Semilinear Sets, Register Machines, and Integer Vector Addition (P) Systems

Artiom Alhazov[1], Omar Belingheri[2], Rudolf Freund[3],
Sergiu Ivanov[4], Antonio E. Porreca[2], and Claudio Zandron[2]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: artiom@math.md
[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
E-mail: {o.belingheri@campus,porreca@disco,zandron@disco}.unimib.it
[3] Faculty of Informatics, TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria
E-mail: rudi@emcc.at
[4] Université Paris Est, France
E-mail: sergiu.ivanov@u-pec.fr

Summary. In this paper we consider P systems working with multisets with integer multiplicities. We focus on a model in which rule applicability is not influenced by the contents of the membrane. We show that this variant is closely related to blind register machines and integer vector addition systems. Furthermore, we describe the computational power of these models in terms of linear and semilinear sets of integer vectors.

1 Introduction

P systems have been traditionally viewed as hierarchical processors of multisets [11]. In the list of open problems disseminated in 2015 [10], Gheorghe Păun suggested going beyond the traditional setting and considering multisets in which objects would not be restricted to having natural multiplicities. Several possible approaches have been suggested since then, including the one from [3], which defines generalised multisets as taking multiplicities from finitely generated, totally ordered Abelian groups.

The work [1] takes a different approach — the objects of the P system are partitioned into two classes: regular objects, which may have any integer multiplicity, and "catalysts", which may only appear in a bounded number of copies and cannot be consumed without being immediately reproduced. Thus, the regular objects cannot influence the applicability of rules, while the always bounded

catalysts induce a finite set of "rule teams" which can be applied in parallel in one step. The virtual absence of applicability conditions and the finiteness of the "teams" hints at the possibility of seeing them as integer vectors; in this case the P system itself can be seen as evolving by sequentially adding such vectors to the contents of its membranes.

Even though this vision is quite reminiscent of the folklore vector addition systems (VAS), the latter model is actually limited to having natural vectors as configurations [2, 8]. On the other hand, P systems manipulating integer multisets allow symbols with negative multiplicities to appear. It turns out that vector addition systems *without* the limitation of having natural configurations (integer VAS) have received relatively little attention in the literature [7].

Another related model which has received notoriously little attention are the blind register machines, whose registers are allowed to range over the whole set of integers. Blind counter automata have been introduced and studied as string recogniser devices by Sheila Greibach in [6]; their adaptation to recognising vectors of integer numbers seems quite relevant to the study of multisets with integer multiplicities.

In the present work we bring together the three models — P systems over integer multisets as defined in [1], integer vector addition systems, and blind register machines — and formally show the connections between their different variants. We also give detailed characterisations of their computing power in terms of linear and semilinear sets of natural and integer vectors.

The article is structured as follows. Section 2 recalls some notions used throughout the paper, in particular semilinear sets and vector addition systems. Section 3 gives a general definition of a register machine over a set A, and then defines blind, partially blind, and conventional register machines within this general framework. Section 4 defines the model of integer vector addition P systems and gives some details as to their semantics. Section 5 investigates the power of blind register machines and gives characterisations in terms of semilinear sets of vectors. Finally, Section 6 studies the power integer vector addition systems with and without membranes, and compares different variants of the models between themselves and with blind register machines.

2 Preliminaries

The reader is assumed to be familiar with the basic notions of formal languages and membrane computing; see [12] for a comprehensive introduction to both.

2.1 Linear Sets

The $\mathbb{N}$-*linear* set of $\mathbb{Z}$-vectors (or just *linear set* of $\mathbb{Z}$-vectors) generated by a set of vectors $A = \{\mathbf{a}_1, \cdots, \mathbf{a}_d\} \subseteq \mathbb{Z}^n$ and an offset $\mathbf{a}_0 \in \mathbb{Z}^n$ is defined as follows:

$$\langle A, \mathbf{a}_0 \rangle_{\mathbb{N}} = \left\{ \mathbf{a}_0 + \sum_{i=1}^{d} k_i \mathbf{a}_i \;\middle|\; k_i \in \mathbb{N},\ 1 \leq i \leq d \right\}.$$

We underline that the vectors are over $\mathbb{Z}$, but the coefficients are from $\mathbb{N}$ (we will also consider the special case when $A \cup \{a_0\} \subseteq \mathbb{N}^n$; this would be an $\mathbb{N}$-*linear* set of $\mathbb{N}$-vectors, or just a linear set, a well-known concept from Formal Language Theory).

A $\mathbb{Z}$-linear set of $\mathbb{Z}$-vectors

$$\langle A, \mathbf{a}_0 \rangle_{\mathbb{Z}} = \left\{ \mathbf{a}_0 + \sum_{i=1}^{d} k_i \mathbf{a}_i \mid k_i \in \mathbb{N},\ 1 \leq i \leq d \right\}$$

can be considered, too. It corresponds precisely to the *linear vector space* notion from the classic course of Linear Algebra. However, it is also a particular case. Indeed, it is easy to see that $\langle A, \mathbf{a}_0 \rangle_{\mathbb{Z}} = \langle B, \mathbf{a}_0 \rangle_{\mathbb{N}}$ for

$$B = \{\mathbf{a}_1, \cdots, \mathbf{a}_d, -\mathbf{a}_1, \cdots, -\mathbf{a}_d\}.$$

If the offset $\mathbf{a}_0$ is the zero vector, we call the corresponding linear set *homogeneous*.

