REAL-TIME VECTOR AUTOMATA

by

Özlem Salehi

B.S., Mathematics, Boğaziçi University, 2011

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2013

REAL-TIME VECTOR AUTOMATA

APPROVED BY:

Prof. Ahmet Celal Cem Say ..... (Thesis Supervisor)

Assist. Prof. Albert Ali Salah .....

Abuzer Yakaryılmaz, Ph.D. .....

DATE OF APPROVAL: 31.05.2013

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis advisor Prof A. C. Cem Say for his continuous support throughout my thesis. Without his guidance, patience and motivation this thesis would not have been possible. I have learned a lot from his vast knowledge and experience.

I would like to especially thank Abuzer Yakaryılmaz for introducing me the subject of this thesis. He was a source of inspiration by kindly answering all my questions and always coming up with new ideas. Without his contributions, this thesis could not have been completed.

I would like to thank my thesis committee member Assist. Prof. Albert Ali Salah for his review of this thesis.

I wish to thank Assoc. Prof. Tunga Güngör for encouraging me towards pursuing graduate studies in computer engineering and for believing in me.

I want to thank my friends Aybüke Özgün, Deniz Nemli, Ezgi Işınay, Serhat Yüksel and Emre Doğan for all the good times we have shared in the Department of Mathematics during four years and for their trust in me. I would like to thank Zeynep Gözen Sarıbatur for our lasting discussions.

I am grateful to my professors in the Department of Mathematics and Computer Engineering who have influenced me. I would also like to thank my teaching assistant friends in the Mathematics Department with whom I had the pleasure of working with. I would like to thank Merve Şahinsoy for the times we have spent together.

Finally, this thesis is dedicated to my beloved family, whose endless love and support throughout my life made everything possible and to Oktay, for his love.

ABSTRACT

REAL-TIME VECTOR AUTOMATA

Finite automaton has been one of the most studied models in automata theory. The limited power of the standard model has led researchers to make various extensions to the standard model. Counter automaton, automaton with multiplication, finite automaton over groups are some of the examples of such extensions. In this thesis, we study the computational power of real-time finite automaton that has been augmented with a vector of dimension k, and programmed to multiply this vector at each step by an appropriately selected $k \times k$ matrix. Only one entry of the vector can be tested for equality to 1 at any time. We study the classes of languages recognized by deterministic, nondeterministic, and "blind" versions of these machines and compare them with each other. It turns out that these machines are closely related to some of the classical models like counter automata and generalized finite automata.

ÖZET

GERKÇEK ZAMANLI VEKTÖR MAKİNELER

Sonlu durum makinesi otomata teorisinin en çok incelenen modellerinden biri olmuştur. Standart modelin gücünün sınırlı olması araştırmacıları bu standart modelin üzerine farklı eklemeler yapmaya itmiştir. Sayaçlı makineler, çarpımlı sonlu durum makineleri, grup üzerinde tanımlı sonlu durum makineleri bu eklemelere örnektir. Bu tezde k boyutlu bir vektörle güçlendirilmiş ve her adımda bu vektörü uygun $k \times k$ matrislerle çarpmaya programlanmış gerçek zamanlı sonlu durum makineleri incelenmiştir. Bir adımda vektörün sadece tek bir girdisi 1'e eşit mi diye kontrol edilebilir. Makinelerin belirlenimci, belirlenimci olmayan, "kör" versiyonları tarafından tanınan diller incelenmiş ve birbirleriyle karşılaştırılmışlardır. Bu makineler ile, sayaçlı makinelerin ve genellenmiş sonlu durum makinelerin birbirleriyle yakından ilişkili olduğu ortaya çıkmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

$E_k^i(c)$	Matrix obtained by setting the ith entry of the first column of the $k \times k$ identity matrix to c.
$\mathfrak{L}(A)$	Class of languages recognized by automata of type A
q_0	Initial state
Q	Set of states
Q_a	Set of accept states
$\mathbb{Q}$	Set of rational numbers
$\mathbb{R}$	Set of real numbers
$\mathbb{Z}$	Set of integers
δ	Transition function
Σ	Alphabet
Σ^*	The set of all strings written by the symbols in Σ

LIST OF ACRONYMS/ABBREVIATIONS

GFA	Generalized finite automaton
rtDBVA(k)	Real-time deterministic blind vector automaton
rtDkBCA	Real-time deterministic blind multicounter automaton
rtDkCA	Real-time deterministic multicounter automaton
rtDkRCA	Real-time deterministic reset multicounter automaton
rtDVA(k)	Real-time deterministic vector automaton
rtNBVA(k)	Real-time nondeterministic blind vector automaton
rtNkBCA	Real-time nondeterministic blind multicounter automaton
rtNkCA	Real-time nondeterministic multicounter automaton
rtNVA(k)	Real-time nondeterministic vector automaton
rtPFA	Real-time probabilistic finite automaton
TuFA	Turakainen finite automaton
1DFAM	One-way deterministic finite automaton with mutliplication
1DFAMW	One-way deterministic finite automaton with mutliplication without equality
1NFAM	One-way nondeterministic finite automaton with mutliplication
1NFAMW	One-way nondeterministic finite automaton with mutliplication without equality
2PFA	Two-way probabilistic finite automaton

1. INTRODUCTION

It is well known that viewing computation as multiplications of matrices induced by the transition functions of the machines under consideration is a useful approach in the study of, for instance, probabilistic and quantum models. The standard way to trace a program in this manner [1] is to use a vector which has a separate entry for each possible configuration of the simulated machine, necessitating matrices whose dimensions depend on the configuration space of the automaton. In this paper, we consider the alternative of using fixed-size matrices, and representing the infinitely many different configurations employing the values of the entries of a fixed-size vector. We introduce the vector automaton, which is linked to many generalizations of the standard deterministic finite automaton model like counter automata, automata with multiplication, and generalized stochastic automata [2–5].

A vector automaton is a finite automaton endowed with a k-dimensional vector, and the capability of multiplying this vector with an appropriately selected matrix at every computational step. Only one of the entries of the vector can be tested for equality to 1 at any step. Since equipping these machines with a "one-way" input head, which is allowed to pause on some symbols during its left-to-right traversal of the input, would easily make them Turing-equivalent, we focus on the case of real-time input, looking at the deterministic and nondeterministic versions of the model. We make a distinction between general vector automata and "blind" ones, where the equality test can be performed only at the end of the computation.

Chapter 2 begins with the preliminaries in which the notation and some basic information are presented that will be useful throughout the thesis. We review some classical models like multicounter automata, generalized finite automata with their formal definitions.

We introduce our model in Chapter 3. We state a generalization which allows us to test any entry of the vector for equality to any desired number c. We give some examples of language recognition by real-time vector automaton to familiarize the reader with the model.

We start further examining real-time deterministic vector automata in Section 3.1. We make comparisons between the real-time deterministic automata and the closely related multicounter automata. The case where the dimension of the vector k is equal to 1 corresponds to the real-time version of one-way deterministic finite automata with multiplication introduced by Ibarra *et al.* in [4]. We show that real-time deterministic vector automata are more powerful when $k > 1$.

In Section 3.2, we look at the deterministic blind version of the model which turns out to be equivalent to Turakainen's generalized finite automata [5] in one language recognition mode. We state a hierarchy result based on the number of states and the vector dimension k. Restricting k to 1, we obtain the real-time version of Ibarra *et al.*'s one-way deterministic finite automata with multiplication without equality and the two models turn out to be equivalent.

We look at the real-time nondeterministic vector automata in Section 3.3 and we show that even the blind versions can recognize some NP-complete languages. We conclude that it is highly likely that they are more powerful than their deterministic versions since the equivalence of the two models would yield the result P=NP.

Chapter 4 is the conclusion of the thesis. We state some open questions which will guide us for the future studies.

2. PREVIOUS WORK

2.1. Preleminaries

Throughout the paper, the following notation will be used: Q is the set of states, where $q_0 \in Q$ denotes the initial state, $Q_a \subset Q$ denotes the set of accept states, and Σ is the input alphabet. An input string w is placed between two end-marker symbols (except for GFA's) on an infinite tape in the form $¢w\$$. For a given string w, $|w|$ denotes the length of the string and w^r is the reverse of the string.

For a given machine, δ is the transition function which specifies the next move based on the current state and the input symbol read. The next move may also depend on the specific properties of the machine such as the status of its counters.

Status of the counters is described by the set $\{0, \pm\}$ where 0 denotes that the value of the counter is equal to 0 and $\pm$ denotes that the value of the counter is non-zero. For a machine with k counters, value of the counters is a k-tuple from the set $\{0, \pm\}^k$.

For the machines with registers, the symbol $=$ indicates that the value of the register is equal to 1 and the symbol $\neq$ indicates that the register value is not equal to 1.

