta(m) = \Theta(\pi(m)\log(m))$, which together with (1) and (2) imply that

$$lcm(p_1, \ldots, p_{\pi(m)}) \geq e^{\Omega(\sqrt{|V(G)|\log(|V(G)|)})}$$

and then the length of the limit cycle of G is not bounded by any polynomial in $|V(G)|$. □

Notice that trivially from Theorem 2, the problem **MAJORITY** P-Hard for block sequential updating schemes. Clearly this problem belongs to $PSPACE$. Indeed, to simulate any step of the dynamics for any updating scheme, we just need both times the size of the graph, one for read the actual state, and the other for write the new state. We conjecture that this problem is $PSPACE$-Complete. Using the gadgets of Theorem 3 we can show that the problem is NP-Hard.

Theorem 4. *The problem* **MAJORITY** *is NP-Hard for block sequential updating schemes.*

Proof. We will show that **MAJORITY** is NP-Hard by reducing to it the problem 3-SAT. Remember that 3-SAT is the Boolean satisfiability problem to instances with exactly three variables per clause.

Let ϕ an instance of 3-SAT. For every variable x_i of ϕ, we will build a gadget which consists in a subgraph with long cycle length using the *ladders* of the proof of Theorem 3. The idea is that at every step we will be simulating a different truth assignment of the variables. The reduction then consists in 3 layers: The first layer will have the gadget for simulating the variables. In the second layer we simulate every clause by joining three different variables with a node that will simulate the OR function. Finally, in the third layer we join every OR gadget to a vertex that simulate an AND function.

Let $\{x_1, \ldots, x_n\}$ the variables of ϕ, and let $\{p_1, \ldots, p_n\}$ the fist n primes. The variable x_i is simulated by the gadget shown in Figure 4, where we simulate both x_i and $\overline{x}_i$. The gadget is build from two different *ladder* gadgets of length p_i. Two vertices, which are updated after every vertex in the ladder, will represent then x_i and $\overline{x}_i$. Then x_i is active and $\overline{x}_i$ passive in the steps that are multiple of p_i, and in the others is x_i is passive and $\overline{x}_i$ active.

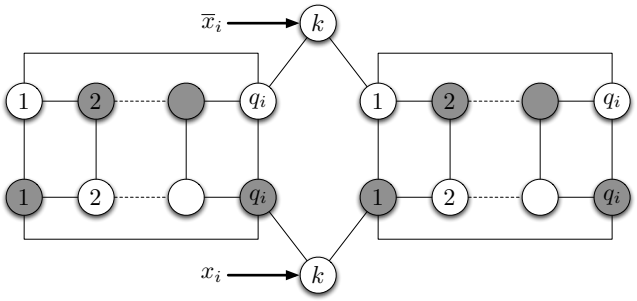


Figure 4: Variable x_i. Gray vertices represent active ones, and vertex numbers represent its value for the updating scheme. In the figure $q_i = p_i + 1$, and then the ladder make a cycle of length p_i. Also in the figure $k = p_n + 2$, and then the vertices that simulate x_i and $\overline{x}_i$ are updated after every ladder.

With this construction, we can go over every combination of input values of ϕ. For example, if $n = 5$ (i.e. ϕ have 5 variables) then the input combination $(x_1, x_2, x_3, x_4, x_5) = (1, 0, 1, 0, 1)$ will occur after $p_1 \times p_3 \times p_5 = 2 * 5 * 11 = 110$ steps.

Let $C_i = (x_{i_1}^{s_1} \vee x_{i_2}^{s_2} \vee x_{i_3}^{s_3})$ the *i-th* clause of ϕ, where $i_1, i_2, i_3 = \{1, \ldots, n\}$, and $s_1, s_2, s_3 \in \{0, 1\}$ such that $x_i^1 = x_i$ and $x_i^0 = \overline{x}_i$. We build then the gadgets for x_{i_1}, x_{i_2} and x_{i_3} and a new vertex that simulate the OR function, and join them according to de values of s_1, s_2, s_3, as shown in Figure 5. The vertex that simulate the OR function will be updated after every vertex in

the gadgets of the variables.

Finally, we connect every clause gadget to a vertex that simulate the AND function, that can be build using an initially passive vertex joined with $m-1$ initially passive vertices, where m is the number of clauses. We have then that this vertex (the one that simulates the AND function) will become active in some step, if and only if there exists a truth value assignation of the variables $x_1, \ldots, x_n$ such that$\phi(x)$ is true.

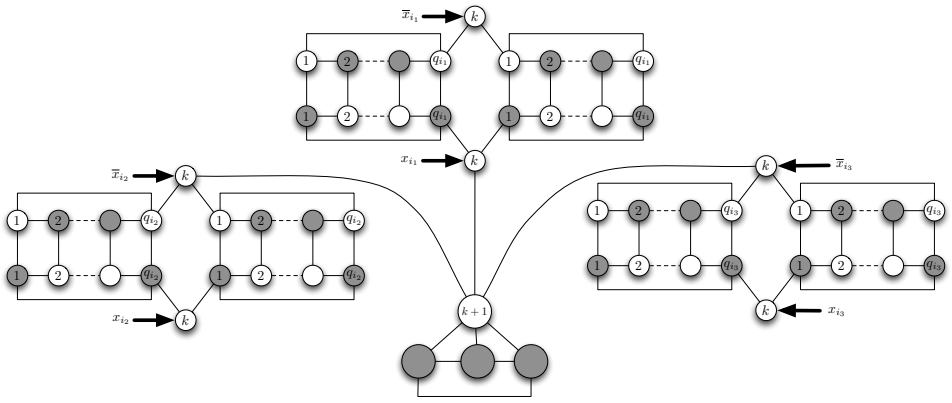


Figure 5: Clause $C_i = (x_{i_1} \vee \overline{x}_{i_2} \vee \overline{x}_{i_3})$. In the figure $q_i = p_i + 1$ and $k = p_n + 2$, where p_i is the *i-th* prime. Gray vertices represent active ones, and the numbers in the vertices are the value of the vertex in the updating scheme. Vertices without number can be updated any time.

From the Prime Number Theorem [6], we have that $p_n = \mathcal{O}(n \log(n))$. Since the input is of size $\mathcal{O}(n)$, then we can compute the first n primes can in polynomial time, and then in polynomial time we can build the gadgets for each variable of ϕ. Clearly all the steps of this reduction can be done in polynomial time, and then **MAJORITY** is NP-Hard. □

4 Conclusions

We have discussed how the computational complexity of majority automata, and how it is affected over changes in the updating scheme. While for the synchronous and sequential updating scheme **MAJORITY** is P-Complete, for the block sequential updating schemes we have shown that the problemis NP-Hard. We conjecture that **MAJORITY** is $PSPACE$-Complete.

Finally, a notion of complexity can be defined for automata networks (in general, not only for majority automata) and updating scheme. If we say that the complexity of an automata is the complexity of the prediction problem, then an automata have a *portable* complexity if the complexity of the problem does not depend on the updating scheme. A future work could involve the search for this kind of automata.

References

[1] E. Goles-Chacc, F. Fogelman-Soulie, D. Pellegrin, Decreasing energy functions as a tool for studying threshold networks, Discrete Applied Mathematics 12 (3) (1985) 261–277.

[2] E. Goles-Chacc, Comportement oscillatoire d'une famille d'automates cellulaires non uniformes, Thèse IMAG, Grenoble.

[3] C. Moore, Majority-Vote Cellular Automata, Ising Dynamics, and P-Completeness, Journal of Statistical Physics 88 (1997) 795–805.

[4] P. Montealegre-Barba, Redes de autómatas y complejidad computacional, Departamento de Ingeniería Matemática, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile, 2012.

[5] M. A. Kiwi, R. Ndoundam, M. Tchuente, E. Goles, No polynomial bound for the period of the parallel chip firing game on graphs, Theoretical Computer Science 136 (2) (1994) 527–532.

[6] G. H. Hardy, E. M. Wright, D. R. Heath-Brown, J. Silverman, An Introduction to the Theory of Numbers, Oxford mathematics, OUP Oxford, 2008.

Identification of Non-Uniform Periodic Boundary Cellular Automata having only Point States*

Nazma Naskar[1], Sumit Adak[2], Sukanta Das[2]
[1]Department of Information Technology
Seacom Engineering college, Howrah,711302, India.
[2]Department of Information Technology
Bengal Engineering and Science University, Shibpur, 711103, India.
E-mails: naskar.preeti@gmail.com,
maths.sumit@gmail.com, sukanta@it.becs.ac.in

Abstract

This paper identifies the 1-dimensional non-uniform cellular automata having only point state attractors under periodic boundary condition. To do this, we first identify the point state attractors of the given automaton. Then the cyclic states of the automaton is counted. If the number of point states (which are also cyclic states) are equal with that of total cyclic states, we declare that the cellular automaton is having only point state attractors. To do these task, reachability tree, a discrete tool for characterizing cellular automata, has been utilized.

Keyword: Periodic boundary CA, Point State Attractor, Reachability Tree, cyclic states, Rule Min Term (RMT).

I Introduction

The cellular automata (CA) are classified into different categories depending on several factors, like dimension, neighbourhood, boundary condition, etc. In this work we concentrate on only one-dimensional CA. Further, we consider the boundary condition as periodic (PBCA). It is traditionally assumed that all the CA cells follow the same local rule. We have, however, considered this work that different cells may follow different rules (non-uniform CA).

In recent years, non-uniform CA have gained a huge attention, and they have been targeted to model several real life applications, such as VLSI design and test, pattern recognition and classification, cryptography, etc [1, 4].

*This work is supported by AICTE Career Award fund (F.No.- 1-51/RID/CA/29/2009-10).

However, in almost all works, null-boundary CA have been considered. A few works on reversibility [2] and number conservation [1] of non-uniform PBCA have been reported recently. No work, however, on non-uniform PBCA that contain only point state attractors in their state space has been reported till date. It has been shown in literature [1, 4] that CA having only point states can be efficiently utilized in pattern classification. In this scenario, this research undertakes the issue of identifying non-uniform PBCA that are having only point state attractors. To do this, we first count the cyclic states in a given CA. Such counting for PBCA, in fact, an unaddressed issue. Our approach is, count the cyclic states and count the point states (which are obviously cyclic states), and if they match, then declare that the automaton contains only point states.

II CA Preliminaries

A CA, consists of regular grid of cells in the form of lattice [5], evolves in discrete space and time. A cell's state depend on its own state and its neighbouring cell's states. In a 3-neighbourhood dependency, the next state S_i^{t+1} of a cell of 1-dimensional CA is denoted as $S_i^{t+1} = f_i(S_{i-1}^t, S_i^t, S_{i+1}^t)$, where f_i is the next state function; S_{i-1}^t, S_i^t and S_{i+1}^t are the present (local) states of the left neighbour, self and right neighbour of the i^{th} CA cell. The collection of states $\mathcal{S}^t(S_1^t, S_2^t, \cdots, S_n^t)$ of cells at time t is the present state of CA having n cells. If $S_0^t = S_n^t$ and $S_{n+1}^t = S_1^t$, then the CA is called periodic boundary CA. Here we are concerned about periodic boundary CA (PBCAS).

The function of next state of i^{th} cell CA can be expressed in the form of a table (Table 1a). The decimal equivalent of the 8 outputs is traditionally called as *Rule* [5]. In such CA, there are a total of 2^{2^3} (256) rules. The first row of the Table 1a lists the possible 2^3 (8) combinations of the present states of left, self and right neighbours. We refer each column as Rule Min Term (RMT) [3].

The set of rules $\mathcal{R} = \langle \mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_n \rangle$ where cell i acts with R_i is called rule vector. If $\mathcal{R}_1 = \mathcal{R}_2 = \cdots = \mathcal{R}_n$, then the CA is uniform CA, otherwise non-uniform CA. Hence, uniform CA are special cases of non-uniform CA.

A CA contains *cyclic* and *acyclic* states. If a single state recursively (RMTs without any merging are self merged) evolved around itself we call it as a point state attractor. If a cycle has more than one state then we call it multi length cycle attractor. The state transition (sequence of states generated) diagram of a four cell CA is shown in Fig. 1, from where we conclude that the states 0100, 0101, 0110, 0001, 0011, 0111, 1001, 1010, 1000, 1011, 1111, 1101 are non-reachable states, whereas 1110 and 1100 are having more than one predecessor [3] as well as they are in a multi length cycle attractor. States 0000 is point state attractor.

Present state: (RMT)	111 (7)	110 (6)	101 (5)	100 (4)	011 (3)	010 (2)	001 (1)	000 (0)	Rule
(i) Next State:	1	1	0	0	1	1	0	0	204
(ii) Next State:	1	1	1	1	0	0	0	0	240
(iii) Next State:	0	0	1	1	0	0	0	0	48
(iv) Next State:	0	0	0	0	0	0	0	0	0

(a)

i^{th} RMT	$(i+1)^{th}$ RMTs
0 or 4	0, 1
1 or 5	2, 3
2 or 6	4, 5
3 or 7	6, 7

(b)

Table 1: (a) Truth table for rule 204, 240, 48 and 0. (b) Relationship between RMTs of cell i and cell $(i+1)$ for next state computation

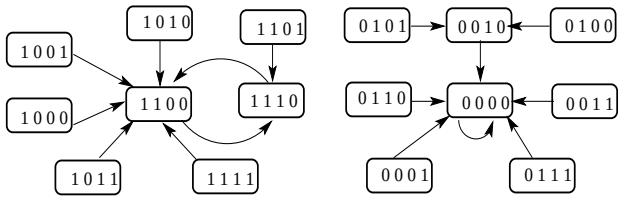


Figure 1: State transition diagram of CA $\langle 204, 240, 48, 0\rangle$

A CA state can be viewed as a sequence of RMTs, called RMT sequence (RS) [3]. To get an RS for a state, we consider a 3-bit window that slides over the state and contains a 3-bit binary value which is equivalent to RMT. Using this mechanism, we get relation between different RMTs (Table 1b). It tells that if i^{th} cell RMT is 2, then the RMT of $(i+1)^{th}$ cell is 4 or 5.

Definition 1 *Two RMTs* **sibling RMTs** *if they are resulted from an RMTs i. For example, RMTs 0 and 1 are sibling RMTs (Table 1b)*

Reachability tree: Reachability tree (RT) is a characterization tool for periodic boundary CA. It is a binary tree that represents the reachable states [2] of a CA. We can define reachability tree (RT) as set of nodes (N) and edges (E). Each edge (E)/ node (N) represents set of RMTs, where RMTs are divided into four groups. In case of root node, sibling RMTs are form single group.

The left edge of tree represent 0-edge, where as right edge represent 1-edge. For i^{th} level, each edge is constructed according to the RMTs of Rule R_{i+1}. Nodes of i^{th} level is constructed by successor RMTs of corresponding edge at $(i-1)^{th}$ level. Edge (node) of RT is denoted by $E_{i.j}$ ($N_{i.j}$) where i is

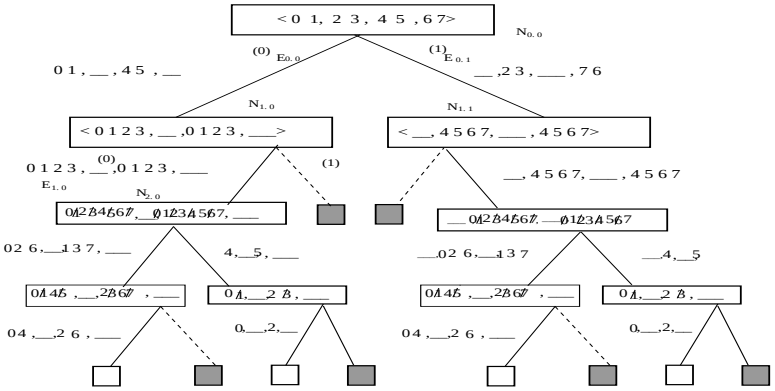


Figure 2: Reachability tree of PBCA $\langle 121, 192, 12, 224\rangle$

level index and j is the j^{th} edge (node) of i^{th} level, value of j varies from 0 to $2^i - 1$. If any edge (node) of reachability tree does not contain any RMT, then call that edge (node) as non-reachable edge (node). For example $E_{1.1}$ ($N_{2.1}$) and $E_{1.2}$ ($N_{2.2}$) is non-reachable edge (node) of Fig. 2.