A *positive-restricted* $\mathbb{N}$-linear set of $\mathbb{Z}$-vectors generated by A and an offset $\mathbf{a}_0$ is defined to be the $\mathbb{N}$-linear set of $\mathbb{Z}$-vectors generated by A, restricted to non-negative vectors only:

$$\langle A, a_0 \rangle_{\mathbb{Z}}^{+} = \{\mathbf{x} \in \langle A, a_0 \rangle_{\mathbb{Z}} \mid \mathbf{x} \geq 0\},$$

where $\mathbf{x} \geq 0$ means that every component of $\mathbf{x}$ is non-negative.

We will use the notations $\mathbb{Z}^n LIN_{\mathbb{N}}$, $\mathbb{N}^n LIN_{\mathbb{N}}$, $\mathbb{Z}^n LIN_{\mathbb{Z}}$, and $\mathbb{Z}_+^n LIN_{\mathbb{N}}$ to refer to the classes of all $\mathbb{N}$-linear sets of $\mathbb{Z}$-vectors, $\mathbb{N}$-linear sets of $\mathbb{N}$-vectors, $\mathbb{Z}$-linear sets of $\mathbb{Z}$-vectors, and positive restricted $\mathbb{N}$-linear sets of $\mathbb{Z}$-vectors of dimension n, correspondingly. Semilinear sets are defined as finite unions of the corresponding types of linear sets. We will use the notations $\mathbb{Z}^n SLIN_{\mathbb{N}}$, $\mathbb{N}^n SLIN_{\mathbb{N}}$, $\mathbb{Z}^n SLIN_{\mathbb{Z}}$ and $\mathbb{Z}_+^n SLIN_{\mathbb{N}}$ to refer to the families of $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors, $\mathbb{N}$-semilinear sets of $\mathbb{N}$-vectors, $\mathbb{Z}$-semilinear sets of $\mathbb{Z}$-vectors, a nd positive-restricted $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors of dimension n, respectively. In case no particular restriction is imposed on the dimension, n will be replaced by $*$. We may omit n if $n = 1$.

We recall the following general result from number theory known as *Bézout's identity*. Given a set of integers $A = \{a_1, \cdots, a_n\} \subseteq \mathbb{Z}$, there exist integers $x_1, \cdots, x_n \in \mathbb{Z}$ such that the following holds:

$$\sum_{i=1}^{n} x_i a_i = \gcd(a_1, \cdots, a_n),$$

where $\gcd(a_1, \ldots, a_n)$ in the greatest common divisor of the integers from A. Furthermore, the greatest common divisor is the smallest positive integer which can be obtained as a linear combination of the elements of A.

2.2 Vector Addition Systems

A *vector addition system* (VAS) of dimension $n \in \mathbb{N}$ is defined to be the pair $(\mathbf{w}_0, W)$, where $\mathbf{w}_0 \in \mathbb{N}^n$ is the start vector, and W is a finite set of vectors from $\mathbb{Z}^n$, called addition vectors. An addition vector $\mathbf{w} \in W$ is said to be enabled in a vector $\mathbf{x} \in \mathbb{N}^n$ if $\mathbf{x} + \mathbf{w} \in \mathbb{N}^n$, i.e. all the components of the vector $\mathbf{x} + \mathbf{w}$ are non-negative. A VAS evolves from the start vector $\mathbf{w}_0$ by sequentially iterating the addition of vectors from W.

A *vector addition system with states* (VASS) is a VAS equipped with a finite state control. Essentially, state labels are assigned to addition vectors and a graph of states is given which defines the possible sequences of application of addition vectors.

We will use the notation *VAS* and *VASS* to refer to the families of sets of natural vectors which can be generated by VAS and VASS, respectively.

It was shown in [8] that VASS are equivalent in expressive power to VAS (without states): any n-dimensional VASS can be simulated by an equivalent $(n+3)$-dimensional VAS.

A variation on the model of vector addition systems consists in lifting the restriction that the valid vectors must have non-negative components. This model has recently been defined in [7].

An *integer vector addition system* ($\mathbb{Z}$-VAS) of dimension $n \in \mathbb{N}$ is the pair $(\mathbf{w}_0, W)$, where $\mathbf{w}_0 \in \mathbb{Z}^n$ is the start vector, and $W \subseteq \mathbb{Z}^n$ is finite set of addition vectors. A $\mathbb{Z}$-VAS evolves from $\mathbf{w}_0$ by sequentially applying the addition vectors from W. The set of vectors generated by a $\mathbb{Z}$-VAS is defined to be the set of reachable vectors.

An *integer vector addition systems with states* ($\mathbb{Z}$-VASS) is a $\mathbb{Z}$-VAS equipped with a state control and is defined as a tuple $(\mathbf{w}_0, Q, q_0, q_h, p, \delta)$, where $\mathbf{w}_0 \in \mathbb{Z}^n$ is the start vector, Q is a finite set of state labels, $q_0 \in Q$ is the starting state, $q_h \in Q$ is the halting state, $p : Q \setminus \{q_h\} \to \mathbb{Z}^n$ is a function assigning a vector to every state from $Q \setminus \{q_h\}$, and $\delta : Q \to 2^Q$ is a state transition function assigning to each state the set of possible successor states.

A $\mathbb{Z}$-VASS starts in $\mathbf{w}_0$ and in state q_0, applies the addition vector $p(q_0)$, and non-deterministically moves into one of the states from $\delta(q_0)$. This process is iteratively repeated, until the halting state q_h is reached. The vector language generated by a $\mathbb{Z}$-VASS is defined as the set of all vectors which are reachable in the halting state q_h.

We will use the notations $\mathbb{Z}$-*VAS* and $\mathbb{Z}$-*VASS* to refer to the sets of integer vectors generated by $\mathbb{Z}$-VAS or $\mathbb{Z}$-VASS.

