Rudolf Freund, František Mráz, and Daniel Průša (eds.)

Ninth Workshop on Non-Classical Models of Automata and Applications (NCMA 2017)

Short Papers

Druck: Druckerei Riegelnik
1080 Wien, Piaristengasse 19

Preface

This volume contains the five short contributions of the *Ninth Workshop on Non-Classical Models of Automata and Applications (NCMA 2017)* held in Prague, Czech Republic, on August 17^{th} and 18^{th}, 2017.

The NCMA workshop series was established in 2009 as an annual forum for researchers working on different aspects of non-classical and classical models of automata and grammars. The purpose of the NCMA workshop series is to provide an opportunity to exchange and develop novel ideas as well as to stimulate research on non-classical and classical models of automata and grammar-like structures. Many models of automata and grammars are studied from different points of view in various areas, both as theoretical concepts and as formal models for applications. The goal of the NCMA workshop series is to motivate a deeper coverage of this particular area and in this way to foster new insights and substantial progress in computer science as a whole.

The previous workshops took place in the following places:

2009 Wroclaw, Poland,
2010 Jena, Germany,
2011 Milan, Italy,
2012 Fribourg, Switzerland,
2013 Umeå, Sweden,
2014 Kassel, Germany,
2015 Porto, Portugal, and
2016 Debrecen, Hungary.

Since 2014, in addition to the invited talks and regular contributions, the NCMA workshops have also included short presentations reporting on recent results or ongoing work. Nevertheless, each short presentation has been evaluated by at least two members of the Program Committee. This volume contains extended abstracts of the five short papers selected for presentation at the Ninth Workshop on Non-Classical Models of Automata and Applications in Prague.

We thank the Department of Software and Computer Science Education of the Faculty of Mathematics and Physics of Charles University in Prague and the Czech Science Foundation (grant projects PRVOUK P46 and 15-04960S) for their support. We also thank the Institute of Computer Languages of the TU Wien for covering the production costs of the proceedings and this collection of short papers. Moreover, we sincerely thank Anna Kotěšovcová (Conforg) for local arrangements and the organization of NCMA 2017.

August 2017

Rudolf Freund, Wien
František Mráz, Prague
Daniel Průša, Prague

Table of Contents

Short Papers

P SYSTEMS WITH COSTS AND THEIR RELATION TO PRICED TIME AUTOMATA AND PRICED TIME PETRI NETS (ABSTRACT)

Bogdan Aman[A] Péter Battyányi[B]
Gabriel Ciobanu[A] György Vaszil[B,C]

[A]Romanian Academy, Institute of Computer Science, Iasi, Romania
bogdan.aman@gmail.com, gabriel@info.uaic.ro

[B] Department of Computer Science, Faculty of Informatics,
University of Debrecen, Kassai út 26, 4028 Debrecen, Hungary
{battyanyi.peter,vaszil.gyorgy}@inf.unideb.hu

Membrane systems of P systems is a natural computational model capturing some of the features of living cells organized in tissues and higher ordered structures [5]. The model describes a distributed, parallel, synchronious computational model, where the objects are contained in compartments which are organized in an embedded, tree-like structure. The objects in each compartment evolve in a nondeterministic, highly parallel manner: the rules enable communication of membranes and membrane dissolution as well. At a computational step in each compartment the rules applied to the objects of the compartment are chosen nondeterministically and in a maximally parallel manner, so that no new rule can be applied to remaining object in that compartment. The rules involve communication of membranes by messages: which means that certain labels direct the movement of objects – they can either ooze into the parent membrane, permeate into one of the children membranes or stay in their places. A special message conveys the dissolution of the present membrane.

Here, in addition to the usual definition of rules, we add a function defining costs for the execution of rules or for the preservation of the content of a compartment. We consider the cost only as an external/observer variable, and thus whether a rule is applicable only depends on available resources (not cost value). In our first model, the costs assigned to the rules of the different compartments are kept constant in the membrane system. Additionally, storage costs for all the elements can be given in the compartments (with multiplicities), which are also fixed for a given P system. In the basic interpretation this storage cost is zero. In this talk we

[C]The work of Gy. Vaszil was supported in part by grant no. MAT120558 of the National Research, Development and Innovation Office, Hungary.

present an abstract syntax of the membrane systems with costs, and then define a structural operational semantics of P systems with costs.

Next we consider the model of linearly priced time automata with costs with both transitions and locations in the spirit of the work by Abdulla and Mayr [4] and Alur et al [1]. Timed automata are tools for modelling real-time processes: a discrete transition graph is equipped with a finite set of non-negative real valued variables, the clock variables. The semantics is given by an infinite transition system: the locations are designated by vertices of the transition graph, while the edges represent the different transitions which can be of two kinds: a change of location (discrete transition) or a time consumption (time transition). An edge is annotated with a guard, an action and a reset set. A transition is enabled only if the guard fulfills by the actual valuation. The actions are taken immediately by the transition, and the reset set resets the clock variables forming the set to zero. Additionally, costs are associated to each location and transition, hence costs can be calculated both for discrete and time transition steps.

Finally, we consider priced timed Petri nets [4]. As usual, the Petri net is constituted by places and transitions with the exception that tokens now denote pairs of places and nonnegative real numbers called the age of a place. A marking is a finite multiset of tokens. We distinguish timed and discrete transition relations: a timed transition relation increases the time components of the tokens in the initial marking by a certain positive real value, while the discrete transition relation alters the initial marking itself. Moreover, a cost is calculated by a cost-function for each type of transitions.

We continue the investigations started by Aman and Ciobanu in this area ([2, 3]). Namely, we target the question of relating the P systems with costs to priced timed automata and priced timed Petri nets. In the latter two computational models with costs mainly two kinds of cost problems are considered: the Cost-Threshold Problem, where an evolution cost under a certain threshold value is intended, and the Cost-Optimality Problem, where the minimal evolution cost is computed. In accordance with this, we examine P systems with costs with respect to the Cost-Threshold Problem and the Cost-Optimality Problem. We also intend to investigate networks of P systems with costs (tissues with costs), as well as networks of priced timed automata. It is also an interesting question how different evolution strategies influence the computed cost of reaching a desired configuration.

References

[1] R. ALUR, S. L. TORRE, G. J. PAPPAS, Optimal paths in weighted timed automata. In: M. D. D. BENEDETTO, A. SANGIOVANNI-VINCENTELLI (eds.), *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001 Rome, Italy, March 28-30, 2001 Proceedings*. LNCS 2034, Springer, Berlin, Heidelberg, 2001, 49–62.

[2] B. AMAN, G. CIOBANU, Time Delays in Membrane Systems and Petri Nets. In: M. MASSINK, G. NORMAN (eds.), *Proceedings Ninth Workshop on Quantitative Aspects of Programming Languages, QAPL 2011, Saarbrücken, Germany, April 1–3, 2011.*. Electronic Proceedings in Theoretical Computer Science 57, 2011, 47–60.

[3] B. AMAN, G. CIOBANU, Verifying P Systems with Costs by Using Priced-Timed Maude. In: C. GRACIANI, D. ORELLANA-MARTÍN, A. RISCOS-NÚEZ, Á. ROMERO-JIMÉNEZ, L. VALENCIA-CABRERA (eds.), *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing, 14th BWMC, Sevilla, Spain, February 1–5, 2016*. LNCS 2034, Springer, Berlin, Heidelberg, 2016, 85–96.

[4] R. M. MAYR, P. A. ABDULLA, Priced Timed Petri Nets. *Logical Methods in Computer Science* 9 (2013) 4, 1–51.

[5] GH. PĂUN, Computing with membranes. *Journal of Computer and System Sciences* 61 (2000) 1, 108–143.

A NOVEL STREAM CIPHER BASED ON DETERMINISTIC FINITE AUTOMATON

Pál Dömösi[(A)] Géza Horváth[(A)]

[(A)]Faculty of Informatics, University of Debrecen, H-4028 Debrecen Kassai u. 26. Hungary
domosi@unideb.hu horvath.geza@inf.unideb.hu

Abstract
In this paper we describe a novel symmetric stream cipher based on finite automaton without outputs such that its transition table forms a Latin square. The state and input sets of the key-automaton coincide with the plaintext and also the ciphertext alphabet. During the encryption the plaintext is read in sequentially character by character. After getting the next (initially the first) plaintext character, the system gets simultaneously the next (initially the first) pseudorandom string which is also an input string of the key-automaton. The corresponding ciphertext character will coincide with the state of the key-automaton into which this pseudorandom input string takes the automaton from the state which coincides with the corresponding plaintext character. The decryption works similarly, using a so-called inverse key-automaton instead of the key automaton such that the input strings will be the mirror images of the corresponding pseudorandom strings.

1. Introduction

In this paper we are going to introduce a novel stream cipher based on deterministic finite automaton without outputs which is more simple and more effective than the previously introduced cryptosystems based on automata theory.

1.1. Preliminaries

Automata theory provides a natural basis for designing cryptosystems and several such systems have been designed. Some of them are based on Mealy automata or their generalization, some of them are based on cellular automata, while others are based on compositions of automata.

Almost all cryptosystems can be modelled with Mealy machines (as sequential machines) or generalized sequential machines. During encryption the system first receives a preliminary input, which contains the encryption key automaton (and sometimes a secondary encryption key as well). Following this key the input contains the plaintext stream, as a string of input signals for the automaton, and starting the automaton from a given state, the encrypted message can

indeed be generated as an output signal string initiated by the input signal string. Decryption is performed in a likewise manner: the encryption key is substituted by the decryption key so that plaintext and ciphertext messages change roles. This method has several variants. Many famous mechanical cryptographic machines are concrete examples of this interpretation. Some of these machines are still used as software versions. A further generation of the cryptosystems based on Mealy machines is the family of public key FAPKC systems. They use the mathematical conjecture that it is difficult to find the reverse automaton for delayed, weakly invertible automata. Unfortunately these systems can be defeated. So as to prevent attacks there was developed a refinement of this system, called FAPKC-3. There are two methods to break this cryptographic system.

Besides the vulnerability of FAPKC and FAPKC-3, the main problem of the most cryptosystems based on Mealy machines is that they suffer from the lack of systematic and massive cryptanalytic research.[1] These facts are serious drawbacks in their practical applications.

Almost from the very beginning of research into cellular automata, there have been serious attempts at cryptographic applications. Cryptosystems of this kind usually use the plaintext as the initial state of the cellular automaton, and the encryption key is the transition rules for the cells. The state reached after a given number of steps provides the encrypted message.

The best-known cellular automaton based cryptosystems all share the common problem of serious realization difficulties: some systems are easy to defeat, the technical realization of others result in slow performance, and still others exhibit difficulties in the choice of the key-automaton. A further common drawback of cellular automaton based cryptosystems is that their micro sized technical realization poses serious difficulties and they are not always economical either.

To overcome the discussed problems, a symmetric cryptosystem is described in [1]. It has a Rabin-Scott automaton as key for encryption and decryption. The applied key automaton performs encryption of the plaintext character by character assigning an encrypted counterpart of variable length to each character, the encryption performs a ciphertext with a length substantially exceeding the length of the plaintext. This system has a serious disadvantage that the ciphertext is significantly longer than the plaintext, with the ciphertext even being multiple times longer than the plaintext.

