ANNALES
UNIVERSITATIS MARIAE CURIE - SKLODOWSKA
LUBLIN - POLONIA
VOL. L, 6 SECTIO A 1996

ZDZISŁAW GRODZKI and JERZY MYCKA (Lublin)

Two-dimensional Markov-like Algorithms

ABSTRACT. Four classes of two-dimensional Markov-like algorithms are introduced which are equivalent to the class $\mathcal{MNA}$ of Markov normal algorithms. Only for illustration the proof of equivalence of one of these classes and $\mathcal{MNA}$ is given in detail.

1. Introduction. Markov normal algorithms have been intensively studied by Russian scientists [7] since 1950. Most results have been obtained in the sixties. Dietlows [4] has shown that the class $\mathcal{MNA}$ of Markov normal algorithms is equivalent to the class $\mathcal{PRF}$ of partial recursive functions, whereas Asser [1] has shown that $\mathcal{MNA}$ is equivalent to the class $\mathcal{TM}$ of Turing machines. Other notions of effective computability, such as λ-notation of Church [2], or unlimited register machine [3], are well known. All above - mentioned notions of algorithm are equivalent to each other. The comprehensive study on this subject can be found in Mendelson's monograph [8], as well as in [6].

Two new classes $\mathcal{LMA}_k$ and $\overline{\mathcal{LMA}_k}$ of Markov-like k-algorithms have been introduced in [5]. They are also equivalent to the class $\mathcal{MNA}$ of Markov normal algorithms.

The aim of this paper is to introduce four classes of two-dimensional Markov-like algorithms which are equivalent to the class $\mathcal{MNA}$ of Markov normal algorithms. Only for one of them, denoted by $\mathcal{MA}^2_{(n,w)}$, the proof of the theorem relating to the equivalence of this class and $\mathcal{MNA}$ is given. An idea of the proofs of theorems related to the equivalence of the remainng three classes and $\mathcal{M}NA$ is similar and therefore the proofs will be omitted here.

The formalization used here allows us to introduce four classes ($n \geq 1$) of n-dimensional Markov-like algorithms and to show easily the equivalence of these classes and $\mathcal{M}NA$.

Let us add that the formalization used here can be easily adopted to define other classes of n-dimensional algorithms (not necessarily Markov-like), for instance n-dimensional matrix of indexed algorithms.

Inspiration of introducing the above classes was Priese's paper [9], where two-dimensional Thue systems have been used for transformation of objects, which are two-dimensional words in Gaussian plane. The Priese's paper is related to the problem how to develope concurent, asynchronous two-dimensional calculi with the computation complexity of safe Petri nets and self-reproducing Turing machines.

But what is the motivation to consider n–dimensional algorithms ($n \geq 2$). Transformation of n–dimensional Gaussian plane appears in a few areas of science or technique, such as cartography, chemistry, cryminology, crystalography, meteorology, mechanics, picture processing, physics (percolation theory) and many others. Therefore it is suitable to study n–dimensional algorithms (not necessarily Markov-like) that would be a good tool to solve uniformly some practical problems from areas mentioned above.

2. Two-dimensional words and productions.

Before we introduce a class $\mathcal{M}A^2_{(n,w)}$ of two-dimensional Markov-like algorithms we must introduce at the beginning some auxiliary notions.

Let Σ be a finite (nonempty) alphabet. The symbol $\varepsilon \notin \Sigma$ will represent the empty symbol. In what follows $\mathbb{Z}$ and N stand for the set of integers, or positive integers, resp..

Let us consider a partial function $\psi : \mathrm{N}^2 \mapsto \Sigma$. The domain of ψ will be denoted by $\mathrm{Dom}(\psi)$ and the set of all functions from N^2 into Σ will be denoted by Ψ_Σ.

Let ψ_1, ψ_2 be two functions, both from Ψ_Σ. Let us introduce a relation $E \in \Psi_\Sigma \times \Psi_\Sigma$ as follows:

$$\psi_1 E \psi_2 \iff (\exists r, s \in \mathbb{Z})(\forall i, j \in \mathrm{N})$$

$$\{[((i,j) \in \mathrm{Dom}(\psi_1)) \wedge ((i+r, j+s) \in \mathrm{Dom}(\psi_2)) \wedge (\psi_1(i,j) = \psi_2(i+r, j+s))]$$

$$\vee$$

$$[((i,j) \notin \mathrm{Dom}(\psi_1)) \wedge ((i+r, j+s) \notin \mathrm{Dom}(\psi_2))]\}.$$

Then E is an equivalence relation.

For every function $\psi \in \Psi_\Sigma$ let we set

$$||\psi||_E = \{\xi \in \Psi_\Sigma : \xi E \psi\}.$$

The eqivalence class $||\psi||_E$ will be called *two-dimensional word in an alphabet* Σ.

