Matrix Languages, Register Machines, Vector Addition Systems

Rudolf Freund[1], Oscar H. Ibarra[2], Gheorghe Păun[3], Hsu-Chen Yen[4]

[1] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9–11, A–1040 Vienna, Austria
E-mail: rudi@emcc.at

[2] Department of Computer Science, University of California
Santa Barbara, CA 93106, USA
E-mail: ibarra@cs.ucsb.edu

[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: gpaun@us.es

[4] Dept. of Electrical Engineering, National Taiwan University
Taipei, Taiwan 106, ROC
E-mail: yen@cc.ee.ntu.edu.tw

Summary. We give a direct and simple proof of the equality of Parikh images of languages generated by matrix grammars with appearance checking with the sets of vectors generated by register machines. As a particular case, we get the equality of the Parikh images of languages generated by matrix grammars without appearance checking with the sets of vectors generated by partially blind register machines. Then, we consider pure matrix grammars (i.e., grammars which do not distinguish terminal and nonterminal symbols), and prove the inclusion of the family of Parikh images of languages generated by such grammars (without appearance checking) in the family of sets of vectors generated by blind register machines, as well as the inclusion of reachability sets of vector addition systems in the family of Parikh images of pure matrix languages. For pure matrix grammars with a certain restriction on the form of matrices, also the converse of the latter inclusion is obtained. Thus, in view of the result from [9], we obtain the semilinearity of languages generated by pure matrix grammars (without appearance checking) with alphabets with at most five letters, with the considered restrictions on the form of matrices. A pure matrix grammar with five symbols, but without restrictions on the form of matrices, is produced which generates a non-semilinear language.

1 Introduction

For representing recursively enumerable sets of (vectors of) natural numbers since long time various models can be found in the literature, and many results are folklore. On the other hand, it seems to us that a direct proof of the equivalence of different models sometimes is missing or at least not present as a folklore result.

Just recently, new motivations came from the area of membrane computing, where in various papers different models were used to establish the computational power of the specific models of P systems under consideration; in many cases, register machines (or counter automata) and matrix grammars with appearance checking were frequently used (see [12]). Most recently, some new results on P systems working in the sequential mode were elaborated, e.g., see [3], [5], and [6], thereby establishing connections to vector addition systems, matrix grammars without appearance checking, and partially blind counter automata, respectively. Interpreting some results proved in [5] and [6] in a suitable way, it turns out that via specific equivalent models of (tissue) P systems with antiport rules working in the sequential mode the equivalence in the generative power of two old (non-universal) models of computation was (re-)established, i.e., the Parikh sets of languages generated by matrix grammars without appearance checking coincide with the sets of vectors of natural numbers generated (or accepted) by partially blind counter automata (or partially blind register machines, respectively).

In view of this quite astonishing coincidence, we are going to give a direct construction of a register machine simulating a matrix grammar and vice versa in Section 3; moreover, as an immediate consequence of these constructions we get a direct simulation of a partially blind register machine by a matrix grammar without appearance checking and vice versa. In Section 4 we investigate the relation of vector addition systems and vector replacement systems with pure matrix grammars and blind register machines.

2 Prerequisites

By $\mathbf{N}$ and $\mathbf{Z}$ we denote the set of natural numbers (non-negative integers) and the set of integers, respectively. We refer to [14], [15], and [17] for the general elements of formal language theory we use here. We only specify that, for a string $x \in V^*$ and a symbol $a \in V$, by $|x|$ we denote the length of x and by $|x|_a$ the number of occurrences of the symbol a in the string x. For $w \in V^*$ with $V = \{a_1, \ldots, a_n\}$, by $\Psi_V(w)$ we denote the Parikh vector of w, i.e., $\Psi_V(w) = (|w|_{a_1}, \ldots, |w|_{a_n})$; this is extended to languages in a natural way. For a family FL of languages, by NFL we denote the family of length sets of language from FL and by $PsFL$ the family of Parikh sets of vectors associated with languages in FL. By CF and RE we denote the families of context-free and of recursively enumerable languages, respectively. Thus, NRE and $PsRE$ are the families of sets of natural numbers and of sets of vectors of natural numbers, respectively, which can be computed by Turing machines.

A *matrix grammar* (with appearance checking) is a construct

$$G = (N, T, S, M, F)$$

where N and T are disjoint alphabets of nonterminal and terminal symbols, $S \in N$ is the start symbol, M is a finite set of sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, of context-free productions over $N \cup T$ (with $A_i \in N$, $x_i \in (N \cup T)^*$, in all cases), and F is a subset of the set of productions occurring in the matrices in M.

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there are a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and strings $w_i \in (N \cup T)^*$, $1 \leq i \leq n+1$, such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \leq i \leq n$, either

(1) $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or

(2) $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F.