In [2, 3] new block ciphers based on compositions of automata are introduced. Both systems use the following simple idea: Consider a giant-size permutation automaton such that the set of states and the set of inputs consist of all given length of strings over a non-trivial alphabet as all possible plaintext/ciphertext blocks. Moreover consider a cryptographically secure pseudorandom number generator with large periodicity having the property that, getting its really random kernel, it serves a sequence of pseudorandom strings as inputs for the automaton. For each plaintext block the system calculates the new state into which the recent pseudorandom string takes the automaton from the state which is identified as the recent plaintext block.

[1] Among others, the vulnerability of these systems may be due to the well-known fact that automaton mappings are length and prefix preserving, and that gsm mappings are prefix preserving. Therefore, they seem to be vulnerable to adaptive chosen-ciphertext attacks.

The string, identified as the new state, will be the ciphertext block ordered to the considered plaintext block. Of course, the ciphertext will be the catenation of the generated ciphertext blocks. The giant size of the automaton makes it infeasible to break the system by brute-force method.

The problem of this idea is that to store the transition matrix having 2^{128} states and 2^{128} input letters is impossible. The basic idea of this cipher is to operate on a giant secret transition matrix which is compressed into the memory using automata-theoretic methods. This problem can be overcome considering automata which consists of composition of automata. In this case, we should store only the component-automata and the structure of the composition. Moreover, if the component automata are isomorphic to each others then it is enough to store the transition matrix of one component automaton and the structure of the isomorphisms. By this recognition, the storage of automata having 2^{128} states and 2^{128} input letters can be easily solved. Because of the giant size of the matrix, there is no hope to attack the system by brute-force method. On the other hand, this giant matrix can be generated unambiguously by a bitstring of 782 bytes length. Note that this less than 1 kilobyte long string can be generated by an appropriate hash function using a secret password of any length. This cipher overcomes all of the discussed disadvantages, however our new cryptosystem uses more simple operations, because it does not work with automata compositions, the only operator we use is the direct access to an element in the transition matrix.

1.2. Basic Terms and Notations

We start with some standard concepts and notations. By an *alphabet* we mean a finite nonempty set. The elements of an *alphabet* are called *letters*. A *word* over an alphabet Σ is a finite string consisting of letters of Σ. A word over a binary alphabet is called a *bit string*. The string consisting of zero letters is called the *empty word,* written by λ. The *length* of a word w, in symbols $|w|$, means the number of letters in w when each letter is counted as many times as it occurs. By definition, $|\lambda| = 0$. At the same time, for any set H, $|H|$ denotes the cardinality of H. In addition, for every nonempty word w, denote by $\overrightarrow{w}$ the last letter of w. ($\overrightarrow{\lambda}$ is not defined.) If $u = x_1 \cdots x_k$ and $v = x_{k+1} \cdots x_\ell$ are words over an alphabet Σ (with $x_1, \ldots, x_k, x_{k+1}, \ldots, x_\ell \in \Sigma$), then their *catenation* $uv = x_1 \cdots x_k x_{k+1} \cdots x_\ell$ is also a word over Σ. In this case we also say that u is a *prefix* of uv and v is a *suffix* of uv. Catenation is an associative operation and, by definition, the empty word λ is the identity with respect to catenation: $w\lambda = \lambda w = w$ for any word w. For every word $w \in \Sigma^*$, put $w^0 = \lambda$, moreover, $w^n = ww^{n-1}, n \geq 1$. Let Σ^* be the set of all words over Σ, moreover, let $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. Σ^* and Σ^+ are the *free monoid* and the *free semigroup,* respectively, generated by Σ under catenation. In particular, we put $\Sigma^0 = \{\lambda\}, \Sigma^n = \{w : |w| = n\}, n \geq 1$, and $\Sigma^{(0)} = \Sigma^0, \Sigma^{(n)} = \{w : |w| \leq n\}, n \geq 1$. In addition, for every string $x_1 \cdots x_k$ with $x_1, \ldots, x_k \in \Sigma$, the string $x_k \cdots x_1$ is called a mirror image of $x_1 \cdots x_k$. We will use the notation p^R as the mirror image of p for every $p \in \Sigma^+$. Moreover, by definition, let $\lambda^R = \lambda$.

By an *automaton* we mean a deterministic finite automaton without outputs. In more details, an automaton is an algebraic structure $\mathcal{A} = (A, \Sigma, \delta)$ consisting of the nonempty and finite

state set A, the nonempty and finite *input set* Σ, and a *transition function* $\delta : A \times \Sigma \to A$. The elements of the state set are the *states,* and the elements of the input set are the *input signals.* An element of A^+ is called a *state word* [2] and an element of Σ^* is called an input word. State and input words are also called *state strings* and *input strings,* respectively. If a state string $a_1a_2 \cdots a_s$ $(a_1, \ldots, a_s \in A)$ has at least three elements, the states $a_2, a_3, \ldots, a_{s-1}$ are also called intermediate states. It is understood that δ is extended to $\delta^* : A \times \Sigma^* \to A^+$ with $\delta^*(a, \lambda) = a$, $\delta^*(a, xq) = \delta(a, x)\delta^*(\delta(a, x), q), a \in A, x \in \Sigma, q \in \Sigma^*$. In other words, $\delta^*(a, \lambda) = a$ and for every nonempty input word $x_1x_2 \cdots x_s \in \Sigma^+$ (where $x_1, x_2, \ldots, x_s \in \Sigma$) there are $a_1, \ldots, a_s \in A$ with $\delta(a, x_1) = a_1, \delta(a_1, x_2) = a_2, \ldots, \delta(a_{s-1}, x_s) = a_s$ such that $\delta^*(a, x_1 \cdots x_s) = a_1 \cdots a_s$.

In the sequel, we will consider the transition of an automaton in this extended form and thus we will denote it by the same Greek letter δ. If $\overrightarrow{\delta(a, w)} = b$ holds for some $a, b \in A, w \in \Sigma^*$ then we say that w *takes* the automaton from its state a into the state b, and we also say that the automaton *goes* from the state a into the state b under the effect of w.

The transition function can be written in a matrix form, and we call this *transition matrix* a *Latin square* if each row and column is a permutation of states.

2. Results

The working of the considered system mainly differs from the most of the stream ciphers: it does not generate the ciphertexts in such a way that the plaintext bit stream is combined with a cipher bit stream by an exclusive-or operation (XOR). On the other hand, it has the main property of the stream ciphers: the plaintext digits are encrypted one at a time, and the transformation of successive digits varies during the encryption.

The subject matter of the presented work is a symmetric stream cipher based on finite automaton without outputs. For encryption the system uses the transition matrix of a key-automaton without outputs as the secret key. This transition matrix forms a Latin square. For decryption the system also has a so-called inverse key-automaton which moves from a given state b under a given input sign x^R into the state a if and only if the original key-automaton moves from the state a into the state b under the effect of x. The input alphabet, the state set, the plaintext alphabet and the ciphertext alphabet coincide. A further element of the system is a pseudorandom generator which generates pseudorandom character strings with a given length. During encryption the plaintext is read in sequentially character by character and the key automaton assigns to each plaintext character the corresponding state which is the same as the read plaintext character. The corresponding ciphertext character will be the state into which the key-automaton moves from the assigned state under the effect of the next (initially the first) pseudorandom string (with a given length). The apparatus creates the ciphertext by linking these character strings together. During decryption the ciphertext is read in sequentially character by character and the inverse key automaton assigns to each ciphertext

[2] The empty word is not considered as a state word.

character the corresponding state which is the same as the original plaintext character. The corresponding plaintext character will be the state into which the inverse key-automaton moves from the assigned state under the effect of the mirror image of the next pseudorandom string. The apparatus recreates the plaintext by linking these character strings together.

Next we give a formal description of this system.

2.1. Key Automaton

Consider an automaton $\mathcal{A} = (A, \Sigma, \delta)$ with $A = \Sigma$, where for every $a, b \in A$ $(a \neq b)$ and $x, y \in \Sigma$ $(x \neq y)$, $\delta(a, x) \neq \delta(b, x)$ and $\delta(a, x) \neq \delta(a, y)$. Thus, $\mathcal{A}$ is a permutation automaton, i.e., each row of the transition matrix forms a permutation of the state set. This is an essential property to ensure the unambiguity of the ciphertext for any plaintext.

For the security, we also assume that all columns of the transition table also form a permutation of the state set.

Let $\mathcal{A}^{-1} = (A, \Sigma, \delta^{-1})$ be the automaton for which $\delta^{-1}(b, x) = a$ with $a, b \in A,\ x \in \Sigma$ if and only if $\delta(a, x) = b$.

In what follows $\mathcal{A}$ will be called the *key-automaton* and $\mathcal{A}^{-1}$ will be called the *inverse key automaton.*

2.2. Encryption

Let $p_1, \ldots, p_k \in A$ be a plaintext and let $r_1, \ldots, r_k \in \Sigma^+$ be random strings generated by the pseudorandom number generator starting by a seed r_0. We note that $|r_0|, \ldots, |r_k| = n$ holds for a fixed positive integer n.

The ciphertext will be $c_1 \cdots c_k$ with $c_1 = \overrightarrow{\delta(p_1, r_1)}, \ldots, c_k = \overrightarrow{\delta(p_k, r_k)}$.

2.3. Decryption

Let $c_1, \ldots, c_k \in A$ be a ciphertext and let $r_1, \ldots, r_k \in \Sigma^+$ be the same random strings generated by the pseudorandom number generator starting by a seed r_0.

The decrypted plaintext will be $p_1 \cdots p_k$ with $p_1 = \overrightarrow{\delta^{-1}(c_1, (r_1)^R)}, \ldots p_k = \overrightarrow{\delta^{-1}(c_k, (r_k)^R)}$.

2.4. Example

In the following very simple example we are going to use a key automaton $\mathcal{A} = (A, \Sigma, \delta)$, where $A = \Sigma = \{0, 1, 2, 3\}$. We use the $\mathcal{A}^{-1} = (A, \Sigma, \delta^{-1})$ automaton for decryption.

	A			
δ	0	1	2	3
0	1	2	3	0
1	2	0	1	3
2	3	1	0	2
3	0	3	2	1

	A			
δ^{-1}	0	1	2	3
0	3	0	1	2
1	1	2	0	3
2	2	1	3	0
3	0	3	2	1

plaintext: 0123

pseudorandom strings: 11, 21, 30, 31

ciphertext: 1030

3. Conclusions

In this paper we introduced a novel stream cipher based on deterministic finite automaton without outputs. This is just a principle of our novel cryptosystem, we can increase the security of the stream cipher by using different core for the pseudorandom number generator during each encryption and we can also create a block cipher based on this stream cipher which can be a true alternative of the recently used complex block ciphers.

References

[1] P. DÖMÖSI, A Novel Cryptosystem Based on Finite Automata without Outputs. In: M. ITO, Y. KOBAYASHI, K. SHOJI (eds.), *Automata, Formal Languages, and Algebraic Systems*. World Scientific, 2010, 23–32.

[2] P. DÖMÖSI, G. HORVÁTH, A novel cryptosystem based on abstract automata and Latin cubes. *Studia Scientiarum Mathematicarum Hungarica* 52 (2015) 2, 221–232.

[3] P. DÖMÖSI, G. HORVÁTH, A novel cryptosystem based on Gluškov product of automata. *Acta Cybernetica* 22 (2015), 359–371.

VARIANTS OF P COLONY AUTOMATA

Kristóf Kántor György Vaszil

Department of Computer Science, Faculty of Informatics
University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
{kantor.kristof, vaszil.gyorgy}@inf.unideb.hu

Abstract
We give an overview of P colony automata presenting recent results and research directions of the area.