Let us assume the following convention. For every function $\psi \in \Psi_\Sigma$ we will consider such an element $\xi \in ||\psi||_E$ that $\{(0,j),(i,0)\} \subset \mathrm{Dom}(\xi)$ for some numbers $i,j \in \mathbb{N}$. In the majority of cases such representants ξ of $||\psi||_E$ will be considered. For simplicity such ξ will be called a two-dimensional word and denoted by t,u,v,w,x,y,z (possibly with subscripts). We will write in this case $t_{i,j}$ instead of $\xi(i,j)$ and $\mathrm{Dom}(t)$ instead of $\mathrm{Dom}(\xi)$. The set of all two-dimensional words in an alphabet Σ will be denoted by $\Sigma_2^\star$.

The two-dimensional word t will be called over an alphabet Σ iff t is in an alphabet Σ', where $\Sigma' \supset \Sigma$.

Now let us define a shape of a two-dimensional word $t \in \Sigma_2^\star$. The pair (m,p) is said to be *a shape* of a two-dimensional word t iff

$$m = \sup\{i : (\exists j \in N)(i,j) \in \mathrm{Dom}(t)\} + 1$$
$$p = \sup\{j : (\exists i \in N)(i,j) \in \mathrm{Dom}(t)\} + 1.$$

The word t will be usually written in the form:

$$\begin{array}{ccc} t_{m,0} & \dots & t_{m,p} \\ \dots & \dots & \dots \\ t_{0,0} & \dots & t_{0,p} \end{array}$$

where $t_{i,j} = \varepsilon$ if $(i,j) \notin \mathrm{Dom}(t)$.

For arbitrary two-dimensional words $u, v \in \Sigma_2^\star$ a word u is *a subword* of v, $u \preceq v$, iff there exist nonnegative integers k, l such that for all $(i,j) \in \mathrm{Dom}(u)$ we have:

$$u_{i,j} = v_{i+k,j+l}.$$

Example 2.1. Let us consider two two-dimensional words u and v of $\Sigma_2^\star$, $\Sigma = \{a,b\}$, where u and v have the form:

$$u = \begin{array}{ccccc} & a & a & a & \\ b & b & & & \\ & & a & a & a \end{array}$$

$$v = \begin{array}{ccccccc} a & a & a & a & a & a & \\ & b & b & b & b & b & a \\ a & a & a & a & a & a & a \\ b & b & b & b & b & b & \\ & a & a & a & a & a & a \end{array}$$

It is simple to see, that $u \preceq v$.

By a restriction of a two-dimensional word v to a Gaussian plane P, $v|_P$, we mean such a word u that

$$u_{i,j} = v_{i,j}$$

for all $(i,j) \in P$.

Let us introduce the notions of a first (f,g)-occurrence of a two-dimensional word into another one for $(f,g) \in \{(n,w),(n,e),(s,w),(s,e)\}$, where (n,w), (n,e), (s,w), (s,e) are the abbreviations of north-western, north-eastern, south-western and south-eastern.

Let u and v be two-dimensional words of Σ_2^*. Then a restricted word $v|_P$, $P \subset \mathrm{Dom}(v)$, is said to be the *first* (n,w)*-occurrence* of u in v iff the following conditions hold:

(1) $v|_P = u$

(2) for arbitrary two–dimensional Gaussian plane $P' \subset \mathrm{Dom}(v)$, if $v|_{P'} = u$ then there exists a pair $(i,j) \in P$, such that for all $(i',j') \in P'$ we have $i' \leq i$ or $j' \geq j$.

For the first (n,e)*–occurrence*, first (s,w)*–occurrence* and *first* (s,e)*–occurrence* the condition (2) should be replaced by the following ones:

(2′) for arbitrary two-dimensional Gaussian plane $P' \subset \mathrm{Dom}(v)$, if $v|_{P'} = u$ then there exists a pair $(i,j) \in P$, such that for all $(i',j') \in P'$ we have $i' \leq i$ or $j' \leq j$.

(2″) for arbitrary two-dimensional Gaussian plane $P' \subset \mathrm{Dom}(v)$, if $v|_{P'} = u$ then there exists a pair $(i,j) \in P$ such that for all $(i',j') \in P'$ we have $i' \geq i$ or $j' \geq j$.

(2‴) for arbitrary two-dimensional Gaussian plane $P' \subset \mathrm{Dom}(v)$, if $v|_{P'} = u$ then there exists a pair $(i,j) \in P$, such that for all $(i',j') \in P'$ we have $i' \geq i$ or $j' \leq j$.

Remark 2.2. The analogy of the first north-western occurrence of a word u in a word v is the same as in topography. We localize any object on a map most in the western and northern directions (analogously for the other occurrences).