3 Register Machines

Definition 1. *A* register machine over the set A *is the tuple* $M_A = (n, A, Q, q_0, q_h, P)$*, where* $n \in \mathbb{N}$*,* A *is a (possibly infinite) register alphabet,* Q *is a finite set of state labels,* q_0 *is the initial state,* $q_h \in Q$ *is the halting state, and*

P is a mapping associating an instruction to every state of M_A. An instruction *is a function $p : A^n \to A^n \times 2^Q$ associating to every n-tuple of values from A another n-tuple of such values and a set of states from Q. A configuration $C \in Q \times A^n$ of M_A is a tuple combining a state and n values from A.*

M_A can be seen as storing values of type A in its n registers. A configuration of M_A therefore defines its current state and the values of its n registers. When in state $q \in Q$, M_A can execute the instruction $P(q)$, which will compute (1) new values for *all* registers of M_A and (2) a set of possible new states; M_A can non-deterministically transition into one of these states.

Definition 2. *A k-step (finite)* computation *of the register machine $M_A = (n, A, Q, q_0, q_h, P)$ is a finite sequence of configurations $(C_i)_{0 \leq i \leq k}$ such that,*

1. *$C_0 = (q_0, \mathbf{a}_0)$, where some of the components of $\mathbf{a}_0$ (registers) may contain input values;*
2. *$C_k = (q_h, \mathbf{a}_k)$, where some of the components of $\mathbf{a}_n$ (registers) may contain output values;*
3. *for every $0 \leq i < k$, $C_i = (q_i, \mathbf{a}_i)$, $C_j = (q_j, \mathbf{a}_j)$, $P(q_i)(a_i) = (\mathbf{a}_j, H)$, and $q_j \in H$.*

M_A therefore transitions from a configuration to another by sequentially applying its instructions. Whenever M_A is in state q_i, it retrieves the corresponding instruction $P(q_i)$ and applies it to the tuple describing the values of the registers. The result, $P(q_i)(a_i)$, gives the new values for the registers and a set of states H from which M_A picks q_j and moves into it. The last configuration C_h is habitually referred to as the *halting configuration*.

Often, in order to be able to express the instructions in a sensible way, one considers some kind of structure over the set A; one example of such a structure may be a finitely generated Abelian group. Classical definitions of register machines rely on (sub)sets of integers and on the associated structure of a linearly ordered finitely generated Abelian group.

In what follows, we describe the existing models of register machines using the abstract language we have just introduced, and we show that blind register machines actually represent the *least* restricted variant.

Definition 3. *A* blind register machine *is a register machine B over the finitely generated Abelian group $(\mathbb{Z}, +)$. The instructions of blind register machines can be of the following two types:*

1. *$(ADD(i), q, s)(a_1, \ldots, a_i, \ldots, a_n) = \big((a_1, \ldots, a_i + 1, \ldots, a_n), \{q, s\}\big)$, and*
2. *$(SUB^*(i), q)(a_1, \ldots, a_i, \ldots, a_n) = \big((a_1, \ldots, a_i - 1, \ldots, a_n), \{q\}\big)$.*

The computations of blind register machines are defined as a computation of the corresponding register machine over $(\mathbb{Z}, +)$.

Definition 4. *A blind register machine* accepts *an input vector by resetting all registers to zero in the halting configuration. A blind register machine* generates *(or* computes *from an input) a vector of numbers by resetting all registers not containing the output to zero.*

We will use the notation $Ps_{\mathbb{Z}}BRM$ (resp., $Ps_{\mathbb{N}}BRM$) to refer to the class of sets of vectors of integer (resp., natural) numbers *accepted* by blind register machines.

All other well known types of register machines can be defined as subtypes of blind register machines.

Definition 5. *A* partially blind register machine *is a blind register machine whose registers are only allowed to contain non-negative numbers: for any computation $(C_i)_{1\leq i\leq k}$ of a partially-blind register machine and for any $C_i = (q_i, \mathbf{a}_i)$, $1 \leq i \leq k$, every component of $\mathbf{a}_i$ is non-negative.*

Thus, if the partially blind register machine B' decides at some point to decrement a register whose value is already zero, it will produce an illegal configuration which will render the whole computation invalid. This means that B' still cannot check its registers for zero, but it knows that all of them are non-negative at any given time. The computations of partially blind machines therefore satisfy a condition which renders them strictly stronger than blind register machines [6]: the registers may never go below zero.

Definition 6. *A partially blind register machine* accepts *an input vector by resetting all registers to zero in the halting configuration. A partially blind register machine* generates *(or* computes *from an input) a vector of numbers by resetting all registers not containing the output to zero in the halting configuration.*

We will use the notation $PsPBRM$ to refer to the class of sets of vectors of natural numbers accepted by partially blind register machines.

We can now also define conventional register machines in our general framework.

Definition 7. *A (conventional)* register machine *is a register machine over $(\mathbb{Z}, +)$ with the following two types of instructions:*

1. $(ADD(i), q, s)(a_1, \ldots, a_i, \ldots, a_n) = \big((a_1, \ldots, a_i + 1, \ldots, a_n), \{q, s\}\big)$, *and*
2. $(SUB(i), q, z)(a_1, \ldots, a_i, \ldots, a_n) =$
$$\begin{cases} \big((a_1, \ldots, a_i - 1, \ldots, a_n), \{q\}\big), & \text{if } a_i > 0, \\ \big((a_1, \ldots, a_i, \ldots, a_n), \{z\}\big), & \text{if } a_i = 0. \end{cases}$$

Computations of conventional register machines are defined as computations of the corresponding register machines over $(\mathbb{Z}, +)$ with the restriction that, in the initial configuration, all registers must contain non-negative values.

It follows from the form of instructions allowed in conventional register machines that their registers contain non-negative values at any time. Therefore, one

can see such register machines as an even more powerful form of partially blind register machines (and thus a particular case of blind register machines), in which the machine is allowed to check whether any given register is zero.