1. Introduction

P colonies are tissue-like membrane systems modeling a community of very simple computing agents (cells), living together and interacting in a shared environment, see [17, 18]. The name *colony* comes from the theory of grammar systems (see [7]), from one of the grammatical models studied in the field called a *colony of grammars*. Such a colony (see also [16]), is a collection of very simple generative grammars (generating finite languages each), but by behaving as a cooperating system, they are able to generate fairly complicated languages, thus, the computing power of the system as a whole increases considerably when compared to the power of the individual components.

P colonies represent this approach in the framework of membrane computing. The model is similar to tissue-like membrane systems, the environment and the computing agents are both described by multisets of objects which are processed by the colony member cells using rules which enable the transformation of the objects and the exchange of objects between the cells and the environment. The capabilities of the computing agents are very restricted, both from the points of view of information storage and information processing possibilities. First, only a limited number of objects are allowed to be present inside the cells simultaneously (this is called the capacity of the system), and second, the rules are very simple, they are either of the form $a \to b$ (for changing an object a into an object b inside the cell), or $a \leftrightarrow b$ (for exchanging an object a inside a cell with an object b in the environment). The rules are grouped into programs. If the capacity of the colony is k, then each program consists of k rules which (when the program is applied) are applied to the k objects simultaneously. A computation of the colony consists of a sequence of computational steps during which the colony member cells execute their programs in parallel, until the system reaches one of the final configurations (usually given as the set of halting configurations, that is, those situations when no programs can be applied by any of the cells).

P colonies have been extensively studied, it has been shown, for example, that although they are extremely simple, they are computationally complete computing devices even with very restricted size parameters and other syntactic or functioning restrictions. For these, and more topics, results, see [5, 6, 4, 3, 8, 9, 12, 13].

As P colonies work with multisets of objects, it is natural to look at the result of the computations as sets of numbers (sets of vectors) represented by the multiplicities of certain objects present in the final configurations. On the other hand, being able to describe sets of strings instead of numbers, is also of interest. P colony automata were introduced in [2] for this purpose: to be able to accept with P colonies strings and string languages (instead of multiset collections). The idea of P colony automata is to assume the presence of an input tape with an input string, and designate certain rules as tape rules. When such a tape rule is applied, the symbol which is processed by the rule should also be read from the input tape. The difficulty of this idea comes from the possibility of applying several programs, and thus, several tape rules simultaneously, which gives rise to possible conflicts between tape rules which would need to read different symbols from the same input. Nevertheless, several variants of P colony automata turned out to be computationally complete, as shown in [2] and later in [1].

Here we consider a different way of dealing with strings in P colonies. The model was introduced in [15] under the name of generalized P colony automata, and studied further in [14]. The idea is to define the reading of input symbols in a way that is more close to the nature of other kinds of membrane computing systems, especially antiport P systems and P automata in particular. P automata, introduced in [11] (see also [10]), are P systems using symport and antiport rules (see [19]), characterizing string languages in a different way than "ordinary" P colony automata. They do not have input tapes with predefined input strings. Instead of reading input tapes, they associate strings to their computations by keeping track of the communication with the environment. They are not forced to a certain behavior through the given input, but operate and communicate freely with the environment, where each object which can be requested for input by the communication rules of the P system is assumed to be available in an unbounded supply. The accepted strings (the strings which are said to be read by the system) are determined by the sequences of those multisets which enter the system from the environment during computations. Such a sequence of multisets is mapped to a string, a sequence of symbols which constitute the string accepted by a particular computation.

A similar idea is employed in generalized P colony automata, the idea of characterizing strings through the sequences of multisets processed during computations. The computations of the colony define accepted multiset sequences, which are turned into accepted strings by mapping the multiset sequences to symbol sequences (strings) over some previously given alphabet. They also have rules distinguished as tape rules, and the application of such a tape rule also implies the reading of the processed symbol from the input (as in "ordinary" P colony automata), but unlike in the original model, the automaton is allowed to read more than one such symbol in a single computational step. This way generalized P colony automata avoid the conflicts that would arise by the simultaneous use of tape rules processing (and therefore reading) different symbols, but they may read several symbols (a multiset of symbols) in one computational step. This means that during a computation consisting of a sequence of computational steps, a sequence of multisets is read from the input. This sequence of multisets then can be mapped

into a string (a sequence of symbols) in a similar way as in P automata.

In [15], some basic variants of the model were introduced and studied from the point of view of their computational power. In [14] we continued the investigations structuring our results around the capacity of the systems, and different types of restrictions imposed on the use of tape rules in the programs of the systems. We considered three possible ways of dealing with tape rules in the programs: (1) the unrestricted case, (2) the case when all programs must contain at least one tape rule (all-tape programs), and (3) the case when all communication rules are tape rules (com-tape programs).

2. P Automata and Generalized P Colony Automata

A *genPCol automaton* of capacity k and with n cells, $k, n \geq 1$, is a construct

$$\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$$

where

- V is an *alphabet*, the alphabet of the automaton, its elements are called *objects*;
- $e \in V$ is the *environmental object* of the automaton, the only object which is assumed to be available in an arbitrary, unbounded number of copies in the environment;
- $w_E \in (V - \{e\})^*$ is a string representing the multiset of objects different from e which is found in the environment initially;
- $(w_i, P_i), 1 \leq i \leq n$, specifies the i-th *cell* where w_i is a multiset over V, it determines the initial contents of the cell, and its cardinality $|w_i| = k$ is called the *capacity* of the system. P_i is a set of *programs*, each program is formed from k rules of the following types (where $a, b \in V$):
  - *tape rules* of the form $a \overset{T}{\to} b$, or $a \overset{T}{\leftrightarrow} b$, called rewriting tape rules and communication tape rules, respectively; or
  - *nontape rules* of the form $a \to b$, or $a \leftrightarrow b$, called rewriting (nontape) rules and communication (nontape) rules, respectively.

  A program is called a *tape program* if it contains at least one tape rule.
- F is a set of *accepting configurations* of the automaton which we will specify in more detail below.

A genPCol automaton reads an input word during a computation. A part of the input (possibly consisting of more than one symbols) is read during each configuration change: the processed part of the input corresponds to the multiset of symbols introduced by the tape rules of the system.

A *configuration* of a genPCol automaton is an $(n + 1)$-tuple $(u_E, u_1, \ldots, u_n)$, where $u_E \in (V - \{e\})^*$ represents the multiset of objects different from e in the environment, and $u_i \in V^*$, $1 \leq i \leq n$, represent the contents of the i-th cell. The *initial configuration* is given by

$(w_E, w_1, \ldots, w_n)$, the initial contents of the environment and the cells. The elements of the set F of *accepting configurations* are given as configurations of the form $(v_E, v_1, \ldots, v_n)$, where

- $v_E \subseteq (V - \{e\})^*$ represents a multiset of objects different from e being in the environment, and each
- $v_i \in V^*,\ 1 \leq i \leq n$, is the contents of the i-th cell.

Instead of the different computational modes used in [2], in genPCol automata, we apply the programs in the maximally parallel way, that is, in each computational step, every component cell non-determinically applies one of its applicable programs. Then we collect all the symbols that the tape rules "read" (these multisets are denoted by $read(p)$ for a program p in the definition below), this is the multiset read by the system in the given computational step. A successful computation defines this way an accepted sequence of multisets: the sequence of multisets entering the system during the steps of the computation.

Let $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ be a genPCol automaton. The *set of input sequences accepted by* Π is defined as

$$\begin{aligned} A(\Pi) = \{u_1 u_2 \ldots u_s \mid\ & u_i \in (V - \{e\})^*,\ 1 \leq i \leq s, \text{ and there is a configuration} \\ & \text{sequence } c_0, \ldots, c_s, \text{ with } c_0 = (w_E, w_1, \ldots, w_n),\ c_s \in F, \text{ and} \\ & c_i \Longrightarrow c_{i+1} \text{ with } \bigcup_{p \in P_{c_i}} read(p) = u_{i+1} \text{ for all } 0 \leq i \leq s-1\}. \end{aligned}$$

Let Π be a genPCol automaton, and let $f : (V - \{e\})^* \to 2^{\Sigma^*}$ be a mapping, such that $f(u) = \varepsilon$ if and only if u is the empty multiset.

The *language accepted by* Π with respect to f is defined as

$$L(\Pi, f) = \{f(u_1)f(u_2)\ldots f(u_s) \in \Sigma^* \mid\ u_1 u_2 \ldots u_s \in A(\Pi)\}.$$

We define the following language classes.

- $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{com-tape}(k))$ is the class of languages accepted by generalized PCol automata with capacity k and with mappings from the class $\mathcal{F}$ where all the communication rules are tape rules,
- $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{all-tape}(k))$ is the class of languages accepted by generalized PCol automata with capacity k and with mappings from the class $\mathcal{F}$ where all the programs must have at least one tape rule,
- $\mathcal{L}(\text{genPCol}, \mathcal{F}, *(k))$ is the class of languages accepted by generalized PCol automata with capacity k and with mappings from the class $\mathcal{F}$ where programs with any kinds of rules are allowed.

Let the mapping f_{perm} and the class of mappings TRANS be defined as follows:

- $f_{perm} : V^* \to 2^{\Sigma^*}$ where $V = \Sigma$ and for all $v \in V^*$, we have $f(v) = \{a_1 a_2 \ldots a_s \mid |v| = s,$ and $a_1 a_2 \ldots a_s$ is a permutation of the elements of $v\}$;

- for some $f : V^* \to 2^{\Sigma^*}$, we say that $f \in$ TRANS if for any $v \in V^*$, we have $f(v) = \{w\}$ for some $w \in \Sigma^*$ which is obtained by applying a finite transducer to the string representation of the multiset v, (as w is unique, the transducer must be constructed in such a way that all string representations of the multiset v as input result in the same $w \in \Sigma^*$ as output, and moreover, as f should be nonerasing, the transducer produces a result with $w \neq \varepsilon$ for any nonempty input).

We denote these language classes as $\mathcal{L}_X(\text{genPCol}, Y(k))$, where $X \in \{f_{perm}, \text{TRANS}\}$, $Y \in \{\text{com-tape}, \text{all-tape}, *\}$.

Now we present an example to demonstrate the above defined notions.

Example 2.1 Let $\Pi = (\{a, b, c\}, e, \emptyset, (ea, P), F)$ be a genPCol automaton where

$$P = \{\langle e \to a, a \overset{T}{\leftrightarrow} e\rangle, \langle e \to b, a \overset{T}{\leftrightarrow} e\rangle, \langle e \to b, b \overset{T}{\leftrightarrow} a\rangle, \langle e \to c, b \overset{T}{\leftrightarrow} a\rangle, \\ \langle a \to b, b \overset{T}{\leftrightarrow} a\rangle, \langle a \to c, b \overset{T}{\leftrightarrow} a\rangle\}$$

with all the communication rules being tape rules. Let also $F = \{(\varepsilon; v, ca) \mid a \notin v\}$ be the set of final configurations. A possible computation of this system is the following:

$$(\emptyset, ea) \Rightarrow (a, ea) \Rightarrow (aa, ea) \Rightarrow (aaa, eb) \Rightarrow (aab, ba) \Rightarrow (bba, ba) \Rightarrow (bbb, ac)$$

where the first three computational steps read the multiset containing an a, the last three steps read a multiset containing a b, thus the accepted multiset sequence of this computation is $(a)(a)(a)(b)(b)(b)$.