The productions of a matrix are applied in the given order, possibly skipping those in F if they cannot be applied – therefore we say that these productions are applied in the *appearance checking* mode. (Note that this directly corresponds to the checking for zero in subtract instructions of register machines.)

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$, where $\Longrightarrow^*$ is the reflexive and transitive closure of the relation $\Longrightarrow$. The family of languages of this form is denoted by MAT_{ac}. If the set F is empty, then the grammar is said to be without appearance checking; the corresponding family of languages is denoted by MAT.

Already at the end of sixties, directly or through the equivalence with programmed grammars of [13], it was proved that $CF \subset MAT \subset MAT_{ac} = RE$, but the problem was formulated whether or not there are one-letter languages in the family MAT which are not regular. The problem was solved only in 1994, in [8], confirming a conjecture from [15] that such languages are regular (this situation resembles the case of one-letter context-free languages, which are also regular).

In our proof given in the next section we will use a well-known normal form for matrix grammars: a matrix grammar $G = (N, T, S, M, F)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and the matrices in M are in one of the following forms:

1. $(S \to XA)$, with $X \in N_1$, $A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1$, $A \in N_2$, $x \in (N_2 \cup T)^*$, $|x| \leq 2$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1$, $A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1$, $A \in N_2$, and $x \in T^*$, $|x| \leq 2$.

Moreover, there is only one matrix of type 1 (that is why we usually write it in the form $(S \to X_0 A_0)$, in order to fix the symbols X, A present in it), and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3; $\#$ is a trap-symbol, because once introduced, it is never removed. A matrix of type 4 is used only once, in the last step of a derivation.

For each matrix grammar there is an equivalent matrix grammar in the binary normal form. Details can be found in [4] and in [15]. The result holds true both for grammars with appearance checking and without appearance checking – in this latter case we no longer have matrices of type 3.

In fact, in the next section we shall use a slightly modified version of this binary normal form called *f-binary normal form* in [5], i.e., instead of the final matrices of type 4 there is only one single final matrix of the form $(f \to \lambda)$, for some special symbol $f \in N_1$.

In general, a grammar is called *pure* if it does not distinguish between terminal and nonterminal symbols. Thus, a *pure matrix grammar* (without appearance checking) is a triple $G = (V, w, M)$, where V is an alphabet, $w \in V^*$ is the axiom (starting string), and M is a finite set of matrices of the form $(a_1 \to u_1, \ldots, a_n \to u_n)$, where $n \geq 1$ and $a_i \in V$, $u_i \in V^*$ for all $1 \leq i \leq n$. The language generated by G is $L(G) = \{z \in V^* \mid w \Longrightarrow^* z\}$, with the derivation relation $\Longrightarrow$ defined as in a non-pure matrix grammar. The family of languages of this form is denoted by $pMAT$. It is known (e.g., see [4]) that $pMAT$ is incomparable with CF and that $pMAT \subset MAT$.

We now introduce register machines, in the generative (non-deterministic) version. Informally speaking, such a device consists of a specified number of counters which can hold any natural number, and which are handled according to a program consisting of labelled instructions. A counter can be increased or decreased by 1 – the decreasing being possible only if the counter holds a number greater than or equal to 1 (we say that it is non-empty) – and it can be checked whether it is empty.

Formally, a (non-deterministic) *register machine* is a device

$$R = (m, B, l_0, l_h, P)$$

where $m \geq 1$ is the number of counters, B is the (finite) set of instruction labels, l_0 is the initial label, l_h is the halting label, and P is the finite set of instructions labelled (hence uniquely identified) by elements from B (P is also called the *program* of the machine). The labelled instructions are of the following forms:

- $l_1 : (\mathtt{ADD}(r), l_2, l_3)$, $1 \leq r \leq m$ (add 1 to counter r and non-deterministically go to one of the instructions with labels l_2, l_3),
- $l_1 : (\mathtt{SUB}(r), l_2, l_3)$, $1 \leq r \leq m$ (if counter r is not empty, then subtract 1 from it and go to the instruction with label l_2, otherwise go to the instruction with label l_3),
- $l_h : \mathtt{HALT}$ (the halt instruction, which can only have the label l_h).

A register machine generates a k-dimensional vector of natural numbers in the following way: we distinguish k counters as output counters (without loss of generality, these can be the first k counters), and we start computing with all m counters being empty, from the instruction labelled by l_0; if the computation reaches the instruction $l_h : \mathtt{HALT}$ (we say that it halts), with all counters $k+1, \ldots, m$ being empty, then the values of counters $1, 2, \ldots, k$ form the vector generated by this computation. The set of all vectors from $\mathbf{N}^k$ generated in this way by R is denoted by $Ps(R)$. If we want to generate only numbers (1-dimensional vectors), then we have the result of a computation in counter 1, and the set of numbers computed by R in this way is denoted by $N(R)$.