A machine can be real-time, 1-way or 2-way depending on the allowed tape head movements. The set $\{\downarrow, \rightarrow, \leftarrow\}$ represents the possible head directions. The tape head can stay in the same position ($\downarrow$), move one square to the right ($\rightarrow$), or move one square to the left ($\leftarrow$) in one step. A machine which is allowed to move in both directions and stay put is said to be a 2-way machine. If the tape head is allowed only to move right and stay put, then the machine is 1-way. A machine is real-time if the tape head can move only right at each step. Note that there is no need to specify tape head

movement in the transition function for the real-time machines.

$\{-1, 0, 1\}$ is the set of counter operations, where -1 indicates that the value of the counter is decremented by 1, 0 indicates that the value of the counter is not changed, and 1 indicates that the value of the counter is increased by 1.

For a machine model A, $\mathfrak{L}(A)$ denotes the class of languages recognized by automata of type A.

For a matrix M, $M(i, j)$ denotes the entry in the i'th row and j'th column of M. Let $E_k^i(c)$ denote the matrix obtained by setting the i'th entry of the first column of the $k \times k$ identity matrix to c. For a row vector v, the product $vE_k^i(c)$ is the vector obtained by adding c times the i'th entry of v to the first entry when $i > 1$, and the vector obtained by multiplying the first entry of v by c when $i = 1$.

2.2. Multicounter Automata.

A *real-time deterministic multicounter automaton* (rtDkCA) [2] is a 5-tuple

$$\mathcal{M} = (Q, \Sigma, \delta, q_0, Q_a).$$

The transition function δ of $\mathcal{M}$ is specified so that $\delta(q, \sigma, \theta) = (q', c)$ means that $\mathcal{M}$ moves the head to the next symbol, switches to state q', and updates its counters according to the list of increments represented by $c \in \{-1, 0, 1\}^k$, if it reads symbol $\sigma \in \Sigma$, when in state $q \in Q$, and with the counter values having signs as described by $\theta \in \{0, \pm\}^k$. At the beginning of the computation, the tape head is placed on the symbol ¢, and the counters are set to 0. At the end of the computation, that is, after the right end-marker \$ has been scanned, the input is accepted if $\mathcal{M}$ is in an accept state.

A *real-time deterministic blind multicounter automaton* (rtDkBCA) [3] $\mathcal{M}$ is a rtDkCA which can check the value of its counters only at the end of the computation. Formally, the transition function is now replaced by $\delta(q, \sigma) = (q', c)$. The input is accepted at the end of the computation if $\mathcal{M}$ enters an accept state, and all counter values are equal to 0.

A *real-time nondeterministic multicounter automaton* (rtNkCA) is a rtDkCA which is allowed to make nondeterministic moves. A *real-time nondeterministic blind multicounter automaton* (rtNkBCA) is a rtNkCA which can check the value of its counters only at the end of the computation.

2.3. Finite Automata With Multiplication.

A *one-way deterministic finite automaton with multiplication* (1DFAM) [4] is a 6-tuple

$$\mathcal{M} = (Q, \Sigma, \delta, q_0, Q_a, \Gamma),$$

where Γ is a finite set of rational numbers (multipliers). The transition function δ is defined as $\delta : Q \times \Sigma \times \Omega \to Q \times \{\downarrow, \rightarrow\} \times \Gamma$, where $\Omega = \{=, \neq\}$. $\mathcal{M}$ has a register which can store any rational number, and is initially set to 1. Reading input symbol $\sigma \in \Sigma$ in state $q \in Q$, $\mathcal{M}$ compares the current value of the register with 1, thereby calculating the corresponding value $\omega \in \Omega$, and switches its state to $q' \in Q$, moves its head in direction $d \in \{\downarrow, \rightarrow\}$, and multiplies the register by $\gamma \in \Gamma$, in accordance with the transition function value $\delta(q, \sigma, \omega) = (q', d, \gamma)$. The input string is accepted if $\mathcal{M}$ enters an accept state with the register value equaling 1 after it scans the right end-marker symbol.

A 1DFAM *without equality* (1DFAMW) is a 1DFAM which can not check whether or not the register has value 1 during computation. One can think of 1DFAMW as a blind 1DFAM since it can check the value of its register only at the end of the

computation. The transition function δ is replaced by $\delta(q, \sigma) = (q', d, \gamma)$. The accept condition of the 1DFAMW is the same with the 1DFAM.

A *one-way nondeterministic finite automaton with multiplication* (1NFAM) is a 1DFAM which is allowed to make nondeterministic moves. A 1NFAM *without equality* (1NFAMW) is a 1NFAM which can not check whether or not the register has value 1 during the computation.

2.4. Finite Automata Over Groups

Let $K = (M, \circ, e)$ be a group under the operation denoted by $\circ$ with the neutral element denoted by e.

A *(deterministic) finite automaton over the group* K [6] is a 6-tuple,

$$\mathcal{A} = \{Q, \Sigma, \delta, K, q_0, Q_a\}.$$

The transition function δ is defined as $Q \times \Sigma \to Q \times M$. We can think of $\mathcal{A}$ as having a register in which any element of M can be stored. The transition function is specified as $\delta(q, \sigma) = (q', m)$ so that reading input symbol $\sigma \in \Sigma$, when in state $q \in Q$, $\mathcal{A}$ moves the head to the next symbol, switches to state $q' \in Q$ and writes in the register $x \circ m$ where $m \in M$ and x is the old content of the register. At the beginning of the computation, the register is initialized to e, the neutral element of the group K. The input string is accepted at the end of the computation if $\mathcal{A}$ enters an accept state, and the register value is equal to e.

2.5. Probabilistic Finite Automata

A *real-time probabilistic finite automaton* (rtPFA) [7] is a 5-tuple

$$\mathcal{P} = (Q, \Sigma, A_{\sigma \in \tilde{\Sigma}}, q_0, Q_a)$$

where $\tilde{\Sigma} = \Sigma \cup \{¢, \$\}$ and $A_{\sigma \in \tilde{\Sigma}}$'s are real-valued stochastic transition matrices for symbol σ, that is, $A_\sigma(i, j)$ is the transition probability from state q_i to state q_j when reading symbol σ.

The computation of $\mathcal{P}$ can be traced using a row vector v, where the i'th entry corresponds to the i'th state. q_0 is the initial state and v_0 is the initial vector whose first entry is equal to 1. A given input string $w \in \Sigma^*$ is placed as $\tilde{w} = ¢w\$$ on the tape. After reading the i'th symbol, the vector is equal to $v_i = v_0 A_{\tilde{w}_1} A_{\tilde{w}_2} \dots A_{\tilde{w}_i}$ where $\tilde{w}_i$ is the i'th symbol of $\tilde{w}$. The acceptance probability for w is defined as $f_{\mathcal{P}}(w) = \sum_{q_i \in Q_a} v_{|\tilde{w}|}(i)$ where $v_{|\tilde{w}|}(i)$ denotes the i'th entry of the vector.

A probabilistic finite automaton whose tape head can move in both directions and can also stay in place is called a *two-way probabilistic finite automaton* (2PFA) [8].

2.6. Generalized Finite Automata.

A *generalized finite automaton* (GFA) [5] is a 5-tuple

$$\mathcal{G} = (Q, \Sigma, \{A_{\sigma \in \Sigma}\}, v_0, f),$$

where the $A_{\sigma \in \Sigma}$'s are $|Q| \times |Q|$ are real valued transition matrices, and v_0 and f are the real valued initial row vector and final column vector, respectively. The acceptance value for an input string $w \in \Sigma^*$ is defined as $f_{\mathcal{G}}(w) = v_o A_{w_1} \dots A_{w_{|w|}} f$.

A GFA whose components are restricted to be rational numbers is called a *Tu-*

rakainen finite automaton (TuFA) in [9].

2.7. Language Recognition

The language $L \subseteq \Sigma^*$ is said to be recognized by machine $\mathcal{M}$ with *bounded error* [10] if there exists an ϵ such that $0 \leq \epsilon < \frac{1}{2}$ if

- $f_{\mathcal{M}}(w) \geq 1 - \epsilon$ when $w \in$ L
- $f_{\mathcal{M}}(w) \leq \epsilon$ when $w \notin$ L.

The class of languages recognized by 2PFA's with bounded error is denoted as $\mathfrak{L}$(2PFA) .