We would like to remark that by considering other types of instructions or restrictions on the class of valid computations, one can characterise many other variants of register machines. For example, reversal-bounded counter automata are register machines in which one can only switch from incrementing to decrementing a register (and conversely) a bounded number of times [9].

4 Integer Vector Addition P Systems

In the article [10], Gheorghe Păun suggested exploring multisets with negative multiplicities. Several possible answers were suggested. In [3], the authors define generalised multisets as having multiplicities from totally ordered Abelian groups. The work [1] takes a different approach and partitions the alphabet of objects into two categories: the regular objects, which may have any integer multiplicity, and the so-called "catalysts", which are only allowed to appear in a bounded number of copies. Like in purely catalytic P systems, the "catalysts" in this model are used to guide the applicability of rules.

In this work, we generalise this model to the concept of integer vector addition P systems. Before defining this model, we define the following natural extension of multisets.

Definition 8. *A* $\mathbb{Z}$-multiset *over the (finite) alphabet* O *is a mapping* $w : O \to \mathbb{Z}$. *The value* $w(a)$ *is called the* multiplicity *of* a *in* w. *An object* $a \in O$ *is said to* apear in w *if* $w(a) \neq 0$. *A multiset* w *is said to be* empty *if no objects appear in it.*

Thus, $\mathbb{Z}$-multisets can also be seen as *vectors* of integers, indexed by elements of O. We will use the notation $\mathbb{Z}^O$ to refer to the set of all $\mathbb{Z}$-multisets over O.

Definition 9. *An* integer vector addition P system *($\mathbb{Z}$-VAPS) is the construct*

$$\Pi = (O, T, \mu, w_1, \ldots, w_n, R, h_i, h_o),$$

where O *is a finite alphabet of objects,* $T \subseteq O$ *is the set of terminal objects,* μ *is the membrane structure injectively labelled by the numbers from* $\{1, \ldots, n\}$ *and usually given by a sequence of correctly nested brackets,* w_i *are the* $\mathbb{Z}$*-multisets giving the initial contents of every membrane* i, $1 \leq i \leq n$, R *is a finite set of rules of the form* $r : \{1, \ldots, n\} \to \mathbb{Z}^O$, *and* h_i *and* h_o *are the labels of the input and the output membranes, respectively (*$1 \leq h_i \leq n$, $1 \leq h_o \leq n$*).*

Thus, integer vector addition P systems manipulate vectors of integers, indexed by the objects from O ($\mathbb{Z}$-multisets). A rule $r \in R$ assigns such a vector to every membrane of Π; applying r means adding (by componentwise addition) the vector $r(i)$ to the vector representing the contents of the membrane i, for every $1 \leq$

$i \leq n$. Therefore, one may see r as only having a right-hand side and as being unconditionally applicable. Such a form comes in naturally, since, as also pointed out in [3, 1], considering multiplicities over $\mathbb{Z}$ renders the usual rule applicability conditions irrelevant. We also remark that this way of defining the rules generalises naturally to a tissue-like membrane structure, i.e. a membrane structure which is not required to be a tree, but can be an arbitrary graph (cf. [5]).

We will use the tuple notation to describe rules of vector addition P systems — a rule r will be given by the set $\big\{(i, r(i)) \mid 1 \leq i \leq n, r(i) \text{ is not empty}\big\}$.

In [1], the authors use a special symbol δ to command the dissolution of the membrane in which it is produced. To allow for the same possibility in vector addition P systems, we will define the rules as functions of the form $\{1, \ldots, n\} \to \mathbb{Z}^{O \cup \{\delta\}}$, i.e. as functions assigning $\mathbb{Z}$-multisets over $O \cup \{\delta\}$ to each membrane. If $r(i)(\delta) = 1$ for the elementary membrane i, the application of r will dissolve this membrane after adding the multiplicities of symbols different from δ to its contents. We may even allow $r(i)(\delta) = k > 1$, in which case k successive membranes in the hierarchy will be dissolved, but only the contents of the innermost dissolved membrane will be copied into the corresponding parent membrane (the contents of the intermediary membranes will be lost).

Allowing dissolution makes it possible to introduce a rule applicability condition: r is applicable if every membrane i, for which $r(i)$ is not empty, is still present in the system.

The integer vector addition P system Π evolves by *sequentially* applying rules from R until a halting configuration is reached. Remark that, because of the use of $\mathbb{Z}$-multisets, the only way to use the classical halting condition is to dissolve all the membranes to which the rules of Π may contribute. This corresponds to the approach proposed in [1] which consists in dissolving all the working membranes until the result reaches a membrane without any rules. Thus, the classical halting condition becomes somewhat degenerate; it is therefore only natural to discuss other halting conditions, for example:

- *unconditional halting* — the system may halt at any moment, independently of rule applicability or contents of the membranes;
- *halting by zero* — the system halts when it reaches a configuration in which all multisets representing the contents of all membranes are empty.

One may see unconditional halting as corresponding to the way in which the language generated by a grammar is defined [4]: essentially, the contents of the output membrane of Π in any configuration Π can reach, projected on the terminal alphabet T, is part of the vector language generated by Π. On the other hand, halting by zero corresponds to the way in which blind register machines recognise input vectors.