It is not difficult to see that similarly to the one above, the computations which end in a final configuration (a configuration which does not containg the object a in the environment) accept the set of multiset sequences $A(\Pi) = \{(a)^n(b)^n \mid n \geq 1\}$.

If we consider f_{perm} as the input mapping, we have $L(\Pi, f_{perm}) = \{a^n b^n \mid n \geq 1\}$. On the other hand, if we consider a mapping $f_1 \in$ TRANS with $f_1 : \{a, b\}^* \to \{c, d, e, f\}^*$ and $f_1(a) = \{cd\}$, $f_1(b) = \{ef\}$ (and f_1 undefined in all other cases), we get the language $L(\Pi, f_1) = \{(cd)^n(ef)^n \mid n \geq 1\}$.

3. Recent Results on Systems with Input Mappings from TRANS

For any class of mappings $\mathcal{F}$, we have (see [14])

1. $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{com-tape}(k)) \subseteq \mathcal{L}(\text{genPCol}, \mathcal{F}, *(k))$ and
   $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{all-tape}(k)) \subseteq \mathcal{L}(\text{genPCol}, \mathcal{F}, *(k)$ for $k \geq 1$; and
2. $\mathcal{L}(\text{genPCol}, \mathcal{F}, X(k)) \subseteq \mathcal{L}(\text{genPCol}, \mathcal{F}, X(k+1))$ for $k \geq 1$, $X \in \{\text{com-tape}, \text{all-tape}, *\}$.

The computational capacity of genPCol automata with input mappimg f_{perm} was investigated in [15] and [14]. It was shown that $\mathcal{L}_{perm}(\text{genPCol}, *(1)) = \mathcal{L}(RE)$, thus, it is not surprising, but the same holds also for the class of mappings TRANS.

Proposition 3.1

$$\mathcal{L}_{TRANS}(\text{genPCol}, *(1)) = \mathcal{L}(RE).$$

A similar result holds for all-tape systems with capacity at least two. From [14] we have that $\mathcal{L}_{perm}(\text{genPCol}, \text{all-tape}(k)) = \mathcal{L}(\text{RE})$ for $k \geq 2$, and we can show the same for systems with input mappings from TRANS.

Proposition 3.2

$$\mathcal{L}_{TRANS}(\text{genPCol}, \text{all-tape}(k)) = \mathcal{L}(\text{RE}) \textit{ for } k \geq 2.$$

For systems with capacity one, it is not difficult to see that all regular langugaes can be characterized, but a more precise characterization of the corresponding langugae classes are still missing.

Proposition 3.3

$$\text{REG} \subseteq \mathcal{L}_{\text{TRANS}}(\text{genPCol}, \text{X}(1)), \textit{ for } \text{X} \in \{\text{all-tape}, \text{com-tape}\}.$$

The characterization of langugaes of com-tape systems is an interesting research direction. Similarly to systems with input mapping f_{perm}, we have the following, where r-1LOGSPACE denotes the class of languages characterized by so-called *restricted one-way logarithmic space* bounded Turing machines, see [10] for more on this complexity class.

Proposition 3.4

$$\mathcal{L}_{TRANS}(\text{genPCol}, \text{com-tape}(2)) \subseteq \text{r-1LOGSPACE}.$$

4. Conclusions

As the class of languages characterized by P automata is strictly included in r-1LOGSPACE, the above theorem does not give any information on the relationship of the power of P automata and genPCol automata. We know, however, that genPCol automata with f_{perm} and com-tape programs are more powerful than P automata using the mapping f_{perm}.

As P automata with sequential rule application and input mappings from TRANS characterize exactly the language class r-1LOGSPACE, the relationship of this language class and genPCol automata with input mappings from TRANS seems to be an especially interesting research direction.

Further, the effect of using *checking rules*, as defined in [17] for P colonies, is also an interesting topic for further investigations, just as the investigation of systems with other classes of input mappings besides f_{perm}.

Acknowledgements

This research was supported in part by grant no. MAT120558 of the National Research, Development and Innovation Office, Hungary.

References

[1] L. CIENCIALA, L. CIENCIALOVÁ, P Colonies and Their Extensions. In: J. KELEMEN, A. KELEMENOVÁ (eds.), *Computation, Cooperation, and Life – Essays Dedicated to Gheorghe Paun on the Occasion of His 60th Birthday*. Lecture Notes in Computer Science 6610, Springer, 2011, 158–169.

[2] L. CIENCIALA, L. CIENCIALOVÁ, E. CSUHAJ-VARJÚ, G. VASZIL, PCol automata: Recognizing strings with P colonies. In: M. A. MARTÍNEZ DEL AMOR, G. PĂUN, I. PÉREZ HURTADO, A. RISCOS NUÑEZ (eds.), *Eighth Brainstorming Week on Membrane Computing, Sevilla, February 1–5, 2010*. Fénix Editora, 2010, 65–76.

[3] L. CIENCIALA, L. CIENCIALOVÁ, A. KELEMENOVÁ, On the Number of Agents in P Colonies. In: G. ELEFTHERAKIS, P. KEFALAS, G. PAUN, G. ROZENBERG, A. SALOMAA (eds.), *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25–28, 2007 Revised Selected and Invited Papers*. Lecture Notes in Computer Science 4860, Springer, 2007, 193–208.

[4] L. CIENCIALA, L. CIENCIALOVÁ, A. KELEMENOVÁ, Homogeneous P Colonies. *Computing and Informatics* 27 (2008) 3+, 481–496.

[5] L. CIENCIALOVÁ, L. CIENCIALA, Variation on the theme: P colonies. In: D. KOLĂR, A. MEDUNA (eds.), *Proc. 1st Intern. Workshop on Formal Models*. Ostrava, 2006, 27–34.

[6] L. CIENCIALOVÁ, E. CSUHAJ-VARJÚ, A. KELEMENOVÁ, G. VASZIL, Variants of P colonies with very simple cell structure. *International Journal of Computers, Communication and Control* 4 (2009) 3, 224–233.

[7] E. CSUHAJ-VARJÚ, J. DASSOW, J. KELEMEN, G. PĂUN, *Grammar Systems – A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach, London, 1994.

[8] E. CSUHAJ-VARJÚ, J. KELEMEN, A. KELEMENOVÁ, Computing with Cells in Environment: P Colonies. *Multiple-Valued Logic and Soft Computing* 12 (2006) 3–4, 201–215.

[9] E. CSUHAJ-VARJÚ, M. MARGENSTERN, G. VASZIL, P Colonies with a Bounded Number of Cells and Programs. In: H. J. HOOGEBOOM, G. PAUN, G. ROZENBERG, A. SALOMAA (eds.), *Membrane Computing, 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 17–21, 2006, Revised, Selected, and Invited Papers*. Lecture Notes in Computer Science 4361, Springer, 2006, 352–366.

[10] E. CSUHAJ-VARJÚ, M. OSWALD, G. VASZIL, P automata. In: G. PAUN, G. ROZENBERG, A. SALOMAA (eds.), *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., 2010.

[11] E. CSUHAJ-VARJÚ, G. VASZIL, P Automata or Purely Communicating Accepting P Systems. In: G. PAUN, G. ROZENBERG, A. SALOMAA, C. ZANDRON (eds.), *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19–23, 2002, Revised Papers*. Lecture Notes in Computer Science 2597, Springer, 2002, 219–233.

[12] R. FREUND, M. OSWALD, P Colonies Working in the Maximally Parallel and in the Sequential Mode. In: D. ZAHARIE, D. PETCU, V. NEGRU, T. JEBELEAN, G. CIOBANU, A. CICORTAS, A. ABRAHAM, M. PAPRZYCKI (eds.), *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005), 25–29 September 2005, Timisoara, Romania*. IEEE Computer Society, 2005, 419–426.

[13] R. FREUND, M. OSWALD, P colonies and prescribed teams. *Int. J. Comput. Math.* 83 (2006) 7, 569–592.

[14] K. KÁNTOR, G. VASZIL, On the Class of Languages Characterized by Generalized P Colony Automata. *Theoretical Computer Science* to appear.

[15] K. KÁNTOR, G. VASZIL, Generalized P Colony Automata. *Journal of Automata, Languages and Combinatorics* 19 (2014) 1–4, 145–156.

[16] J. KELEMEN, A. KELEMENOVÁ, A grammar-theoretic treatment of multiagent systems. *Cybernetics and Systems* 23 (1992), 621–633.

[17] J. KELEMEN, A. KELEMENOVA, G. PAUN, Preview of P colonies: A biochemically inspired computing model. In: *Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems, ALIFE IX, Boston, Mass*. 2004, 82–86.

[18] A. KELEMENOVÁ, P colonies. In: G. PAUN, G. ROZENBERG, A. SALOMAA (eds.), *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., 2010, 584–593.

[19] A. PAUN, G. PĂUN, The Power of Communication: P Systems with Symport/Antiport. *New Generation Comput.* 20 (2002) 3, 295–306.

QUANTUM AUTOMATA FOR ONLINE MINIMIZATION PROBLEMS

Kamil Khadiev[(A)] Aliya Khadieva[(B)]

[(A)]University of Latvia, Riga, Latvia
and Kazan Federal University, Kazan, Russia
kamilhadi@gmail.com

[(B)]Kazan Federal University, Kazan, Russia
aliya.khadi@gmail.com

Abstract
Online minimization problems is a well-known problems in the online algorithms complexity. We explore a model with restricted memory and use automata as algorithms for solving problems. We allow to use a non-constant number of states for automata and consider quantum and classical automata. We show that quantum automata can be better than classical ones (deterministic or probabilistic) for a subpolynomial number of states. Additionally, we consider a polynomial number of states. And we show that in this case, quantum automata can be better than deterministic ones as well.

1. Introduction

Online algorithms are well-known computational model for solving optimization problems. The peculiarity is that algorithm reads an input piece by piece and should return an answer piece by piece immediately, even if an answer can depend on future pieces of the input. An online algorithm should return an output for minimizing an objective function. There are different methods to define the effectiveness of the algorithms [13, 14, 16]. But a most standard is the competitive ratio [19, 22]. It is a ratio between output's price for an online algorithm and optimal offline algorithms.

We suggest a new model for online algorithms, *quantum online algorithms* that use a power of quantum computing for solving online minimization problem. We discuss the online algorithms with restricted memory. For this kind of algorithms, we allow to use only s bits of memory, for given integer s. In fact, we consider automaton with non-constant number of states [10, 11] or streaming algorithm [21, 17] as an online algorithm. Classical streaming algorithms for online minimization problems were considered in [12, 18, 15]. In this case, we show that quantum automata with single qubit can be better than any classical (deterministic or probabilistic) automata with a subpolynomial number of states. It is also interesting to investigate the model with a polynomial number of states. Here the automaton can use $n^{O(1)}$ states. We show that

for (n, k, r, w)-Parity for Number of Equality Hats problem a quantum automaton can be better than any deterministic one.

The paper is organized in the following way. We present definitions in Section 2. In Section 3 we explore automata for online minimization problems.

2. Preliminaries

Firstly, let us define an online optimization problem. All following definitions we give with respect to [20].