By NRM and $PsRM$ we denote the families of sets of natural numbers and of sets of vectors of natural numbers, respectively, generated by register machines. It is known (e.g., see [11], [17]) that $PsRE = PsRM$ and $NRE = NRM$ (actually, three counters suffice in order to generate any set from the family NRE, but this detail is not of special interest for what follows).

In the case when a register machine cannot check whether a counter is empty we say that it is *partially blind*: the counters are increased and decreased by one as usual, but if the machine tries to subtract from an empty counter, then the computation aborts without producing any result (that is to say that the subtract instructions are of the form $l_1 : (\mathtt{SUB}(r), l_2, abort)$). Note that there is an implicit test for zero, at the end of a (successful) computation, where all counters $k+1$, $\dots$, m should be empty, that is why we say that the device is *partially* blind. It is known (e.g., see [7]) that partially blind register machines (counter automata) are strictly less powerful than general register machines (hence than Turing machines). By $NPBRM$ and $PsPBRM$ we denote the families of sets of natural numbers and of sets of vectors of natural numbers, respectively, computed by partially blind register machines.

If a partially blind register machine accepts the value of all counters $1, 2, \dots, m$, then this register machine is said to be *blind* (no counter is checked for zero during the computations or when reaching the final label l_h). By $NBRM$ and $PsBRM$ we denote the families of sets of natural numbers and of sets of vectors of natural numbers, respectively, generated by blind register machines.

A **technical detail**:

In the proofs of Theorems 1 and 2, we will construct register machines of a slightly more relaxed form than defined above, i.e., the same label $l_1 \in B$ will be allowed to be assigned to several instructions; when such a label is introduced by another instruction, then in the next step any of the instructions labelled by l_1 can be executed, non-deterministically chosen. However, it is easy to pass from such a "super-non-deterministic" register machine to a usual non-deterministic register machine: For instance, assume that there are two instructions

- $l_1 : (\mathtt{op}'(r'), l_2', l_3')$ and
  $l_1 : (\mathtt{op}''(r''), l_2'', l_3'')$
  with the same label l_1 ($\mathtt{op}'$ and $\mathtt{op}''$ are any operations ADD or SUB). We can replace these instructions by the following ones:
- $l_1 : (\mathtt{ADD}(1), l_1', l_1'')$,
  $l_1' : (\mathtt{SUB}(1), l_1''', l_1''')$,
  $l_1'' : (\mathtt{SUB}(1), l_1^{iv}, l_1^{iv})$,
  $l_1''' : (\mathtt{op}'(r'), l_2', l_3')$, and
  $l_1^{iv} : (\mathtt{op}''(r''), l_2'', l_3'')$,
  where $l_1', l_1'', l_1''', l_1^{iv}$ are new labels.

In this way, the number of instructions labelled with l_1 has been decreased by one and no new pair of instructions having the same label was introduced. Continuing this way, we can eliminate all duplicate labelling.

A $k \times m$ *vector replacement system (VRS),* see [10], is a triple (w_0, U, W), where $w_0 \in \mathbf{N}^k$ *(start vector)*, $U \in \mathbf{N}^{k\times m}$ *(check matrix)*, and $W \in \mathbf{Z}^{k\times m}$ *(addition matrix)* such that, for any i, j with $1 \leq i \leq k$ and $1 \leq j \leq m$, we have $U_i(j) + W_i(j) \geq 0$. A vector $W_i \in W$ is said to be *enabled* in a vector $x \in \mathbf{N}^k$ if and only if $x \geq U_i$; as $U_i + W_i \geq 0$, adding W_i to x yields $x + W_i \in \mathbf{N}^k$. For a VRS $\gamma = (w_0, U, W)$, $Ps(\gamma)$ denotes the sets of vectors from $\mathbf{N}^k$ that can be reached from w_0 by iteratively adding vectors from W enabled in the vector computed so far. By VRS we denote the family of all sets of vectors of natural numbers generated by vector replacement systems in that way.

A *vector addition system (VAS)* is a pair (w_0, W), where $w_0 \in \mathbf{N}^k$ *(start vector)* and $W \in \mathbf{Z}^{k\times m}$ *(addition matrix)*; for a vector addition system $\gamma = (w_0, W)$, by $Ps(\gamma)$ we denote the set of vectors reachable in γ from w_0 by iteratively adding vectors from W in such a way that the resulting vectors always are in $\mathbf{N}^k$. The family of all sets of vectors of natural numbers generated in that way by vector addition systems is denoted by VAS. From [9] it is known that the sets of vectors form VAS generated by vector addition systems of dimension $k \leq 5$ are semilinear, while vector addition systems of dimension six can already produce non-semilinear sets of vectors. Some counterparts of these results for pure matrix languages will be given in Section 4.