At $(n-2)^{th}$ level we discard odd RMTs from first two sets (first set and second set) and discard even RMTs from last two sets (third set and fourth set) at each node. For example in 2^{nd} level of Fig. 2 in node $N_{2.1}$ RMTs 1, 3, 5 and 7 from first set and RMTs 0, 2, 4 and 6 from fourth set are dropped. In $(n-1)^{th}$ level, we allow those RMTs only in a set which is capable to produce RMTs at corresponding set number in edge of first level. For any node of $(n-1)^{th}$ level in first group only RMTs 0 and 4 can reside. Whereas, in second group only RMTs 1 and 5, in third group only RMTs 2 and 6 and in fourth group only RMTs 3 and 7 can reside respectively. For example, in $N_{3.1}$ we have four RMTs 0, 1, 4 and 5 in first set. Whereas at first set, in $E_{1.1}$ we have two RMTs 0 and 1. Between RMTs 0, 1, 4 and 5 only RMTs 0 and 4 is capable to generate RMTs 0 and 1, so RMTs 1 and 5 will be discarded.

III Processing of reachability tree

Reachability tree does not explicitly depict the transitions of states like state transition diagram Fig. 1b. But the transitions of states can be traced from the reachability tree. We know, in the tree, a sequence of edges from root to a leaf node associates a reachable state and at least one RMT sequence which corresponds to the predecessor of the state. To process the reachability tree, we first find the predecessors RMT of each RMT and then make the links.To decide whether an automaton contains only point states, we process corresponding RT to count cyclic states. Now, to count cyclic states firstly we process the reachability tree of CA and remove the non-reachable states. Then identify new non-reachable states. To implement this procedure we

introduce the concept of *linkcount*, which indicates that how many time we have to process an RMT.

Definition 2 *At i^{th} level, an RMT r has linkcount n implies, at $(i-1)^{th}$ level n number of predecessor RMTs or link exist of RMT r.*

Definition 3 *An RMT $x0y$ ($x1y$) is said to be self replicating if RMT $x0y$ ($x1y$) is 0 (1).*

For example, RMTs 0, 1, 6 and 7 of rule 240 is self replicating, whereas RMTs 2, 3, 4 and 5 of rule 240 are not self replicating (Table 1a). A state refers to a point state attractor if all the RMTs of the RS are self replicating.

We consider in the tree processing that all the cyclic states form point state attractors. So, only self replicating RMTs are to be present in the finally processed tree. If an RMT is self replicating then we call this position of the RMT on this edge of the tree is *stable*. During processing, if an RMT is not in its stable position, we make a link of the RMT from its current position (edge) to stable position (edge). We call such implication of an RMT as *merging*. The merging of an RMT depends on the RMT value and the merging of its predecessor RMT.

Based on position of merged RMT and destination edge, we have three type of merging. If any RMT appears in any edge $E_{i.j}$ and merged to other edge $E_{i.k}$ and $j < k$ ($j > k$), then we call the merging as forward (backward) merging. If any RMT appears and merges in same edge $E_{i.j}$, then we call the merging as self merging.

IV Identification of point state attractors in PBCA

Firstly we try to identify point states of PBCA. A point state occurs when corresponding RMTs of RS are self replicating. Self replicating RMTs help to create self merging, sequences of self merging from root level to leaf level results point state attractor. Using the concept of merging, we can identify the point states where the self merging RMTs are allowed to stay, and rest are discarded. In this way, we process the reachability tree from root node to leaf nodes. At leaf level, if some nodes are left, we consider them as point state attractors. Following algorithm counts the number point states in a given PBCA.

Algorithm 1 IdentifyPointAttractor
Input*: $\langle \mathcal{R}_1, \cdots, \mathcal{R}_i, \cdots, \mathcal{R}_n \rangle$ (n-cell CA).*
Output*: number of point attractors.*
Step 1*: Repeat Step 2 to Step 4 for $i = 1$ to n*
Step 2*: Identify the left and right edges of each node for i^{th} level (based on R_i).*

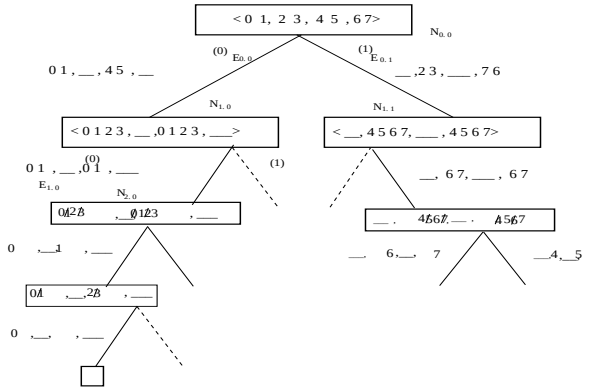


Figure 3: Tree for attractors

(i) *Form a link from the edge of an RMT to the edge which is the stable position of the RMT.*

Step 3*: Check at i^{th} level any self merging is there ?*

If yes, then goto step 4.

If No, then Report CA does not contain any point attractor.

Step 4*:*(i) *Store only self merged RMTs.*

(ii) *Create nodes for $(i+1)^{th}$ level with only successor RMTs of self merged RMT at i^{th} level.*

Step 5*: If $(i = n)$ then,*

count total reachable state and Report the number and state(s) are as point attractor.

Step 6*:End.*

Performance Analysis: Merging depends on $(i-1)^{th}$ level and i^{th} level (except root). But in this algorithm, in i^{th} node we process those RMTs which predecessor RMT at $(i-1)^{th}$ edge was self merged and others RMTs are discarded. So, knowledge about the present level is enough in here and therefore the complexity of the Algorithm 1 is linear.

Example 1: Fig 3 extracts point states from Fig. 2 using Algorithm 1. At root level RMTs 0 to 8 all are self merged, so in Fig.3 these RMTs stay. We store the successor RMTs of RMTs 0 to 8 for next level. In the 1^{st} level successor RMTs 0, 1, 6 and 7 are self merged, so we preserve them. In the 2^{nd} level, we get only one effective edge, namely $E_{2.0}$. In $E_{2.0}$ edge RMT 0 (first set) and RMT 1 (third set) are eligible. At the 3^{rd} level, as per periodic boundary condition we get only RMT 0 (first set) in the edge $E_{3.0}$ which is self merged. In Fig. 3, from root level to leaf level we have only one sequence of RMTs that are self merged. So we get only one point state attractors in the CA $\langle 204, 240, 48, 0\rangle$.

V PBCA having only point states

To identify only point state PBCA, first we count cyclic state(s), then it compare the result with result of Algorithm 1 which gives number of point state attractors. If number of cyclic states is more than point states, then the CA contains multi length cycle attractor(s), otherwise it contains only point states.

Using the concept of *linkcount* (discussed in Section III), in first and second level all RMTs have *linkcount* 1. From third level to $(n-1)^{th}$ level, *linkcount* depends on number of predecessors or links at $(i-1)^{th}$ level for i^{th} level. *linkcount* decrease by 1, if RMT is link to a non-reachable edge. Find the existing non-reachable state and then find new non-reachable state in the same level using *linkcount* and continuously using this process until two consecutive count of non-reachable states are equal to leaf level of tree. So, at the end of leaf level, we conclude that the number of cyclic states is equal to number of reachable edge (node). Now, we present an algorithm to calculate the number of cyclic states of CA. Input of the algorithm is CA rule vector and output of the algorithm is number of cyclic states.

Algorithm 2 FindNoOfCyclicStateForPBCA
Input*: $\langle \mathcal{R}_1, \cdots, \mathcal{R}_i, \cdots, \mathcal{R}_n \rangle$ (n-cell CA).*
Output*: number of cyclic states.*
Step 1*: Form the root and identify its left and right edges following $\mathcal{R}_1$).*
For each RMT, set linkcount $\leftarrow 1$.

(i) *Form a link from the edge of an RMT to the edge which is the stable position of the RMT.*

(ii) *If any RMT does not exist in an edge, the edge is marked as a non-reachable edge.*

Set oldcount $\leftarrow$ *number of non-reachable edge* $\times 2^{n-1}$
Step 2*: Repeat Step 3 to Step 6 for* $i = 2$ *to* n
Step 3*: Get the edges of reachability tree for level* $(i-1)$.

(i) *Get the edges at level (i) using* $\mathcal{R}_i$.

(a) *If level (i) equal to* $(n-2)$ *then,*

We discard odd RMTs from first two set and discard even RMTs from last two set.

(b) *If level (i) equal to* $(n-1)$ *then,*

In first group only RMTs 0 and 4 can reside. Whereas, in second group, third group and fourth group only RMTs 1 and 5, RMTs 2 and 6, and RMTs 3 and 7 can reside respectively.

(ii) *If no RMT exists in an edge, the edge is marked as a non-reachable edge.*

Step 4*: For each RMT (r) on each edge of level (i),*

(i) *identify number of RMTs (p) of level* $(i-1)$ *that derive* r.

(ii) *set linkcount (r) sum of linkcount (p).*

Step 5*:* (i) *Find stable position of the RMT (r) on an edge and make a link from the current edge to the final edge.*

(ii) *If the stable position does not exist on the tree (non-reachable edge), decrement linkcount (r) by 1.*

(iii) *If linkcount*$(r) = 0$*, then mark the RMT (r) as non-reachable for its current edge position.*

(iv) *If each RMT of an edge is non-reachable mark that edge as non-reachable edge.*

Step 6*: Set newcount* $\leftarrow$ *number of non-reachable edges* $\times 2^{n-i}$.

If newcount $\neq$ oldcount*, then*

(i) *set* oldcount = newcount*.*

(ii) *Repeat Step 5 and Step 6.*

Step 7*: Report number of cyclic state* $\leftarrow$ *(*2^n *- oldcount).*

Step 8*: Calculate number of point attractors using the Algorithm 1.*

Step 9*: If number of cyclic state > number of point attractor*

Report the CA is multi length cycle CA.

Otherwise, Report the CA is only single length cycle CA.

Performance Analysis: The performance of Algorithm 2 depends in which level we encounter first non-reachable state and number of cyclic states. If the cyclic states are exponential, time and space complexity become exponential. However the multi attractor CA, which are targeted to application like pattern classification , are having a limited number of attractors. For such Algorithm 2 is excellent. Using tree merging technique further help to develop a set of rules with specific arrangement technique by which we design only point state PBCA in linear time complexity.

Since the cyclic states can only form the attractors, the number of attractors are less than or equal to the number of of cyclic states (which is the output of the Algorithm 2). If all the attractors are point states, then the number of cyclic states is equal to the number of point states. Using Algorithm 1, we can answer whether all attractors are of cycle length 1.

Example 2: Considering, the CA $\langle 204, 240, 48, 0\rangle$ in Fig 4 the root level are same as figure of reachability tree 2. At the 1^{st} level, we discard RMT 2, 3 merged in $E_{1.1}$ and RMT 4, 5 merged in $E_{1.2}$ which are non-reachable edges. In the 2^{nd} level, we discard RMT 2,3 merged in $E_{2.1}$ which is non-reachable. In the 3^{rd} level RMT 2, 3 and 7 are discarded as they merged to non-reachable edge $E_{3.1}$, $E_{3.13}$ and $E_{3.15}$ respectively. In the leaf level of Fig 4, we can get only three node, so the result of applying the CA $\langle 204, 240, 48, 0\rangle$ on Algorithm 2 is three cyclic states. In the CA $\langle 204, 240, 48, 0\rangle$ the number of cyclic state (using Algorithm 2) is more than point state (using Algorithm 1), so the CA contain multi length cycle attractor.

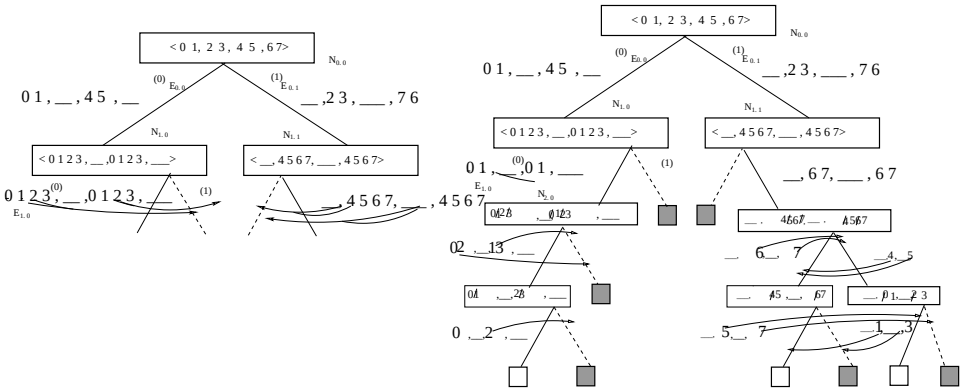


Figure 4: Tree merging for getting cyclic states (RMTs without any merging are self merged)

VI Conclusion

Here we have covered some aspects of non-uniform periodic boundary CA having only point states. To do this we take help of tree merging technique. Using the concept of tree merging, an algorithm for identifying point states attractors from CA state space is reported. Another algorithm is also reported to count cyclic states in a CA state space. Using these two algorithm we can identify a PBCA having only point states.

References

[1] P Pal Chaudhuri, D Roy Chowdhury, S Nandi, and S Chatterjee. *Additive Cellular Automata – Theory and Applications*, volume 1. IEEE Computer Society Press, USA, ISBN 0-8186-7717-1, 1997.

[2] Sukanta Das and Biplab K. Sikdar. Characterization of 1-d periodic boundary reversible ca. *Electr. Notes Theor. Comput. Sci.*, 252:205–227, 2009.

[3] Nazma Naskar, Avik Chakraborty, Pradipta Maji, and Sukanta Das. Analysis of reachability tree for identification of cyclic and acyclic ca states. In *ACRI*, pages 63–72, 2012.

[4] Nazma Naskar, Sukanta Das, and Biplab K. Sikdar. Characterization of nonlinear cellular automata having only single length cycle attractors. *J. Cellular Automata*, 7(5-6):431–453, 2012.

[5] Stephen Wolfram. *Cellular Automata and Complexity*, chapter 4, pages 159–202. Westview Press, 2002.

Complexity Analysis in Cyclic Tag System Emulated by Rule 110

Shigeru Ninagawa
Kanazawa Institute of Technology, Ishikawa, Japan.
University of the West of England, Bristol, United Kingdom.
shigeruninagawa@gmail.com

Genaro J. Martínez
Departamento de Ciencias e Ingeniería de la Computación,
Escuela Superior de Cómputo, Instituto Politécnico Nacional, México, D. F.
University of the West of England, Bristol, United Kingdom.
genaro.martinez@uwe.ac.uk

August 17, 2013

Abstract

It is known that elementary cellular automaton rule 110 is capable of supporting universal computation by emulating cyclic tag system. Since the whole information necessary to perform computation is stored in the configuration, it is reasonable to investigate the complexity of configuration for the analysis of computing process. In this research we employed Lempel-Ziv complexity as a measure of complexity and calculated it during the evolution of emulating cyclic tag system by rule 110. As a result, we observed the stepwise decline of complexity during the evolution. That is caused by the transformation from table data to moving data and the elimination of table data by a rejector.

1 Introduction

Cellular automaton (CA) is a model of information processing system. Since CAs have no memory except for cell, all the information necessary to perform computation is stored in its configuration. That means the complexity of configuration is in some way related with the complexity of information the CA is processing. Therefore it is reasonable to investigate the complexity of configuration for the analysis of computing process by CA. Elementary CA (ECA) rule 110 is supporting universal computation [1]. In this research we focus on the complexity of configurations during the computing process by rule 110. In the next section, we make a brief explanation of cyclic tag

system emulated by rule 110. The results of complexity analysis of cyclic tag system are shown in section 3. Finally we discuss the results and a future plan.