Example 2.3. Let us consider two-dimensional words u and v of the Example 2.1. Then we have:

$v|_{P_1}$ is the first (n,w) – occurrence of u in v,

$v|_{P_2}$ is the first (n,e) – occurrence of u in v,

$v|_{P_3}$ is the first (s,w) – occurrence of u in v

$v|_{P_4}$ is the first (s,e) – occurrence of u in v,

where

$$P_1 = \{(2,3),(2,4),(2,5),(3,1),(3,2),(4,2),(4,3),(4,4)\},$$
$$P_2 = \{(2,4),(2,5),(2,6),(3,2),(3,3),(4,3),(4,4),(4,5)\},$$
$$P_3 = \{(0,2),(0,3),(0,4),(1,0),(1,1),(2,1),(2,2),(2,3)\},$$
$$P_4 = \{(0,4),(0,5),(0,6),(1,2),(1,3),(2,3),(2,4),(2,5)\}.$$

Now let us introduce a notion of a two-dimensional production.

By *a two-dimensional production* in an alphabet Σ we mean a pair (x,y) of two equally shaped two-dimensional words in Σ with the property

$$x_{i,j} \quad \text{is undefined} \iff y_{i,j} \quad \text{is undefined}$$

for $1 \le i \le m$, $1 \le j \le p$.

Let $\mathcal{P}_\Sigma$ denote the set of all two-dimensional productions over an alphabet Σ of arbitrary shapes.

The elements of $\mathcal{P}_\Sigma$ will be usually denoted by $x \longrightarrow y$ (possibly with subscripts). Some production will be called *final* and will be denoted by $x \longrightarrow \cdot y$ (by analogy with notation for Markov normal algorithms). If we do not underline that a production is final or nonfinal then it will be denoted by $x \longrightarrow (\cdot)y$.

A two-dimensional production $x \longrightarrow (\cdot)y$ is said to be *applicable* to a two-dimensional word t iff $x \preceq t$.

Now let us define *a resulting function* $\mathrm{Res}_{(n,w)} : \mathcal{P}_\Sigma \times \Sigma_2^\star \mapsto \Sigma_2^\star$ as follows.

For arbitrary two-dimensional production $x \longrightarrow (\cdot)y \in \mathcal{P}_\Sigma$ and a two-dimensional word $t \in \Sigma_2^\star$ we have:

$$\mathrm{Res}_{(n,w)}(x \longrightarrow (\cdot)y, t) = \begin{cases} u & \text{if } x \preceq t \\ t & \text{if } \neg(x \preceq t), \end{cases},$$

where u is a two-dimensional word of $\Sigma_2^\star$, which is obtained from t by replacing the first (n,w)-occurrence of x in t by y.

In the first case (if $x \preceq t$) a production $x \longrightarrow (\cdot)y$ is said to be *effectively used*, whereas in the second one (if $\neg(x \preceq t)$) *noneffectively used* to a word t.

3. The two-dimensional Markov-like algorithms. Four classes $\mathcal{M}A^2_{(n,w)}$, $\mathcal{M}A^2_{(n,e)}$, $\mathcal{M}A^2_{(s,w)}$, $\mathcal{M}A^2_{(s,e)}$ of two-dimensional algorithms are introduced. The classes mentioned above differ from one another only by transformations.

Namely, for the class $\mathcal{M}A^2_{(f,g)}$ and $(f,g) \in \{(n,w),(n,e),(s,w),(s,e)\}$ the transformation replaces the first (f,g) occurrence of the left side of a production in a transformed word by the right side of this production.

Only for illustration the class $\mathcal{M}A^2_{(n,w)}$ will be studied.

By *a two-dimensional Markov-like algorithm* of the class $\mathcal{M}A^2_{(n,w)}$ in an alphabet Σ we mean a sixtuple

$$A = (P_\Sigma, L, L_i, L_f, Contr, Tr_{(n,w)}),$$

where
P_Σ is a finite (nonempty) subset of $\mathcal{P}_\Sigma$ of two-dimensional productions in an alphabet Σ,
L is *set of labels* of P_Σ (we assume $L = \{1,\ldots|P_\Sigma|\}$),
$L_i = \{1\}$ and L_f are subsets of L whose elements are called *initial* and *final* productions[1].