3 The Equivalence of Register Machines and Matrix Grammars

We now elaborate direct constructions for showing the equivalence of register machines and matrix grammars as already mentioned in the Introduction, in its general form.

Theorem 1. $PsRM = PsMAT_{ac}$.

Proof. **Inclusion $PsRM \subseteq PsMAT_{ac}$:**

Let us consider a register machine $R = (m, B, l_0, l_h, P)$ with m counters, meant to generate a set $Ps(R) \subseteq \mathbf{N}^k$, for some k with $1 \leq k \leq m$. We construct the matrix grammar $G = (N, T, S, M, F)$ where

$$\begin{aligned}
N &= \{A_i \mid 1 \leq i \leq m\} \cup B \cup \{\#\},\\
T &= \{a_i \mid 1 \leq i \leq k\},\\
S &= l_0,\\
M &= \{(l_1 \to A_r l_2),\\
&\qquad (l_1 \to A_r l_3) \mid l_1 : (\mathtt{ADD}(r), l_2, l_3) \in P\}
\end{aligned}$$

$$\begin{aligned}
&\cup \{(l_1 \to l_2, A_r \to \lambda),\\
&\quad (l_1 \to l_3, A_r \to \#) \mid l_1 : (\texttt{SUB}(r), l_2, l_3) \in P\}\\
&\cup \{(l_h \to l_h, A_i \to a_i) \mid 1 \le i \le k\}\\
&\cup \{(l_h \to \lambda)\},
\end{aligned}$$

and the set F consists of all productions of the form $A_r \to \#$, $1 \le r \le m$, from the matrices of M.

The equality $Ps(R) = \Psi_T(L(G))$ is obvious: a computation in R halts correctly only after reaching the instruction l_h : HALT, with all registers $k+1, \ldots, m$ being empty, and this corresponds to the fact that a derivation in G reaches a terminal string only after introducing the nonterminal symbol l_h, in the presence of which the nonterminal symbols A_i are transformed into the terminal symbols a_i, $1 \le i \le k$ (if any nonterminal symbol A_j, $j > k$, is still present, then it cannot be removed, hence, the sentential form cannot be turned into a terminal one).

Inclusion $PsRM \supseteq PsMAT_{ac}$:

Now consider a matrix grammar $G = (N, T, S, M, F)$ in the f-binary normal form, i.e., with $N = N_1 \cup N_2 \cup \{S, \#\}$, and the matrices only being of the four forms specified in the previous section.

Without any loss of generality (because we are interested in the Parikh image of the language $L(G)$ only), we can assume that $T = \{a_1, \ldots, a_k\}$ and $N_2 = \{A_{k+1}, \ldots, A_s\}$, for some k, s with $1 \le k < s$. Hence, by α_j, $1 \le j \le s$, we denote a symbol from $T \cup N_2$ with the understanding that $\alpha_i = a_i$ for $1 \le i \le k$ and $\alpha_i = A_i$ for $k+1 \le i \le s$. Let us suppose all matrices of types 2 from M to be labelled in a one-to-one manner by $m_1, \ldots, m_n$.

We then construct the register machine $R = (s, B, l_0, l_h, P)$ where

$$\begin{aligned}
B = &\{S, \#\} \cup N_1\\
&\cup \{\langle m_i, 1\rangle \mid m_i : (X \to \beta, A \to x) \in M, \beta \in N_1 \cup \{\lambda\}, 1 \le i \le n,\\
&\quad \text{with } |x| = 1\}\\
&\cup \{\langle m_i, 1\rangle, \langle m_i, 2\rangle \mid m_i : (X \to \beta, A \to x) \in M, \beta \in N_1 \cup \{\lambda\}, 1 \le i \le n,\\
&\quad \text{with } |x| = 2\},\\
l_0 = &\ S,\\
P = &\{S : (\texttt{ADD}(r), X, X) \mid (S \to XA_r) \in M, \text{ for some } k+1 \le r \le s\}\\
&\cup \{X : (\texttt{SUB}(r), Y, \#) \mid (X \to Y, A_r \to \lambda) \in M, \text{ for some } k+1 \le r \le s\}\\
&\cup \{X : (\texttt{SUB}(r), \langle m_i, 1\rangle, \#),\\
&\quad \langle m_i, 1\rangle : (\texttt{ADD}(j), Y, Y) \mid m_i : (X \to Y, A_r \to \alpha_j) \in M\\
&\quad \text{for some } k+1 \le r \le s \text{ and } 1 \le j \le s\}\\
&\cup \{X : (\texttt{SUB}(r), \langle m_i, 1\rangle, \#),\\
&\quad \langle m_i, 1\rangle : (\texttt{ADD}(j_1), \langle m_i, 2\rangle, \langle m_i, 2\rangle),\\
&\quad \langle m_i, 2\rangle : (\texttt{ADD}(j_2), Y, Y) \mid m_i : (X \to Y, A_r \to \alpha_{j_1}\alpha_{j_2}) \in M,
\end{aligned}$$