2 Cyclic Tag System by Rule 110

The transition function of ECA rule 110 is given by:

$$\frac{111}{0}\,\frac{110}{1}\,\frac{101}{1}\,\frac{100}{0}\,\frac{011}{1}\,\frac{010}{1}\,\frac{001}{1}\,\frac{000}{0}.$$

The upper line represents the state of the neighborhood and the lower line specifies the state of the cell at the next time step. Cook proved the computational universality of rule 110 by showing that rule 110 can emulate cyclic tag systems [1].

A cyclic tag system works on a finite tape which is read from the front and appended to based on what is read. An appendant is cyclically chosen from the appendant table. The alphabet on the tape consists of $\{0, 1\}$. At each step, the system reads one character and deletes it, and if that is '1', then it appends the appendant, while an '0' causes the appendant to be skipped. At the next step, the system moves on to the next appendant in the table. The system halts if the word on the tape is empty. For example, the transition of the initial word '1' with the appendant table (1, 101) is given as following:

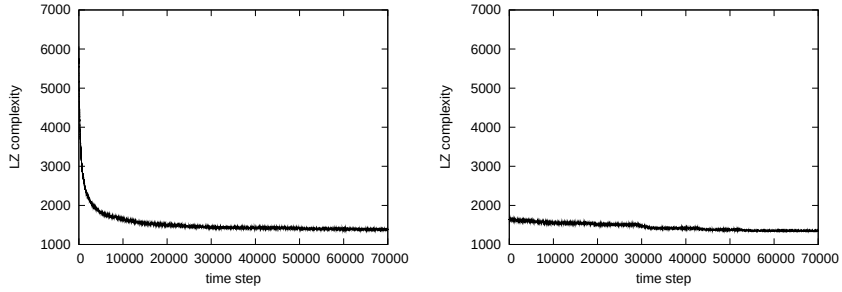
$$1 \vdash 1 \vdash 101 \vdash 011 \vdash 11 \vdash 11 \vdash 1101 \vdash \cdots .$$

A detailed explanation of the emulation of cyclic tag systems by rule 110 is in Ref. [7].

3 Complexity Analysis

As a measure of complexity, we focus on the compressibility of configuration. Compression-based CA classification was implemented by Zenil [2] using the DEFLATE algorithm [3]. We use Lempel-Ziv (LZ) complexity used in the data compression algorithm called LZ78 [4]. In LZ78, a string is divided into phrases. Given a string $s_1 s_2 \cdots s_k s_{k+1} \cdots$ where a substring $s_1 s_2 \cdots s_k$ has already been divided is constructed by searching the longest substring $s_{k+1} \cdots s_{k+n} = w_j$, $(0 \leq j \leq m)$ and by setting $w_{m+1} = w_j s_{k+n+1}$ where $w_0 = \epsilon$. The LZ complexity of the string is defined as the number of divided phrases. The complexity analysis based on LZ78 was applied to the study of the parity problem solving process by rule 60 [5].

In this research, we calculate the LZ complexity of configuration at each time step during the computing process of cyclic tag system emulated by rule 110. Figure 1 shows the evolution of LZ complexity staring from a



]

Figure 1: Evolution of LZ complexity staring from a random configuration (left) and from the one designed to emulate cyclic tag system (right).

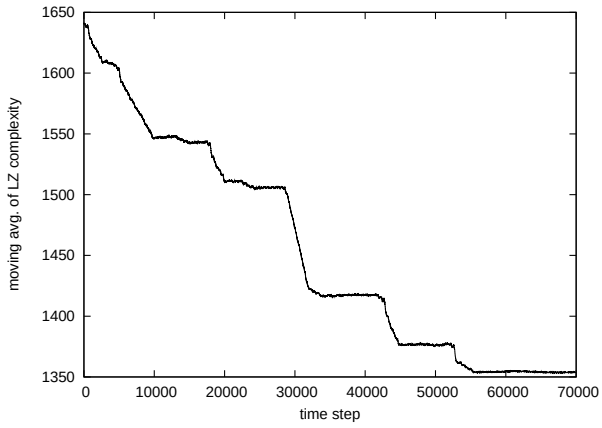


Figure 2: Moving average of LZ complexity in the evolution of cyclic tag system.

random configuration (left) and from the one designed to emulate cyclic tag system. The array size is 65900 in both cases. We made use of the initial configuration on the web site [6] from which you can download the file of initial configuration of rule 110 emulating the cyclic tag system exemplified in the previous section.

In the case of a random initial configuration, the LZ complexity starts with the value of 6068 and decreases quickly, meanwhile, it shows a more slighter decrease in the case of cyclic tag system emulation. To investigate the evolution of LZ complexity in the process of cyclic tag system emulation, we calculated the simple moving average of the data with period 100 and showed with a finer scale in Fig. 3. We can see the repetition of significant decline and temporary equilibrium in the evolution of LZ complexity. Figure 3, however, does not inform us about the regional difference of LZ complexity, because LZ complexity is a measure of complexity from a global

perspective. So we divide the array into 20 sections (section $0 \sim 19$ starting from the left) of 3295 cells and calculate the LZ complexity of each section individually. Figure 3 shows the evolution of the LZ complexity in section $2 \sim 17$.

By viewing the figures from section 2 to 11 in sequence, we can observe four bumps are moving from the left to the right. Each bump corresponds to the four packages of A^4 gliders that construct an appendant from a moving data. Those packages of A^4 gliders are called ossifier (4_A^4). The term in the parenthesis is the one according to the naming convention employed in Ref [7].

The initial configuration in section 16 contains several patterns such as tape data, table data or leader that separates packages of table data (Fig. 4). The group of these complicated structures makes the complexity high in this section. As time goes by, however, these patterns move to the left and there remains the periodic background called ether that causes the low value of complexity.

For the detailed analysis of section 14, we divide it into three parts of array size 1100 and calculated the moving average (period:100) of LZ complexity for each part during time step from $t = 10,000$ to $t = 50,000$ as shown in Fig. 5. The explanation in this paragraph is shown diagrammatically in Fig. 6. The right of Fig. 5 shows LZ complexity in the part of $x = 48,200 \sim 49,299$ (the index of the leftmost cell of the array is given by $x = 0$). As a moving data '0' (0Add_$\overline{E}$) is coming from the right, the LZ complexity starts increasing from $t = 20,000$. While the collision between the moving data '0' and an ossifier creates a tape data '0' (0Ele_C_2) from $t = 22,000$ to $t = 24,000$, the LZ complexity does not vary a lot because tape data do not move. The LZ complexity increases from $t = 25,000$ as a leader and three table data come from the left. When the leader collides with a tape data '0' at about $t = 27,000$, the LZ complexity reaches a maximum. While a rejector created by the collision is erasing table data as show in the left of Fig. 7, the LZ complexity decreases a lot from $t = 28,000$ to $t = 32,000$.

When the moving data '1' created in section 15 pass through the part of $x = 47,100 \sim 48,199$ to the left, the LZ complexity temporarily increase from $t = 25,000$ to $33,000$ as shown in the middle of Fig. 5. Finally the moving data '1' collides with an ossifier coming from the left and converts into tape data '1' as shown in the right of Fig. 5. That corresponds to the bump at $t = 28,000 \sim 37,000$ in the part of $x = 46,000 \sim 47,099$ shown in the left of Fig. 5. The sharp increase from $t = 32,000$ in the right of Fig. 5 is caused by leaders and table data moving from the right. These structures bring a same result from $t = 35,000$ in the middle of Fig. 5 and from $t = 39,000$ in the left of Fig. 5, as they move to the left. They collide with tape data '1' and convert into a moving data '1' that goes away to the left. That causes the sharp decrease from $t = 43,000$ to $t = 47,000$ in the

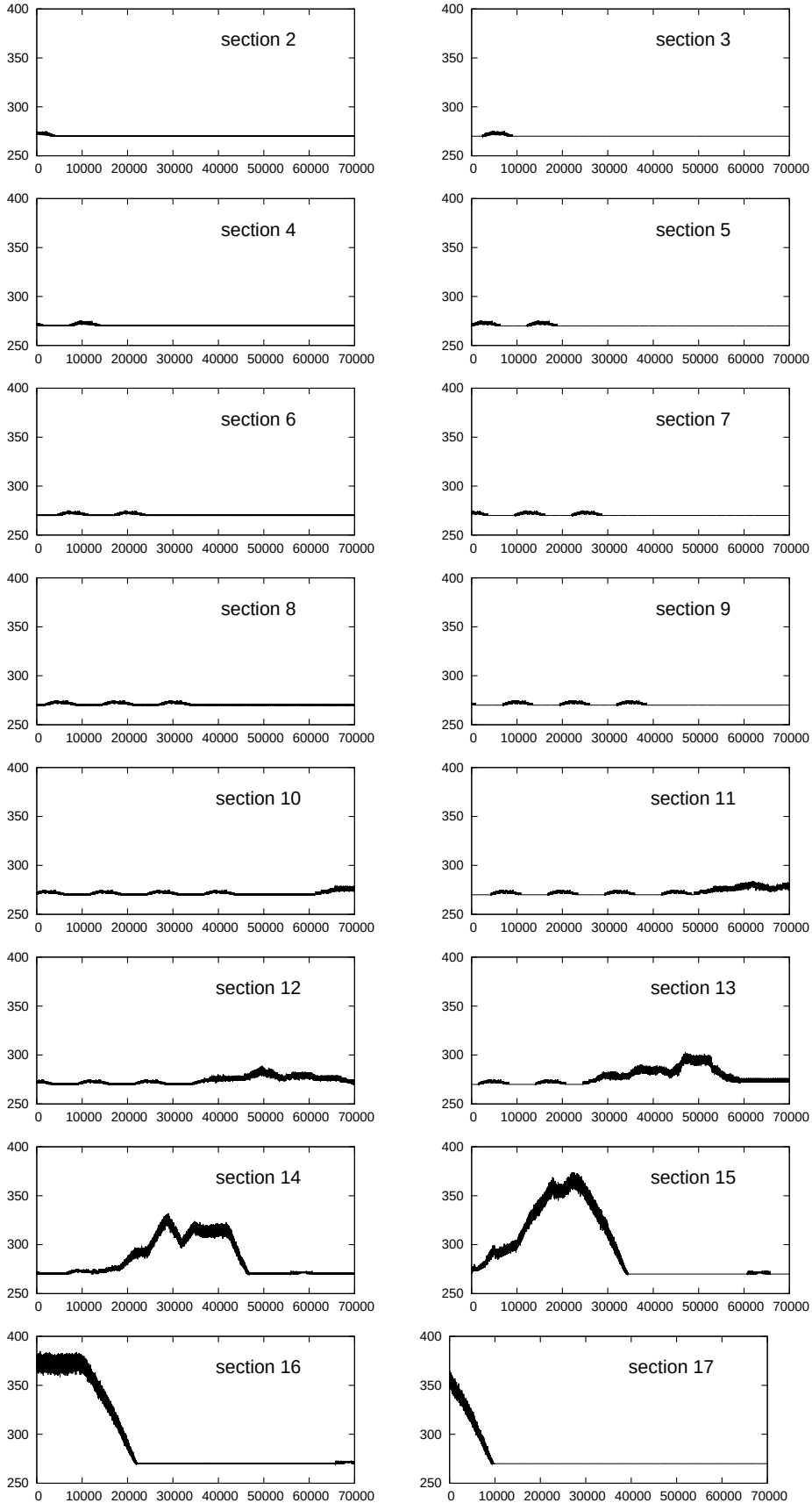


Figure 3: Evolution of LZ complexity in section 2 - 17. Vertical axis is LZ complexity and horizontal axis is time step.

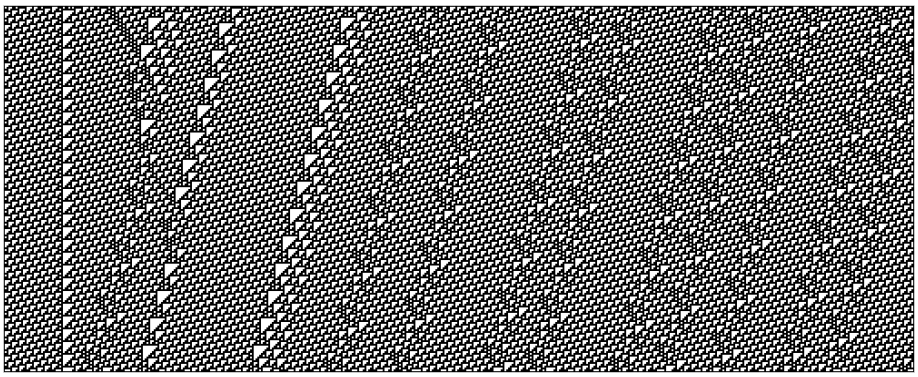


Figure 4: Space-time pattern of the leftmost part of section 16 in Fig.3 for the first 200 steps. Array size is 500. Starting from the left, there are the rightmost part of tape data '1' (1Ele_C_2), leader (SepInit_$E\overline{E}$), and the left part of table data '1' (1BloP_$\overline{E}$).

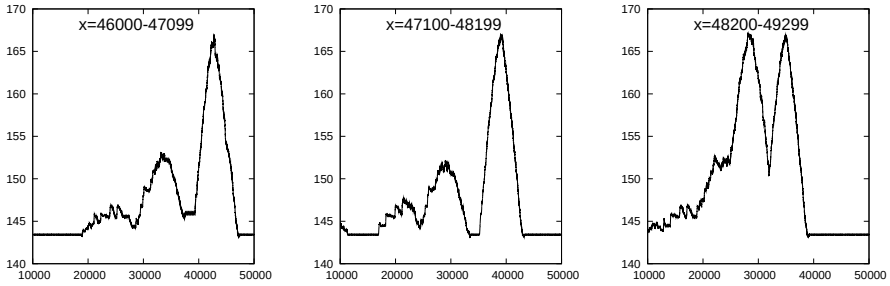


Figure 5: Moving average of the LZ complexity in the three parts of the section 14 in Fig.3 during time step from $t = 10,000$ to $t = 50,000$.

left of Fig. 5. Since there remains only ether after $t = 47,000$, the value of LZ complexity is low.

4 Discussion

As we explained in the previous section, by dividing the whole array into small parts, we can correspond the change of LZ complexity to the events occurring on the array such as incoming of propagating patterns, collision between several patterns, and outgoing of them. In particular the significant decline of LZ complexity is caused by a transformation from table data to moving data or an elimination of table data. Figure 8 shows the space-time pattern of table data '1' (1BloP_$\overline{E}$) (top) and moving data '1' (1Add_$\overline{E}$) (bottom). It is apparent that the pattern of table data '1' is more complicated than that of moving data '1'. Therefore the transformation from table data to moving data brings about the decline of LZ complexity. And the

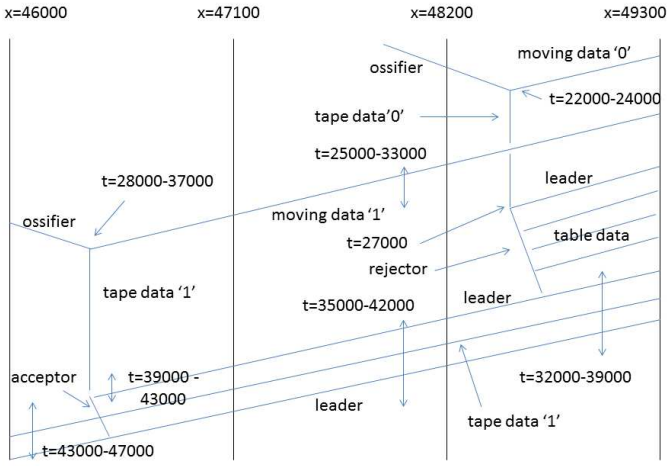


Figure 6: Diagram of patterns in section 14 in Fig.3. Time goes from top to bottom.