The language $L \subseteq \Sigma^*$ recognized by machine $\mathcal{M}$ with *cutpoint* $\lambda \in \mathbb{R}$ is defined as [10]

$$L = (\mathcal{M}, > \lambda) = \{w \in \Sigma^* | f_{\mathcal{M}}(w) > \lambda\}.$$

When $\mathcal{M}$ is a probabilistic finite automaton, λ is restricted so that $0 \leq \lambda < 1$ since the components of a probabilistic finite automaton are stochastic. For a GFA with cutpoint $\lambda_1 \in \mathbb{R}$, there exists a rtPFA with cutpoint $\lambda_2 \in [0, 1)$ such that the two recognizes the same language [5]. Similarly, for a 2PFA with cutpoint $\lambda_2 \in [0, 1)$, there exists a rtPFA with cutpoint $\lambda_2 \in [0, 1)$ such that the two recognizes the same language [11]. rtPFA's, 2PFA's and GFA's recognize the same class of languages with cutpoint, the class of *stochastic languages* denoted by $S^{>}$.

Furthermore, $S^{=}$ is the class of languages of the form

$$L = (\mathcal{M}, = \lambda) = \{w \in \Sigma^* \mid f_{\mathcal{M}}(w) = \lambda\}$$

where $\mathcal{M}$ is a rtPFA (GFA or 2PFA) and $\lambda \in \mathbb{R}$. [12]

$\mathrm{S}_{\mathbb{Q}}^{=}$ is the class of the languages of the form

$$L = (\mathcal{G}, = \lambda) = \{w \in \Sigma^* \mid f_{\mathcal{G}}(w) = \lambda\}$$

where $\mathcal{G}$ is a Turakainen finite automaton and $\lambda \in \mathbb{Q}$. [5]

3. VECTOR AUTOMATA

A *real-time deterministic vector automaton of dimension k* (rtDVA(k)) is a 6-tuple

$$\mathcal{V} = (Q, \Sigma, \delta, q_0, Q_a, v),$$

where v is a k-dimensional initial row vector, and the transition function δ is defined as

$$\delta : Q \times \Sigma \times \Omega \to Q \times S,$$

where S is the set of $k \times k$ rational-valued matrices, and $\Omega = \{=, \neq\}$, as in the definition of 1DFAM's.

Specifically, $\delta(q, \sigma, \omega) = (q', M)$ means that when $\mathcal{V}$ is in state q reading symbol $\sigma \in \Sigma$, and the first entry of its vector corresponds to $\omega \in \Omega$ (with ω having the value $=$ if and only if this entry is equal to 1), $\mathcal{V}$ moves to state q', multiplying its vector with the matrix $M \in S$. As in the definition of 1DFAM's, ω is taken to be $=$ if the first entry of the vector equals 1, and $\neq$ otherwise. The string is accepted if $\mathcal{V}$ enters an accept state, and the first entry of the vector is 1, after processing the right end-marker symbol \$.

Remark 3.1. *The designer of the automaton is free to choose the initial setting v of the vector, where $v \in \mathbb{Q}^k$.*

In the definition, it is stated that the machine can only check the first entry of the vector for equality to 1. Sometimes we find it convenient to design programs that check for equality to some number other than 1. One may also wish that it were possible to check not the first, but some other entry of the vector. In the following theorem, we

show that we can assume our rtDVA(k)'s have that flexibility. For the purposes of that theorem, let a rtDVA$(k)^i_c$ be a machine similar to a rtDVA(k), but with a generalized definition that enables it to check the i'th entry, for equality to the number c.

Theorem 3.2. (i) *Given a* rtDVA$(k)^i_1$ *recognizing a language* L*, one can construct a* rtDVA(k) *that recognizes* L*.* (ii) *For any* $c \in \mathbb{Q}$*, given a* rtDVA$(k)^1_c$ *recognizing a language* L*, one can construct a* rtDVA(k+1) *that recognizes* L*.*

Proof. (i) Suppose that we are given a rtDVA$(k)^i_1$ $\mathcal{V} = (Q, \Sigma, \delta, q_0, Q_a, v)$. We will construct an equivalent rtDVA(k) $\mathcal{V}' = (Q, \Sigma, \delta', q_0, Q_a, v')$. Let J denote the matrix obtained from the $k \times k$ identity matrix by interchanging the first and i'th rows. We will use multiplications with J repeatedly to swap the first and i'th entries of the vector when it is time for that value to be checked, and then to restore the vector back to its original order, so that the rest of the computation is not affected. The initial vector of $\mathcal{V}'$ has to be a reordered version of v to let the machine check the correct entry at the first step, so $v' = vJ$. We update the individual transitions so that if $\mathcal{V}$ has the move $\delta(q, \sigma, \omega) = (q', M)$, then $\mathcal{V}'$ has the move $\delta'(q, \sigma, \omega) = (q', JMJ)$ for every $q \in Q$, $\sigma \in \Sigma$, and $\omega \in \Omega$.

(ii) Suppose that we are given a rtDVA$(k)^1_c$ $\mathcal{V} = (Q, \Sigma, \delta, q_0, Q_a, v)$. We construct an equivalent rtDVA(k+1) $\mathcal{V}' = (Q, \Sigma, \delta', q_0, Q_a, v')$. The idea is to repeatedly subtract $(c-1)$ from the first entry of the vector when it is time for that value to be checked, and then add $(c-1)$ to restore the original vector. We will use the additional entry (which will always equal 1 throughout the computation) in the vector of $\mathcal{V}'$ to perform these additions and subtractions, as will be explained soon. Let v'' be a $(k+1)$-dimensional vector equaling $[v_1, v_2, ..., v_k, 1]$, where $v = [v_1, v_2, ..., v_k]$. The initial vector of $\mathcal{V}'$ has to be a modified version of v'' to accommodate the check for equality to 1 in the first step, so $v' = v''E^{k+1}_{k+1}(-c+1)$. For every individual transition $\delta(q, \sigma, \omega) = (q', M)$ of $\mathcal{V}$, $\mathcal{V}'$ has the move $\delta'(q, \sigma, \omega) = (q', E^{k+1}_{k+1}(c-1)NE^{k+1}_{k+1}(-c+1))$, where the $(k+1) \times (k+1)$ matrix N has been obtained by adding a new row-column pair to M, i.e. $N_{i,j} = M_{i,j}$ for $i, j = 1, ..., k$, $N_{(k+1)j} = 0$ for $j = 1, ..., k$, $N_{i(k+1)} = 0$ for $i = 1, ..., k$ and $N_{(k+1)(k+1)} = 1$.

Note that when $c \neq 0$, there is an alternative method for constructing an equivalent rtDVA(k) which does not require an extra entry in the vector, where the first entry is modified simply by repeated multiplications with $E_k^1(1/c)$ and $E_k^1(c)$ when necessary. □

We conclude this section with two examples that will familiarize us with the programming of rtDVA(k)'s.

Example 3.3. UFIBONACCI $= \{a^n \mid n \textit{ is a Fibonacci number}\} \in \mathfrak{L}(\text{rtDVA}(5))$.

Proof. We construct a rtDVA(5) $\mathcal{V}$ recognizing UFIBONACCI as follows: We let the initial vector equal $[0, 1, 0, 0, 1]$. Reading each a, we multiply the vector with the matrix M_1 if the first entry of the of the vector is equal to 0, and with M_2 otherwise.

$$M_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 & 1 \end{bmatrix} \quad M_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

After reading the i'th a, the fourth entry of the vector equals i. The second and third entries of the vector hold consecutive Fibonacci numbers. The first entry is equal to 0 whenever i equals the second entry, which triggers the next Fibonacci number to be computed and assigned to the second entry in the following step. Otherwise, the second and third entries remain unchanged until i reaches the second entry. $\mathcal{V}$ accepts if the computation ends with the first entry equaling 0, which occurs if and only if the input length n is a Fibonacci number. □

Theorem 3.4. UGAUSS $= \{a^{n^2+n} \mid n \in \mathbb{N}\} \in \mathfrak{L}(\text{rtDVA}(2))$.

Proof. We construct a rtDVA(2) $\mathcal{V}$ with initial vector $[1, 1]$. If the input is the empty string, $\mathcal{V}$ accepts. Otherwise, $\mathcal{V}$ increments the first entry of the vector by multiplying it by 2 on reading the first a which is performed by multiplying the vector with the matrix $M_1 = E_2^1(2)$.

$$M_1 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

It then repeats the following procedure for the rest of the computation: Decrement the first entry of the vector by multiplying it by $\frac{1}{2}$ until it reaches one, while parallelly incrementing the second entry of the vector by multiplying it by 2 with the help of matrix M_2. The second entry stops increasing exactly when the first counter reaches 1. Then the directions are swapped, with the second entry now being decremented, and the first entry going up by multiplying the vector with the matrix M_3.

$$M_2 = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 2 \end{bmatrix} M_3 = \begin{bmatrix} 2 & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$

When the second entry of the vector reaches 1, the first entry of the vector is multiplied by 2 one more time with the help of matrix M_1. Throughout this loop, the accept state is entered only when the first entry of the vector is equal to 1.