A partial function $Contr : \Sigma_2^\star \times L \mapsto L$, called *a control* of A, and a total function $Tr_{(n,w)} : \Sigma_2^\star \times L \mapsto \Sigma_2^\star$, called *a transformation* of A, are defined as follows:

For arbitrary words $t, u \in \Sigma_2^\star$ and $i \in L$ we have:

$$Contr(t,i) = \begin{cases} 1 & \text{if } x_i \preceq t \text{ and } i \notin L_f \\ i+1 & \text{if } \neg(x_i \preceq t) \text{ and } i \leq |L| \\ \text{undefined} & \text{if } x_i \preceq t \text{ and } i \in L_f \\ & \text{or } \neg(x_i \preceq t) \text{ and } i = |L|, \end{cases}$$

$$Tr_{(n,w)}(t,i) = \begin{cases} u & \text{if } x_i \preceq t \text{ and } \mathrm{Res}_{(n,w)}(x_i \longrightarrow (\cdot)y_i, t) = u \\ t & \text{if } \neg(x_i \preceq t). \end{cases}$$

We denote some production from P_Σ by P_i iff $\phi(i) = P_i$, where ϕ is one-to-one mapping of P_Σ onto L.

[1] We will sometimes identify productions with their labels.

Thus if a production P_i has been effectively used to a word t and it is nonfinal then $Contr(t,i) = 1$ or if $\neg(x_i \preceq t)$ and $i < |L|$ then $Contr(t,i) = i+1$. $Contr$ is undefined if a production P_i has been effectively used to a word t and it is final ($i \in L_f$) or if $\neg(x_i \preceq t)$ and $i = |L|$.

If a production P_i has been effectively used to a word t then $Tr_{(n,w)}$ transforms a word t in a such way that the first (n,w)-occurrence of the left side x_i of a production P_i in t is replaced by the right side y_i of this production. If a production P_i has been noneffectively used to a word t then $Tr_{(n,w)}(t,i) = t$ and we continue a computation in every case (if P_i is final or nonfinal) with only such restriction that $i < |L|$. Let us observe that the process stops iff the last used production is final and additionally it has been effectively used to an actually transformed word or if the last used production P_j has been noneffectively used and $j = |L|$.

Following Markov [7] let us assume the following convention.

As for all algorithms of $\mathcal{MA}^2_{(n,w)}$ the controls $Contr$ and transformation $Tr_{(n,w)}$ are defined in the same manner. Therefore to define an algorithm $A \in \mathcal{MA}^2_{(n,w)}$ it is sufficient to define a sequence of productions in an alphabet Σ by omitting their labels (such sequence will be called *a schema of productions*).

In some examples it is necessary to introduce some additional letters (separators, parantheses). This leads to the following definition.

A two-dimensional Markov-like algorithm is said to be over an alphabet Σ iff it is an algorithm in some alphabet Σ' such that $\Sigma \subset \Sigma'$.

Now let us introduce the notion of a computation of a two-dimensional Markov-like algorithm.

A sequence $T = t_1, t_2, \ldots \in (\Sigma_2^\star)^\omega$ (finite or infinite) is said to be *a computation* of a two-dimensional algorithm $A = (P_\Sigma, L, L_i, L_f, Contr, Tr_{(n,w)})$ iff there exists a sequence $I = i_1, i_2, \ldots \in L^\infty$, called *a trace* of T, such that the following conditions hold:

(1) Both sequences T and I are infinite, $i_1 \in L_i$, and for every $j \geq 1$ we have: $t_{j+1} = Tr_{(n,w)}(t_j, i_j)$ and $i_{j+1} = Contr(t_j, i_j)$;

(2) Both sequences T and I are of finite length equal to m for some $m > 1$, $i_1 \in L_i$ and for every $1 \leq j < m$ we have: $t_{j+1} = Tr_{(n,w)}(t_j, i_j)$ and $i_{j+1} = Contr(t_j, i_j)$.[2]

We additionally assume that $i_m = |L|+1$ and this label indicates the fact that a computation T stops. Two cases imply that T is finite. The first one is when a production $P_{i_{m-1}}$ with the label i_{m-1} has been effectively used to a word t_{m-1} and it is final or if $P_{i_{m-1}}$ has been noneffectively used to a word t_{m-1} and $i_{m-1} = |L|$.

[2]Let us denote by $A(t_1)$ the last element t_m of a finite computation $T = t_1, \ldots, t_m$ of A and we assume that $A(t)$ is undefined if T is infinite.

The set of all computations of a two-dimensional Markov-like algorithm A is said to be its *computation set* and denoted by $\mathcal{C}(A)$.

From the above definitions we immediately obtain the following

Corollary 3.1. *Let $T = t_1, t_2, \ldots$ and $U = u_1, u_2, \ldots$ be arbitrary computations of a two-dimensional algorithm A in an alphabet Σ. Let $T_i = t_i, t_{i+1}, \ldots$ for every $i \geq 1$. Then we have:*

(3) *Every sequence T_i, $l(T_i) \geq 2$, is a computation of A;*

(4) *For every numbers $i, j \in N$, $i \leq l(T)$, $j \leq l(U)$ if $t_i = u_j$ then $t_{i+1} = u_{j+1}$ or both t_{i+1}, u_{j+1} are undefined.*

Remark 3.2. The conditions (3) and (4) are necessary (not sufficient) for a nonempty set $S \subset (\Sigma_2^\star)^\infty$ to be a computation set of any two-dimensional Markov-like algorithm.