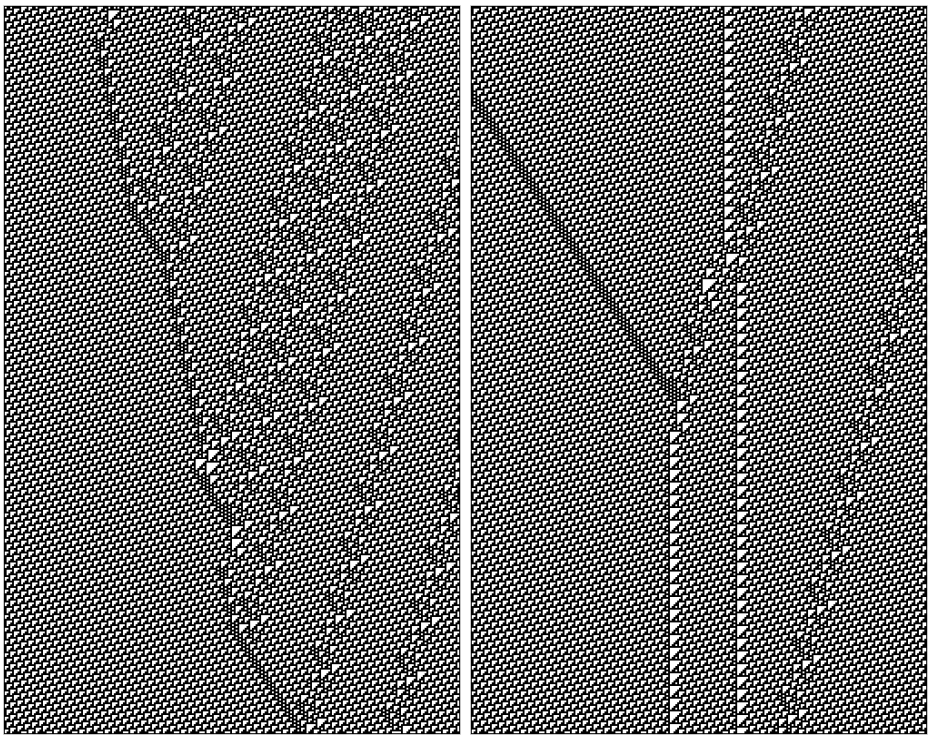


Figure 7: Space-time pattern of the part of section 14 in Fig.3. Left: A rejector is erasing table data. Right: Collision between an ossifier moving from the left and moving data from the right is constructing a table data. Both pictures show 400 steps of evolution of the area of 250 cells.

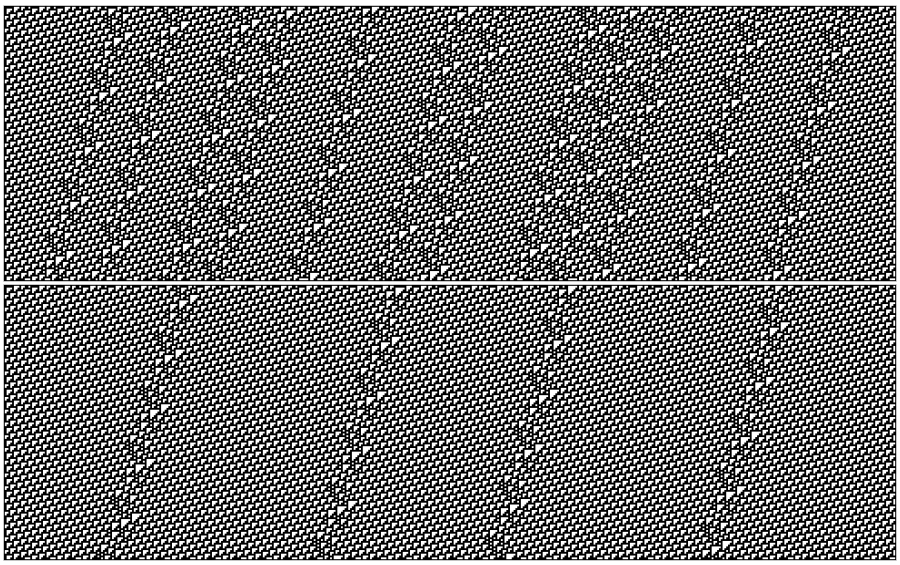


Figure 8: Space-time pattern of table data '1' (top) and moving data '1' (bottom). The array size is 490 in both cases.

elimination of table data by a rejector has the same result. It seems that the decline of LZ complexity observed in Fig. 3 is caused by the events of either of these two types.

In this research we employed the periodic boundary conditions. The ossifiers, the table data, and the leaders are built in advance in the initial configuration. They are consumed during the evolution and are not supplied from the outside. It is uncertain about how LZ complexity varies in time if they are supplied regularly and eternally. We are planning to employ "discharging" boundary conditions that can supply these patterns from the outside.

References

[1] Cook, M.: Universality in elementary cellular automata. Complex Systems 15, 1–40 (2004)

[2] Zenil, H.: Compression-based investigation of the dynamical properties of cellular automata and other systems. Complex Systems 19, 1–28 (2010)

[3] Deutsch, L. P.: DEFLATE compressed data format specification version 1.3. (May 1996) http://www.rfc-editor.org/rfc/rfc1951.txt

[4] Ziv, J., and Lempel, A.: Compression of individual sequences via variable-rate coding IEEE Transactions on Information Theory 24, 530–536 (1978).

[5] Ninagawa, S.: Solving the parity problem with rule 60 in array size of the power of two. J. Cellular Automata in press

[6] Martínez, G. J.: Elementary cellular automaton rule 110 http://uncomp.uwe.ac.uk/genaro/Rule110.html

[7] Martínez, G. J., McIntosh, H. V., Seck-Tuoh-Mora, J. C., Vergara, S. V. C.: Reproducing the cyclic tag system developed by Mattew Cook with rule 110 using the phase f_i_1. J. Cellular Automata 6, 121–161 (2011)

Experimental study on convergence time of elementary cellular automata under asynchronous update *

Biswanath Sethi, Souvik Roy, Sukanta Das
Department of Information Technology
Bengal Engineering & Science University, Shibpur
Howrah, West Bengal, India–711103
E-mails: sethi.biswanath@gmail.com, svkr89@gmail.com,
sukanta@it.becs.ac.in

Abstract

This paper investigates the convergence of elementary cellular automata (ECA) under fully asynchronous update. There are 146 ECA rules which converge to some fixed-point attractors. We have experimentally studied the convergence time of such ECA. For this experimental study, we have followed hybrid iterative refinement method of the empirical curve bounding technique. It is found that there are rules with six classes of convergence time, and these are $O(1)$, $O(\log(\log n))$, $O(\log n)$, $\mathrm{O}(n^{1/2})$, $O(n^2)$ and $O(2^n)$, where n is the number of cells in the ECA.

Keywords: Elementary cellular automata, asynchronous cellular automata (ACA), fixed-point, convergence time, empirical curve bounding technique.

I Introduction

This work concentrates only on the elementary cellular automata (ECA). Here, we explore a set of ECA states, called fixed-point attractors, towards which the neighboring states asymptotically approach in the course of dynamic evolution. We experimentally study the rate of growth of convergence time of these ECA under asynchronous update. As per our knowledge very few target have been fixed in this direction.

The study of convergence property of ECA under asynchronous update has been initiated in [1, 2, 4]. However, most of the works on convergence

*This work is supported by DST Fast Track Project Fund (No. SR/FTP/ETA-0071/2011)

Table 1: Look-up table for rule 64, 88 and 251

Present state :	111	110	101	100	011	010	001	000	*Rule*
(*RMT*)	(7)	(6)	(5)	(4)	(3)	(2)	(1)	(0)	
(i) Next state :	0	1	0	0	0	0	0	0	64
(ii) Next state :	0	1	0	1	1	0	0	0	88
(iii) Next state :	1	1	1	1	1	0	1	1	251

have concentrated on two-dimensional ACA. A probabilistic study on the convergence and their convergence time of ECA under asynchronous update is provided in [3]. However, the work has dealt with only a set of CA (64 out of 256 ECA).

In this scenario, this paper targets to extensively study the convergence property and convergence time of ECA under asynchronous update. We study the rate of growth of convergence time with respect to the size of automaton. The work concentrates on all the 256 ECA rules. Like [3, 5], we have also considered that in each discrete time step, only one arbitrary cell is updated. Hereafter, by ACA we shall refer ECA under asynchronous update (that is, one arbitrary cell is updated in a single step). We have designed one theorem for the characterization of ACA. From the theorem we identify 146 ACA (out of 256 ACA) rules that converge to some fixed-point attractors during their dynamic evolution. For the experimental study of convergence time, we have used the empirical curve bounding method and guess the rate of growth of convergence time of the characterized ACA [6]. An algorithm is designed to setup the environment for experimentation and find the average convergence time of the characterized ACA. We have identified six different convergence time with respect to the size of ACA and these are $O(1)$, $O(\log(\log n))$, $O(\log n)$, $\mathrm{O}(n^{1/2})$, $O(n^2)$ and $O(2^n)$ where, n is the number of cells of ACA. ACA (146 ACA) with their convergence time are listed in Table 2.

II Definitions

The one-dimensional two state 3-neighborhood cellular automata are commonly known as elementary cellular automata (ECA) [10]. In asynchronous update, the cells are considered as independent and so, updated independently. We have considered here that a single arbitrary cell is updated in each time step. Such CA are referred as *fully asynchronous* CA in [3]. Three such rules (64, 88 and 251) are shown in Table 1. We referred, each column of the first row of Table 1 as Rule Min Term (RMT).

Asynchronous update of ECA is studied in [5, 8]. We denote the cell, updated at time t as u_t. Therefore, we can get an *update pattern*, $U = \langle u_1, u_2, \cdots, u_t, \cdots \rangle$ which notes the cells, updated in different time [7]. With

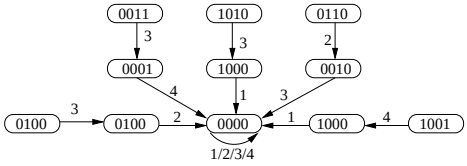


Figure 1: Partial state transition diagram of 4 cells rule 64 ACA.

the update pattern, initial state and the ACA rule, one can determine the state transition for the ACA (Figure 1). The cells updated during state transition are noted over arrows.

Definition 1 *An RMT r of a rule R is active if an ACA cell flips its state (1 to 0 or 0 to 1) on r. Otherwise, the RMT r is passive.*

For example, RMT 0 of rule 251 (see Table 1) is active. Because, while a cell is acting on that particular RMT, the cell's present state is 0 and next state of the cell for the rule is 1. Hence, transition occurs. So, it can be said that if the middle bit of an RMT is unequal with the RMT value, the RMT is active. RMT 3 of rule 251, on the other hand, is passive.

Definition 2 *A fixed-point attractor is an ACA state, next state of which is the state itself for any style of update of cells. That is, if an ACA reaches to a fixed-point attractor, the ACA remains in the state forever.*

In Figure 1, the state 0000 is a fixed-point attractor for rule 64 ACA. The RMT sequence for the state 0000 is $\langle 0000 \rangle$. The next state of 0000 is always 0000 for the update of any cell in any sequence. The RMT 0 is passive for rule 64. It can be observed that the RMTs in an RMT sequence of the state of a fixed-point attractor are to be passive. Hence, we get the following lemma.

Lemma 1 *Rule R ACA forms a fixed-point attractor with state S if the RMTs of R that are present in the RMT sequence of S are passive.*

III Identification of convergent ACA rules

Following theorem states the condition of ECA that converges to some fixed-point attractors under asynchronous update. Due to page constrain, the proof of the theorem is omitted.

Theorem 1 *Rule R ACA converges to fixed-point attractor for at least one update pattern if one of the following condition is verified [9]:*
(i) RMT 0 (RMT 7) of R is passive and RMT 2 (RMT 5) is active.
(ii) RMTs 0, 1, 2 and 4 (RMTs 3, 5, 6 and 7) are passive and RMT 3 or 6 (RMT 1 or 4) is active.
(iii) RMTs 1, 2, 4 and 5 (RMTs 2, 3, 5 and 6) are passive.

Table 2: Convergence time of ACA

Rate of growth	ACA
$O(1)$	204
$O(\log(\log n))$	0, 4, 5, 8, 12, 13, 32, 64, 68, 69, 72, 76, 77, 79, 93, 94, 128, 132, 133, 160, 164, 200, 205, 207, 218, 221, 222, 223, 232, 235, 236, 237, 239, 250, 251, 253, 254, 255
$O(\log n)$	2, 10, 16, 18, 24, 34, 36, 42, 40, 44, 48, 50, 56, 66, 78, 80, 92, 95, 96, 98, 100, 104, 112, 130, 136, 140, 141, 144, 162, 168, 171, 172, 175, 176, 179, 183, 185, 186, 187, 189, 190, 191, 192, 196, 197, 202, 203, 206, 216, 217, 219, 220, 224, 227, 228, 234, 238, 231, 233, 241, 242, 243, 245, 246, 247, 248, 249, 252
$O(n^{1/2})$	26, 58, 74, 82, 88, 106, 114, 120,163, 167, 169, 173, 177, 181, 225, 229
$O(n^2)$	138, 146, 152, 170, 174, 178, 182, 184, 188, 194, 208, 226, 240, 244, 230
$O(2^n)$	90, 122, 154, 161, 165, 166, 180, 210

Example 1 *Let us consider the rule 36 ACA, in which RMTs 0, 1, 2 and 4 are passive. Therefore, the rule satisfies the condition for fixed-point attractor for at least one update pattern (Theorem 1 (ii)). Here, we assume number of cells is 4 and the initial state is 1111. Since the cells are updated arbitrary, we may get the update pattern* $\langle 1, 4, 4, 2, \cdots \rangle$. *In this case the ACA reaches to a fixed-point attractor. The detail transitions are:* $1111(1) \rightarrow 0111(4) \rightarrow 0110(4) \rightarrow 0110(2) \rightarrow 0010$ *(the cell updated in a step is noted in bracket).*

There are 64 rules where the RMT 0 is passive and RMT 2 is active and 64 rules where RMT 7 is passive and RMT 5 is active (Theorem 1(i)). Similarly, identifying this way theorem 1 dictates that there are 146 ACA out of 256 (see second column of Table 2), which converge to some fixed-point attractors during their dynamic evolution.

III.1 Finding average convergence time

The ultimate goal of calculating the average convergence time of ACA, would be to find the closed form expressions for the run time in terms of input CA size. Since this is so complicated, we have estimated the average convergence time (asymptotic performance) depending upon small number of inputs which are usually represented in big-oh notation [6].

To calculate the average convergence time for an ACA, the convergence time for all possible states are considered. There are 2^n possible states for an n-cell ACA. So, average convergence time can be, $\overline{X_{2^n}} = \frac{\sum_{i=1}^{2^n} X_i}{2^n}$, where X_i is the convergence time for i^{th} initial state. For large size of ACA, it is difficult to calculate the average convergence time, considering all the possible states. So, we choose m states out of total 2^n possible states randomly. Hence, the estimated average convergence time is, $\widehat{\overline{X_1}}=\frac{\sum_{i=1}^{m} X_i}{m}$, where m is the number of samples and X_i is the convergence time of i^{th} random initial state. Here, $\overline{X_{2^n}}$ and $\widehat{\overline{X_1}}$ may not be close as we have taken only m states out of 2^n states, and generally $m \ll 2^n$.

To get better estimation, we take again m samples randomly from total state space and calculate the average convergence time, considering total $2m$ states. Hence, $\widehat{\overline{X_2}}=\frac{1}{2}$ $(\widehat{\overline{X_1}}+\frac{1}{m}\ \sum_{i=1}^{m} X_i)$. Similarly, we can get $\widehat{\overline{X_3}}$, $\widehat{\overline{X_4}},\cdots,\widehat{\overline{X_k}}$, where,

$$\widehat{\overline{X_k}} = \frac{k-1}{k}(\widehat{\overline{X_{k-1}}} + \frac{1}{m}\sum_{i=1}^{m} X_i) \tag{1}$$

This is obvious that the series $\widehat{\overline{X_1}}$, $\widehat{\overline{X_2}}$, $\cdots$, $\widehat{\overline{X_k}}$, $\cdots$ converges to $\widehat{\overline{X_{2^n}}}$. However, we declare that $\widehat{\overline{X_k}}$ is the estimated convergence time, if $|\widehat{\overline{X_k}}-\widehat{\overline{X_{k-1}}}| < \delta$, where δ is a very small number. To find the estimated convergence time, we have experimented all 146 ACA considering the value of δ as 0.0001. We designed an algorithm, reported next, to find the estimated convergence time.