$$\text{for some } k+1 \leq r \leq s \text{ and } 1 \leq j_1, j_2 \leq s\}$$
$$\cup \{X : (\texttt{SUB}(r), \#, Y) \mid (X \to Y, A_r \to \#) \in M, \text{ for some } k+1 \leq r \leq s\}$$
$$\cup \{f : \texttt{HALT}\}$$
$$\cup \{\# : (\texttt{ADD}(s), \#, \#)\}.$$

The matrices of G are simulated by the register machine R as follows: the nonterminal symbols from N_1 are labels of instructions. A rule $A_r \to \alpha_{j_1}\alpha_{j_2}$ is simulated by first subtracting one from register r and then adding one to each of the registers j_1, j_2 (with the cases when one or two of the symbols $\alpha_{j_1}, \alpha_{j_2}$ is missing being still simpler); these steps are controlled by the labels $\langle m_i, 1\rangle, \langle m_i, 2\rangle$, and they start in the presence of label X and end with introducing the label Y, thus completing the simulation of the matrix $m_i : (X \to Y, A_r \to \alpha_{j_1}\alpha_{j_2})$.

The matrices $(X \to Y, A \to \#)$ of type 3 are directly simulated by a subtract instruction $X : (\texttt{SUB}(r), \#, Y)$. Entering label $\#$ (the "trap") leads to an infinite loop with $\# : (\texttt{ADD}(s), \#, \#)$.

The simulation of the terminal matrix $(f \to \lambda)$ introduces the label f of the halt instruction. If the derivation in G has been a terminal one, then all registers $k+1, \ldots, s$ are empty at that moment, hence, the computation in R ends correctly. Thus, the equality $\Psi_T(L(G)) = Ps(R)$ follows, and observing the technical detail from the previous section concerning the elimination of duplicate labels concludes the proof. □

If we start from a partially blind register machine, then the matrices of the form $(l_1 \to l_3, A_r \to \#)$ can be omitted in the construction of the matrix grammar from the first part of the previous proof, hence, we obtain a matrix grammar without appearance checking.

Conversely, if we start the second part of the proof from a matrix grammar without appearance checking, then the register machine we obtain will be partially blind: the only subtract instructions $l_1 : (\texttt{SUB}(r), l_2, l_3)$ having the label l_3 different from the trap $\#$ are those corresponding to matrices of type 3 – which now are missing.

Consequently, from the proof of Theorem 1 we immediately infer the following result:

Corollary 1. $PsMAT = PsPBRM$ *and* $NMAT = NPBRM$.

4 Pure Matrix Grammars and Vector Addition Systems

The proof of Theorem 1 cannot be repeated for pure matrix grammars, because we cannot use any auxiliary symbol, but some counterparts of it can be obtained in this case, too.

Since a (pure) matrix grammar with λ as the starting string can only generate the language $\{\lambda\}$, in our subsequent discussion we assume that the VAS under consideration starts with a non-zero initial vector. More about this will be said later.

Theorem 2. *$VAS \subset PspMAT \subseteq PsBRM$.*

Proof. (Sketch) Given a vector addition system (with vectors of dimension $k \geq 1$) $\gamma = (w_0, W)$, we construct a pure matrix grammar $G = (V, w, M)$ with $V = \{a_1, \ldots, a_k\}$, $\Psi_V(w) = w_0$, and with the matrices in M associated with vectors $W_i \in W$ constructed as follows: for a vector W_i, we consider the set of matrices obtained by examining each component $W_i(j)$ of the vector and proceeding as follows: if $W_i(j)$ is positive, then we take a rule $a_q \to a_q a_j^{W_i(j)}$, for some $1 \leq q \leq k$, otherwise we have to take $W_i(j)$ rules of the form $a_j \to \lambda$. Thus, for the same vector W_i we get a set of matrices, obtained by combining all possibilities to take the rules $a_q \to a_q a_j^{W_i(j)}$ for those components $W_i(j)$ which are positive. The idea is that we have to add $W_i(j)$ occurrences of a_j, and to this aim we need a rewriting rule $a_q \to u$ for some symbol a_q which is already present in the string.