Definition 2.1 (Online Minimization Problem) *An online minimization problem consists of a set $\mathcal{I}$ of inputs and a cost function. Every input $I \in \mathcal{I}$ is a sequence of requests $I = (x_1, \ldots, x_n)$. Furthermore, a set of feasible outputs (or solutions) is associated with every I; every output is a sequence of answers $O = (y_1, \ldots, y_n)$. The cost function assigns a positive real value $cost(I, O)$ to every input I and any feasible output O. For every input I, we call any feasible output O for I that has the smallest possible cost (i. e., that minimizes the cost function) an optimal solution for I.*

Let us define an online algorithm for this problem as an algorithm which gets requests I one by one and should return answers O immediately, even if an optimal solution can depend on future requests.

Definition 2.2 (Deterministic Online Algorithm) *Consider an input I of an online minimization problem. An online algorithm A computes the output sequence $A(I) = (y_1, \ldots, y_n)$ such that y_i is computed from $x_1, \ldots, x_i$, $y_1, \ldots, y_{i-1}$. We denote the cost of the computed output by $cost(A(I)) = cost(I, A(I))$.*

This setting can also be regarded as a request-answer game: an adversary generates requests, and an algorithm has to serve them one at a time [6].

As the main measure of quality of an online algorithm, we use a competitive ratio. It is the ratio of two costs: cost for an online algorithm's solution; and cost for an optimal offline algorithm solution. We consider the worst case.

Definition 2.3 (Competitive Ratio) *An online algorithm A is c-competitive if there exists a non-negative constant α such that, for every input I, we have: $cost(A(I)) \leq c \cdot cost(Opt(I)) + \alpha$, where Opt is an optimal offline algorithm for the problem. We also call c the competitive ratio of A. If $\alpha = 0$, then A is called strictly c-competitive; A is optimal if it is strictly 1-competitive.*

Next, let us define a randomized online algorithm.

Definition 2.4 (Randomized Online Algorithm) *Consider an input I of an online minimization problem. A randomized online algorithm R computes the output sequence $R^{\psi} :=$*

$R^\psi(I) = (y_1, \cdots, y_n)$ such that y_i is computed from $\psi, x_1, \cdots, x_i$, where ψ is the content of a random tape, i. e., an infinite binary sequence, where every bit is chosen uniformly at random and independently of all the others. By $cost(R^\psi(I))$ we denote the random variable expressing the cost of the solution computed by R on I. R is c-competitive in expectation if there exists a non-negative constant α such that, for every I, $\mathbb{E}[cost(R^\psi(I))] \leq c \cdot cost(Opt(I)) + \alpha$, where Opt is an optimal offline algorithm for the problem.

Now we are ready to define a *quantum online algorithm*. You can read more about quantum computation in [11]

Definition 2.5 (Quantum Online Algorithm) *Consider an input I of an online minimization problem. A quantum online algorithm Q computes the output sequence $Q(I) = (y_1, \cdots, y_n)$ such that y_i is computed from $x_1, \cdots, x_i$. Q can have only quantum part. The algorithm can measure qubits several times during computation. By $cost(Q(I))$ we denote the cost of the solution computed by Q on I. Note that quantum computation is probabilistic process. Q is c-competitive in expectation if there exists a non-negative constant α such that, for every I, $\mathbb{E}[cost(Q(I))] \leq c \cdot cost(Opt(I)) + \alpha$, where Opt is an optimal offline algorithm for the problem.*

As algorithms, we will consider automata or online algorithms with restricted memory. Let deterministic online algorithm A_s be an algorithm which uses at most s bits of memory on processing any input I. Or A_s is automata which use 2^s states. We can define similar restrictions for randomized algorithms and algorithms with advice.

3. Main Results

Let us focus on space complexity of online algorithms. It is interesting to analyze size of memory that is required by the algorithm. In a case of the restricted memory, a quantum algorithm can be better than classical ones (deterministic or probabilistic). We present this result in Theorems 3.1, 3.4 and 3.5.

Let us consider the special problem which allows us to show the separation: (n, k, r, w)-*Parity for Number of Hats* ((n, k, r, w)-PNH).

Definition of (n, k, r, w)-PNH problem is based on definition of $PartialMOD_n^k$ function from [10, 1, 2]. Feasible inputs for the problem are $X = (x_1, \ldots, x_n)$, for $x_1, \ldots, x_n \in \{0, 1\}$ such that $\#_1(X) = v \cdot 2^k$, where $\#_1(X)$ is the number of 1s and $v \geq 2$ is a positive integer. $PartialMOD_n^k(X) = v \mod 2$.

Firstly, let us describe (n, k, r, w)-PNH problem informally. There are 3 guardians and 3 prisoners. They stay one by one in a line "$G_1P_1G_2P_2G_3P_3$, G_i is guardian and P_i is prisoner. Prisoner P_i has an input X_i of length m_i and computes function $PartialMOD_{m_i}^k(X_i)$ for $i \in \{1, 2, 3\}$. If the result is 1 then he paints his hat in black. Otherwise, he paints it in white. Each guardian wants to know if the number of following black hats is odd or even. The cost of right guardian's

answer is r, and the cost of the wrong answer is w. We want to minimize the cost of output, and assume that $r < w$.

Formal definition of (n, k, r, w)-PNH is following: Feasible inputs for the problem are $I = (x_1, \ldots, x_n)$ of length n such that $n = m_1 + m_2 + m_3 + 3$, for some integer $m_1, m_2, m_3 \geq 2^{k+1}$. It is guarantied that I is always such that $I = 2, X_1, 2, X_2, 2, X_3$, where $X_i \in \{0,1\}^{m_i}$, for $i \in \{1,2,3\}$. Additionally, $\#_1(X_i) = v_i \cdot 2^k$, where v_i is some integer, $i \in \{1,2,3\}$. Let O be output of (n, k, r, w)-PNH and $O' = y_1, y_3, y_3$ be output bits corresponding to input variables with value 2 (in other words, variables of guardians). Output y_1 corresponds to x_1, y_2 corresponds to x_{2+m_1} and , y_3 corresponds to $x_{3+m_1+m_2}$. Let $z_j(I) = \bigoplus_{i=j}^{3} PartialMOD_{m_i}^k(X_i)$. The cost $cost(I, O) = r$, if $y_j = z_j$ for all $j \in \{1,2,3\}$, and $cost(I, O) = w$ otherwise. We consider numbers r and w such that $r < w$.

Let us present a quantum online algorithm which uses single qubit of memory for this problem. It uses ideas from quantum automata [10] and branching programs theories [1, 2].

Algorithm 1. (Quantum Online Algorithm for (n, k, r, w)-PNH) The quantum algorithm Q_1 uses single qubit.

Step 1. The algorithm emulates guessing for $z_1(I)$. Q_1 starts on a state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. And it measures the qubit before reading any input variables. It gets $|0\rangle$ or $|1\rangle$ with equal probability. The result of measurement is y_1.

Step 2. The algorithm reads X_1. Let angle $\alpha = \pi/2^{k+1}$. Q_1 rotates the qubit by an angle α if the algorithm meets 1. And it does not do anything otherwise.

Step 3. If Q_1 meets 2 then it measures the qubit $|\psi\rangle = a|0\rangle + b|1\rangle$. If $PartialMOD_{m_1}^k(X_1) = 1$ then the qubit is rotated by an angle $\pi/2 + v \cdot \pi$, for some integer v, else the qubit is rotated by an angle $w \cdot \pi$, for some integer w. If $y_1 = 1$, then $a \in \{1, -1\}$ and $b = 0$. And if $y_1 = 0$, then $a = 0$ and $b \in \{1, -1\}$. The result of measurement is y_2.

Step 4. The step is similar to Step 2, but algorithm reads X_2.

Step 5. The step is similar to Step 3, but algorithm outputs y_3.

Step 6. The algorithm reads and skips the last part of the input. Q_1 does not need these variables, because it guesses y_1 and using this value we already can obtain y_2 and y_3 without X_3.

Assume, that algorithm did right guess that $z_1 = y_1$. So, if the parity of the passing part is the same as the parity of the future part of the input, then the algorithm returns the right answer with probability 1. And if the guess is not correct and $z_1 \neq y_1$, then the algorithm returns a wrong answer with probability 1.

With equal probabilities 0.5 we have $z_1 = y_1$ or $z_1 \neq y_1$. Thus competitive ratio is $(0.5 \cdot r + 0.5 \cdot w)/r = (r + w)/(2r)$.

As a result we have the following theorem:

Theorem 3.1 *There is $(r+w)/(2r)$-competitive in expectation quantum algorithms for (n, k, r, w)-PNH Problem Q_1 with a single qubit of memory.*

At the same time, if a deterministic online algorithm for (n, k, r, w)-PNH Problem uses less than k bits; then it is (w/r)-competitive. To show this claim, let us discuss properties of $PartialMOD_n^k$ function, based on results for automata and branching programs[10, 1, 2].

Lemma 3.2 *Let integer s, k be such that $s < k = o(\log(n))$, where n is the length of input. Then there is no deterministic algorithm that reads an input variable by variable, uses s bits of memory and computes $PartialMOD_n^k(X)$.*

Lemma 3.3 *Let integer s, k be such that $s < k = o(\log(n))$, where n is the length of input. Then there is no randomized algorithm that reads an input variable by variable, uses s bits of memory and computes $PartialMOD_n^k(X)$ with probability of error less than 0.5.*

Now we can discuss deterministic online algorithms for (n, k, r, w)-PNH Problem.

Theorem 3.4 *Let integer s, k be such that $s < k = o(\log(n))$, where n is the length of input. Any deterministic online algorithm A_s computing (n, k, w, r)-PNH Problem is (w/r)-competitive.*

Proof. Let us assume that we have such an algorithm A_s. Then we suggest the input $I = (x_1, \ldots, x_n)$ such that A_s returns the wrong answer on all requests of guardians. Let $m_1 = m_2 = m_3 = m = 3 \cdot 2^k$.

The first guardian answers y_i. Due to Lemma 3.2 we can choose input $X_1 \in \{0,1\}^m$ such that A_s cannot compute $PartialMOD_m^k(X_1)$. It means that we can choose X^1 such that $y_2 = v_1 \oplus y_1$, for $v_1 = (\#_1(X_1)/2^k)) \ mod\ 2$. By the same reason we can pick X^2 such that $y_3 = v_2 \oplus y_2$, for $v_2 = (\#_1(X_2)/2^k)) \ mod\ 2$. Let us choose the input X^3 such that $v_3 \oplus y_3 = 1$, for $v_3 = (\#_1(X_3)/2^k)) \ mod\ 2$. Note that we guarantee that $\#_1(X_i)/2^k$ is an integer, for $i \in \{1, 2, 3\}$.

Therefore, we have: $z_3 = (\#_1(X^3)/2^k) \ mod\ 2 \neq y_3$, $z_2 = (\#_1(X_2, X_3)/2^k) \ mod\ 2 = v_2 \oplus v_3 \neq y_2$, $z_1 = (\#_1(X_1, X_2, X_3)/2^k) \ mod\ 2 = v_1 \oplus v_2 \oplus v_3 \neq y_1$. We got a contradiction for input $(2, X^1, 2, X^2, 2, X^3)$. Hence the cost of output is w and the competitive ration is w/r. □

Theorem 3.5 *Let integer s, k be such that $s < k = o(\log(n))$, where n is the length of input. Any randomize online algorithm R_s computing (n, k, w, r)-PNH Problem is $(r + 7w)/(8r)$-competitive.*

Proof. By the same way as in the previous Theorem we can show that for any algorithm R_s we can suggest the input such that it cannot say anything better than just guessing answers with probabilities 0.5. Therefore expected cost is $(r + 7w)/8$ and expected competitive ratio is $(r + 7w)/(8r)$. □

It is easy to see that $(r + 7w)/(8r) > (w + r)/(2r)$ and $w/r > (w + r)/(2r)$ due to $r < w$. Therefore the quantum algorithm is better than any deterministic or randomize online algorithm that uses less than k bits of memory.