Algorithm 1 *: Convergence time*
Input: n (Size of ACA), Rule R ACA from Table 2, m
Output: Average convergence time
Step 1: Initialize: Time $\longleftarrow$ *0*
Step 2: Generate an n-bit binary string randomly and consider it as the initial ACA state
Step 3: Select an ACA cell randomly, and update that cell
Step 4: Repeat step 3 for m times
Step 5: Time $\longleftarrow$ *Time+1*
Step 6: If the ACA state is not a fixed-point, go to Step 3
Step 7: Repeat Step 2 to Step 6 for m times
Step 8: avg_time $\longleftarrow$ *(time/m)*
Step 9: Repeat Step 1 to Step 8 to get a new average time, $Newavg\_time$
Step 10: If $|avg\_time - Newavg\_time|$ *is less than some given threshold report avg_time as the output.*
Step 11: avg_time $\longleftarrow$ $\frac{avg\_time+Newavg\_time}{2}$
Step 12: go to Step 9

III.2 Method of finding rate of growth

It explains the method which we use to find the rate of growth of convergence time of ACA, using the results obtained from Algorithm 1. To find the rate of growth of convergence time of ACA, we have followed the empirical curve bounding technique [6]. Assuming the execution time (t) follows power rule, that is, t $\approx kn^a$, the coefficient a can be found by taking empirical measurements of run time $\{t_1, t_2\}$ at some input size $\{n_1, n_2\}$, and calculating $t_2/t_1=(n_2/n_1)^a$. So,

$$a = \frac{log(t_2/t_1)}{log(n_2/n_1)} \tag{2}$$

We experimented all 146 ACA, using the Algorithm-1 several times varying the size of ACA. A series of average convergence time are obtained from the algorithm for a particular rule. We find a series of co-efficient (a) using equation 2 considering two consecutive average convergence time resulted from Algorithm 1. From these several values of a, we find the rate of growth convergence time of ACA applying the hybrid iterative refinement method of the empirical curve bounding technique [6]. From the experimentation, we also observed that after a certain CA size the growth rate of convergence time always lies under some upper bound . As per the definition of big-oh (O) notation, for a given function g(n), T(n)=O(g(n)), there exist positive constant c and n_0, such that $0 \leq$ T(n)$\leq$ c g(n), for all n$\geq n_0$. As the average convergence time of the ACA satisfies the definition of big-oh, so we represent the rate of growth of convergence time of ACA in big-oh notation. Finally, to validate our guess for convergence time, we also plot the graph for rate of growth with respect to ACA size for each class of convergence time.

IV Experimental results

Here, we discuss about various experimental results and classes of convergence time of ACA. The co-efficient a (rate of growth) is calculated using the equation 2, considering two consecutive average convergence time resulted from the experimentation (Algorithm 1). The rate of growth of convergence time of ACA, remain constant after a certain ACA size.

The average convergence time for rule 204 ACA is constant as all the RMTs for rule 204 ACA are passive. Hence, the convergence time for rule 204 ACA is $O(1)$. The rest 145 ACA rules are categorized in 5 classes based on their upper bound of convergence time. All 146 ACA are listed with their rate of growth of convergence time in Table 2.

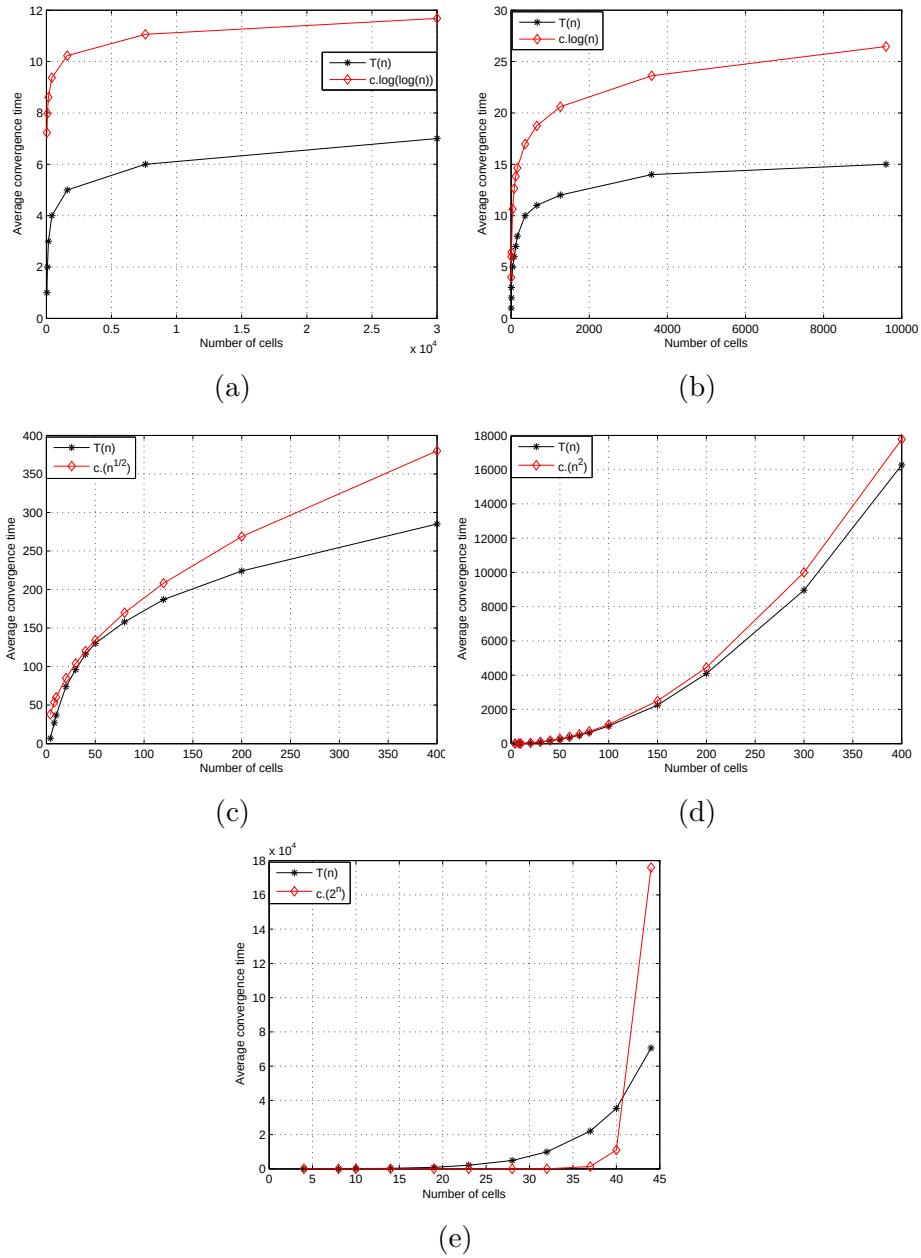


Figure 2: Rate of growth of ACA (a) for rule 76 (b) for rule 78 (c) for rule 169 (d)for rule 240 (e) for rule 90

IV.1 Time: $O(\log(\log n))$

There are 38 rules whose convergence time is O(log(log n)). We calculate the rate of growth of convergence time taking samples from the experimentation. We have calculated the rate of growth (value of a) using equation 2 considering two consecutive average convergence time obtained from the Algorithm 1 and plot the graph of rate of growth of convergence time. Figure 2(a) shows the rate of growth of convergence time for rule 76 ACA.

Example 2 *Consider the rule 76 ACA. The rate of growth of convergence time for this rule satisfies the condition for big-oh. Hence, we can apply* $T(n) \leq c(\log(\log n))$*, for* $c=3$ *and* $n \geq 40$*, where* $T(n)$ *is rate of growth of convergence time. So, we can represent the rate of growth of convergence as* $O(\log(\log n))$ *(see Figure 2(a)).*

IV.2 Time: $O(\log n)$, $O(n^{1/2})$, $O(n^2)$ and $O(2^n)$

68, 16, 15 and 8 rules are identified with $O(\log n), O(n^{1/2}), O(n^2)$ and $O(2^n)$ convergence time respectively (Table 2). The average convergence time is calculated considering the samples resulted from the experimentation using the empirical curve bounding method [6]. Figure 2(b), 2(c), 2(d), 2(e) shows the rate of growth of convergence time for rule 78 ACA of class $O(\log n)$, rule 169 of class $O(n^{1/2})$, rule 240 of class $O(n^2)$ and rule 90 of class $O(2^n)$ respectively.

V Discussion

This paper has reported an experimental study on convergence time of elementary cellular under asynchronous update. For this study we have designed one theorem which characterizes the ACA, converges to some fixed-point attractor during their dynamic evolution. 146 ACA are identified, which are converged to some fixed-point attractors. For the experimental study of convergence time of ACA, we have followed the empirical curve bounding method. For the experimental setup we have designed an algorithm to find the average convergence time and the rate of growth of convergence time have found using the hybrid iterative refinement technique of empirical curve bounding method. We have identified six different classes of convergence time and that are O(1), $O(\log(\log n))$, $O(\log n)$, O($n^{1/2}$), $O(n^2)$ and $O(2^n)$, where n is the number of cells in the ACA.

References

[1] H. Bersini and V. Detour. Asynchrony induces stability in cellular automata based models. In R. A. Brooks and P. Maes, editors, *Artificial*

Life IV, pages 382–387, Cambridge, Massachusetts, 1994. The MIT Press.

[2] Nazim Fatès and Lucas Gerin. Examples of fast and slow convergence of 2d asynchronous cellular systems. *J. Cellular Automata*, 4(4):323–337, 2009.

[3] Nazim Fatès, Eric Thierry, Michel Morvan, and Nicolas Schabanel. Fully asynchronous behavior of double-quiescent elementary cellular automata. *Theor. Comput. Sci.*, 362(1-3):1–16, 2006.

[4] Jörg Hoffmann, Nazim Fatès, and Héctor Palacios. Brothers in arms? on ai planning and cellular automata. In *ECAI*, pages 223–228, 2010.

[5] T. Ingerson and R. Buvel. Structure in asynchronous cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):59–68, 1984.

[6] Catherine McGeoch, Peter Sanders, Ruldolf Fleischer, Paul R.Cohen, and Doina Precup. Using finite experiments to study asymptotic performance. *Experimental Algorithmics, LNCS*, (2547):93–126, 2002.

[7] Anindita Sarkar, Anindita Mukherjee, and Sukanta Das. Reversibility in asynchronous cellular automata. *Complex Systems*, 21(1):71–84, June 2012.

[8] B. Schonfisch and A. De Roos. Synchronous and Asynchronous updating in Cellular Automata. *Biosystems*, 51:123–143, 1999.

[9] Biswanath Sethi and Sukanta Das. Modeling of asynchronous cellular automata with fixed-point attractors for pattern classification. In *International Conference on High Performance Computing and Simulation*, July 2013.

[10] S. Wolfram. *Theory and applications of cellular automata*. World Scientific, Singapore, 1986. ISBN 9971-50-124-4 pbk.

Linear acceleration for one-dimensional cellular automata

Véronique Terrier

GREYC - UMR 6072
Université de Caen Basse-Normandie
Campus Côte de Nacre, 14 032 CAEN cedex 5, France

Abstract

This paper is an attempt to characterize linear acceleration algorithms in one-dimensional cellular automata. Different solutions have been presented in the literature based on geometric transformations of the space-time diagram. We will emphasize their common characteristics and will try to make explicit their generic design.

1 Introduction

Cellular automata are known as a relevant model for massively parallel computation. Various algorithms illustrate their ability to do fast computation ([3, 2]). Here we focus on the classical question of linear acceleration and investigate simulations which achieve speed up.

2 Definitions

In the sequel we only consider one-dimensional cellular automata with two-way communication.

2.1 One dimensional CA

Formally, a *cellular automaton* is a one-dimensional array of finite automata (the cells) indexed by $\mathbb{Z}$. The cells evolve synchronously at discrete time step. Each cell takes on value from a finite set of states S and communicates to its left and right neighbors. At each step, each cell changes its state according to its own state, the states of its left and right neighbors and to a transition function $\delta\colon S^3 \to S$. A site (c,t) denotes the cell c at time t and $\langle c,t\rangle$ denotes its state. For time $t \geq 0$, $\langle c,t+1\rangle = \delta\left(\langle c-1,t\rangle, \langle c,t\rangle, \langle c+1,t\rangle\right)$. The alphabet of the input words is a subset of S: the input set Σ. The input mode is parallel; at initial time 0, the i-th bit of the input word $w \in \Sigma^*$

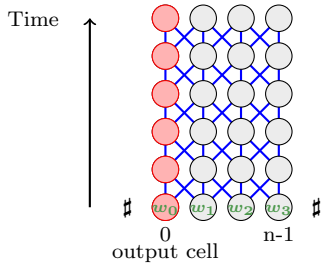


Figure 1: The space time diagram of a CA with parallel input mode and two-way communication

is fed to the cell i: $\langle i, 0\rangle = w_i$. The computation is space bounded: all other cells remain in a persistent state $\sharp$ during all the computation. A configuration is the sequence of cell states at a given time. The evolution of a cellular automaton starting from a given input is represented by a *space-time diagram* where the rows correspond to the configurations at the successive time steps.

2.2 Language recognition

To specify language recognition, we distinguish a subset of S: the set of accepting states S_{accept}. A CA accepts a word w, if, on input w, the distinguished cell 0 enters an accepting state at some time t. A CA recognizes a language L in time f, if it accepts every word $w \in L$ of length $n = |w|$ at time $t \leq f(n)$. Among these time complexities, the real time function corresponds to $f(n) = n$; it is the minimal time for the distinguished cell 0 to read the whole input and to know that the input is completed. We denote by $CA(f)$ the class of languages recognized in time f by some CA.

2.3 Linear acceleration

Here we focus on linear acceleration algorithms which transform an initial CA recognizer $\mathcal{A}$ in a CA recognizer $\mathcal{B}$ working k time faster, where $k > 1$ is any rational. Precisely, it corresponds to an acceleration by a constant factor of the complexity time beyond real time: once the output cell gets the whole input, it proceeds faster. See Figure 2. The linear acceleration theorem has been first established by Beyer [1].

Theorem 1 (linear acceleration)**.** Let f be a function from $\mathbb{N}$ to $\mathbb{N}$. If a language L is recognized in time $n + f(n)$, then for any positive ratio $r \in \mathbb{Q}$, L is recognized in time $n + \lceil r\, f(n)\rceil$.

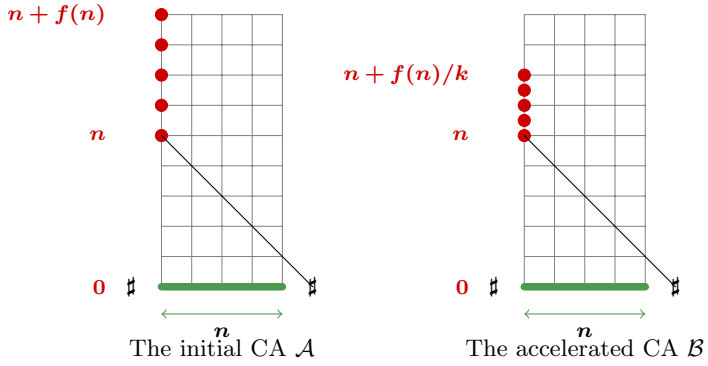


Figure 2: Linear acceleration

2.4 Affine transformation

Simulations between CA make widely use of affine transformations of the working area. Formally, an affine transformation Π of the space-time diagram is specified by three ingredients: a square matrix T, an origin C and a set of sites $\mathcal{X}$. It maps every site Q into $\Pi(Q)$ defined as follows:

$$\Pi(Q) = \begin{cases} C + T \times (Q - C) & \text{if } Q \in \mathcal{X} \\ Q & \text{otherwise} \end{cases}$$

See Figure 3 for examples. With $T_1 = \begin{pmatrix} f & 0 \\ 1-f & 1 \end{pmatrix}$ (where f is $1/2$) and $C = (0, n)$ (where n is the input length), the first transformation applies on the set of sites $\{(c,t)\colon t > 0\}$. With $T_2 = \begin{pmatrix} 1 & 0 \\ f-1 & f \end{pmatrix}$ and $C = (0, n)$, the second transformation applies on the set of sites $\{(c,t)\colon c + t \geq n\}$.
Note that we actually introduce rational coordinates. To revert then to a discrete space-time diagram will require to group and to keep some redundant information in each integer site.