Suppose that at some step, the value of the vector is $[1, 2^c]$. If the input is sufficiently long, $2c + 2$ steps will pass before the first counter reaches 1 again, with the vector having the value $[1, 2^{c+1}]$. On an infinite sequence of a's, the accept state will be entered after reading the second a, and then again with intervals of $2c + 2$ symbols between subsequent entrances, for $c = 1, 2, 3....$ Doing the sum, we conclude that strings of the form a^{n^2+n}, $n \in \mathbb{N}$, are accepted. □

3.1. Deterministic vector automata

We start by specializing a fact stated by Ibarra *et al.* in [4] in the context of 1DFAM's to the case of rtDVA(1)'s. For this purpose, we will use the following well-known fact about counter machines.

Fact 3.5. [2] *Given any k-counter automaton A with the ability to alter the contents of each counter independently by any integer between $+c$ and $-c$ in a single step (for some fixed integer c), one can effectively construct a k-counter automaton which can modify each counter by at most one unit at every step, and which recognizes the same language as A in precisely the same number of steps.*

Fact 3.6. rtDVA(1)*'s are equivalent in language recognition power to real-time deterministic multicounter automata which can only check if all counters are equal to 0 simultaneously.*

Proof. Let us simulate a given rtDVA(1) $\mathcal{V}$ by a real-time deterministic multicounter automaton $\mathcal{M}$. Let $S = \{m_1, m_2, ..., m_t\}$ be the set of numbers the single-entry "vector" can be multiplied with during the computation. Let $P = \{p_1, p_2, ..., p_k\}$ be the set of prime factors of the denominators and the numerators of the numbers in S. $\mathcal{M}$ will have k counters $c_1, ..., c_k$ to represent the current value of the vector. When $\mathcal{V}$ multiplies the vector with $n_i = \frac{a}{b}$, where $a = p_1^{x_1} p_2^{x_2} \dots p_k^{x_k}$ and $b = p_1^{y_1} p_2^{y_2} \dots p_k^{y_k}$, the counters of $\mathcal{M}$ are updated by the values $(x_1 - y_1, x_2 - y_2, ..., x_k - y_k)$. As stated in Fact 3.5, we can update the counter values by any integer between c and $-c$, where c here is equal to the largest exponent in the prime decomposition of the numbers in S. When $\mathcal{V}$ checks if the value of the vector is equal to 1, $\mathcal{M}$ checks if the current value of the counters is $(0, 0, ..., 0)$, since the value of the vector is equal to 1 exactly when all the counters are equal to 0.

For the other direction, we should simulate a rtDkCA $\mathcal{M}$ that can only check if all counters are equal to 0 simultaneously with a rtDVA(1) $\mathcal{V}$. For each counter c_i of $\mathcal{V}$, we assign a distinct prime number p_i for $i = 1, ..., k$. We multiply the "vector"

with p_i and $\frac{1}{p_i}$, when the i'th counter c_i is incremented and decremented, respectively. Whenever $\mathcal{M}$ has all counters equal to 0, $\mathcal{V}$'s vector has value 1, so it can mimic $\mathcal{M}$ as required. □

We now prove a fact about rtDkCA's that will be helpful in the separation of the classes of languages associated with these machines and rtDVA(1)'s.

Theorem 3.7. UGAUSS $= \{a^{n^2+n} \mid n \in \mathbb{N}\} \in \mathfrak{L}$(rtD2CA).

Proof. We construct a real-time deterministic automaton $\mathcal{M}$ with two counters recognizing UGAUSS. The idea of the proof is the same with the proof of Theorem 3.4. If the input is the empty string, $\mathcal{M}$ accepts. Otherwise, $\mathcal{M}$ increments the first counter on reading the first a. It then repeats the following procedure for the rest of the computation: Decrement the first counter until it reaches zero, while parallelly incrementing the second counter. The second counter stops increasing exactly when the first counter reaches 0. The counters then swap directions, with the second counter now being decremented, and the first counter going up. When the second counter reaches 0, the first counter is incremented one more time.

Throughout this loop, the accept state is entered only when the first counter is zero.

Suppose that at some step, the value of the counter pair is $(0, c)$. If the input is sufficiently long, $2c + 2$ steps will pass before the first counter reaches zero again, with the pair having the value $(0, c + 1)$. On an infinite sequence of a's, the accept state will be entered after reading the second a, and then again with intervals of $2c+2$ symbols between subsequent entrances, for $c = 1, 2, 3....$ Doing the sum, we conclude that strings of the form a^{n^2+n}, $n \in \mathbb{N}$, are accepted. □

For $k \geq 1$, let LNG$_k = \{w \in \{a_0, a_1, ..., a_k\}^* \mid |w|_{a_0} = |w|_{a_1} = ... = |w|_{a_k}\}$, where $|w|_x$ denotes the number of occurrences of symbol x in w.

Fact 3.8. [13] $\mathrm{LNG}_k \in \mathfrak{L}(\mathrm{rtD}k\mathrm{CA})$, *and* $\mathrm{LNG}_k \notin \mathfrak{L}(\mathrm{rtD}(k\text{-}1)\mathrm{CA})$, *for every* $k \geq 1$.

Fact 3.9. [4] 1DFAM*'s can only recognize regular languages on unary alphabets.*

We are now able to state several new facts about the computational power of rtDVA(k)'s:

Theorem 3.10. *For any fixed* $k > 0$, $\mathfrak{L}(\mathrm{rtDVA}(1))$ *and* $\mathfrak{L}(\mathrm{rtD}k\mathrm{CA})$ *are incomparable.*

Proof. From Fact 3.8, we know that LNG_{k+1} can not be recognized by any rtDkCA. We can construct a rtDVA(1) $\mathcal{V}$ recognizing LNG_{k+1} as follows: We choose $k+1$ distinct prime numbers $p_1, ..., p_k, p_{k+1}$, each corresponding to a different symbol a_i in the input alphabet, where $i \in \{1, ..., k+1\}$. When it reads an a_i with i in that range, $\mathcal{V}$ multiplies its single-entry vector with p_i. When it reads an a_0, $\mathcal{V}$ multiplies the vector with $\frac{1}{p_1 p_2 \cdots p_k p_{k+1}}$. The input string w is accepted if the value of the vector is equal to 1 at the end of the computation, which is the case if and only if $w \in \mathrm{LNG}_{k+1}$. We conclude that $\mathrm{LNG}_{k+1} \in \mathfrak{L}(\mathrm{rtDVA}(1))$.

From Theorem 3.7, we know that rtDkCA's can recognize some nonregular languages on a unary alphabet. By Fact 3.9, we know that rtDVA(1)'s, which are additionally restricted 1DFAM's, can only recognize regular languages in that case. Hence, we conclude that the two models are incomparable. □

Theorem 3.11. $\mathfrak{L}(\mathrm{rtDVA}(1)) \subsetneq \bigcup_k \mathfrak{L}(\mathrm{rtD}k\mathrm{CA})$.

Proof. By the argument in the proof of Fact 3.6, any rtDVA(1) can be simulated by a rtDkCA for some k. The inclusion is proper, since we know that a rtD2CA can recognize a nonregular language on a unary alphabet (Theorem 3.7), a feat that is impossible for rtDVA(1)'s by Fact 3.9. □

Theorem 3.12. $\mathfrak{L}(\mathrm{rtDVA}(2)) \not\subseteq \bigcup_k \mathfrak{L}(\mathrm{rtD}k\mathrm{CA})$.

Proof. Let GEQ $= \{a^m b^n | m \geq n \geq 1\}$, and let GEQ* be the Kleene closure of GEQ. It is known that no rtDkCA can recognize GEQ* for any k, due to the inability of these machines to set a counter to 0 in a single step [14].

We will construct a rtDVA(2) $\mathcal{V}$ that recognizes GEQ*. The idea is to use the first entry of the vector as a counter, and employ matrix multiplication to set this counter to 0 quickly when needed. $\mathcal{V}$ rejects strings that are not in the regular set $(a^+b^+)^*$ easily. The vector starts out as $[0, 1]$. When it reads an a, $\mathcal{V}$ multiplies the vector with the "incrementation" matrix M_a to increment the counter. When reading a b, $\mathcal{V}$ rejects if the first entry is zero, since this indicates that there are more b's than there were a's in the preceding segment. Otherwise, it multiplies the vector with the "decrementation" matrix M_b.

$$M_a = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} M_b = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

When an a is encountered immediately after a b, the counter has to be reset to 0, so the M_a in the processing of such a's is preceded by the "reset" matrix M_0.

$$M_0 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$\mathcal{V}$ accepts if it reaches the end of the input without rejecting. □

We have mentioned that no multicounter automata can recognize GEQ* in real-time due to the inability of these machines to set a counter to 0 in a single step. Let us briefly talk about reset multicounter automaton introduced by Petersen in [15], which are counter machines with the additional capability that each counter can be reset to zero. With this additional capability, GEQ* is real-time recognizable by a reset counter machine with one counter. Now, let us present a result comparing the power of rtDkCA's and reset counter machines. Let us denote a real-time deterministic reset multicounter automaton with k counters as rtDkRCA.