As an important additional constraint, we can suppose that we always have at least one symbol in the string – except when generating the empty string. More precisely, in the vector addition system γ we may suppose that each "computation" to a vector v different from $(0, 0, \ldots, 0)$ never passes through the vector $(0, 0, \ldots, 0)$. Indeed, if to a vector v' we add a vector W_i, $1 \leq i \leq t$, such that $v' + W_i = (0, 0, \ldots, 0)$, and then we add W_j, we immediately infer that all components of (the non-zero vector) W_j must be non-negative, hence, we can commute W_i and W_j, thus computing $(v' + W_j) + W_i = W_j$ instead of $(v' + W_i) + W_j$; in that way, $(0, 0, \ldots, 0)$ can be avoided avoided as an immediate result. Hence, to the matrices constructed so far, we have to add all matrices associated (in the sense of the preceding arguments) with vectors $W_j + W_i$ where W_j is a positive vector.

In this way, using one of the matrices associated as above with a vector W_i or $W_j + W_i$ exactly corresponds to the addition of the associated vector in the VAS. Thus, $Ps(\gamma) = \Psi_V(L(G))$; hence, we have proved the inclusion $VAS \subseteq PspMAT$.

Now we show the strictness of this inclusion, i.e., $VAS \neq PspMAT$: Consider the following pure matrix grammar $G_1 = (V, aab, M)$ where

$$\begin{aligned} V &= \{a, b\}, \\ M &= \{(a \to \lambda, a \to a, b \to \lambda), (a \to \lambda, a \to a)\}. \end{aligned}$$

Clearly $\Psi_V(L(G_1)) = \{(|w|_a, |w|_b) \mid aab \Longrightarrow^* w\} = \{(2, 1), (1, 0), (1, 1)\}$. We now show that no VAS can have this set as its reachability set. Assume, on the contrary, that there were a VAS $\gamma = (v_0, V)$ with $Ps(\gamma)$=$\{(2, 1), (1, 0), (1, 1)\}$. First note that $v_0 \neq (1, 0)$ and $v_0 \neq (1, 1)$, because otherwise, $(1, 0) \Longrightarrow^* (2, 1)$ or $(1, 0) \Longrightarrow^* (1, 1)$, respectively, thus yielding infinite reachability sets. As a consequence, $v_0 = (2, 1)$. Consider two cases:

1. Case 1: $(2, 1) \Longrightarrow (1, 0)$. Then $(-1, -1) \in X$ and therefore $(1, 1) + (-1, -1) \in Ps(\gamma)$ – a contradiction.
2. Case 2: $(2, 1) \to (1, 1)$. Then $(-1, 0) \in X$ and therefore $(1, 0) + (-1, 0) \in Ps(\gamma)$ – a contradiction.

Hence, we have proved $VAS \neq PspMAT$.

Starting from a pure matrix grammar $G = (V, w, M)$ with $V = \{a_1, \ldots, a_k\}$, we now construct a blind register machine $R = (k, B, l_0, l_h, P)$ as follows: for each matrix $(r_1 : a_{i_1} \to x_1, \ldots, r_n : a_{i_n} \to x_n)$ from M, we simulate the use of the rules $r_1, \ldots, r_n$, in this order, under the control of suitable labels of instructions from P; then, each rule $r_j : a_{i_j} \to a_{s_1} a_{s_2} \ldots a_{s_{q_j}}$ is simulated like in the second part of the proof of Theorem 1, by first subtracting one from the register associated with the symbol a_{i_j} (aborting the computation if this is not possible) and then adding one to all registers associated with the symbols $a_{s_1} a_{s_2} \ldots a_{s_{q_j}}$. Again, the control of the correct sequencing of these operations is ensured by the labels of the instructions. After finishing the simulation of a matrix, we non-deterministically pass to the simulation of any other matrix, or to the halting instruction (because the grammar is pure, any sentential form is accepted). This construction clearly produces a blind register machine which generates the Parikh set of the language $L(G)$.

As described before, the construction leads to a "super-non-deterministic" register machine, but we can pass to a usual register machine as explained in Section 2. Consequently, we have proved the second inclusion from the theorem, too. □

It should be noted that for $VAS \subseteq PspMAT$ to hold, we must exclude those VAS with $(0, \ldots, 0)$ as their initial vectors. To see this, consider the following VAS $A = \{(0,0), \{(1,1), (2,3)\}\}$. Clearly, the reachability set of A consists of those vectors satisfying $i * (1,1) + j * (2,3)$, $i, j \geq 0$. Now suppose there is another VAS $B = (w_B, W_B)$ with the same reachability set, i.e., $Ps(B) = Ps(A)$, such that $w_B \neq (0,0)$. Suppose $w_B = s * (1,1) + t * (2,3)$ for some $s, t \geq 0$. Consider the following cases:

1. $s = 0$, $t > 0$: then $t * (2,3) \overset{\sigma}{\Longrightarrow} (0,0)$ for some sequence σ, because $(0,0)$ is in the reachability set. Now consider the reachable vector $4*(1,1)+(t-1)*(2,3)$. As $4 * (1,1) + (t-1) * (2,3) = ((2,1) + t * (2,3)) \overset{\sigma}{\Longrightarrow} (2,1)$, $(2,1)$ is in the reachability set – a contradiction.
2. $t = 0$, $s > 0$: then $s * (1,1) \overset{\sigma}{\Longrightarrow} (0,0)$ for some sequence σ, because $(0,0)$ is in the reachability set. Now consider the reachable vector $(s-1)*(1,1)+(2,3)$. As $(s-1)*(1,1)+(2,3) = (s*(1,1)+(1,2)) \overset{\sigma}{\Longrightarrow} (1,2)$, $(1,2)$ is in the reachability set – a contradiction.
3. $s > 0, t > 0$: then $s * (1,1) + t * (2,3) \overset{\sigma}{\Longrightarrow} (0,0)$ for some sequence σ, because $(0,0)$ is in the reachability set. Now consider the reachable vector $(s-1)*(1,1)+(t+1)*(2,3)$. As $(s-1)*(1,1)+(t+1)*(2,3) = (s*(1,1)+t*(2,3)+(1,2)) \overset{\sigma}{\Longrightarrow} (1,2)$, $(1,2)$ is in the reachability set – a contradiction.

In view of the considerations above, no such B can exist. In a similar way, we can prove that no pure matrix grammar with a non-λ initial string can generate the reachability set $Ps(A)$ of A.

Lemma 1. *$PspMAT = VRS$.*

Proof. (Sketch) The proof of $VRS \subseteq PspMAT$ is along a similar line as showing $VAS \subseteq PspMAT$ in Theorem 2, with the following modification: let (w_0, U, W) be a VRS; then, for each vector $U_i \in U$, the matrix simulating the addition of the corresponding vector $W_i \in W$ is of the form

$$(\ldots, (a_i \to \lambda,)^{U_i(j)-1} a_j \to (a_j)^{U_i(j)} \ldots, \text{ simulation of } W_i \quad \ldots).$$

Now we show $PspMAT \subseteq VRS$: for each matrix $(a_1 \to x_1, \ldots, a_n \to x_n)$, we let c_j be the minimum number of symbols a_j needed for the matrix to be applicable, and we let d_j be the net effect regarding a_j when the matrix is applied. From $(a_1 \to x_1, \ldots, a_n \to x_n)$, c_j and d_j, $1 \leq j \leq n$, can easily be computed. The associated check vector U_i and addition vector W_i are obtained as $U_i(j) = c_j$ and $W_i(j) = d_j$, $1 \leq j \leq n$. □

Let us call a matrix $(a_1 \to x_1, \ldots, a_n \to x_n)$, of a pure matrix grammar G, *separated* if, for each $1 \leq j \leq n$, a_j does not appear in x_j, and the rules use mutually disjoint sets of symbols (formally, $alph(a_j x_j) \cap alph(a_k x_k) = \emptyset$ for all $1 \leq j < k \leq n$). By $SpMAT$ we then denote the family of languages generated by separated pure matrix grammars.

Starting from a grammar G with all matrices being separated, it is obvious that a vector addition system can be constructed which reaches the same vectors as the Parikh images of strings in $L(G)$ (for each rule $a_j \to x_j$ we have a component with -1 corresponding to a_j and corresponding positive components for the symbols appearing in x_j).

Conversely, consider the following VAS $C = \{(1), \{1\}\}$, whose reachability set is $\mathbf{N} - \{0\}$: as in a separated pure matrix grammar every $a \to x$ with a not occurring in x must have $x = \lambda$, such a grammar clearly cannot generate the reachability set of the VAS C.

As a consequence of these considerations, we have the following result:

Lemma 2. *$PsSpMAT \subset VAS$.*

As an immediate consequence of Theorem 2 as well as Lemmas 1 and 2, we have the following corollary:

Corollary 2. *$PsSpMAT \subset VAS \subset PspMAT = VRS \subseteq PsBRM$.*

The corresponding result in [9] immediately yields the following:

Corollary 3. *The separated pure matrix grammars with at most five symbols generate semilinear languages.*

A natural question now arises concerning the semilinearity of languages generated by pure matrix grammars which are not separated. How many symbols do they need in order to generate non-semilinear languages? We here give a partial answer to this question by providing a pure matrix grammar with five symbols that generates a non-semilinear language. The proof is based on an old example of a non-pure matrix grammar generating a non-semilinear language over an alphabet with two letters, [16] (see also [4]). That language (whose non-semilinearity is easy to prove) is

$$L = \{a^n b^m \mid 1 \leq n < m \leq 2^n\}.$$

Here consider the following pure matrix grammar:

$$\begin{aligned}
G &= (V, a_0 b, M), \text{ where} \\
V &= \{a_0, a_1, a, b, b'\}, \\
M &= \{(a_0 \to a_0, b \to b'b'), \\
&\quad (a_0 \to aa_1, b \to b'b'), \\
&\quad (a_1 \to a_1, b' \to b), \\
&\quad (a_1 \to a_0, b' \to b), \\
&\quad (a_1 \to a)\}.
\end{aligned}$$