3 Examples of linear acceleration algorithms

We recall the known acceleration algorithms and explicit their common characteristics.

3.1 Beyer algorithm

The first linear acceleration algorithm was presented by Beyer in his PhD thesis [1]. See Figure 3. It is the composition of two complementary compressions with $\boldsymbol{u} = (0, 1)$ and $\boldsymbol{v} = (1, -1)$. The first compression $\begin{pmatrix} f & 0 \\ 1-f & 1 \end{pmatrix}$

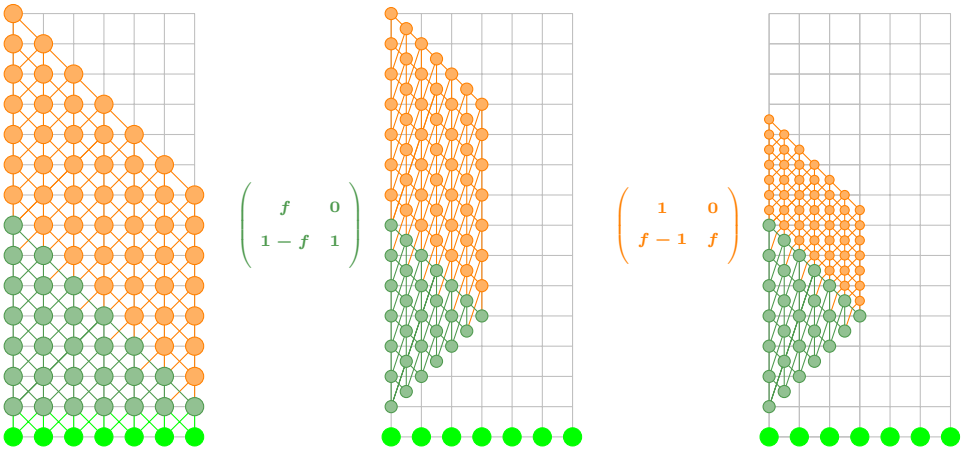


Figure 3: Beyer acceleration

applies on all sites; the second one $\begin{pmatrix} 1 & 0 \\ f-1 & f \end{pmatrix}$ applies on the sites up to the segment $[(0,n)\cdots(n-1,1)]$.

3.2 Algorithm with Firing Squad

Mazoyer and Reimen proposed the solution depicted in Figure 4 which makes use of a Firing Squad [5]. It is the composition of two complementary compressions with $\boldsymbol{u} = (-1,1)$ and $\boldsymbol{v} = (1,0)$. The first compression $\begin{pmatrix} f & -1+f \\ 0 & 1 \end{pmatrix}$ applies on the sites up to the segment $[(0,n)\cdots(n-1,1)]$; the second one $\begin{pmatrix} 1 & 1-f \\ 0 & f \end{pmatrix}$ applies from time n, making use of the Firing Squad to delimitate the horizontal segment.

3.3 Another variant

Another variant without synchronization was proposed by Heen [4]. See Figure 5. It is the composition of two complementary compressions with $\boldsymbol{u} = (-1,1)$ and $\boldsymbol{v} = (1,1)$. The first compression $\begin{pmatrix} (f+1)/2 & (f-1)/2 \\ (f-1)/2 & (f+1)/2 \end{pmatrix}$ applies on the sites up to the segment $[(0,n)\cdots(n-1,1)]$; the second one $\begin{pmatrix} (f+1)/2 & (-f+1)/2 \\ (-f+1)/2 & (f+1)/2 \end{pmatrix}$ applies on the sites up to the segment $[(0,n)\cdots(n-1,2n-1)]$.

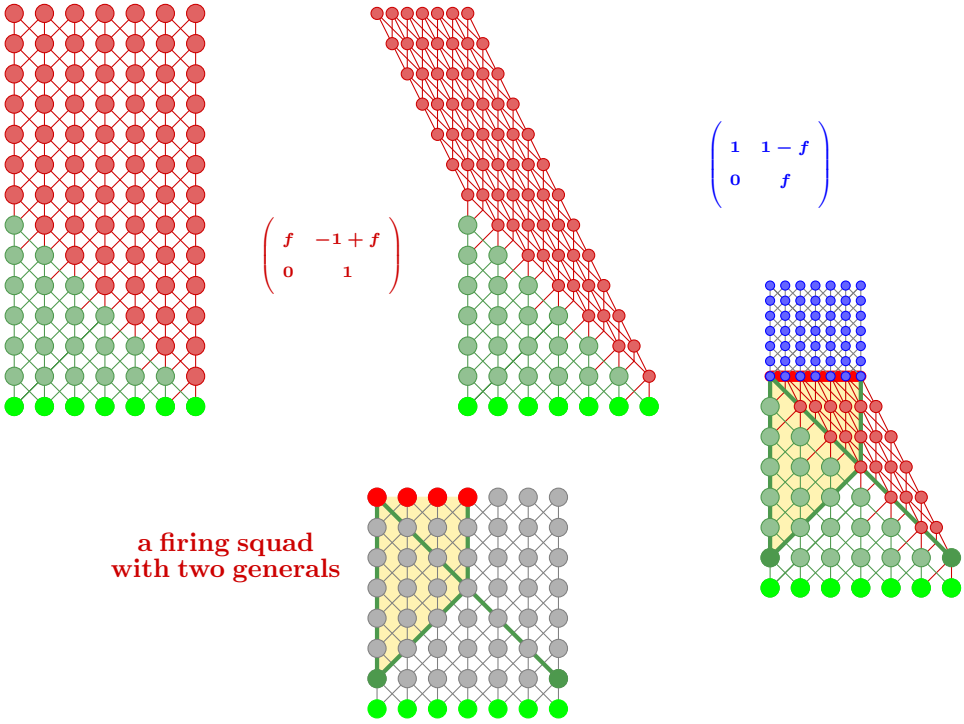


Figure 4: variant with a Firing Squad (Mazoyer & Reimen)

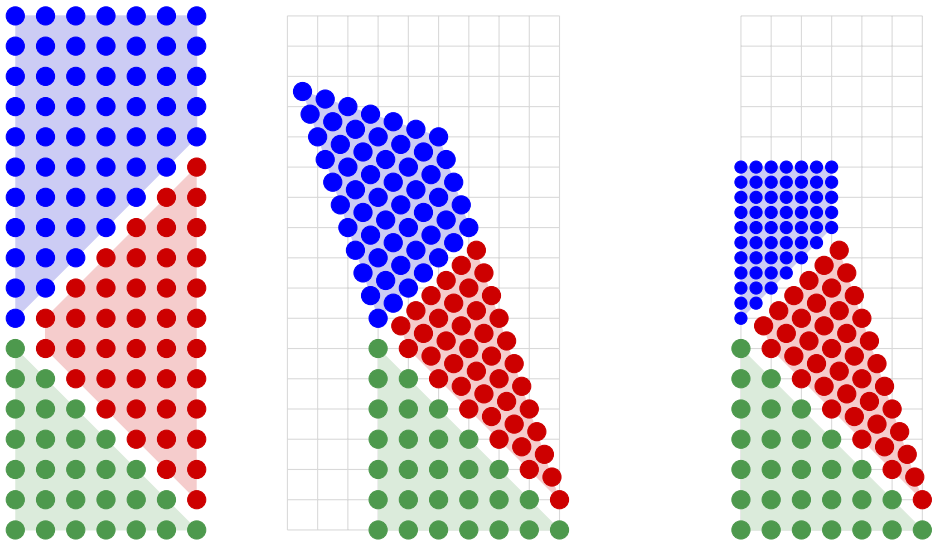


Figure 5: another variant (Heen)

4 Common characteristics

We review some common characteristics shared by these speed-up algorithms.

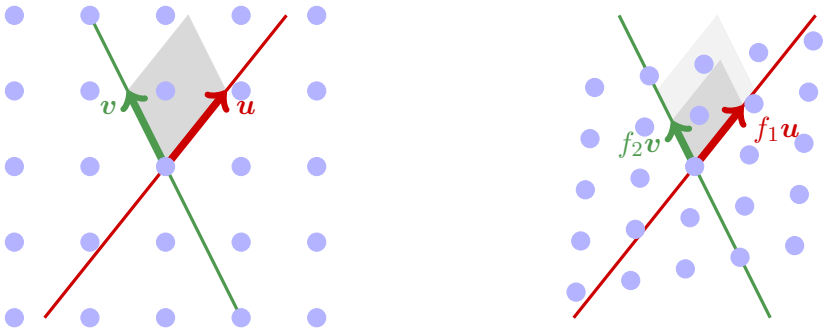
4.1 Contraction

First every speed-up algorithm is the result of the composition of two complementary contractions.

Two axes specified by two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ with $\boldsymbol{u} = (x, 1)$, $\boldsymbol{v} = (1, y)$ and $x\,y \neq 1$
the eigenvectors basis $P = \begin{pmatrix} x & 1 \\ 1 & y \end{pmatrix}$

Two eigenvalues f_1, f_2 with $f_1, f_2 > 0$ and the compression rate $f_1 f_2 < 1$
the diagonal contraction $D = \begin{pmatrix} f_1 & 0 \\ 0 & f_2 \end{pmatrix}$

A contraction is the transformation $T = P \times D \times P^{-1}$ such that $T(\boldsymbol{u}) = f_1\boldsymbol{u}$ et $T(\boldsymbol{v}) = f_2\boldsymbol{v}$
its complementary contraction is $T_c = P \times \begin{pmatrix} f_2 & 0 \\ 0 & f_1 \end{pmatrix} \times P^{-1}$, $T_c \circ T = T \circ T_c = f_1 f_2\,\mathrm{Id}$



Observe that not all contractions are achievable on a CA. Indeed the data move is subjected to the neighborhood constraints. The contraction applied on the space-time diagram must preserve the neighborhood rules. As the initial CA and the contracted CA share the same neighborhood, the image of the neighborhood must satisfy the neighborhood constraint itself. It means that the image of the neighborhood extremities $\boldsymbol{e_1} = (-1, 1)$ and $\boldsymbol{e_2} = (1, 1)$ are contained in the neighborhood cone $\{\alpha\boldsymbol{e_1} + \beta\boldsymbol{e_2} \colon \alpha, \beta \geq 0\}$.

Notably, one can't contract more the time than the space:

Fact 1. The contractions compatible with the nearest neighborhood are such that $f_1 \geq f_2$ and the eigenvectors $\boldsymbol{u} = (x, 1)$, $\boldsymbol{v} = (1, y)$ associated respectively to f_1 et f_2 verify $|x|, |y| <= 1$.

Proof. The linear contraction T is represented by the matrix $\begin{pmatrix} (f_1\,xy - f_2)/(xy-1) & -(f_1 - f_2)\,x/(xy-1) \\ (f_1 - f_2)\,y/(xy-1) & (f_2\,xy - f_1)/(xy-1) \end{pmatrix}$. The images of the neighborhood vectors $\boldsymbol{e_1}$ and $\boldsymbol{e_2}$ are $T(\boldsymbol{e_1}) = a\,\boldsymbol{e_1} + b\,\boldsymbol{e_2}$ and $T(\boldsymbol{e_2}) = c\,\boldsymbol{e_1} + d\,\boldsymbol{e_2}$, with $a = 1/2\,(f_1\,x\,y + f_2\,x\,y + f_1\,x - f_1\,y - f_2\,x + f_2\,y - f_1 - f_2)/(x\,y-1)$, $b = -1/2\,(y+1)\,(x+1)\,(f_1 - f_2)/(x\,y-1)$, $c = -1/2\,(y-1)\,(x-1)\,(f_1 - f_2)/(x\,y-1)$, $d = (f_1\,x\,y + f_2\,x\,y - f_1\,x + f_1\,y + f_2\,x - f_2\,y - f_1 - f_2)/(x\,y-1)$ non negative values. From the condition $b, c \geq 0$, we get either $f_1 \geq f_2$ and $|x|, |y| \leq 1$ or $f_1 \leq f_2$ and $|x|, |y| \geq 1$. For the second case, $f_1 \leq f_2$ and $|x|, |y| \geq 1$, to set $\boldsymbol{v'} = (1, 1/x)$ and $\boldsymbol{u'} = (1/y, 1)$ brings back to the first case. □

We may verify that the images of $\boldsymbol{e_1} = (-1, 1)$ and $\boldsymbol{e_2} = (1, 1)$, by each first contraction involved in the three acceleration examples, are indeed in the neighborhood cone. But the complementary contraction is not necessarily compatible with the neighborhood, as the complementary one in Beyer algorithm ($T_c(\boldsymbol{e_2}) = (1, 2f - 1)$ lies outside of the cone). It is only the composition of the two contractions which is achievable.

4.2 Compression

The contractions involved in the three acceleration examples are specific. They are unidirectional: the deformation applies only along one axis, the second one remains stable.

A compression is a contraction with one of the eigenvalues equals to 1.

The eigenvalues $\{f_1, f_2\} = \{1, f\}$ where f is the compression rate.

The acceleration examples are the results of two complementary compressions:

- $T_1 = P \times \begin{pmatrix} 1 & 0 \\ 0 & f \end{pmatrix} \times P^{-1}$ where the sites on a line $C_1 + \alpha\boldsymbol{u}$ remain stable
- $T_2 = P \times \begin{pmatrix} f & 0 \\ 0 & 1 \end{pmatrix} \times P^{-1}$ where the sites on a line $C_2 + \beta\boldsymbol{v}$ remain stable

Let C be the intersection of the lines $C_1 + \alpha\boldsymbol{u}$ et $C_2 + \beta\boldsymbol{v}$. It specifies the center of the affine compressions.

Now we wonder if other accelerations than the three examples could be implemented as the result of two compressions.

First, notice that $C = (0, n)$ with n the input length, since the two compressions composition maps the sites $(0, n+t)$ to $(0, n+ft)$ by definition. Next, we must take into account the input constraint. Indeed, the way the input word is supplied in parallel to the initial CA remains identical to the compressed CA. This consequently implies access constraints in the compressed CA. It depends also whether the compression apply on all sites (as the first compression in Beyer algorithm) or not (as the first compression in the two other examples).

4.2.1 Case 1. The compression applies on all sites

In this case, the only compression compatible with the input constraint is the first compression involved in Beyer algorithm:

Fact 2. Let $P = (i, j)$ be any site with $0 < i, j$ and $i + j < n$. The only one compression which both applies to P and is compatible with the input constraint, remains stable on the axis $\boldsymbol{u} = (0, 1)$ and compresses along the axis $\boldsymbol{v} = (1, -1)$.

Proof. Below the line $i + j = n$, any site $P = (i, j)$ has no information on the input length n. Equally, its image under the compression $imP = C + T_1 \times (P - C) = \big((f-1)(j-n)\,x/(xy-1) + (xy-f)\,i/(xy-1), -(f-1)\,i\,y/(xy-1) + (f\,xy-1)(j-n)/(xy-1) + n\big)$ is independent from n. That means that $-(f-1)\,x/(xy-1)$ and $-(f-1)\,xy/(xy-1)$ are equal to 0; this leads to $x = 0$.
Furthermore, like P, imP depends on the input bit w_{i+j} supplied to the array on cell $i + j$ at time 0. Hence the site $(i + j, 0)$ must be accessible on the site imP. In other words, the corresponding vector $((y-1)(f-1)\,i/2 + j)\,\boldsymbol{e_1} + (y+1)(f-1)\,i/2\,\boldsymbol{e_2}$ must be in the neighborhood cone. So $(y-1)(f-1)\,i + 2j$ and $(y+1)(f-1)\,i$ are positive, which implies that $y = -1$. □

4.2.2 Case 2. The compression does not apply below the diagonal line $i + j = n$

A common characteristic of the acceleration with a Firing Squad and the another variant is that the stable axis is $\boldsymbol{u} = (-1, 1)$ (See Figures 4, 5). It is also a characteristic of all compressions which leave the sites below the diagonal line unchanged:

Fact 3. Any compression which both does not apply on the sites $P = (i, j)$ with $i + j < n$ and is compatible with the input constraint, remains stable on the axis $\boldsymbol{u} = (-1, 1)$.