Let us consider the following examples.

Example 3.3. Let A have the following schema of productions, where $\Sigma = \{b, \overline{b}, g, \overline{g}, w, r\}$,

$$P_1 : \begin{array}{cc} g & g \\ w & w \\ & b \end{array} \longrightarrow \begin{array}{cc} b & b \\ \overline{b} & \overline{b} \\ & b \end{array}$$

$$P_2 : \begin{array}{cc} g & \overline{g} \\ w & w \\ & b \end{array} \longrightarrow (\cdot) \begin{array}{cc} \overline{g} & \overline{g} \\ \overline{g} & \overline{g} \\ & b \end{array}$$

The following sequence is a computation of A:

$$\begin{array}{ccccccc} & g & g & b & g & \overline{g} & \\ \overline{b} & w & w & \overline{b} & w & w & \\ & & b & b & & b & \overline{b} \end{array}, \begin{array}{ccccccc} & b & b & b & g & \overline{g} & \\ \overline{b} & \overline{b} & \overline{b} & \overline{b} & w & w & \\ & & b & b & & b & \overline{b} \end{array}, \begin{array}{ccccccc} & b & b & b & \overline{g} & \overline{g} & \\ \overline{b} & \overline{b} & \overline{b} & \overline{b} & \overline{g} & \overline{g} & \\ & & b & b & & b & \overline{b} \end{array}$$

4. Equivalence of the classes $\mathcal{MNA}$ and $\mathcal{MA}^2_{(n,w)}$.

Before we formulate the equivalence of the classes $\mathcal{MNA}$ and $\mathcal{MA}^2_{(n,w)}$ we should introduce a manner of coding of two-dimensional words and productions into one-dimensional ones.

Let Σ be an alphabet which does not contain the symbols $\&$ and $-$. Let $c : \Sigma_2^\star \longmapsto (\Sigma \cup \{\&, -\})^\star$ be an injective mapping, called *a coding function*, defined as follows:

For an arbitary two-dimensional word $t \in \Sigma_2^\star$ we have:

$$c(t) = t'_1 \& t'_2 \& \ldots \& t'_m,$$

where t'_i is a sequence $t'_{i,1}, \ldots, t'_{i,p}$ for all $1 \le i \le m$ such that $t'_{i,j} = -$ if $t_{i,j}$ is undefined or $t'_{i,j} = t_{i,j}$ otherwise.

Example 4.1. Let us consider a two-dimensional word u in the alphabet $\Sigma = \{0,1\}$ of the form

$$\begin{array}{ccccc} & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & & \\ & & & 1 & 1 \end{array}$$

Then $c(u)$ is a one-dimensional word over $\{0,1\} \cup \{\&, -\}$ of the form: $c(u) = -1101\&100--\&---11$.

For $t \in \Sigma_2^\star$ by $c(t)_i$ we denote the code of the i-th component of t. Then we have $c(u)_1 = -1101, c(u)_2 = 100--, c(u)_3 = ---11$.

For a two-dimensional production $x \longrightarrow (\cdot)y$ its code $\overline{c}(x \longrightarrow (\cdot)y)$ is defined as $(c(x)_1 \longrightarrow c(y)_1, \ldots, c(x)_{n-1} \longrightarrow c(y)_{n-1}, c(x)_n \longrightarrow (\cdot)c(y)_n)$[3].

Lemma 4.2. *For every two-dimensional Markov-like algorithm $A \in \mathcal{MA}^2_{(n,w)}$ in an alphabet Σ we are able to construct effectively a Markov-normal algorithm M over an alphabet Σ such that for every two-dimensional word $t \in \Sigma_2^\star$*

$$c(A(t)) = M(c(t))$$

unless both algorithms are undefined.

Proof. Let us give at the beginning a short outline of the proof.

Let $A \in \mathcal{MA}^2_{(n,w)}$ be a two-dimensional algorithm in an alphabet Σ with a schema of productions of the form $(P_1, P_2, \ldots, P_n)$, where every production P_i has the form $(x_i \longrightarrow (\cdot)y_i)$ for $1 \le i \le n$.

For every P_i we shall construct one-dimensional productions sequence S_i "simulating" P_i. One can distinguish three parts in every sequence S_i.