We will use the symbols *uncond*, *zero*, and *inappl* to refer to unconditional halting, halting by zero, and halting by inapplicability of rules. Similarly, we will use the symbols *acc* and *gen* to refer to the accepting and generating modes. We will use the notation $Ps_{\mathbb{Z}}VAPS(m, h)$, $m \in \{acc, gen\}$, $h \in \{uncond, zero, inappl\}$,

to refer to the class of sets of vectors of integers accepted or generated by integer vector addition P systems working with the corresponding halting conditions. We will add the symbol δ to refer to the vector languages associated with $\mathbb{Z}$-VAPS with dissolution rules ($Ps_{\mathbb{Z}}VAPS(m,h,\delta)$) and the symbol δ^* to refer to the languages of $\mathbb{Z}$-VAPS which are allowed to dissolve multiple membranes at a time ($Ps_{\mathbb{Z}}VAPS(m,h,\delta^*)$). Finally, we will replace $\mathbb{Z}$ by $\mathbb{N}$ to refer to the languages of vectors of naturals (non-negative integers).

We immediately observe that $Ps_{\mathbb{Z}}VAPS(m,inappl) = \{\varnothing\}$, because if a $\mathbb{Z}$-VAPS has any rules at all, it can never halt by rule inapplicability.

5 On the Power of Blind Register Machines

In this section, we will focus on relating integer vector addition systems to blind register machines, as well as on expressing the power of both models in terms of semilinear vectors of numbers. We will show that blind register machines and $\mathbb{Z}$-VASS generate exactly $\mathbb{N}$-linear sets of $\mathbb{Z}$-vectors.

The work [2] also discusses the computational power of blind and partially blind register machines, but it uses a different definition of blindness: a blind register machine is defined as a partially blind register machine which may halt with any values in the registers. In the present paper we use a definition which is closer to Sheila Greibach's blind and partially register machines [6].

We will start by giving a proof of the quite intuitive result that blind register machines recognise exactly the same sets of integer vectors as integer vector addition systems with states generate.

Theorem 1. *$Ps_{\mathbb{Z}}BRM = \mathbb{Z}$-VASS.*

Proof. Take a blind register machine $B = (n, \mathbb{Z}, Q, q_0, q_h, P)$; we will construct a $\mathbb{Z}$-VASS $\Gamma = (w_0, S, s_0, s_h, p, \delta)$ with $w_0 = (0, \ldots, 0) \in \mathbb{Z}^n$, $S = Q$, $s_0 = q_h$, $s_h = q_0$. The set $\delta(p)$ will contains all the states of B from which p can be reached:

$$\begin{aligned}\delta(s) \;=\; \{q \in Q \mid P(q) = (SUB(i), s) \text{ or } P(q) = (ADD(i), s, s')\\ \text{or } P(q) = (ADD(i), s', s)\}.\end{aligned}$$

The vector $p(s)$ associated with a state $s \in S$ does the opposite effect of the instruction associated with the same state in B:

$$p(s) \;=\; \begin{cases} \mathbf{1}_i, & \text{if } P(s) = (SUB(i), q),\\ -\mathbf{1}_i, & \text{if } P(S) = (ADD(i), q, q'),\end{cases}$$

where $\mathbf{1}_i \in \mathbb{Z}^n$ is a vector whose only non-zero component is the i-th component.

It follows from the construction of Γ that, for every computation of B accepting an input vector $\mathbf{x}$, there exists a computation of Γ halting on the same vector, and conversely, which proves that $Ps_{\mathbb{Z}}BRM \subseteq \mathbb{Z}$-VASS.

To prove the converse inclusion, it suffices to take an arbitrary integer vector addition system and construct a blind register machine by reversing the arrows in the state control graph and by simulating the inverse effect of the addition vectors using multiple states.

The same construction can be used to show that partially blind register machine are equivalent in power to conventional vector addition systems with states. Taking into consideration the result on equivalence between (conventional) VAS and VAS with states from [8], we formulate the following characterisation of the power of partially blind register machines.

Theorem 2. $PsPBRM = VASS = VAS$.

We will now show that blind register machines do not recognise more than $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors.

Lemma 1. $Ps_{\mathbb{Z}}BRM \subseteq \mathbb{Z}^*SLIN_{\mathbb{N}}$.

Proof. Consider a blind n-register machine B. At every step, B can increment or decrement a register, *independently* of the contents of the registers. Consider the alphabet of actions of B: $\mathcal{A}_B = \{ADD(i), SUB(i) \mid 1 \leq i \leq n\}$; every computation of B can be represented as a string over this alphabet. Let $valid(A_B) \subseteq A_B^*$ be the strings over A_B^* which correspond to all computations of B. Pick such a string $w \in valid(A_B)$. Since the actions do not depend on the contents of the registers, any permutation of w which is in $valid(A_B)$ will have the same effect as w. In particular, B will halt with the same values in its registers. Therefore, the set of vectors B recognises can be described as follows:

$$N(B) = \left\{-(a_1, \ldots, a_n) \mid w \in valid(A_B), a_i = |w|_{ADD(i)} - |w|_{SUB(i)}\right\},$$

where $|w|_x$ is the number of copies of the symbol $x \in A_B$ in the string w.

Because B cannot read the values of its registers, the set $valid(A_B)$ is the regular language given by the state control of B. Therefore, the Parikh image $Ps\big(valid(A_B)\big)$ is an $\mathbb{N}$-semilinear set. This means that $N(B)$ is an $\mathbb{N}$-semilinear set of $\mathbb{Z}$-vectors, and so is the set of vectors including only the values of the output registers of B. Consequently, $Ps_{\mathbb{Z}}BRM \subseteq \mathbb{Z}^*SLIN_{\mathbb{N}}$, which is the statement of the lemma.

We will now show that blind register machines can recognise all $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors.

Lemma 2. $Ps_{\mathbb{Z}}BRM \supseteq \mathbb{Z}^*SLIN_{\mathbb{N}}$.