Proof. Consider two sites $R = (n - 1 - i, i)$ and $Q = R + j\,\boldsymbol{e_2}$. Below the diagonal line, R is not affected by the compression. The compression

applies on Q (otherwise it would be no acceleration). Q and, in the same way, its image under the compression $imQ = C + T_1 \times (Q - C)$ depend on R. Hence the vector $R\,imQ$ is in the neighborhood cone. That implies the values $(y-1)\,(f-1)\,((x+1)\,(i+j-n)-2\,j+1)/(xy-1)$ and $(x+1)\,(f-1)\,((i-j-n)\,(y+1)+2\,j+y)/(xy-1)+2\,f\,j-f+1$ are positive. It follows that $x = -1$. □

Finally we wonder what compressions with a stable axis $\boldsymbol{u} = (-1, 1)$ and their complementary compression are achievable on CA. Figure 6 depicts different cases. We have to take care how the two compressions link together along the line $C + \alpha\,\boldsymbol{v}$.
First the segment $\{C+\alpha\,(1, y)\colon 0 \leq \alpha < f\,n\}$ should be characterized by the CA. In the given examples, it is feasible when $y = 0$ (with a Firing Squad), $y = 1$ (a simple signal with speed 1), $y = 1/2$ (with a delayed Firing Squad). But is it feasible when $y = -1/5$?
Second, we have also to clarify how to patch things up on the segment.

5 Conclusion

Investigating linear acceleration on one-dimensinal CA, we have approched the question of geometric transformations and their composition. The difficulty is to make explicit under which conditions these transformations are feasable or not on a CA. Different constraints come into play: the neighborhood constraint is rather easily controlled whereas the input constraint and the problem of sticking together transformations turn out to be more delicate to handle. It would be also worthwhile to explore the feasible transformations in higher dimensional CA.

References

[1] Terry Beyer. *Recognition of topological invariants by iterative arrays.* PhD thesis, Cambridge, MA, USA, 1969.

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

[3] Patrick C. Fischer. Generation of primes by a one-dimensional real-time iterative array. *Journal of the ACM*, pages 388–394, 1965.

[4] Olivier Heen. *Économie de ressources sur automates cellulaires.* PhD thesis, Paris 7, 1996.

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

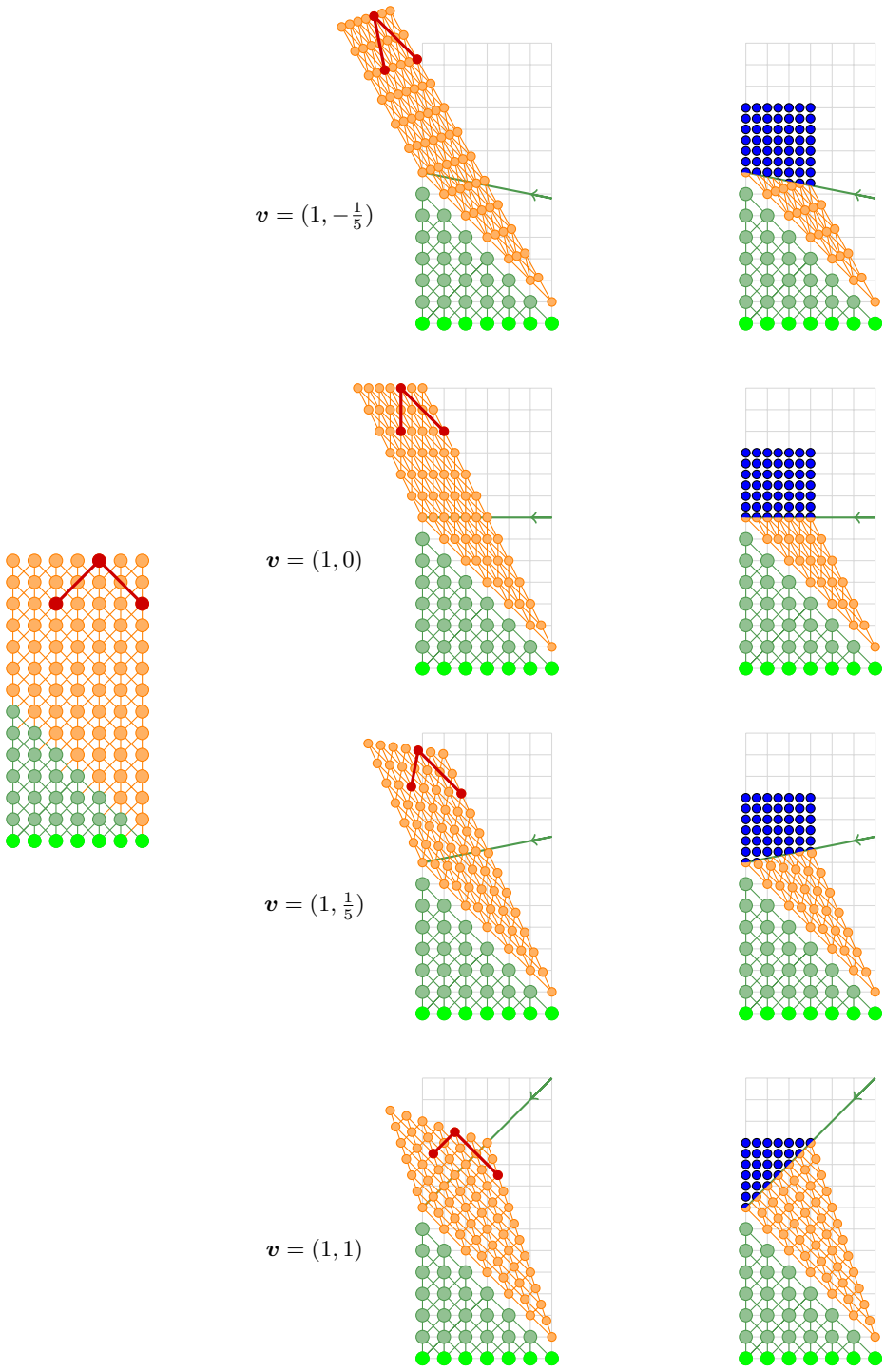


Figure 6: Compressions composition with a stable axis $\boldsymbol{u} = (-1, 1)$

Standardizing the set of states and the neighborhood of asynchronous cellular automata

Thomas Worsch
Karlsruhe Institute of Technology
worsch@kit.edu

August 17, 2013

Abstract

Smith (1971) has shown that any synchronous CA can be simulated by one with only 2 states and by one with von Neumann neighborhood of radius 1. We prove that the same is possible for asynchronous CA.

1 Introduction

More than 40 years ago Smith [4] has shown that each synchronous CA can be simulated by another synchronous one with only 2 states, and that each can be simulated by another synchronous one with von Neumann neighborhood of radius 1. In the present paper we describe constructions showing that indeed analogous standardization of the set of states and the neighborhood are possible for asynchronous CA. To the best of our knowledge this has not been considered in the literature until now. To simplify the presentation we will only consider one-dimensional CA explicitly. But it will be obvious how to generalize the construction to higher dimensions.

The rest of this paper is organized as follows: In Section 2 we recall the basic definitions for asynchronous CA as far as they are needed later on. In Section 3 we recap a construction for the simulation of synchronous CA on asynchronous CA. The underlying idea turns out to be useful also in both constructions presented. In Section 4 we will describe how an ACA with an arbitrary set of states can be simulated by one with $\{0, 1\}$ as its set of states. In Section 5 we will describe how an ACA with arbitrary neighborhood can be simulated by one with neighborhood $\{-1, 0, 1\}$. We conclude in Section 6.

2 Basics

We write $\mathbf{Z}$ for the set of integers, B^A for the set of total functions from A to B, and 2^M for the powerset of M. The cardinality of a set M is $|M|$.

In this paper we are interested in one-dimensional cellular automata. If the set of states of one cell is denoted as Q, the set of all configurations is $Q^{\mathbf{Z}}$. A neighborhood is a finite set $N = \{d_1, \ldots, d_k\}$ of integers. Without loss of generality one may assume that it is a one-dimensional von Neumann neighborhood with radius r, which we denote as $N_r = \{-r, \ldots, 0, \ldots, r\}$. A *local configuration* is a mapping $\ell : N \to Q$; thus Q^N is the set of all local configurations. The local configuration c_{i+N} observed by cell $i \in \mathbf{Z}$ in global configuration c is defined as $c_{i+N} : N \to Q : d \mapsto c(i+d)$. The behavior of a single cell is described by the local transition function $f\colon Q^N \to Q$.

In this paper we consider (purely) asynchronous CA (ACA), where in a global step each cell has two possibilities: to be *active* and make a state transition (according to f) or to be *passive* and not to change its state. If $A \subseteq \mathbf{Z}$ is the *activity set* of cells which are to apply f, then given a configuration c one step of the CA will result in configuration c' defined by

$$c'(i) = \begin{cases} f(c_{i+N}) & \text{iff } i \in A, \\ c(i) & \text{iff } i \notin A. \end{cases}$$

This will be denoted as $c' = \Delta_A(c)$ or as $c \vdash_A c'$; occasionally we will drop the index A if it is irrelevant. A (finite or infinite) sequence $(c^0, c^1, c^2, \ldots)$ of configurations is a *computation*, iff for all pairs (c^i, c^{i+1}) within the sequence it is true that $c^i \vdash c^{i+1}$.

3 A useful idea

As a first step let us consider a construction to modify a local transition function which has been developed for the synchronous case such that in its new form it can be used in the asynchronous case. Probably best known is Nakamura's idea [2]. Because it is more descriptive we will use another method which has first been described in [1], but an equivalent, more explicit version has been discovered independently by Schuhmacher [3]. The description below is adapted from [5]. The idea will be expanded later on.

Given the set Q_S of states of a synchronous CA C_S, the set Q_A of states of the asynchronous CA C_A is constructed as follows

$$\begin{aligned} & & Q_A &= Q_{\mathtt{B}} \cup Q_{\mathtt{T}} \cup Q_{\mathtt{E}} \\ \text{where} & & Q_{\mathtt{B}} &= \{(q, \mathtt{B}) \mid q \in Q_S\} \\ & & Q_{\mathtt{T}} &= \{(q, q') \mid q, q' \in Q_S\} \\ & & Q_{\mathtt{E}} &= \{(\mathtt{E}, q') \mid q' \in Q_S\} \end{aligned}$$

The states in the set $Q_{\mathtt{B}}$, $Q_{\mathtt{T}}$ and $Q_{\mathtt{E}}$ are called *begin*, *transitional* and *end* states respectively. Given an initial configuration $c_S \in Q_S^{\mathbf{Z}}$ of C_S the corresponding initial configuration c_A of C_A is simply defined as

$$\forall i \in \mathbf{Z} : c_A(i) = (c_S(i), \mathtt{B}) \tag{1}$$

Definition 1. The local rules for C_A are as follows:

$\mathtt{B} \to \mathtt{T}$**:** If an active cell is in a state $(q, \mathtt{B})$, it will only change its state, if all neighbors are in states from $Q_\mathtt{B} \cup Q_\mathtt{T}$. If a new state is entered it is (q, q') where $q' = f_S(q_1, \ldots, q_k)$ and the q_i are the first components of the states of the neighbors.

$\mathtt{T} \to \mathtt{E}$**:** If an active cell is in a state $(q, q') \in Q_\mathtt{T}$, it will only change its state, if all neighbors are in states from $Q_\mathtt{T} \cup Q_\mathtt{E}$. If a new state is entered it is $(\mathtt{E}, q')$.

$\mathtt{E} \to \mathtt{B}$**:** If an active cell is in a state $(\mathtt{E}, q)$, it will only change its state, if all neighbors are in states from $Q_\mathtt{E} \cup Q_\mathtt{B}$. If a new state is entered it is $(q, \mathtt{B})$.

Thus a cell always runs in a cycle from a begin state to a transitional state to an end state and then again to a begin state. We say that a cell *makes progress*, if it changes from an end to a begin state. Whenever that happens, the cell has simulated one state transition of the corresponding guest cell.

An initial configuration to be used (see formula (1) above) has an important property: For any cell i the set of states observed in its neighborhood $i + N$ does *not* contain states from *all three* subsets $Q_\mathtt{B}$, $Q_\mathtt{T}$ and $Q_\mathtt{E}$ simultaneously. Let us call this the *"states-not-mixed property"*. Rereading the description of the local rules immediately shows the following:

Lemma 2. *A cell only changes its state if in its neighborhood the states are not mixed. If a cell changes its state then in the resulting local configuration the states are not mixed.*

One should note, that the states-not-mixed property may be destroyed in neighborhoods in which the center cell is not active. As a consequence of Lemma 2 the progress of neighboring cells is always closely related:

Corollary 3. *Whenever a cell has made progress $\tau + 1$ times, its neighbors have already made progress at least τ times.*

As a simple example an asynchronous simulation of the left shift (of symbols _ and #) is shown in Figure 1. For example cells 3 and 5 make progress in step 5 and cell 4 makes progress in step 6.

For another nice property of the construction we use the following definitions. Assume that an arbitrary computation $\mathcal{C} = c^0 \vdash_{A_1} c^1 \vdash_{A_2} c^2 \cdots$ of an ACA is given. Define its *"state change virtual time"* $sc(\mathcal{C}) : \mathbf{N}_0 \times \mathbf{Z} \to \mathbf{N}_0$ by defining its value $sc_i^t = sc(\mathcal{C})(t, i)$ for cell $i \in \mathbf{Z}$ in configuration c^t as follows:

$$\forall i \in \mathbf{Z}: \qquad sc_i^0 = 0$$

$$\forall t \in \mathbf{N}_0\ \forall i \in \mathbf{Z}: \qquad sc_i^{t+1} = \begin{cases} 1 + sc_i^t & \text{iff } c_i^{t+1} \neq c_i^t \\ sc_i^t & \text{otherwise} \end{cases}$$

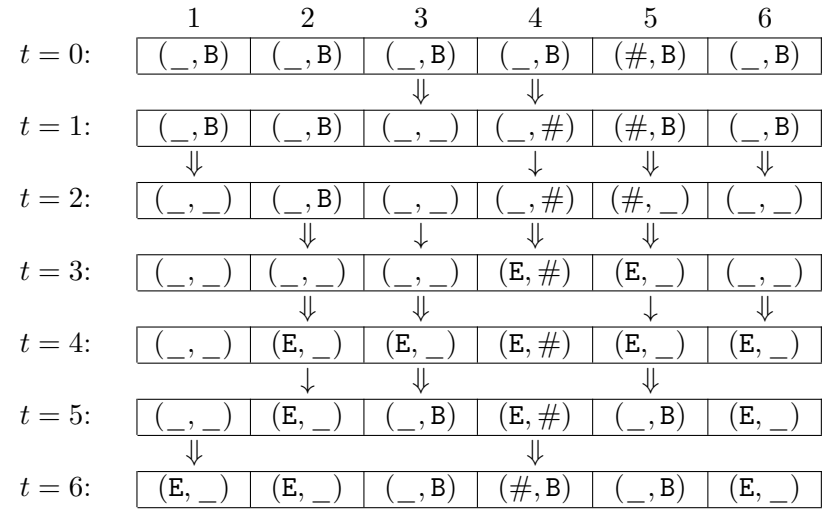


Figure 1: Asynchronous simulation of a left shift using the technique by Lee et al. [1, 3]. The set of original states is $\{\_, \#\}$. Arrows indicate that a cell is active. A double arrow indicates, that an activity actually leads to a change of state. It is assumed that cells outside of the shown segment also make sufficient progress in order to allow cells 1 and 6 to behave as shown.