The first one (denoted by S_i^1) chooses in a code of actually transformed word t the first line $c(x_i)_1$ counting by means of symbols λ_i^1 the distance of this subword from the left beginning of t. If such a subword there exists

[3]If the production $x \longrightarrow (\cdot)y$ is final then only last component $c(x)_n \longrightarrow (\cdot)c(y)_n$ is final

then S_i^1 replaces a subword $c(x_i)_1$ by a word in an additional alphabet Σ' of the form[4] $(c(x_i)_1)'$, otherwise the sequence S_{i+1} is started.

The second part S_i^2 chooses in the next parts of $c(t)$ the subwords $c(x_i)_j, 2 \leq j \leq k_i$, with respect to the distance of these subwords from & equal to the quantity of λ_i^1, and then replaces $c(x_i)_j$ by $(c(x_i)_j)'$ for every $2 \leq j \leq k_i$ If such a subword does not exist then the algorithm replaces the first symbol of the subword $(c(x_i)_1)'$ by the symbol of Σ''. The rest of symbols in the word $(c(x_i)_1)'$ are replaced by symbols of alphabet Σ. All symbols of Σ' are replaced by means the symbols of Σ and then the first part S_i^1 is started. If all subwords $c(x_i)_j, 2 \leq j \leq k_i$, are replaced by $(c(x_i)_j)'$ then the third part S_i^3 is started .

The third part S_i^3 ends "simulation" of P_i. All $(c(x_i)_j)'$ are replaced by $(c(y_i)_j)$, for $1 \leq j \leq k_i$.

Now we will present the full form of the sequence S_i by using the additional symbols (not in the alphabet Σ).

S_1^i is the sequence of one-dimensional productions of the form[5]:

$$
\begin{array}{ll}
\theta_i c(x_i)_1 \longrightarrow \lambda_i^1 (c(x_i)_i)' & \\
\theta_i \lambda_i^1 \longrightarrow \lambda_i^1 \lambda_i^1 & \\
\xi_i \lambda_i^1 \longrightarrow \lambda_i^1 & \\
\theta_i \alpha \theta_i \longrightarrow \alpha \theta_i \theta_i & \alpha \in \Sigma \cup \{-\} \\
\xi_i \alpha \longrightarrow \alpha \xi_i \theta_i & \\
\theta_i \alpha \longrightarrow \alpha \theta_i & \\
\theta_i \& \longrightarrow \& & \\
\xi_i \& \longrightarrow \& \xi_i \theta_i & \\
\theta_i \longrightarrow \varepsilon & \\
\xi_i \longrightarrow \eta_i &
\end{array}
$$

The first three productions are used if there exists the first line $c(x_i)_1$ of the left side of production P_i in a transformed word. In this case quantity of λ_i^1 is equal to the quantity of symbols preceding $c(x_i)_1$.

The productions from 4-th to 6-th line move symbols θ_i in a transformed word and increase their quantity after each symbol α.

The remaining productions in the sequence try to find $c(x_i)_1$ in the other components of a transformed word or (the last two) prepare a transformed word to be used by the next sequence S_{i+1}.

Now let us construct S_i^2 of the form $S_{i,2}^2, \ldots S_{i,k_i}^2$. Every sequence $S_{i,m}^2$ for $m = 2, \ldots, k_i$ is defined as follows:

$$
\begin{array}{ll}
\lambda_i^{m-1} \alpha \longrightarrow \alpha \lambda_i^{m-1} & \alpha \in \Sigma \cup \{-\} \\
\lambda_i^{m-1} \& \longrightarrow \& \lambda_i^m \gamma_i^m &
\end{array}
$$

[4] Every symbol $\alpha \in \Sigma$ is replaced by respective $\alpha' \in \Sigma'$.

[5] ε is the empty symbol.

$\lambda_i^{m-1} \longrightarrow \eta_i$
$\lambda_i^{m-1}\eta_i \longrightarrow \eta_i$

$\gamma_i^m\lambda_i^m \longrightarrow \lambda_i^m\gamma_i^m$
$\lambda_i^m\alpha\lambda_i^m \longrightarrow \alpha\lambda_i^m\lambda_i^m$
$\lambda_i^m\alpha\gamma_i^m \longrightarrow \alpha\lambda_i^m$
$\gamma_i^m\alpha\gamma_i^m \longrightarrow \alpha\gamma_i^m\gamma_i^m$
$\gamma_i^m\gamma_i^m\alpha \longrightarrow \gamma_i^m\alpha\gamma_i^m$
$\lambda_i^m\gamma_i^m c(x_i)_m \longrightarrow \lambda_i^m(c(x_i)_m)'$ for $m \leq k_i$
$\lambda_i^{k_i}\gamma_i^{k_i}c(x_i)_{k_i} \longrightarrow \lambda_i^{k_i}(c(x_i)_{k_i})'\sigma_i^{k_i}$ for $m = k_i$
$\lambda_i^m\gamma_i^m \longrightarrow \tau_i^m$