3.1. Polylogarithmic Space Complexity Separation between Quantum and Deterministic Online Algorithms with Polylogarithmic Memory

Let us consider polylogarithmic memory case. We present separation between quantum and deterministic models in Theorems 3.6 and 3.8

Let us consider modification of (n, k, r, w)-PNH problem called (n, r, w)-Parity Number of Equality Hats or (n, r, w)-PNEH. It is the same problem, but we use $EQ_m(X)$ function instead of $PartialMOD_m^k$. Boolean function $EQ_m : \{0,1\}^n \to \{0,1\}$ is such that $EQ(x_1, \dots x_{\lfloor m/2 \rfloor}, x_{\lfloor m/2 \rfloor+1}, \dots x_m) = 1$, if $(x_1, \dots x_{\lfloor m/2 \rfloor}) = (x_{\lfloor m/2 \rfloor+1}, \dots x_m)$, and 0 otherwise. So $z_j(I) = \bigoplus_{i=j}^{3} EQ_{m_i}(X_i)$. We suppose that $m > 1, r < w$.

Let us construct a quantum online algorithm that uses $O(\log n)$ and solves (n,r,w)$-PNEH$.

Algorithm 2. (Quantum Algorithm for (n, r, w)-PNEH) The quantum algorithm $Q = Q_{O(\log n)}$ uses $O(\log n)$ qubits.

Step 1. The algorithm emulates guessing for $z_1(I)$. Q initializes the qubit $|\psi\rangle = |\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. And it measures the qubit before reading any input variables. It gets $|0\rangle$ or $|1\rangle$ with equal probability. The result of measurement is y_1.

Step 2. The algorithm takes X_1 and obtains a value of the function $v_1 = EQ(X_1)$ using *quantum fingerprinting method* presented in [5, 4, 3], [8, 9, 7]. This method allows to compute the function EQ_{m_1} with probability of error by any fixed constant $\varepsilon > 0$. The method uses $O(\log m_1)$ qubits. For 0-instances, probability of error is ε. And for 1-instances probability of error is 0.

Step 3. If Q meets 2 then it takes a value v_1. Let the value be in qubit $|\phi\rangle$. Then the algorithm applies $CNOT$ gate for $|\phi\rangle|\psi\rangle$. After that $|\psi\rangle = |y_1 \oplus v_1\rangle$. Then Q measure $|\psi\rangle$ and returns y_2.

Step 4. The step is similar to Step 2, but algorithm reads X_2.

Step 5. The step is similar to Step 3, but algorithm outputs y_3.

Step 6. The algorithm reads and skips the last part of the input. Q does not need these variables, because it guesses y_1 and using this value we already can obtain y_2 and y_3 without X_3.

Let us compute an expected cost of pairs $(I, Q(I))$. $cost(Q(I_{000})) = cost(Q(I_{001})) = r(1 - \varepsilon)^2/2 + w(\frac{1-\varepsilon^2}{2} + \varepsilon)$ $cost(Q(I_{010})) = cost(Q(I_{011})) = cost(Q(I_{100})) = cost(Q(I_{101})) = r(1 - \varepsilon)/2 + w(1 + \varepsilon)/2$ $cost(Q(I_{110})) = cost(Q(I_{111})) = r/2 + w/2$

So, expected ratio is $(r(1-\varepsilon)^2/2 + w(\frac{1-\varepsilon^2}{2} + \varepsilon))/r$. As a result we have the following theorem:

Theorem 3.6 *There is $(r(1-\varepsilon)^2/2 + w(\frac{1-\varepsilon^2}{2} + \varepsilon))/r$-competitive in expectation quantum algorithms for (n, r, w)-PNEH Problem $Q_{O(\log n)}$ with $O(\log n)$ qubits of memory.*

At the same time, if a deterministic online algorithm for (n, k, r, w)-PNH Problem use polylogarithmic number of bits; then it is (w/r)-competitive. To show this claim, let us discuss a required property of EQ_m function.

Lemma 3.7 *There is no deterministic algorithm that reads an input variable by variable, uses $s = o(m)$ bits of memory and computes $EQ_m(X)$.*

Now we can discuss deterministic online algorithms for (n, r, w)-PNEH Problem.

Theorem 3.8 *Let integer s be such that $s = o(n)$, where n is the length of input. Any deterministic online algorithm A_s computing (n, w, r)-PNEH Problem is (w/r)-competitive.*

Using Lemma 3.7, we can prove the theorem by the same way as in proof of Theorem 3.4.

It is easy to see that $(r(1-\varepsilon)^2/2 + w(\frac{1-\varepsilon^2}{2} + \varepsilon))/r < w/r$ due to $r < w$. Therefore, quantum algorithm is better than any deterministic online algorithm without memory restriction.

Acknowledgements. Partially supported by ERC Advanced Grant MQC. The work is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University. We thank Abuzer Yakaryılmaz from University of Latvia for helpful comments and discussions.

References

[1] F. ABLAYEV, A. GAINUTDINOVA, K. KHADIEV, A. YAKARYILMAZ, Very Narrow Quantum OBDDs and Width Hierarchies for Classical OBDDs. In: *Descriptional Complexity of Formal Systems*. Lecture Notes in Computer Science 8614, Springer, 2014, 53–64.

[2] F. ABLAYEV, A. GAINUTDINOVA, K. KHADIEV, A. YAKARYILMAZ, Very Narrow Quantum OBDDs and Width Hierarchies for Classical OBDDs. *Lobachevskii Journal of Mathematics* 37 (2016) 6, 670–682.

[3] F. ABLAYEV, A. KHASIANOV, A. VASILIEV, *On Complexity of Quantum Branching Programs Computing Equality-like Boolean Functions*. Technical Report TR08-085, ECCC, 2010.

[4] F. ABLAYEV, A. VASILIEV, *On the Computation of Boolean Functions by Quantum Branching Programs via Fingerprinting*. Technical Report TR08-059, ECCC, 2008.

[5] F. ABLAYEV, A. VASILIEV, On Quantum Realisation of Boolean Functions by the Fingerprinting Technique. *Discrete Mathematics and Applications* 19 (2009) 6, 555–572.

[6] S. ALBERS, *BRICS, Mini-Course on Competitive Online Algorithms*. Aarhus University, 1996.

[7] A. AMBAINIS, R. FREIVALDS, 1-way Quantum Finite Automata: Strengths, Weaknesses and Generalizations. In: *FOCS'98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*. 1998, 332–341. (http://arxiv.org/abs/quant-ph/9802062).

[8] A. AMBAINIS, N. NAHIMOVS, Improved Constructions of Quantum Automata. In: *TQC*. Springer, 2008, 47–56.

[9] A. AMBAINIS, N. NAHIMOVS, Improved Constructions of Quantum Automata. *Theoretical Computer Science* 410 (2009) 20, 1916–1922.

[10] A. AMBAINIS, A. YAKARYILMAZ, Superiority of Exact Quantum Automata for Promise Problems. *Information Processing Letters* 112 (2012) 7, 289–291.

[11] A. AMBAINIS, A. YAKARYILMAZ, *Automata and Quantum Computing*. Technical Report 1507.01988, arXiv, 2015.

[12] L. BECCHETTI, E. KOUTSOUPIAS, Competitive Analysis of Aggregate Max in Windowed Streaming. In: *Automata, Languages and Programming: 36th International Colloquium, ICALP 2009, Proceedings, Part I*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, 156–170.

[13] J. BOYAR, S. IRANI, K. S. LARSEN, A Comparison of Performance Measures for Online Algorithms. In: *Workshop on Algorithms and Data Structures*. Springer, 2009, 119–130.

[14] J. BOYAR, S. IRANI, K. S. LARSEN, A Comparison of Performance Measures for Online Algorithms. *Algorithmica* 72 (2015) 4, 969–994.

[15] J. BOYAR, K. S. LARSEN, A. MAITI, The Frequent Items Problem in Online Streaming under Various Performance Measures. *International Journal of Foundations of Computer Science* 26 (2015) 4, 413–439.

[16] R. DORRIGIV, A. LÓPEZ-ORTIZ, A Survey of Performance Measures for On-line Algorithms. *SIGACT News* 36 (2005) 3, 67–81.

[17] D. GAVINSKY, J. KEMPE, I. KERENIDIS, R. RAZ, R. DE WOLF, Exponential Separations for One-way Quantum Communication Complexity, with Applications to Cryptography. In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM, 2007, 516–525.

[18] Y. GIANNAKOPOULOS, E. KOUTSOUPIAS, Competitive Analysis of Maintaining Frequent Items of a Stream. *Theoretical Computer Science* 562 (2015), 23–32.

[19] A. R. KARLIN, M. S. MANASSE, L. RUDOLPH, D. D. SLEATOR, Competitive Snoopy Caching. In: *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 1986, 244–254.

[20] D. KOMM, *An Introduction to Online Computation: Determinism, Randomization, Advice*. Springer, 2016.

[21] F. LE GALL, Exponential Separation of Quantum and Classical Online Space Complexity. In: *Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 2006, 67–73.

[22] D. D. SLEATOR, R. E. TARJAN, Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM* 28 (1985) 2, 202–208.

MEMBERSHIP PROBLEM FOR TWO-DIMENSIONAL JUMPING FINITE AUTOMATA

Grzegorz Madejski[A] Andrzej Szepietowski[A]

[A]Institute of Informatics, Faculty of Mathematics, Physics and Informatics,
University of Gdańsk, 80-308 Gdańsk, Poland
{gmadejsk,Andrzej.Szepietowski}@inf.ug.edu.pl

Abstract
Two-dimensional row jumping finite automata were recently introduced as an interesting computational model for accepting two-dimensional languages. These automata are nondeterministic. They guess the order in which rows of the input array are read and they jump to the next row only after reading all symbols in the previous row. In each row, they choose, also nondeterministcally, the order in which symbols are read. In this paper, we prove that uniform membership problem for these automata is NP-complete, even if we restrict the number of rows or the length of the automaton's rules.

1. Introduction

A two-dimensional language, or a picture language, is a set of rectangular arrays over a finite alphabet. The study of such languages is motivated by their importance in many areas, e.g. pattern recognition, image processing or cellular automata. Many models were introduced to recognize 2D languages, such as array grammars [6] or picture-walking automata [4].

Most of the models in formal language theory process the input in a continuous manner e.g. finite automata read the word from left to right. In the modern era, there are types of data that need to be parsed discontinuously, i.e. the pointer of the parser can jump to any place of the input in search of a specific piece of data. Recently, jumping finite automata were presented in [5] as an attractive model to handle such type of information. They caught a lot of attention and were studied in numerous papers, with the lastest results presented in [1]. In one computation step, jumping finite automata (JFA for short) read a letter from the input word, the letter is removed and the head of the automaton jumps to an arbitrary position in the word. The generalisation of JFA are general jumping finite automata (GJFA) which can in one step read and remove a whole subword of the input, not only single letter.