Proof. Consider an $\mathbb{N}$-semilinear set A of $\mathbb{Z}$-vectors. There exists a finite collection of sets of generators $A_i \subseteq \mathbb{Z}^n$ and offsets $\mathbf{a}_i \in \mathbb{Z}^n$ such that $A = \bigcup_i \langle A_i, \mathbf{a}_i \rangle_{\mathbb{Z}}$. Consider a blind register machine B which starts by non-deterministically choosing a set of generators A_i and the corresponding offset $\mathbf{a}_i$. B then repeats the following procedure until the set A_i is exhausted:

1. remove a generator $\mathbf{a}$ from A_i;
2. subtract $\mathbf{a}$ from the vector describing the registers of B a number of times chosen non-deterministically.

At the end, B subtracts the vector $\mathbf{a}_i$ from its registers. If B manages to reset all its registers using this procedure, then, by construction, the input vector belongs to $\langle A_i, \mathbf{a}_i \rangle_{\mathbb{Z}} \subseteq A$ (and the computation of the machine gives a way to construct this vector from A_i and $\mathbf{a}_i$). This implies the statement of the lemma.

It follows from Lemmas 1 and 2 that blind register machines recognise exactly the $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors.

Theorem 3. $Ps_{\mathbb{Z}}BRM = \mathbb{Z}^*SLIN_{\mathbb{N}}$.

Consequently, if one takes only the natural vectors recognised by blind register machines, one obtains positive-restricted $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors.

Corollary 1. $Ps_{\mathbb{N}}BRM = \mathbb{Z}^*_+SLIN_{\mathbb{N}}$.

6 On the Power of $\mathbb{Z}$-VA(P)S

In this section we will describe the power of integer vector addition (P) systems in terms of semilinear sets of vectors and blind register machines. We will start by pointing out that $\mathbb{Z}$-VAPS without dissolution and with unconditional halting generate exactly the sets reachable by $\mathbb{Z}$-VAS.

Lemma 3. $Ps_{\mathbb{Z}}VAPS(gen, uncond) = \mathbb{Z}\text{-}VAS$.

Proof. The effect of rules of $\mathbb{Z}$-VAPS without dissolution does not depend on the contents of the membranes. Consider the set of rules R of such a P system Π; we will construct a $\mathbb{Z}$-VAS Γ whose starting vector is the initial contents of the output membrane h_o of Π, and whose addition vectors are given by the projection $\{r(h_o) \mid r \in R\}$. Since Π can halt at any moment, its output is exactly the reachable vectors of Γ. Therefore $Ps_{\mathbb{Z}}VAPS(gen, uncond) \subseteq \mathbb{Z}\text{-}VAS$.

The converse inclusions follows from the fact that any $\mathbb{Z}$-VAS can be seen as a one-membrane $\mathbb{Z}$-VAPS working with unconditional halting.

The same kind of reasoning allows us to characterise the power of $\mathbb{Z}$-VAPS working in recognising mode and halting by reaching zero vectors in all membranes.

Lemma 4. $Ps_{\mathbb{Z}}VAPS(acc, zero) = \mathbb{Z}\text{-}VAS$.

On the other hand, because of the direct equivalence between $\mathbb{Z}$-VAS and $\mathbb{N}$-linear sets of $\mathbb{Z}$-vectors, we can write the previous two results in the following way.

Theorem 4. $Ps_{\mathbb{Z}}VAPS(gen, uncond) = Ps_{\mathbb{Z}}VAPS(acc, zero) = Z^*LIN_{\mathbb{N}} = \mathbb{Z}\text{-}VAS$.

Allowing membrane dissolution together with unconditional halting allows generating at most unions of vector languages generated by families of $\mathbb{Z}$-VAS which may only differ in their start vectors. We will call such families *uniform* and will denote the class of vector languages generated by such families by $\mathbb{Z}$-$VAS_{\cup}$.

Lemma 5. $Ps_{\mathbb{Z}}VAPS(gen, inappl, \delta) \subseteq \mathbb{Z}\text{-}VAS_{\cup}$.

Proof. Consider a $\mathbb{Z}$-VAPS Π with normal membrane dissolution and halting by inapplicability of rules. First of all, we remark that the contents of the output membrane h_o only depend on the evolution of the membranes located within. Furthermore, h_o must have no rules associated, otherwise the system will never halt (or will end up dissolving h_o if h_o is not the skin, in which case no output will be yielded either). Finally, only those inner membranes of h_o which are dissolved contribute to its final contents.

Consider one of the membranes h located somewhere within h_o. If it has no inner membranes, its evolution is described by a $\mathbb{Z}$-VAS. If h has one inner membrane h' which is elementary (it contains no other membranes), then we can distinguish two phases in the evolution of h: before and after the dissolution of h' (and before the dissolution of h itself). Given that the contents of h' must eventually be merged with those of h, we can just as well consider that, during the first phase, the rules contributing to h are extended by the corresponding additions carried out by the rules contributing to h'. Since the order in which the rules of $\mathbb{Z}$-VAPS are applied does not affect the result, we can consider that h contains *no inner membranes* at all, but possesses a double set of contributing rules instead: one which combines the original rules contributing to h and to h', and another which only includes the original rules contributing to h. Therefore, we can correctly describe the evolution of h by taking at least some of the vectors a $\mathbb{Z}$-VAS can reach.

We remark that the rule dissolving h' in this case may only be applied once, and its effect can be simulated by adding the vector it produces to the starting multiset of the containing membrane h.