$\lambda_i^m\tau_i^m \longrightarrow \tau_i^m$
$\alpha\tau_i^m \longrightarrow \tau_i^m\alpha$
$\&\tau_i^m \longrightarrow \tau^{m-1}\&$
$\alpha'\tau_i^m \longrightarrow \tau_i^m\alpha$
$\alpha\tau_i^{m-1} \longrightarrow \tau_i^{m-1}\alpha$
$\&\tau_i^{m-1} \longrightarrow \tau^{m-2}\&$
$\alpha'\tau_i^{m-2} \longrightarrow \tau_i^{m-2}\alpha$
$\alpha\tau_i^{m-2} \longrightarrow \tau_i^{m-2}\alpha$
$\vdots$
$\&\tau_i^2 \longrightarrow \tau^1\&$
$\alpha'\beta'\tau_i^1 \longrightarrow \alpha'\tau_i^{m-2}\beta$
$\alpha\beta'\tau_i^1 \longrightarrow \alpha\chi_i\beta''$

In the first part of $S_{i,m}^2$ the algorithm changes the symbols λ_i^{m-1} by the symbols λ_i^m at the end (&) of some component of a transformed word. Then the algorithm adds symbols γ_i^m (the quantity of symbols γ_i^m is equal to quantity of λ_i^m). In the last component of transformed word a new symbol η_i is introduced.

In the second part of $S_{i,m}^2$ the algorithm replaces (if it is possible) $c(x_i)_m$ by $(c(x_i)_m)'$, otherwise introduces the symbol τ_i^m. If the algorithm finds the last component $c(x_i)_{k_i}$ of x_i then new symbol $\sigma_i^{k_i}$ is introduced.

In the third part of $S_{i,m}^2$ the symbols τ_i^m are used to replace all symbols of Σ' into symbols of Σ. The first symbol in $c(x_i)_1$ is changed into a symbol of Σ''.

The sequence S_i^3 has the form:

$\alpha\eta_i \longrightarrow \eta_i\alpha$
$\alpha'\eta_i \longrightarrow \eta_i\alpha$
$\alpha''\eta_i \longrightarrow \eta_i\alpha$
$\lambda_i^{k_i}\eta_i \longrightarrow \eta_i$

$\vdots$

$\lambda_i^1 \eta_i \longrightarrow \eta_i$

$\eta_i \longrightarrow \xi_{i+1}\theta_{i+1}$

$\alpha\chi_i \longrightarrow \chi_i\alpha$

$\alpha'\chi_i \longrightarrow \chi_i\alpha$

$\alpha''\chi_i \longrightarrow \chi_i\alpha''$

$\lambda_i^{k_i}\chi_i \longrightarrow \chi_i$

$\vdots$

$\lambda_i^1\chi_i \longrightarrow \chi_i$

$\chi_i \longrightarrow \xi_i\theta_i$

$(c(x_i)_{k_i})'\sigma_i^{k_i} \longrightarrow \sigma_i^{k_i}c(y_i)_{k_i}$

$\vdots$

$(c(x_i)_m)'\sigma_i^m \longrightarrow \sigma_i^m c(y_i)_m$

$\vdots$

$(c(x_i)_1)'\sigma_i^1 \longrightarrow \sigma_i^1 c(y_i)_1$

$\alpha\sigma_i^{k_i} \longrightarrow \sigma_i^{k_i}\alpha$

$\alpha''\sigma_i^{k_i} \longrightarrow \sigma_i^{k_i}\alpha$

$\lambda_i^{k_i}\sigma_i^{k_i} \longrightarrow \sigma_i^{k_i}$

$\vdots$

$\alpha\sigma_i^1 \longrightarrow \sigma_i^1\alpha$

$\alpha''\sigma_i^1 \longrightarrow \sigma_i^1\alpha$

$\lambda_i^1\sigma_i^1 \longrightarrow \sigma_i^1$

$\&\sigma^{k_i} \longrightarrow \sigma^{k_i-1}\&$

$\vdots$

$\&\sigma^2 \longrightarrow \sigma^1\&$

$\sigma_i^1 \longrightarrow \cdot\varepsilon$ if P_i is final

$\sigma_i^1 \longrightarrow \xi_1\theta_1$ if P_i is nonfinal

In this part of S_i the algorithm replaces all occurrences of $(c(x_i)_m)'$ by $c(y_i)_m$ for $m = 1, \ldots, k_i$ If x_i is a subword of a transformed word then algorithm stops if production P_i is final, otherwise algorithm starts with "simulation" of P_1 using symbol $\xi_1\theta_1$. If x_i is not a subword of a transformed word then an algorithm continues "simulation" of P_{i+1} using symbol $\xi_{i+1}\theta_{i+1}$.