Assume that we have a sentential form $a_0 x$, with $x \in \{b, b'\}^*$; initially, $x = b$. In the presence of symbol a_0, by using the matrix $(b \to b'b')$, we can double any number of occurrences of b (introducing primed versions of b). At least one b is replaced by $b'b'$ when passing from a_0 to a_1 – at that time, also a copy of a is introduced. In the presence of a_1, we can remove the primes of symbols b; at some step, the matrix $(a_1 \to a_0, b' \to b)$ should be used, hence the doubling of the number of occurrences of b can be repeated. Instead of this matrix we can also use the matrix $(a_1 \to \lambda)$, which closes the derivation, because all matrices need either a_0 or a_1 in order to be applied.

Anyway, each cycle of doubling the number of occurrences of b corresponds to introducing one copy of a; in each cycle we can double all occurrences of b, or less, but at least one b is replaced by $b'b'$. Consequently,

$$\Psi_V(L(G)) \cap \{(0,0,i,j,0) \mid i,j \geq 1\} = \{(0,0,n,m,0) \mid 1 \leq n < m \leq 2^n\},$$

which is not a semilinear set. As the family of semilinear sets of vectors is closed under intersection, it follows that $\Psi_V(L(G))$ is not semilinear.

We now state this result as a theorem, too; moreover, we also pose as an *open problem* whether this result is optimal, i.e., whether we really need five symbols for generating a non-semilinear language by a pure matrix grammar:

Theorem 3. *Pure matrix grammars with five symbols can generate non-semilinear languages.*

Note that the previous grammar is not separated, because of the rules $a_0 \to a_0$ and $a_1 \to a_1$, but we can easily construct an equivalent separated grammar: we replace the matrix $(a_0 \to a_0,\, b \to b'b')$ by the two matrices $(a_0 \to a_0', b \to b'b')$ and $(a_0' \to a_0)$ and the matrix $(a_1 \to a_1, b' \to b)$ by the two matrices $(a_1 \to a_1', b' \to b)$ and $(a_1' \to a_1)$. In this way, we get a separated grammar which unfortunately now has seven symbols, i.e., we need one symbol more than it was needed in the non-semilinearity result from [9].

References

1. S. Abraham: Some questions of phrase-structure grammars. *Computational Linguistics*, 4 (1965), 61–70.
2. H.K. Büning, T. Lettmann, E.W. Mayr: Projections of vector addition system reachability sets are semilinear. *Theoretical Computer Science*, 64 (1989), 343–350 (Technical Report No. STAN-CS-88-1199 of Stanford University, March 1988).
3. Z. Dang, O. H. Ibarra: On P systems operating in sequential mode. In *Pre-proceedings of the Workshop Descriptional Complexity of Formal Systems (DCFS) 2004* (L. Ilie, D. Wotschke, eds.), Report No. 619, Univ. of Western Ontario, London, Canada (2004), 164–177.
4. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, 1989.
5. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue P systems with channel states. *Theoretical Computer Science*, 330 (2205), 101–116.
6. P. Frisco: About P systems with symport/antiport. In *Second Brainstorming Week on Membrane Computing, Sevilla, Spain, Feb. 2-7, 2004* (Gh. Păun, A. Riscos–Nuñez, A. Romero–Jiménez, F. Sancho–Caparrini, eds.), Techn. Report 01/2004, Research Group on Natural Computing, Sevilla University, 224–236.
7. S.A. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7 (1978), 311–324.
8. D. Hauschild, M. Jantzen: Petri nets algorithms in the theory of matrix grammars. *Acta Informatica*, 8 (1994), 719–728.
9. J.E. Hopcroft, J.J. Pansiot: On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8 (1979), 135–159.
10. R. Keller: Vector replacement systems: A formalism for modelling asynchronous systems. Tech. Rept. 117, Computer Science Lab., Princeton Univ. 1972.
11. M.L. Minsky. *Computation. Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.
12. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, Berlin, 2002.
13. D.J. Rosenkrantz: Programmed grammars and classes of formal languages. *Journal of the ACM*, 16 (1969), 107–131.
14. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1987.
15. A. Salomaa: *Formal Languages.* Academic Press, New York, 1973.
16. E.D. Stotskij: Remarks on a paper by M.K. Levitina. *NTI*, Ser. 2, No. 4 (1972), 40–45.
17. D. Wood: *Theory of Computation.* Harper and Row, New York, 1987.