The reasoning from the previous paragraphs can be applied to a membrane which contains multiple elementary membranes, as well as, inductively, to all inner membranes of the output membrane h_o: we can replace h_o by a new elementary membrane h'_o, and take some of the vectors generated in it into the output language. Remark now that the moment at which a membrane is dissolved is not correlated with its contents and only depends on whether all of its inner membranes have been dissolved already. This means that, if the depth of the membrane structure contained in h_o is d, d steps of evolution are necessary and suffice for dissolving all the inner membranes of h_o. Therefore, the contents of the membrane h_o in the halting configurations of Π are given by all the vectors that can be reached in membrane h'_o in at least d steps. These are the vectors which can be reached by the family of $\mathbb{Z}$-VAS $\{(\mathbf{w}_i, W) \mid \mathbf{w}_i \in W_d\}$, where W contains the addition vectors defined by the rules contributing to the new membrane h'_o, and W_d is the (finite) set of vectors which h'_o can reach in exactly d steps.

It turns out that this family of $\mathbb{Z}$-VAPS can generate all vector languages from $\mathbb{Z}$-$VAS_{\cup}$.

Lemma 6. $Ps_{\mathbb{Z}}VAPS(gen, inappl, \delta) \supseteq \mathbb{Z}$-$VAS_{\cup}$.

Proof. Consider a finite family of $\mathbb{Z}$-VAS $F = \{(\mathbf{w}_i, W) \mid 1 \leq i \leq n\}$. We will define a $\mathbb{Z}$-VAPS Π generating the vectors reachable by the systems from F in the following way. Π will have two nested membranes and two groups of rules. The first group of rules will apply the vectors from W to the inner membrane. A rule of the second group will add one of the vectors $\mathbf{w}_i$ to the inner membrane and dissolve it immediately. By construction, the vectors appearing in the halting configurations of Π are exactly the vectors which can be reached by the $\mathbb{Z}$-VAS from F, which proves the lemma.

The following theorem summarises the two preceding lemmas.

Theorem 5. $Ps_{\mathbb{Z}}VAPS(gen, inappl, \delta) = \mathbb{Z}$-$VAS_{\cup}$.

Interestingly, the class $\mathbb{Z}$-$VAS_{\cup}$ is strictly in between the classes $\mathbb{Z}$-VAS and $\mathbb{Z}$-$VASS$.

Lemma 7. $\mathbb{Z}$-$VAS \subsetneq \mathbb{Z}$-$VAS_{\cup}$.

Proof. The inclusion is trivial. Consider now two $\mathbb{Z}$-VAS having the axioms $(0, 0)$ and $(0, 1)$, and sharing the only addition vector $(1, 1)$. The language of vectors reachable by these two systems is $L = \{(a, a), (a, a + 1) \mid a \in \mathbb{N}\}$. Suppose there exists a $\mathbb{Z}$-VAS Γ generating the same vector language L. In order to generate all pairs of natural numbers (a, a), it must start with the axiom $(0, 0)$ and have an addition vector of the form $(1, 1)$. Then, in order to generate the pairs $(a, a + 1)$, Γ needs to have an addition vector of the form $(x, x + 1)$. However, applying this addition vector twice yields the vector $(2x, 2x + 2) \notin L$, which contradicts the supposition and proves that the inclusion from the statement of the lemma is strict.

The following lemma describes the relationship between $\mathbb{Z}$-$VAS_{\cup}$ and $\mathbb{Z}$-$VASS$.

Lemma 8. $\mathbb{Z}$-$VAS_{\cup} \subsetneq \mathbb{Z}$-$VASS$.

Proof. The work of a finite family F of $\mathbb{Z}$-VAS can be simulated by a $\mathbb{Z}$-VASS by non-deterministically choosing a state in which one of the start vectors of F will be added, and by subsequent direct simulation of the application of the shared addition vectors.

Consider now the $\mathbb{Z}$-VASS Γ with the starting vector $(0, 0)$, which applies the addition vector $(0, 0)$ in the starting state q_0 and the non-deterministically chooses between $q_{(1,0)}$ and $q_{(0,1)}$. In $q_{(1,0)}$, Γ may apply the addition vector $(1, 0)$ indefinitely, before transitioning into q_h. Similarly, in $q_{(0,1)}$, Γ may apply the addition vector $(0, 1)$ indefinitely, before moving into q_h. Thus, the vector language generated by Γ is $L = \{(a, 0), (0, a) \mid a \in \mathbb{N}\}$.

Suppose there exists a family of $\mathbb{Z}$-VAS which generate the same language. The shared addition vectors of this family must therefore include both $(1,0)$ and $(0,1)$. But then, this family must also generate vectors in which both components are non-zero and which therefore do not belong to L. This contradicts our supposition and proves that the inclusion in the statement of the lemma is strict.

The previous lemma also gives an example of a $\mathbb{Z}$-semilinear set which cannot be generated by uniform family of $\mathbb{Z}$-VAS systems, which implies the following result.

Corollary 2. $\mathbb{Z}\text{-}VAS_{\cup} \subsetneq \mathbb{Z}^*SLIN_{\mathbb{N}}$.

It follows from the Theorem 5, Lemmas 7 and 8, as well as from the characterisations from the previous section, that the languages recognised by $\mathbb{Z}$-VAPS with normal dissolution and conventional halting are situated strictly in between $\mathbb{N}$-linear sets of $\mathbb{Z}$-vectors and $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors.

Theorem 6. $\mathbb{Z}^*LIN_{\mathbb{N}} \subsetneq Ps_{\mathbb{Z}}VAPS(gen, inappl, \delta) \subsetneq \mathbb{Z}^*SLIN_{\mathbb{N}}$.

Finally, we show that allowing dissolution of *multiple* membranes by one rule allows generating all $\mathbb{Z}$-semilinear languages and therefore renders such $\mathbb{Z}$-VAPS equivalent in power to blind register machines.

Theorem 7. $Ps_{\mathbb{Z}}VAPS(gen, inappl, \delta^*) = \mathbb{Z}^*SLIN_{\mathbb{N}}$.