Let us add at the end an important remark. In the productions of the form $(c(x_i)_{k_i})' \longrightarrow c(y_i)_{k_i}$, $(c(x_i)_{k_i}) \longrightarrow c(x_i)'_{k_i}$ there can occur some sequences over $\{-\}$. In this case productions $(c(x_i)_{k_i})' \longrightarrow c(y_i)_{k_i}$, $(c(x_i)_{k_i}) \longrightarrow c(x_i)'_{k_i}$ will represent two sequences. These sequences consist of all productions of the above form in which all symbols "–" on the left and right sides of production are replaced by all combination of symbols from Σ, the same on the left and right side of production.

It is clear that the above construction implies equivalence with respect to coding function the algorithms $M = (S_1, \dots, S_n, \alpha \longrightarrow \xi_1\theta_1\alpha), (\alpha \in \Sigma)$ of $\mathcal{MNA}$ and $A = (P_1, \dots, P_n)$ of $\mathcal{MA}^2_{(n,w)}$. ■

Lemma 4.3. *For every Markov normal algorithm M in an alphabet Σ we are able to construct effectively a two–dimensional Markov–like algorithm $A \in \mathcal{MA}^2_{(n,w)}$ over an alpahbet Σ such that for every $v \in \Sigma^*$ $\psi(M(v)) = A(\psi(v))$ unless both algorithms are undefined, where $\psi : \Sigma^* \mapsto (\Sigma \cup \lambda)^*_2$ ($\lambda \notin \Sigma$) is a total funcion, defined as follows (Λ is $\lambda \dots \lambda$ with the same length as v):*

$$\psi(v) = \begin{matrix} v \\ \Lambda \end{matrix}$$

for every $v \in \Sigma^*$.

Proof. Let M be an arbitary Markov normal algorithm with a schema of productions

$P_1 : x_1 \longrightarrow (\cdot)y_1$

$\vdots$

$P_n : x_n \longrightarrow (\cdot)y_n$

Let A be an two-dimensional algorithm with the schema of productions of the form:

$\overline{P_1} : \psi(x_1) \longrightarrow (\cdot)\psi(y_1)$

$\vdots$

$\overline{P_n} : \psi(x_n) \longrightarrow (\cdot)\psi(y_n)$

with a restriction that $\overline{P_i}$ is final iff P_i is final. Then two–dimensional algorithm A is equivalent with respect to coding function ψ to Markov normal algorithm M. ■

As a consequence of two previous lemmas we have the following theorem.

Theorem 4.4. *The classes $\mathcal{MA}^2_{(n,w)}$ and $\mathcal{MNA}$ are equivalent with respect to coding function.*

5. Final remarks. Using the idea of the proof of Lemmas 4.2 and 4.3 one can prove the equivalence of the classes $\mathcal{M}A^2_{(n,e)}$, $\mathcal{M}A^2_{(s,w)}$, $\mathcal{M}A^2_{(s,e)}$ of two-dimensional algorithms with the class $\mathcal{M}NA$ of Markov normal algorithms.

This paper is an initial stage of development related to two-dimensional Markov-like algorithms. Many problems remain open, for example the complexity problem of algorithms and their computations. Another open problem is to study n-dimensional algorithms (not necessarily Markov-like).

References

[1] Asser, G., *Turing Maschinen und Markovsche Algorithmen*, Z. Math. Logik Grundlag. Math. **5** (1959), 326-359.

[2] Church, A., *An unsolvable problem for elementary number theory*, J. of Math. **58** (1936), 345-363.

[3] Cutland, N. J., *Computability and introduction to recursive function theory*, Cambridge University Press, Cambridge-London-New York-Sydney-Melbourne, 1980.

[4] Dietlows W.K., *Equivalence of Markov normal algorithms and recursive functions*, Trudy Mat. Inst. Steklov. IV (1952), 66-69 (Russian).

[5] Grodzki, Z. and J. Mycka, *The equivalence of some classes of algorithms*, Ann. Univ. Mariae Curie-Skłodowska **XLIX, 6** (1995), 85-99.

[6] Kleene, S.C. λ-definability and recursivennes, Duke Math. J. **2** (1936), 340-358.

[7] Markov, A., *The Theory of Algorithms*, Trudy Mat. Inst. Steklov. **XLII** (1954 (Russian)).

[8] Mendelson, E., *Introduction to Mathematical Logic*, Princeton, 1964.

[9] Priese, L., *A note of asynchronous cellular automata*, J. Comput. System Sci. **17** (1978), 237-252.

Institute of Management
and Foundation of Technique
Department of Applied Mathematics
Technical University of Lublin
ul. Bernardyńska 13
20-950 Lublin, Poland

Instytut Matematyki UMCS
Plac Marii Curie-Skłodowskiej 1
20-031 Lublin, Poland

received October 1, 1996