That is, sc_i^t is the number of proper changes cell i has made up to time t. Now we define a new asynchronous computation. For $\tau \geq 1$ the new subsets $B_\tau \subseteq \mathbf{Z}$ of active cells are chosen as follows:

$$B_\tau = \{i \in \mathbf{Z} \mid \exists t\colon sc_i^t = \tau\} \tag{2}$$

B_τ contains all cells which have changed their states at least τ times. Starting with the same configuration $d^0 = c^0$ as $\mathcal{C}$ have a look at the computation

$$\mathcal{D} = d^0 \vdash_{B_1} d^1 \vdash_{B_2} d^2 \cdots \tag{3}$$

If computation $\mathcal{C}$ is finite or reaches a fixed point configuration, then only a finite number of sets B_t is non-empty and $\mathcal{D}$ is finite (too).

The essence of the above construction is captured by the following lemma.

Lemma 4. *Given an arbitrary computation $\mathcal{C}$, the corresponding computation $\mathcal{D}$ defined in Equations 2 and 3 above satisfies for each cell i the following properties:*

1. *Whenever cell i is active in $\mathcal{D}$, it changes its state.*
2. *In both computations cell i changes its state the same number of times.*
3. *The resulting states in $\mathcal{C}$ and $\mathcal{D}$ are always the same.*

This immediately implies:

Corollary 5. *If computation $\mathcal{C}$ is reaching an end, so does $\mathcal{D}$ and the final configurations of both computations are the same.*

Furthermore computation $\mathcal{D}$ is "basically" a synchronous one in the following sense. Consider a computation $\mathcal{C}$ with the property *INA* ("identical number of activities") that each cell makes progress for the same number of times. That implies that the all non-empty sets $B_\tau = \mathbf{Z}$. Hence:

Lemma 6. *If computation $\mathcal{C}$ has property INA then $\mathcal{D}$ is a* synchronous *computation.*

Since we are considering deterministic CA one gets:

Corollary 7. *Any two INA computations of the same length result in the same final configuration. Any INA computations with length $3T$ results in the configuration obtained after T steps of the original CA.*

In other words, if all cells make progress for the same number of times this simulates a computation of the original synchronous CA.

4 Standardizing the set of states

In this section we are given an arbitrary ACA $C = (Q, f, N_C)$. The goal is to construct another ACA $D = (\{0,1\}, g, N_D)$ which simulates C using only 2 states per cell. The construction below has been chosen with an eye to a simple reasoning not with an eye to a particularly small neighborhood (see subsection 4.3 for a further remark in this respect). The correctness of the approach will be obvious since we will make use of the idea explained above.

4.1 Construction

Since different ACA C_i may be simulated by different corresponding ACA D_i the "knowledge" about $k = |Q|$ can be "built into" into the set of states and the local rules of the D_i.

Each cell of C is encoded in a *block* of cells of D which conceptually is further divided into three segments:

- a *marker* consisting of the bit pattern `011110` which never changes;
- an *old state segment* consisting of $k+1$ bits which are used for a one-hot encoding of the "states" of $Q \cup \{\mathtt{E}\}$. State i, $1 \le i \le |Q|$, is encoded as $\mathtt{0}^{i-1}\mathtt{10}^{k-i}\mathtt{0}$ and `E` is encoded as $\mathtt{0}^{k}\mathtt{1}$.
- a *new state segment* also consisting of $k+1$ bits which are used for a one-hot encoding of the "states" of $Q \cup \{\mathtt{B}\}$. State i, $1 \le i \le |Q|$, is encoded as $\mathtt{0}^{i-1}\mathtt{10}^{k-i}\mathtt{0}$ and `B` is encoded as $\mathtt{0}^{k}\mathtt{1}$.

It should be noted that during a computation it will happen, that for some transitional time two (or zero bits) are set in a state segment. Nevertheless the bit pattern `011110` contained in the marker will always stay unique: it will never occur anywhere else in a block.

Figure 2 shows a sketch of one block of cells. In both state segments there are k bits, $o_1, \ldots, o_k$ and $n_1, \ldots, n_k$ respectively, for encoding a "real" state. The bit for the representation of `B` is denoted $x_{\mathtt{B}}$, and the one for `E` is $x_{\mathtt{E}}$.

marker						old state					new state				
0	1	1	1	1	0	o_1	o_2	$\cdots$	o_k	$x_{\mathtt{E}}$	n_1	n_2	$\cdots$	n_k	$x_{\mathtt{B}}$

Figure 2: Binary encoding of a state; $o_1, \ldots, o_k$ are used for a one-hot encoding of the old state of the cell, and $n_1, \ldots, n_k$ for the new state.

According to the neighborhood N_C of C we will speak of one block of D being a neighbor of another block. The real neighborhood N_D of D has to be chosen large enough, so that each single cell of a block of D can observe each block which is neighbor of it as well as each block for which it is a neighbor.

Each state of an initial configuration c of the ACA C is embedded into a block as shown in Figure 3.

marker						old state					new state				
0	1	1	1	1	0	o_1	o_2	$\cdots$	o_k	0	0	0	$\cdots$	0	B

Figure 3: Initial embedding of a state into a block. Exactly one of $o_1, \ldots, o_k$ is set to `1`. All n_j are `0`. We have written `B` to indicate that $x_{\mathtt{B}} = 1$.

The simulation of the activity of one cell of C consists of three phases. Assume that old and new state have number i and j respectively (see Figure 4).

`B` $\rightarrow$ `T`: Initially a block is in a situation as shown in Fig. 3. The transition to a situation corresponding to a state (q_i, q_j) happens in two subphases:

- **`BT1`:** A transition is started when cell n_j in the new state segment (corresponding the new state) is active and it observes that in no neighboring block the $x_{\mathtt{E}}$ is still set to `1`.
  If this condition is satisfied n_j is set to `1`.
- **`BT2`:** As soon as afterwards the $x_{\mathtt{B}}$ cell (which is currently `1`) becomes active and observes that another bit in the new state is set it resets itself to `0` and the representation of a `T` state has been reached.

`T` $\rightarrow$ `E`: From this situation a block may continue to change its states, if no block for which it is a neighbor needs the old state any longer.

- **`TE1`:** Once this is the case and the $x_{\mathtt{E}}$ cell is active it sets its bit to `1`.

TE2: As soon as the cell storing the old state bit o_i which still has a `1` becomes active and observes the set $x_{\mathtt{E}}$ bit, the bit o_i is set to `0`.

E → B : For the simulation of a transition from an `E` to a `B` state the corresponding x bits have to be adjusted and the `1` bit has to be moved from the new to the old state segment. This is acheived using four subphases. As above, each subphase only happens if the responsible cell observes in its neighborhood that it is allowed to change its state.

EB1: The $x_{\mathtt{B}}$ bit of the new state segment is set to `1`.

EB2: The bit corresponding of the old state segment corresponding to the new state is set to `1`.

EB3: The bit corresponding of the new state segment corresponding to the new state is reset to `0`.

EB4: The $x_{\mathtt{E}}$ bit of the old state segment is reset to `0`.

In each case each cell can determine whether it is its turn to change a bit or not by looking at a sufficiently large neighborhood.

marker	old state					new state					corresponds to
		o_i	o_j					n_j			
011110	0	1	0	0	0	0	0	0	0	B	$(q_i, \mathtt{B})$
011110	0	1	0	0	0	0	0	1	0	B	
011110	0	1	0	0	0	0	0	1	0	0	(q_i, q_j)
011110	0	1	0	0	0	0	0	1	0	0	(q_i, q_j)
011110	0	1	0	0	E	0	0	1	0	0	
011110	0	0	0	0	E	0	0	1	0	0	$(\mathtt{E}, q_j)$
011110	0	0	0	0	E	0	0	1	0	0	$(\mathtt{E}, q_j)$
011110	0	0	0	0	E	0	0	1	0	B	
011110	0	0	1	0	E	0	0	1	0	B	
011110	0	0	1	0	E	0	0	0	0	B	
011110	0	0	1	0	0	0	0	0	0	B	$(q_j, \mathtt{B})$

Figure 4: All subphases for one block

4.2 Correctness

It is clear that for any global step $c \vdash_A c'$ of the original ACA C there is a sequence of steps of D simulating it. On the other in each configuration in each block there is at most one cell that will change its state when it becomes active. It is therefore clear that the corresponding changes lead to a simulation of the CA constructed in Section 3.

4.3 Improvements

As described the simulator needs $2|Q|+O(1)$ bits per block, i. e. per simulated cell. This can be reduced to $O(\log|Q|)$ bits by encoding old and new state using $\log|Q|+O(1)$ bits. It just has to be made sure that the bit pattern representing the marker will not appear anywhere else.

A new problem arises for the $\mathtt{E}\to\mathtt{B}$ phase. For each of the $\log|Q|$ bits of a state it must be possible to find out whether it has already been copied or not. One possibility is to represent each of the $\log|Q|$ "logical" bits in a kind of dual-rail coding as `01` and `10` respectively. In this case one can also have "empty" location(pair)s as in the one-hot encoding case and it is easy to detect which bits already have been copied and which not.

5 Standardizing the neighborhood

In this section we are given an arbitrary ACA $C=(Q,f,N)$. Without loss of generality $N=N_r$ for some positive integer $r\geq 2$.

5.1 Construction overview

The goal is to construct another ACA $D=(S,g,N_1)$ which simulates C using only von Neumann neighborhood of radius 1 using the following approach.

- First, from the given CA C one constructs a CA C' where each cell (of the original) has an additional *activity bit* a which influences the local transition function:
  - If $a=1$ an active cell will use the old f as defined by C to compute its new state.
  - If $a=0$ an active cell will not change its state.
- Given C' the construction for the reduction of the neighborhood in the synchronous case by Smith [4] is applied resulting in a CA C''.
- To C'' the construction from Section 3 is applied to make the local transition function robust for asychronous updating. *This is done in such a way, that any arbitrary subset of cells can have its activity bits set.* That way one gets the required CA D which can simulate global transitions of the original CA C for arbitrary subsets of active cells.

5.2 An extension of a useful idea

We reuse a concept called *asynchronous coin* from [5]. There two cells are needed and different asynchronous coins have to be separated by other cells. In the present paper each cell of D should simulate a cell of C and contain an asynchronous coin. To this end first of all, the cells are augmented with

two additional bits: an auxiliary "random" bit r and the "activity" bit a (initially both are 0). For their computation the cycle $\mathtt{B} \to \mathtt{T} \to \mathtt{E} \to \mathtt{B}$ used in the construction in Section 3 is extended to $\mathtt{B} \to \mathtt{R} \to \mathtt{A} \to \mathtt{T} \to \mathtt{E} \to \mathtt{B}$. Roughly speaking the R subphase will be used to set the r bits of some cells. Subsequently during the A subphase the a bits will be computed. Everything has to be set up in such a way that for every subset of all cells it can happen (due to different asynchronous updatings) that exactly the cells in the subset have a set to 1. Conceptually this happens in two steps:

1. An arbitrary non-empty subset of cells set their r bit to 1.
2. Each cell computes its a bit as the exclusive or of its own r bit and the r bit of its left neighbor.

This allows *any* pattern of a bits to be realized by always setting at least one r bit to 1:

- If at least one a should be set to 1, setting the r bit of one of those cells to 1 uniquely determines the r bits of all other cells.
- If all a bits should be 0, choose all r bits set to one.

Formally given an ACA $C = (Q, f, N)$ an ACA $C' = (Q', f', N)$ is constructed as follows. First of all, Q' is chosen as

$$
\begin{aligned}
& Q' = (Q_{\mathtt{B}} \cup Q_{\mathtt{R}} \cup Q_{\mathtt{A}} \cup Q_{\mathtt{T}} \cup Q_{\mathtt{E}}) \times \{0,1\} \times \{0,1\} \\
\text{where} \quad & Q_{\mathtt{B}} = \{(q, \mathtt{B}) \mid q \in Q\} \\
& Q_{\mathtt{R}} = \{(q, \mathtt{R}) \mid q \in Q\} \\
& Q_{\mathtt{A}} = \{(q, \mathtt{A}) \mid q \in Q\} \\
& Q_{\mathtt{T}} = \{(q, q') \mid q, q' \in Q\} \\
& Q_{\mathtt{E}} = \{(\mathtt{E}, q') \mid q' \in Q_\}
\end{aligned}
$$

The transition function will ensure that cells in a B state will always have their r and a bits set to 0. To this end Definition 1 extended: The direct transitions from B states to T states is broken up into substeps as described above, using R and A states. It should be noted that in some cases a cell may "jump" directly from B to A (omitting an R).

Definition 8. Compared to Definition 1 the important changes are detailed below. $\mathtt{B} \to \mathtt{T}$ is replaced by the first three points:

$\mathtt{B} \to \mathtt{R}$**:** An active cell in state $(q, \mathtt{B})$ which does not observe an E states at its right neigbor and observes a B state at its left neighbor, will enter state $(q, \mathtt{R})$ and set its r bit to 1.

A cell in state $(q, \mathtt{B})$ which already observes an R state at its left neighbor, will enter state $(q, \mathtt{A})$ and leave its r bit at 0 but set its a bit to 1 (because it left neighbor will have $r = 1$).

$\mathtt{R} \to \mathtt{A}$: An active cell in a R state computes its a bit as the exclusive or of the r bits of itself and of its left neighbor.

$\mathtt{A} \to \mathtt{T}$: An active cell in a state $(q, \mathtt{A})$, it will only change its state, if all neighbors are in states from $Q_{\mathtt{A}} \cup Q_{\mathtt{T}}$. If this is the case the new state depends on the activity bit of the cell:

- If $a = 1$ the new state is (q, q') where $q' = f_S(q_1, \ldots, q_k)$ and the q_i are the first components of the states of the neighbors.
- If $a = 0$ the new state is (q, q).

$\mathtt{T} \to \mathtt{E}$: unchanged;

$\mathtt{E} \to \mathtt{B}$: as before; additionally each cell resets both its r and its b bit to 0.

5.3 Correctness

First, the activity bits have been added to C and f was defined as: If $a = 1$ then use the original f, and if $a = 0$ then do nothing If A is the set of cells with activity bit 1, this ensures that the synchronous execution of one step of C' corresponds to the asynchronous step of C with set A of active cells.

To C' the construction for reducing the neighborhood in the synchronous case [4] is applied resulting in a CA C''. If A is the set of cells with $a = 1$, this ensures that the synchronous execution of a fixed number of steps of C'' corresponds to the asynchronous step of C with set A of active cells.

Finally, to C'' the construction from Sec. 3 is applied to make f robust for asychronous updating resulting in the ACA D. This does not change the neighborhood, it is $\{-1, 0, 1\}$ as for C''. As described in Sec. 5.2 the asynchronicity D can be exploited to have the activity bits of C'' set in a arbitrary subset of cells. Hence, using different asynchronous updatings D can simulate global transitions of the original CA C for arbitrary subsets of active cells.

6 Summary and Outlook

We have shown that each ACA can be simulated by another one with only 2 states, and that each ACA can be simulated by another one with von Neumann neighborhood of radius 1.

Concerning the standardization of the neighborhood the construction presented above is considerably simpler and thus also lends itself to less involved notions of simulation than other constructions (e. g. [5]) which use neighborhood radius 1. Concerning the standardization of the set of states constructions of intrinsically universal ACA do not help at all.

It remains to be seen whether there are constructions for α-asynchronous CA which yield similar results as presented above.

The author gratefully acknowledges interesing disucssions with Christian Kühnle and Alfred Schuhmacher.

References

[1] Jia Lee, Susumi Adachi, Ferdinand Peper, and Kenichi Morita. Asynchronous game of life. *Physica D*, 194(3-4):369–384, 2004.

[2] Katsuo Nakamura. Asynchronous cellular automata and their computational ability. *Systems, Computers, Control*, 5(5):58–66, 1974.

[3] Alfred Schuhmacher. Simulationsbegriffe bei asynchronen Zellularautomaten. Report for a student seminar, 2012.

[4] Alvy Ray Smith III. Cellular automata complexity trade-offs. *Information and Control*, 18:466–482, 1971.

[5] Thomas Worsch. Intrinsically universal asynchronous CA. *Natural Computing*, page (accepted for publication), 2013.

Author Index