Theorem 3.13. $\mathfrak{L}(\text{rtDVA}(3)) \not\subseteq \bigcup_k \mathfrak{L}(\text{rtD}k\text{RCA})$.

Proof. Let BIN $= \{0^{v(w)}cw^r | w \in 1\{0,1\}^*\}$ where $v(w)$ denotes the number represented by w in binary encoding. It is known that no rtDkRCA can recognize BIN for any k [15].

We will construct a rtDVA(3) $\mathcal{V}$ that recognizes BIN. $\mathcal{V}$'s initial vector is [0 0 1]. Reading each 0, $\mathcal{V}$ first increments the first entry of the vector at each step. After reading c, while reading w^r the number represented by w is encoded in the second entry of the vector. $\mathcal{V}$ multiplies the vector with matrix M_0 (resp. M_1) for each scanned 0 (resp. 1).

$$M_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \quad M_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

After reading the input string, $\mathcal{V}$ subtracts the second entry from the first entry by multiplying the vector with the matrix $E_4^3(-1)$. $\mathcal{V}$ accepts if the first entry of the vector is equal to 0, and rejects otherwise. □

We are now able to compare the power of rtDVA(1)'s with their one-way versions, namely, the 1DFAM's of Ibarra et al [4].

Theorem 3.14. $\mathfrak{L}(\text{rtDVA}(1)) \subsetneq \mathfrak{L}(\text{1DFAM})$.

Proof. We construct a 1DFAM $\mathcal{M}$ recognizing the language GEQ* that we saw in the proof of Theorem 3.12. $\mathcal{M}$ uses its register to simulate the counter of a one-way single-counter automaton. When it reads an a, $\mathcal{M}$ multiplies the register by 2. When reading a new b, $\mathcal{M}$ rejects if the register has value 1, and multiplies with $\frac{1}{2}$ otherwise. When a new block of a is seen to start, $\mathcal{M}$ pauses its input head while repeatedly multiplying

the register with $\frac{1}{2}$ to set its value back to 1 before processing the new block. $\mathcal{M}$ accepts if it has processed the whole input without rejecting.

By the already mentioned fact that no rtDkCA for any k can recognize GEQ*, and Theorem 3.11, we conclude that GEQ$^* \notin \mathfrak{L}$(rtDVA(1)). □

We can say the following about the power of rtDVA(k)'s when we increase k.

Corollary 3.15. $\mathfrak{L}(\text{rtDVA}(1)) \subsetneq \mathfrak{L}(\text{rtDVA}(2))$.

Proof. Note that Fact 3.9 and Theorem 3.4 let one conclude that rtDVA(2)'s outperform rtDVA(1)'s when the input alphabet is unary. □

A language L is in class TISP($t(n), s(n)$) if there is a deterministic Turing Machine that is both $t(n)$-time bounded and $s(n)$-space bounded and that decides L. It is easy to state the following simultaneous Turing machine time-space upper bound on the power of deterministic real-time vector automata:

Theorem 3.16. $\bigcup_k \mathfrak{L}(\text{rtDVA}(k)) \subseteq \text{TISP}(n^3, n)$.

Proof. A Turing machine that multiplies the vector with the matrices corresponding to the transitions of a given rtDVA(k) requires only linear space, since the numbers in the vector can grow by at most a fixed number of bits for each one of the $O(n)$ multiplications in the process. Using the primary-school algorithm for multiplication, this takes $O(n^3)$ overall time. □

If one gave the capability of one-way traversal of the input tape to vector automata of dimension larger than 1, one would gain a huge amount of computational power. Even with vectors of dimension 2, such machines can simulate one-way 2-counter automata, and are therefore Turing equivalent [16]. This is why we focus on real-time vector automata.

3.2. Blind vector automata

A *real-time deterministic blind vector automaton* (rtDBVA(k)) is a rtDVA(k) which is not allowed to check the entries of the vector until the end of the computation. Formally, a rtDBVA(k) is a 6-tuple

$$\mathcal{V} = (Q, \Sigma, \delta, q_0, Q_a, v),$$

where the transition function δ is defined as $\delta : Q \times \Sigma \to Q \times S$, with S as defined earlier. $\delta(q, \sigma) = (q', M)$ means that when $\mathcal{V}$ reads symbol $\sigma \in \Sigma$ in state q, it will move to state q', multiplying the vector with the matrix $M \in S$. The acceptance condition is the same as for rtDVA(k)'s.

Let us begin with an example of programming with rtDBVA(k)'s.

Example 3.17. PAL $= \{w | w = w^r, w \in \{a, b\}^*\} \in \mathfrak{L}$(rtDBVA(4)).

Proof. We construct a rtDBVA(4) $\mathcal{V}$ recognizing the palindrome language PAL as follows. We let the initial vector equal [0,1,0,1]. While reading the input string w, $\mathcal{V}$ encodes w and w^r using the matrices M_a and M_b.

$$M_a = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \text{ and } M_b = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 2 & 1 \end{bmatrix}$$

Each time reading an a and b, $\mathcal{V}$ multiplies the vector with the matrices M_a and M_b respectively. Note that M_a and M_b consist of 2 blocks of submatrices. We use

$$M_{a_1} = \begin{bmatrix} 1 & 0 \\ 1 & 10 \end{bmatrix} \text{ and } M_{b_1} = \begin{bmatrix} 1 & 0 \\ 2 & 10 \end{bmatrix}$$

to encode w^r and

$$M_{a_2} = \begin{bmatrix} 10 & 0 \\ 1 & 1 \end{bmatrix} \text{ and } M_{b_2} = \begin{bmatrix} 10 & 0 \\ 2 & 1 \end{bmatrix}$$

to encode w. The encoding is done so that each a is represented by 1 and each b is represented by 2. When we finish reading the string, we get w^r encoded in the first entry and w encoded in the third entry. When reading the right end-marker \$, the vector is multiplied with $E_4^3(-1)$. $\mathcal{V}$ accepts if the computation ends with the first entry equaling 0, which occurs if and only if $w = w^r$ so that the encodings of w and w^r are equal. □

Remark 3.18. *Let us note that* $\mathfrak{L}(\text{rtDBVA}(1)) = \bigcup_k \mathfrak{L}(\text{rtD}k\text{BCA})$*, unlike the general case considered in Theorem 3.11. Since blind counter automata only check if all counters are zero at the end, the reasoning of Fact 3.6 is sufficient to conclude this.*

Let us now present a result stating the equivalence between 1DFAMW's and rtDBVA(1).

Theorem 3.19. $\mathfrak{L}(\text{rtDBVA}(1)) = \mathfrak{L}(\text{1DFAMW})$.

Proof. A rtDBVA(1) is clearly a 1DFAMW, so we look at the other direction of the equality. Given a 1DFAMW $\mathcal{V}_1$, we wish to construct a rtDBVA(1) $\mathcal{V}_r$ which mimics $\mathcal{V}_1$, but without spending more than one computational step on any symbol. When $\mathcal{V}_1$ scans a particular input symbol σ for the first time in a particular state q, whether it will ever leave this symbol, and if so, after which sequence of moves, are determined by its program. This information can be precomputed for every state/symbol pair by examining the transition function of $\mathcal{V}_1$. We program $\mathcal{V}_r$ so that it rejects the input if it ever determines during computation that $\mathcal{V}_1$ would have entered an infinite loop. Otherwise, upon seeing the simulated $\mathcal{V}_1$ moving on a symbol σ while in state q, $\mathcal{V}_r$ simply retrieves the aforementioned information from a lookup table, moves the head to the right, entering the state that $\mathcal{V}_1$ would enter when it moves off that σ, and

multiplies its single-entry vector with the product of the multipliers corresponding to the transitions $\mathcal{V}_1$ executes while the head is pausing on σ. □

Remark 3.20. *In* [6]*, it is noted that* 1DFAMW *can be regarded as a finite automaton over the multiplicative group of non-null rational numbers. Having proven that* 1DFAMW *and* rtDBVA(1) *are equivalent models,* rtDBVA(1) *can be also regarded analogously. With a similar reasoning, in* [17] *it is stated that blind multicounter automata are finite automata over the monoid* $\mathbb{Z}^n$ *(a group where the elements do not necessarily have inverses).*

We now give a full characterization of the class of languages recognized by real-time deterministic blind vector automata.