The jumping mode was used for picture languages in [3], where two-dimensional row jumping finite automata, or 2-RJFA for short, were presented. These automata read the input array row

by row, but the order in which the rows are read is guessed nondeterministcally. Each row is processed using a jumping mode and the automaton erases the letters it reads. More precisely, after reading the subword y, it is replaced by $\boxtimes^{|y|}$. Only after a successful computation on a row, the automaton checks if all letters were erased and moves to the next row. If more than one letter can be removed in a computational step, then we call such automaton general, or 2-GRJFA for short.

It is important to note that 2-GRJFA use a different jumping mode than GJFA considered in [1]. GJFA delete the read symbols from the input word and shorten the word, while 2-GRJFA put a special erase symbol $\boxtimes$ for each read letter. We illustrate this with a short example: using a rule $qab \to q$, we can read a word $aaabbb$ using the deleting mode $aaqabbb \Rightarrow aqabb \Rightarrow qab \Rightarrow q$, but not the erasing mode in which only words of the form $(ab)^*$ can be processed: $abqabab \Rightarrow qab\boxtimes\boxtimes ab \Rightarrow \boxtimes\boxtimes\boxtimes\boxtimes qab \Rightarrow q\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes\boxtimes$. The erasing mode is well suited for 2D arrays, since it does not break the shape of the input. In this paper, we deal only with this mode.

One of the important aspects of studying a model is establishing time complexity for the membership problem. To be of practical use, it is desirable that this problem is solvable in polynomial time. It is important to note that there are two types of membership problems. For the fixed membership problem the automaton and the alphabet is set and only the word is given as input. For example, let A be a 2-GRJFA with alphabet Σ, then:

FIXED M.P. FOR 2-GRJFA A
INPUT: $X \in \Sigma^{**}$
QUESTION: $X \in L(A)$?

A more general case, and computationally harder one, is the uniform membership problem. Here, the automaton and the alphabet is given as input together with the array:

UNIFORM M.P. FOR 2-GRJFA
INPUT: a 2-GRJFA A and an array $X \in \Sigma^{**}$
QUESTION: $X \in L(A)$?

The complexity of these problems was studied for JFA and $GJFA$ in [1]:

	JFA	GJFA
Fixed M.P.	P	NPC[2]
Uniform M.P.	NPC / P[1]	NPC[2]
[1] under the restriction that $\|\Sigma\| = k$, for some constant k [2] even under the restriction that $\|\Sigma\| = 2$		

In this paper, we deal with the uniform membership problem for 2-RJFA and 2-GRJFA. We show that in both cases the problem is NP-complete, even under some restrictions.

	2-RJFA	2-GRJFA
Uniform M.P.	NPC[1]	NPC[2]
[1] even under the restriction that $\|\Sigma\| = 2$ [2] even under the restriction that $\|\Sigma\| = 2$ and the input arrays have one row		

2. Preliminaries

We assume the reader is familiar with the basics of formal language and automata theory. An alphabet Σ is a finite set of letters and Σ^* is a set of all words over the alphabet. An empty word is denoted by λ. A language over alphabet Σ is a set $L \subseteq \Sigma^*$.

In the two-dimensional case, the definitions are similar. Instead of words, we consider rectangular arrays, or pictures, consisting of symbols in Σ. A set of all such arrays is denoted by Σ^{**}. If X has m rows and n columns, then the array can be written in the form $X = \begin{array}{cccc} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{array}$ where $a_{i,j} \in \Sigma$ for all $1 \leq i \leq m, 1 \leq j \leq n$ or in short form $X = [a_{i,j}]_{m \times n}$. An array with no rows and columns is an empty array Λ. A (picture) language over alphabet Σ is a set $L \subseteq \Sigma^{**}$.

For the purpose of reading arrays by the 2D row jumping automata, we need to mark the beginning and end of each row with special guard symbols. For an array $X \in \Sigma^{**}$ with m rows and n columns, we define an array $\widehat{X}$ of size $(m, n+2)$, whose every row is guarded by the symbol $\$ \notin \Sigma$ from the left and by the symbol $\# \notin \Sigma$ from the right.

We are ready to give the definition of 2D row jumping automata, as in [3].

Definition 2.1 *A two-dimensional general row jumping finite automaton (2-GRJFA) is an octuple $A = (Q, Q', \Sigma, R_1, R_2, R_3, s, F)$, where*

- *Q is a finite set of states;*
- *$Q' \subset Q$ is a set of check and row jump states;*
- *Σ is an input alphabet;*
- *$R_1 \subseteq Q \times \Sigma^* \times Q$ is a set of jumping rules;*
- *$R_2 \subseteq (Q \times \$ \times Q') \cup \{r \times \boxtimes \times r : r \in Q'\}$, where $\$, \boxtimes \notin \Sigma$, is a set of (standard, non-jumping) row checking rules;*
- *$R_3 \subseteq Q' \times \# \times Q$, where $\# \notin \Sigma$, is a set of row jump rules;*
- *$s \in Q$ is the initial state;*
- *$F \subseteq Q$ is the set of accepting states.*

If for every rule $(p, y, q) \in R_1$, we have $|y| \leq 1$, then we call the automaton a two-dimensional row jumping finite automaton (2-RJFA).

Instead of writing (p, y, q), we shall denote rules in R_1 by $py \curvearrowright q$, rules in R_2 by $py \to q$, and rules in R_3 by $py \circlearrowright q$.

The automaton A works with $\widehat{X}$ on its input tape. In general, a 2-GRJFA guesses the order in which rows of the input array are read. It jumps to the next row using a rule in R_3 only after reading all symbols in the previous row. In each row, it reads symbols using rules in R_1. The

order in which symbols are read is chosen nondeterministically. Next, it checks if the whole row was read using rules in R_2. We see that 2-GRJFA rely heavily on nondeterminism, as both jumping and row jump are nondeterministic actions.

To illustrate in detail how the automaton processes the input, we first consider only one row from an array. First the automaton chooses nondeterministcally the place where it starts. For example, let us consider the row $\boxed{\$\ 0\ 0\ p0\ 0\ \#}$, where the automaton starts observing the third 0 in a state p. Next, it chooses a rule $py \curvearrowright q \in R_1$ to use, removes a word y from the row, puts $\boxtimes^{|y|}$ in its place, jumps to a different position in the row, and changes its state to q. For example, if $p00 \curvearrowright q \in R_1$ and the automaton chooses to jump to the first letter, we have the following step:

$$\boxed{\$\ 0\ 0\ p0\ 0\ \#} \curvearrowright \boxed{\$\ q0\ 0\ \boxtimes\ \boxtimes\ \#}.$$

After reading all letters in the row, the automaton goes to the second phase of the computation, where it checks if all letters were read and replaced by $\boxtimes$. This is done in a non-jumping way, from left to right, using rules in R_2. To be more precise, after reading the last input letters using a rule in R_1 and erasing them, the automaton should jump to \$, otherwise the computation cannot be continued. It reads this symbol and moves right in state $r \in Q'$. In this state, it moves right and checks if all symbols are $\boxtimes$. Example:

$$\boxed{p\$\ \boxtimes\ \boxtimes\ \boxtimes\ \boxtimes\ \#} \to \boxed{\$\ r\boxtimes\ \boxtimes\ \boxtimes\ \boxtimes\ \#} \to \boxed{\$\ \boxtimes\ r\boxtimes\ \boxtimes\ \boxtimes\ \#} \to ... \to \boxed{\$\ \boxtimes\ \boxtimes\ \boxtimes\ \boxtimes\ r\#}.$$

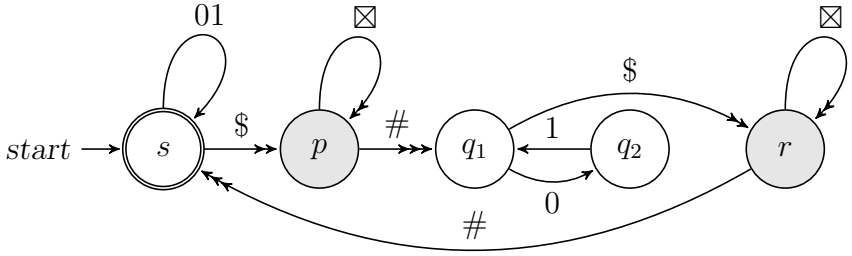
If the automaton successfully reaches #, it proceeds to the final step. It uses a rule in R_3 to jump to another row. The row which was just read is removed from the array. The automaton continues to do so until all rows are removed.

We say that A accepts X, if there is a computation of A that starts in the state s, reads all rows of an array $\widehat{X}$, and finishes with the empty array Λ in an accepting state $q_f \in F$. The language accepted by the automaton A is a set

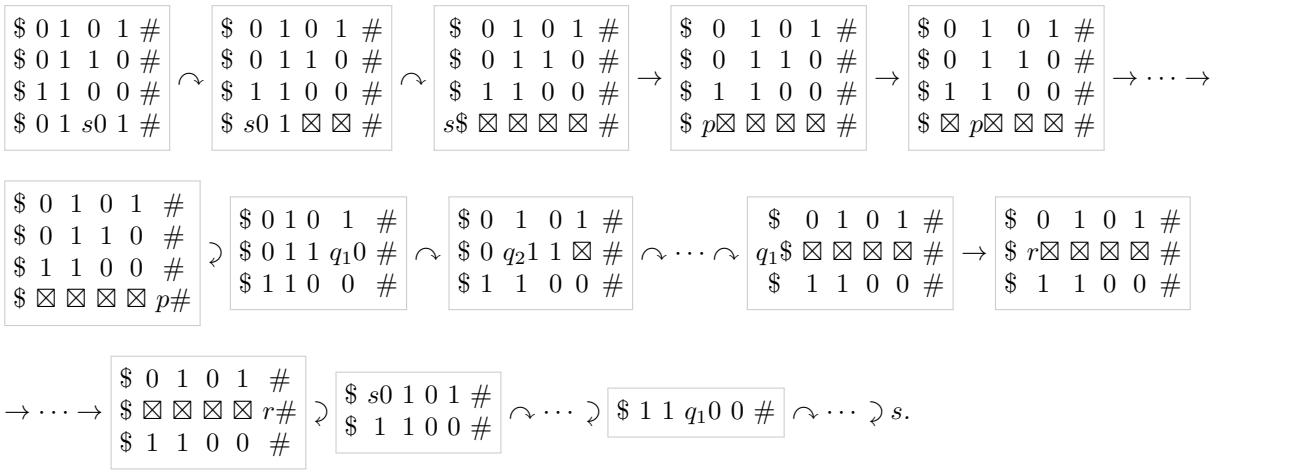
$$L(A) = \{X : A \text{ accepts } X\}.$$

A class of all languages accepted by 2-GRJFA will be denoted as $\mathcal{L}$(2-GRJFA), and for the 2-RJFA as $\mathcal{L}$(2-RJFA).

Example 2.2 *Let us consider the automaton* $A = (\{s, p, q_1, q_2, r\}, \{p, r\}, \{0, 1\}, \{s01 \curvearrowright s, q_10 \curvearrowright q_2, q_21 \curvearrowright q_1, \}, \{s\$ \to p, p\boxtimes \to p, q_1\$ \to r, r\boxtimes \to r\}, \{p\# \circlearrowright q_1, r\# \circlearrowright s\}, s, \{s\})$*, see a graphic form below. The row checking states* $\{p, r\}$ *are coloured in gray. The jumping rules are represented by single-head arrows, row checking rules by double-head arrows and row jump rules by triple-head arrows.*



An example computation on an array:



It is easy to see, that $L(A) = \{X \in \{0,1\}^{**} : X$ *has* n *rows of the form* $(01)^*$ *and* n *rows containing the same number of* 0*'s and* 1*'s, where* $n \geq 0\}$. *Obviously,* $L(A) \in \mathcal{L}$(2-GRJFA). *However,* $L(A) \notin \mathcal{L}$(2-RJFA), *as it was shown in [3] using a similar example.*

3. Results

First, we consider the following problem:

Uniform m.p. for 2-rjfa
Input: a 2-RJFA A and an array $X \in \Sigma^{**}$
Question: $X \in L(A)$?