Theorem 3.21. $\bigcup_k \mathfrak{L}(\text{rtDBVA}(k)) = \text{S}_{\mathbb{Q}}^{=}$.

Proof. For any language $L \in \text{S}_{\mathbb{Q}}^{=}$, we can assume without loss of generality that $L = (\mathcal{G}, =1)$ [5] for some TuFA $\mathcal{G}$ with, say, m states. Let us construct a rtDBVA(k) $\mathcal{V}$ simulating $\mathcal{G}$. We let $k = m$, so that the vector is in $\mathbb{Q}^k$. The initial vector values of $\mathcal{V}$ and $\mathcal{G}$ are identical. $\mathcal{V}$ has only one state, and the vector is multiplied with the corresponding transition matrix of $\mathcal{G}$ when an input symbol is read. When processing the right end-marker, $\mathcal{V}$ multiplies the vector with a matrix whose first column is the final vector f of $\mathcal{G}$. $\mathcal{V}$ accepts input string w if the first entry of the vector is 1 at the end of the computation, which happens only if the acceptance value $f_{\mathcal{G}}(w) = 1$.

For the other direction, let us simulate a rtDBVA(k) $\mathcal{V}$ recognizing some language L by a TuFA $\mathcal{G}$. If $\mathcal{V}$ has m states, then $\mathcal{G}$ will have km states. For any symbol σ, the corresponding transition matrix A is constructed as follows. View A as being tiled to m^2 $k \times k$ submatrices called $A_{i,j}$, for $i, j \in \{0, 1, ..., m-1\}$ which can be seen in Figure 3.2. If $\mathcal{V}$ moves from q_i to q_j by multiplying the vector with the matrix M when reading symbol σ, then $A_{i,j}$ will be set to equal M. All remaining entries of A are zeros. The initial vector v' of $\mathcal{G}$ will be a row vector with km entries, viewed as being segmented to m blocks of k entries. The first k entries of v', corresponding to the initial

Figure 3.1. Matrix A and submatrices $A_{i,j}$

$A_{0,0}$	$A_{0,1}$	$\dots$	$A_{0,m-1}$
$A_{1,0}$	$A_{1,1}$	$\dots$	$A_{1,m-1}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$A_{m-1,0}$	$A_{m-1,1}$	$\dots$	$A_{m-1,m-1}$

state of $\mathcal{V}$, will equal v, and the remaining entries of v' will equal 0. The km entries of the final column vector f of $\mathcal{G}$ will again consist of m segments corresponding to the states of $\mathcal{V}$. The first entry of every such segment that corresponds to an accept state of $\mathcal{V}$ will equal 1, and all remaining entries will equal 0. $\mathcal{G}$ imitates the computation of $\mathcal{V}$ by keeping the current value of the vector of $\mathcal{V}$ at any step within the segment that corresponds to $\mathcal{V}$'s current state in the vector representing the portion of $\mathcal{G}$'s own matrix multiplication up to that point. We therefore have that $L = (\mathcal{G}, =1)$.

□

Remark 3.22. *Note that if the components of a* rtDBVA(k) *are not restricted to be rational, then one can obtain the fact that* $\bigcup_k \mathfrak{L}(\text{rtDBVA}(k)) = \text{S}^=$.

In the next theorem, we prove that general real-time vector automata are strictly more powerful than their blind versions.

Theorem 3.23. $\bigcup_k \mathfrak{L}(\text{rtDBVA}(k)) \subsetneq \bigcup_k \mathfrak{L}(\text{rtDVA}(k))$

Proof. It is obvious that any rtDBVA(k) can be simulated by a rtDVA(k). In order to prove that the inclusion is strict, we are going to construct a rtDVA(1) $\mathcal{V}$ that recognizes the language NH $= \{a^x b a^{y_1} b a^{y_2} \cdots a^{y_k} | \exists m, 1 \leq m \leq k, \sum_{i=1}^{m} y_i = x\}$. Starting with the

initial value 0, $\mathcal{V}$ increases the vector by 1 at each step until reading the first b. After the first b is encountered, the vector is decremented while reading each a. When $\mathcal{V}$ reads a b, it checks if the current value of the vector is equal to 0. The input string is accepted if the value of the vector becomes 0 at any time of the computation and rejected otherwise. In [18], it is proven that NH is nonstochastic, meaning that there exists no probabilistic finite automaton recognizing NH. Since $\bigcup_k \mathfrak{L}(\text{rtDBVA}(k)) = \text{S}_{\mathbb{Q}}^{=}$ by Theorem 3.21, and $\text{S}_{\mathbb{Q}}^{=}$ is a proper subset of stochastic languages $\text{S}^{<}$ [19], we conclude the result. □

We can also give a characterization for the case where the alphabet is unary, thanks to the following fact, which is implicit in the proof of Theorem 7 in [20]:

Fact 3.24. *All languages on a unary alphabet in* $\text{S}_{\mathbb{Q}}^{=}$ *are regular.*

Note that in order to prove Theorem 3.23, we could also use the above fact that the unary languages recognized by blind vector automata are regular. Since we have already showed that some nonregular unary languages can be recognized by the standard model, the result follows.

Now, we can say the following about the effect of increasing k on the power of rtDBVA(k)'s:

Theorem 3.25. $\mathfrak{L}(\text{rtDBVA}(1)) \subsetneq \mathfrak{L}(\text{rtDBVA}(2))$.

Proof. Let us construct a rtDBVA(2) $\mathcal{V}$ recognizing the marked palindrome language MPAL $= \{wcw^r | w \in \{a, b\}^*\}$. We let the initial vector equal $[0, 1]$. A similar construction is given in Example 3.17 for the palindrome language. While reading the input string, $\mathcal{V}$ first encodes the string w in the first entry of the vector using the matrices M_{a_1} and M_{b_1} of Example 3.17.

Upon reading the symbol c, $\mathcal{V}$ finishes reading w and starts reading the rest of the string. $\mathcal{V}$ now makes a reverse encoding and multiplies the vector with M_{a_2} and

M_{b_2} each time it reads an a and a b, respectively.

$$M_{a_2} = \begin{bmatrix} \frac{1}{10} & 0 \\ -\frac{1}{10} & 1 \end{bmatrix} M_{b_2} = \begin{bmatrix} \frac{1}{10} & 0 \\ -\frac{2}{10} & 1 \end{bmatrix}$$

When the computation ends, the first entry of the vector is equal to 0 iff the string read after the symbol c is the reverse of the string w so that the input string is in MPAL.

Now, we are going to prove that MPAL $\notin \mathfrak{L}$(2PFA). Suppose for a contradiction that there exists a 2PFA $\mathcal{M}$ recognizing MPAL with bounded error. Then it is not hard to show that PAL can be recognized by a 2PFA $\mathcal{M}'$ such that $\mathcal{M}'$ sees the input, say w, as $u = wcw$ and then executes $\mathcal{M}$ on u. Note that $\mathcal{M}$ accepts u if and only if w is a member of PAL. Since PAL $\notin \mathfrak{L}$(2PFA) [21], we get a contradiction. Hence, we conclude that MPAL can not be in $\mathfrak{L}$(2PFA).

It is known [22] that $\mathfrak{L}$(2PFA) includes all languages recognized by one-way deterministic blind multicounter automata, and we already stated that rtDBVA(1) and rtDkBCA are equivalent models in Remark 3.18. Since MPAL $\notin \mathfrak{L}$(2PFA), MPAL cannot be in $\mathfrak{L}$(rtDBVA(1)). Having proven that MPAL $\in \mathfrak{L}$(rtDBVA(2)), we conclude that $\mathfrak{L}$(rtDBVA(1)) $\subsetneq \mathfrak{L}$(rtDBVA(2)). □

For an m-state rtDBVA(k) $\mathcal{V}$, we define the *size* of $\mathcal{V}$ to be the product mk. For all $i \geq 1$, let $\mathfrak{L}$(rtDBVASIZE(i)) denote the class of languages that are recognized by real-time deterministic blind vector automata whose size is i. We use the following fact to prove a language hierarchy on this metric.

Fact 3.26. [23] *(Recurrence Theorem) Let L be a language belonging to $\mathrm{S}_{\mathbb{Q}}^{=}$ in the alphabet Σ. Let $\mathcal{G}$ be a TuFA recognizing L such that $L = (\mathcal{G}, = \lambda)$ for some $\lambda \in \mathbb{Q}$ and let n be the number of states of $\mathcal{G}$. Then for any words $x, y, z \in \Sigma^*$, if $yz, yxz, ..., yx^{n-1}z \in$ L,*

then $yx^mz \in \mathrm{L}$ *for any* $m \geq 0$. [1]

Theorem 3.27. *For every* $i > 1$, $\mathfrak{L}(\mathrm{rtDBVASIZE}(i-1)) \subsetneq \mathfrak{L}(\mathrm{rtDBVASIZE}(i))$.