To prove NP-hardness, we show a reduction from the 3-partition problem, which is know to be strongly NP-complete [2]. Therefore, the input numbers can be given in unary.

3-Partition
Input: a sequence of natural numbers $S = (n_1, n_2, ..., n_{3m})$ given in unary, where $m \geq 0$,
Question: can S be partitioned into triplets $(n_{i_1}, n_{j_1}, n_{k_1})$,$(n_{i_2}, n_{j_2}, n_{k_2})$, ..., $(n_{i_m}, n_{j_m}, n_{k_m})$, such that $\bigcup_{1 \leq r \leq m} \{i_r, j_r, k_r\} = \{1, 2, ..., 3m\}$ and for each r, $n_{i_r} + n_{j_r} + n_{k_r} = B$ (each triplet sums to the same number)?

Theorem 3.1 Uniform m.p. for 2-rjfa *is NP-complete, even under the restriction that* $|\Sigma| = 2$.

Proof. It is easy to see that Uniform m.p. for 2-rjfa $\in$ NP. To prove NP-hardness, we show that there exists a deterministic polynomially time bounded Turing machine M which reduces 3-Partition to Uniform m.p. for 2-rjfa. For a sequence $S = (n_1, n_2, ..., n_{3m})$, the machine M computes a pair (A, X), where A is a 2-RJFA and $X \in \Sigma^{**}$, such that A accepts X if and only if S has a 3-partition.

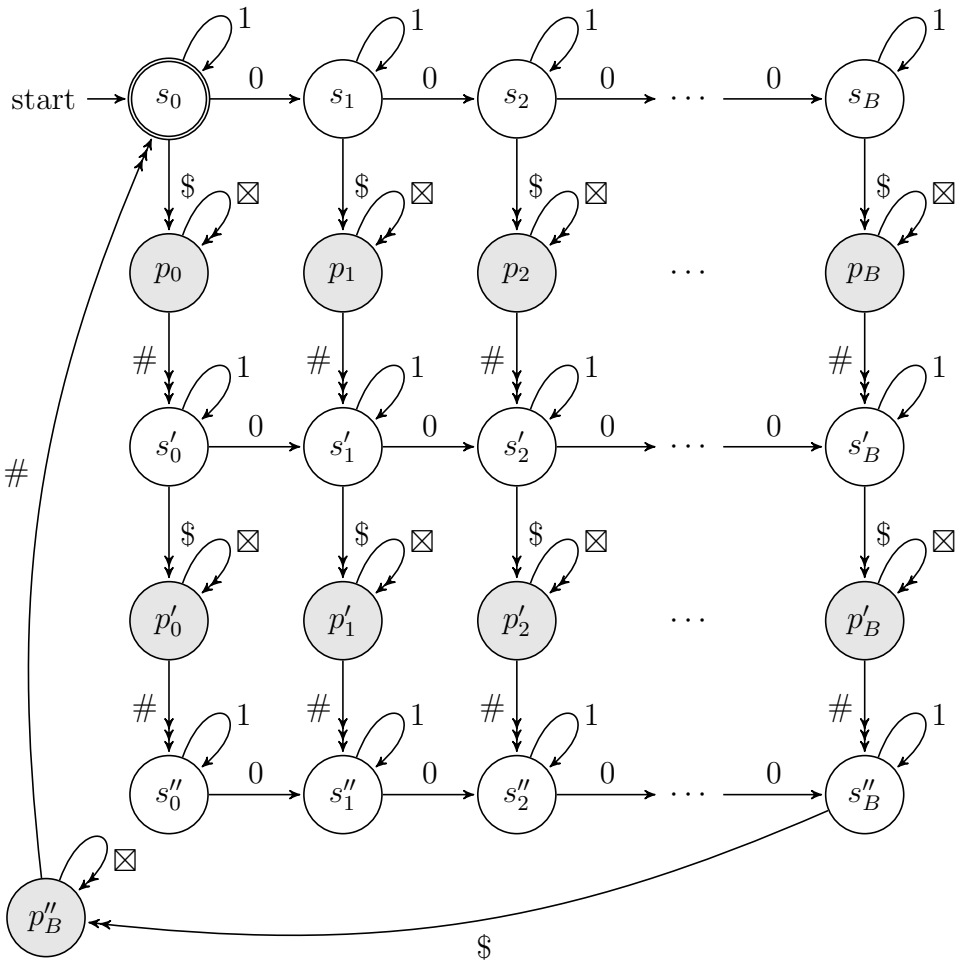


Figure 1: 2-RJFA constructed based on 3-partition instance.

Machine M works as follows. First, it computes $N = \sum_{i=1}^{3m} n_i$ and $B = \frac{N}{m}$. Then it constructs the automaton A presented in Fig. 1 and the array $X = \begin{array}{c} 0^{n_1}1^{B-n_1} \\ 0^{n_2}1^{B-n_2} \\ \cdots \\ 0^{n_{3m}}1^{B-n_{3m}} \end{array}$. We may assume that each $n_i \leq B$.

Suppose that there is a 3-partition $(n_{i_1}, n_{j_1}, n_{k_1})$,$(n_{i_2}, n_{j_2}, n_{k_2})$, ..., $(n_{i_m}, n_{j_m}, n_{k_m})$ of S, then there is an accepting computation of A on X. The computation starts in i_1-th row of X. After reading the whole row it reaches the state $s_{n_{i_1}}$, jumps in the state $p_{n_{i_1}}$ into \$, checks if all letters are read and jumps in the state $s'_{n_{i_1}}$ to the row j_1. After reading and checking the j_1-th row it jumps in the state $s''_{n_{i_1}+n_{j_1}}$ to the k_1-th row. The automaton can read and check the k_1-th row only if $n_{i_1} + n_{j_1} + n_{k_1} = B$. After that, the automaton takes the next triplet and jumps in the state s_0 to i_2-th row, and so on.

On the other hand, if there is an accepting computation of the automaton A on the array X,

then the order in which the rows of X are visited gives the 3-partition of S. □

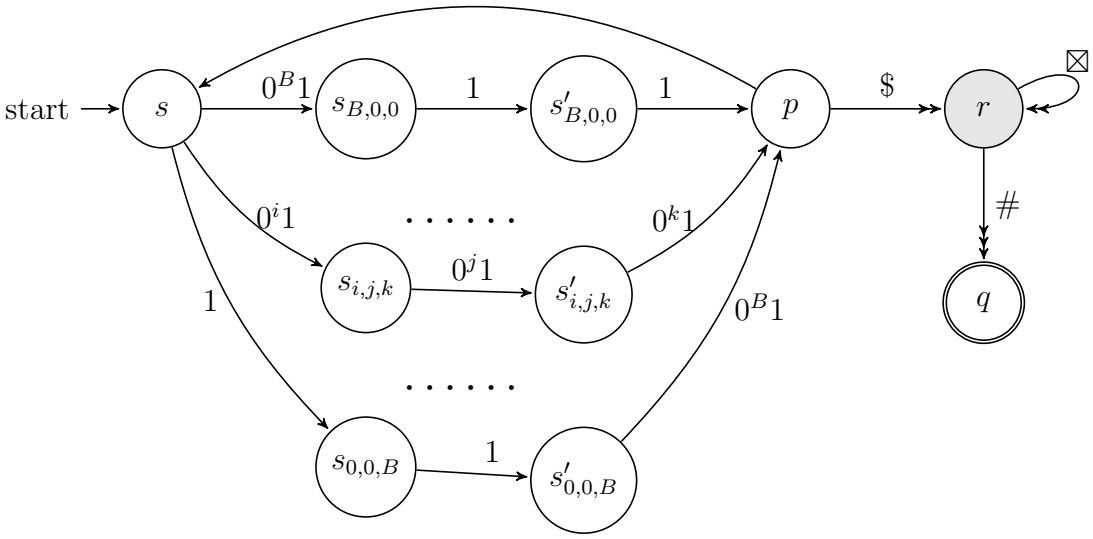
Corollary 3.2 UNIFORM M.P. FOR 2-GRJFA *is NP-complete, even under the restriction that* $|\Sigma| = 2$.

Proof. NP-hardness follows from Theorem 3.1. It is obvious that UNIFORM M.P. FOR 2-GRJFA $\in$ NP. □

We can give a stronger result as the one above by restricting the number of rows in an input array to one.

Theorem 3.3 *The uniform membership problem for single-row 2-GRJFA is NP-complete, even under the restriction that* $|\Sigma| = 2$.

Proof. We reduce the 3-partition problem with $S = \{n_1, n_2, ..., n_{3m}\}$ to the uniform membership problem for single-row 2-GRJFA. First we compute B, then we construct an automaton with the set of states $Q = \{p, q, r, s\} \cup \{s_{i,j,k}, s'_{i,j,k} : i + j + k = B\}$ and $Q' = \{r\}$. The rules are presented below.



The 3-partition problem has a solution if and only if $0^{n_1}10^{n_2}1 \cdots 0^{n_{3m}}1 \in L(A)$. We see that the automaton reads three numbers from the input that sum up to B (a cycle from state s to p and p to s). This corresponds to a triplet from the 3-partition of S. It repeats the cycle until all triplets are read. The problem is in NP which is easy to show. □

4. Conclusions

We analysed the time complexity of the uniform membership problem for 2-GRJA and 2-RJFA. In both cases, we obtained NP-completeness results, even if we restricted the alphabet to having only two symbols. For 2-GRJFA, this problem is NP-complete even in the single-row case. For

2-RJFA it is crucial that we have an arbitrary number of rows, otherwise the problem would collapse to the uniform membership problem for JFA, which is in P if the size of the alphabet is set [1].

References

[1] H. FERNAU, M. PARAMASIVAN, M. L. SCHMID, V. VOREL, Characterization and complexity results on jumping finite automata. *Theor. Comput. Sci.* 679 (2017), 31–52.
https://doi.org/10.1016/j.tcs.2016.07.006

[2] M. R. GAREY, D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[3] S. J. IMMANUEL, D. G. THOMAS, Two-Dimensional Jumping Finite Automata. *Math. Appl.* 5 (2016), 105–122.
https://doi.org/10.13164/ma.2016.08

[4] J. KARI, V. SALO, Algebraic Foundations in Computer Science. chapter A Survey on Picture-walking Automata, Springer-Verlag, Berlin, Heidelberg, 2011, 183–213.
http://dl.acm.org/citation.cfm?id=2172429.2172438

[5] A. MEDUNA, P. ZEMEK, Jumping Finite Automata. *Int. J. Found. Comput. Sci.* 23 (2012) 7, 1555–1578.
https://doi.org/10.1142/S0129054112500244

[6] P. S. P. WANG (ed.), *Array Grammars, Patterns and Recognizers*. World Scientific Series in Computer Science 18, World Scientific, 1989.
https://doi.org/10.1142/0996

Author Index