Proof. We first establish a hierarchy of complexity classes for TuFA's based on the number of states, and use this fact to conclude the result.

Let us construct a k state TuFA $\mathcal{G}$ recognizing $\mathrm{MOD}_k = \{a^i \mid i \neq 0 \mod k\} \in \mathrm{S}^{=}_{\mathbb{Q}}$ such that $\mathrm{MOD}_k = (\mathcal{G}, = 0)$. Note that since MOD_k is a regular language, what we actually do is to simulate the deterministic finite automaton recognizing it. Let v_0 be the $1 \times k$ initial vector of $\mathcal{G}$ which is equal to [1,0,...,0]. Transition matrix A_a is as follows:

$$A_a = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & & & & & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

In the beginning of the computation, $\mathcal{G}$ is in the initial state q_0. Reading each a, $\mathcal{G}$ will move to the next state until it reaches q_{k-1}. When in q_{k-1}, reading one more a $\mathcal{G}$ will move back to the initial state q_0 and the same procedure will be repeated for the rest of the computation. $\mathcal{G}$ will enter q_0 with intervals of k symbols between subsequent entrances. $v_0{A_a}^i$ is the vector describing the current state of $\mathcal{G}$ after reading i many a's, whose j'th entry is equal to 1 if $\mathcal{G}$ is in state q_{j-1}. An input string w is accepted if $\mathcal{G}$ is not in q_0 at the end of the computation. The acceptance value for string w is equal to $f_{\mathcal{G}}(w) = v_0{A_a}^{|w|}f$ where f is the final column vector whose first entry is equal to 1

[1] Recurrence Theorem in [23] is stated for the languages in $\mathrm{S}^{=}$. We were able to adapt it to our case since the theorem also works for the languages in $\mathrm{S}^{=}_{\mathbb{Q}}$.

and remaining entries are equal to 0. $f_{\mathcal{M}}(w)$ will be equal to 0 iff $\mathcal{M}$ is not in state q_0 at the end of the computation which will guarantee that the strings of the form a^i where $i \neq 0 \mod k$ are accepted.

Having proven that $\text{MOD}_k \in \text{S}^{=}_{\mathbb{Q}}$, we claim that any TuFA $\mathcal{G}$ recognizing MOD_k such that $\text{MOD}_k = (\mathcal{G}, = \lambda)$ should have at least k states. In order to prove our claim, we are going to use Fact 3.26 as follows: Let n be the number of states of $\mathcal{G}'$ and let us suppose that $n < k$. Let $x = a$, $y = a$ and let z be the empty string. Since the strings $a, a^2, ..., a^n$ are in MOD_k, by Fact 3.26 we see that the strings of the form a^+ are also in MOD_k and we get a contradiction. Hence, we conclude that $n \geq k$ should hold, and that $\mathcal{G}$ should have at least k states.

By Theorem 3.21 there exists a real-time blind deterministic vector automaton with size k (a rtDBVA(k) with just one state) recognizing the same language. Suppose that there exists another real-time blind vector automaton $\mathcal{V}$ with size k' such that $k' < k$. Then by Theorem 3.21, there exists a TuFA with k' states recognizing MOD_k. Since we know that any TuFA recognizing MOD_k should have at least k states, we get a contradiction.

Hence, for every $i > 1$, we have showed that there is a language (MOD_i) which is in $\mathcal{L}$(rtDBVASIZE(i)), but not in $\mathcal{L}$(rtDBVASIZE(i-1)) . □

3.3. Nondeterministic vector automata

We now define the *real-time nondeterministic vector automaton* (rtNVA(k)) by adding the capability of making nondeterministic choices to the rtDVA(k). The transition function δ is now replaced by $\delta : Q \times \Sigma \times \Omega \to \mathbb{P}(Q \times S)$, where $\mathbb{P}(A)$ denotes the power set of the set A. We will also study blind versions of these machines: A *real-time nondeterministic blind vector automaton* (rtNBVA(k)) is just a rtNVA(k) which does not check the vector entries until the end of the computation.

We start by showing that it is highly likely that rtNVA(k)'s are more powerful than their deterministic versions.

Theorem 3.28. *If* $\bigcup_k \mathfrak{L}(\text{rtNVA}(k)) = \bigcup_k \mathfrak{L}(\text{rtDVA}(k))$, *then* P=NP.

Proof. We construct a rtNBVA(3) $\mathcal{V}$ recognizing the NP-complete language SUBSET-SUM, which is the collection of all strings of the form $t\#a_1\#...\#a_n\#$, such that t and the a_i's are numbers in binary notation ($1 \leq i \leq n$), and there exists a set $I \subseteq \{1, ..., n\}$ satisfying $\sum_{i \in I} a_i = t$, where $n > 0$. $\mathcal{V}$'s initial vector is $[0, 0, 1]$. The main idea of the construction is similar to the one performed in 3.13 that we can encode the numbers appearing in the input string to certain entries of the vector, and perform arithmetic on them, all in real time. We use a similar encoding given in [24]. When scanning the symbols of t, $\mathcal{V}$ multiplies the vector with the matrix M_0 (resp. M_1) for each scanned 0 (resp. 1).

$$M_0 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad M_1 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

When $\mathcal{V}$ finishes reading t, the vector equals $[t, 0, 1]$. In the rest of the computation, $\mathcal{V}$ nondeterministically decides which a_i's to subtract from the second entry. Each selected a_i is encoded in a similar fashion to the second entry of the vector, using the matrices

$$N_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad N_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

After encoding the first selected a_i, the vector equals $[t, a_i, 1]$. $\mathcal{V}$ subtracts the second entry from the first entry by multiplying the vector with the matrix $E_3^2(-1)$. After this subtraction, the second entry is reinitialized to 0. $\mathcal{V}$ chooses another a_j if it wishes, and the same procedure is applied. At the end of the input, $\mathcal{V}$ accepts if the first entry of the vector is equal to 0, and rejects otherwise.

If $\bigcup_k \mathfrak{L}(\text{rtNVA}(k))=\bigcup_k \mathfrak{L}(\text{rtDVA}(k))$, then SUBSETSUM would be in P by Theorem 3.16, and we would have to conclude that P=NP. □

We can say the following when we restrict k to 1.

Theorem 3.29. $\mathfrak{L}(\text{rtDVA}(1)) \subsetneq \mathfrak{L}(\text{rtNVA}(1))$.

Proof. We construct a rtNVA(1) $\mathcal{V}$ recognizing the language ISK $= \{a^n b^n | n \geq 1\} \cup \{a^n b^{2n} | n \geq 1\}$ as follows. $\mathcal{V}$ nondeterministically branches into two, multiplying its single-entry vector with 2 in the first branch and with 4 in the second branch while reading each a. While reading b's, $\mathcal{V}$ multiplies the single-entry vector by $\frac{1}{2}$. At the end of the computation, the string is accepted if one of the branches leads to an accept state with the single-entry vector having value 1.

In [4], it is proven that the language ISK can not be recognized by a 1DFAM. Having proven that $\mathfrak{L}(\text{rtDVA}(1)) \subsetneq \mathfrak{L}(\text{1DFAM})$ in Theorem 3.14, we conclude the result. □

Now we move our consideration to nondeterministic blind vector automata.

Remark 3.30. *Let us note that* $\mathfrak{L}(\text{rtNBVA}(1)) = \bigcup_k \mathfrak{L}(\text{rtN}k\text{BCA})$*, similar to the deterministic case considered in Remark 3.18. We can similarly state the equivalence* $\mathfrak{L}(\text{rtNBVA}(1)) = \mathfrak{L}(\text{1NFAMW})$ *for the nondeterministic case. The idea of Theorem* 3.19 *is enough to conclude this.*

Having stated the above remark, the following fact allows us to make a characterization for rtNBVA(1)'s where the alphabet is unary.

Fact 3.31. [4] *All* 1NFAMW *recognizable languages over a one letter alphabet are regular.*

Let us first prove a separation result between the blind and non-blind versions.

Theorem 3.32. $\mathfrak{L}(\text{rtNBVA}(1)) \subsetneq \mathfrak{L}(\text{rtNVA}(1))$.

Proof. We construct a rtDVA(1) $\mathcal{V}$ recognizing the language $\text{EQ}^* = \{a^n b^n | n \geq 0\}^*$. Reading each a and b, $\mathcal{V}$ multiplies its single-entry vector by 2 and $\frac{1}{2}$ respectively which is initially set to 1. After finishing reading a block of b's, $\mathcal{V}$ checks if the single-entry of the vector is equal to 1 and rejects the input string if this is not the case. An input string is accepted if $\mathcal{V}$ can process the string without rejecting.

It is known that EQ^* can not be recognized by a 1NFAMW [4]. Having stated that $\mathfrak{L}(\text{rtDVA}(1)) \subsetneq \mathfrak{L}(\text{rtNVA}(1))$ in Theorem 3.29 and recalling that $\mathfrak{L}(\text{rtNBVA}(1)) = \mathfrak{L}(\text{1NFAMW})$, the result follows. (Note that it also follows that $\mathfrak{L}(\text{rtDBVA}(1)) \subsetneq \mathfrak{L}(\text{rtDVA}(1))$.) □

Now, we prove the following unconditional separation between the deterministic and nondeterministic versions.

Theorem 3.33. $\mathfrak{L}(\text{rtNBVA}(2)) \nsubseteq \bigcup_k \mathfrak{L}(\text{rtDBVA}(k))$.

Proof. Let us construct a rtNBVA(2) $\mathcal{V}$ recognizing the language $\text{POW} = \{a^{k+2^k} \mid k > 0\}$. The initial value of $\mathcal{V}$'s vector is $[1, 1]$. $\mathcal{V}$'s computation consists of two stages. In the first stage, $\mathcal{V}$ doubles the value of the first entry for each a that it scans, by multiplying the vector with the matrix M_1. At any step, $\mathcal{V}$ may nondeterministically decide to enter the second stage. In the second stage, $\mathcal{V}$ decrements the first entry by

1, for each a that is scanned, using the matrix M_2, and accepts if the first entry equals 0 at the end.

$$M_1 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} M_2 = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

If the input length is n, and if $\mathcal{V}$ decides to enter the second stage right after the k'th a, the vector value at the end of the computation equals $[2^k - (n-k), 1]$. We see that $2^k - (n-k) = 0$ if and only if $n = k + 2^k$ for some k.

Having proven that the nonregular language POW $\in \mathfrak{L}$(rtNBVA(2)), we note that POW can not be in $\bigcup_k \mathfrak{L}$(rtDBVA(k)), by Theorem 3.21, and Fact 3.24. □

We can also state the following corollary by the same reasoning:

Corollary 3.34. $\mathfrak{L}$(rtNBVA(1)) $\subsetneq \mathfrak{L}$(rtNBVA(2)).

Proof. We have already stated that unary languages recognized by rtNBVA(1)'s are regular in Fact 3.31. Having proven that POW $\in$ rtNBVA(2) in Theorem 3.33, we conclude the result. □

4. CONCLUSIONS

4.1. Open Questions and Future Work

- Can we show a hierarchy result based on the dimension of the vector for general deterministic vector automata or for nondeterministic vector automata? Theorem 3.27 suggests that such a hierarchy can be possible based on the "size" of the automata which we defined as the number of states times the dimension of the vector.
- We have proved that for the deterministic case real-time vector automata are more powerful than their blind versions. Is this also true for the nondeterministic case?
- We have proved that the class of languages recognized by deterministic blind vector automata correspond to $S^{=}_{\mathbb{Q}}$. Would properly defined probabilistic versions of vector automata correspond to larger classes? Would quantum vector automata outperform the probabilistic ones?
- Can we show any succinctness results with vector automata? Is it possible to recognize some regular languages with fewer states? Can we simulate a finite automaton using a deterministic blind vector automaton so that the size of the vector automaton is smaller than the number of states of the finite automaton?
- We have mentioned that we focus on real-time computation since one-way deterministic vector automaton would be Turing-equivalent. Is it possible to make time efficient Turing Machine simulation using a one-way deterministic vector automaton?
- Assume that we limit the number of queries made to the vector so that we can check the value of the vector only a constant number of times. Can we show hierarchy results based on the number of queries? Similarly, what happens when we limit the number of queries made to the counters in a multicounter automaton? Will the obtained model be more powerful than the blind version?
- If we change the $=$ comparison with $>$ in the deterministic blind vector automata,

we obtain the class of stochastic languages. Can we properly make a modification to the = control so that the class of languages recognized will correspond to the languages recognized by bounded error?

- The real-time vector automaton has a limitation that we can only check one of the entries of the vector at each step. This limitation prevents us from simulating a real-time deterministic multicounter automaton since we can not detect the status of the counters in a single step. What happens if we allow to check multiple entries of the vector? Would it be as powerful as the one-way model if all entries of the vector can be checked in a single step?

REFERENCES

1. Fortnow, L., "One Complexity Theorist's View of Quantum Computing", *Theoretical Computer Science*, Vol. 292, No. 3, pp. 597–610, 2003.

2. Fischer, P. C., A. R. Meyer and A. L. Rosenberg, "Counter Machines and Counter Languages", *Mathematical Systems Theory*, Vol. 2, No. 3, pp. 265–283, 1968.

3. Greibach, S. A., "Remarks on Blind and Partially Blind One-Way Multicounter Machines", *Theoretical Computer Science*, Vol. 7, pp. 311–324, 1978.

4. Ibarra, O. H., S. K. Sahni and C. E. Kim, "Finite Automata with Multiplication", *Theoretical Computer Science*, Vol. 2, No. 3, pp. 271 – 294, 1976.

5. Turakainen, P., "Generalized Automata and Stochastic Languages", *Proceedings of the American Mathematical Society*, Vol. 21, pp. 303–309, 1969.

6. Mitrana, V., R. Stiebe, T. Centre and C. Science, "The Accepting Power of Finite Automata Over Groups", *New Trends in Formal Languages*, pp. 39–48, Springer, 1997.

7. Rabin, M. O., *Sequential Machines: Selected Paper*, chap. Probabilistic automata, pp. 98–114, Addison-Wesley, 1964.

8. Freivalds, R., "Probabilistic Two-Way Machines", *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pp. 33–45, 1981.

9. Yakaryılmaz, A., "Superiority of One-way and Realtime Quantum Machines", *RAIRO - Theoretical Informatics and Applications.*, Vol. 46, No. 4, pp. 615–641, 2012.

10. Rabin, M. O., "Probabilistic Automata", *Information and Control*, Vol. 6, pp.

230–243, 1963.

11. Kaŋeps, J., "Stochasticity of the Languages Acceptable by Two-way Finite Probabilistic Automata", *Diskretnaya Matematika*, Vol. 1, pp. 63–67, 1989, (Russian).

12. Paz, A., *Introduction to Probabilistic Automata*, Academic Press, New York, 1971.

13. Laing, R., *Realization and Complexity of Commutative Events*, Tech. rep., University of Michigan, 1967.

14. Fischer, P. C., A. R. Meyer and A. L. Rosenberg, "Real Time Counter Machines", *Proceedings of the 8th Annual Symposium on Switching and Automata Theory (SWAT 1967)*, FOCS '67, pp. 148–154, IEEE Computer Society, Washington, DC, USA, 1967.

15. Petersen, H., "Simulations by Time-Bounded Counter Machines", *Proceedings of the 13th International Conference on Developments in Language Theory*, DLT '09, pp. 410–418, Springer-Verlag, Berlin, Heidelberg, 2009.

16. Minsky, M. L., *Computation: Finite and Infinite Machines*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

17. Kambites, M., "Formal Languages and Groups as Memory", *Communications in Algebra*, Vol. 37, No. 1, pp. 193–208, 2009.

18. Masakazu Nasu, N. H., "A Context-Free Language Which is not Acceptable by a Probabilistic Automaton", *Information and Control*, Vol. 18, No. 3, pp. 233–236, 1971.

19. Turakainen, P., "On Languages Representable in Rational Probabilistic Automata", *Annales Academiae Scientiarum Fennicae, Ser.A*, , No. 439, pp. 4–10, 1969.

20. Diêu, P. D., "Criteria of representability of languages in probabilistic automata", *Cybernetics and Systems Analysis*, Vol. 13, No. 3, pp. 352–364, 1977, translated from Kibernetika, No. 3, pp. 39–50, May–June, 1977.

21. Dwork, C. and L. Stockmeyer, "Finite State Verifiers I: The Power of Interaction", *Journal of the ACM*, Vol. 39, No. 4, pp. 800–828, 1992.

22. Ravikumar, B., "Some Observations on 2-way Probabilistic Finite Automata", *Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science*, pp. 392–403, Springer-Verlag, 1992.

23. Diêu, P. D., "On a Class of Stochastic Languages", *Mathematical Logic Quarterly*, Vol. 17, No. 1, pp. 421–425, 1971.

24. Yakaryılmaz, A., "Quantum Alternation", *Proceedings of the 8th International Computer Science Symposium in Russia*, 2013 (to appear).