Proof (Sketch). Consider a family F of n $\mathbb{Z}$-VAS, each of which generates a $\mathbb{Z}$-linear set of vectors. One can construct an integer vector addition P system Π with multiple dissolution in the following way. Π will have $n+2$ membranes organised in a linear structure. The rules of Π will simulate the i-th $\mathbb{Z}$-VAS in the membrane at depth $i+1$ (the depth of the skin is 0); moreover, Π will have a rule producing the start vector $\mathbf{w}_i$ and introducing $n-i$ copies of δ into the membrane at depth $i+1$, for $1 \leq i \leq n$. These rule effectively finalise the simulation of the i-th $\mathbb{Z}$-VAS. Finally, Π will have rules introducing i copies of δ into the innermost membrane, for $1 \leq i \leq n$, which will "select" the membrane at depth $i+1$ and will allow it to eventually apply its dissolution rules and put the result into the skin. Thus, Π generates the semilinear language generated by the family F.

To prove the inverse inclusion, we will rely on Lemma 1. Consider a $\mathbb{Z}$-VAPS Π with multiple dissolution. We will construct a blind register machine B which recognises the vector language generated by Π in the following way. B will have a group of working register per membrane of Π which will represent the multiplicities of the symbols in this membrane. B will start with the vector $\mathbf{x}$ in its input registers, and will simulate the applications of rules of Π in its working registers. Whenever Π dissolves a membrane (or multiple membranes), B will nondeterministically guess the multiplicities of each symbol in the dissolved membrane and will copy the guessed values into the working registers representing the corresponding parent membrane. When all inner membranes of the output membranes have been dissolved (B can encode the information about the membrane structure

in its state), B will simultaneously decrement the working registers representing the contents of the skin and the input registers. If, earlier during the simulation, B had guessed the value of a register wrongly, or, at the end of the simulation, the values of the input registers and the working registers representing the skin did not match, some registers of B will be zero and the vector $\mathbf{x}$ will be rejected. It follows from the construction that B will accept exactly the vector generated by Π, which implies the statement of the theorem.

7 Conclusion

In this paper we continued the investigation of P systems with multisets with integer multiplicities, proposed in [10] and already studied in [3] and [1]. We focused on the model originally described in [1] and generalised it to integer vector addition P systems, in which the applicability of rules does not in any way depend on the contents of the membranes. Interestingly enough, this P system variant exhibits very strong connection with blind register machines and integer vector addition systems — two models which have received little to no attention in the scientific literature up to now.

We studied a number of working modes of and halting conditions for integer vector addition P systems and gave exact characterisations of the power of the corresponding variants in terms of linear and semilinear sets over $\mathbb{Z}$ and over $\mathbb{N}$. We also pointed out a number of relations between the classes of languages generated or accepted by the model.

Some non-trivial open questions are revealed by our research. One of them concerns the semantics of multiple dissolution. In P systems, dissolution typically concerns one membrane at a time; in the present paper we suggest considering the possibility of dissolving multiple containing membranes in one step. The semantics we propose discards the contents of the dissolved intermediary membranes, so only the multiset of the innermost dissolved membrane is transferred to the corresponding parent membrane. Other semantics of multiple dissolution may be possible and are certainly worth exploring.

A very interesting open question concerns the types of semilinear sets. In this paper we only deal with semilinear sets with generators and initial offsets from $\mathbb{N}^n$ and $\mathbb{Z}^n$, restricted to non-negative values or not. It is however possible to consider the generators, the offsets, and the *coefficients* to belong to $\mathbb{N}^n$ or $\mathbb{Z}^n$, alternatively. This yields 8 possibly different kinds of semilinear sets, not including restrictions to non-negative values. Exploring the relations between these kinds of semilinear sets may be useful in further refining certain characterisations.

Finally, we point out that classical halting by inapplicability of rules is not necessarily well adapted for dealing with generalisations of multisets to integers. We give examples of different halting conditions inspired by other models of computing, but our list is far from exhaustive and is definitely worth to be extended.

References

1. O. Belingheri, A. E. Porreca, and C. Zandron. P systems with hybrid sets, 2016. Workshop on Membrane Computing, submitted.
2. R. Freund, O. Ibarra, Gh. Păun, and H.-C. Yen. Matrix languages, register machines, vector addition systems. *Third Brainstorming Week on Membrane Computing*, pages 155–167, 2005.
3. R. Freund, S. Ivanov, and S. Verlan. P systems with generalized multisets over totally ordered abelian groups. In *Int. Conf. on Membrane Computing*, volume 9504 of *Lecture Notes in Computer Science*, pages 117–136. Springer, 2015.
4. R. Freund, M. Kogler, and M. Oswald. A general framework for regulated rewriting based on the applicability of rules. In J. Kelemen and A. Kelemenová, editors, *Computation, Cooperation, and Life*, volume 6610 of *Lecture Notes in Computer Science*, pages 35–53. Springer Berlin Heidelberg, 2011.
5. R. Freund and S. Verlan. A formal framework for static (tissue) P systems. In G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–284. Springer Berlin Heidelberg, 2007.
6. S. A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3):311–324, 1978.
7. C. Haase and S. Halfon. Integer vector addition systems with states. In J. Ouaknine, I. Potapov, and J. Worrell, editors, *Reachability Problems: 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, pages 112–124. Springer, 2014.
8. J. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979.
9. O. H. Ibarra. Automata with reversal-bounded counters: A survey. In H. Jürgensen, J. Karhumäki, and A. Okhotin, editors, *Descriptional Complexity of Formal Systems: 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings*, pages 5–22. Springer, 2014.
10. Gh. Păun. Some quick research topics. `http://www.gcn.us.es/files/OpenProblems_bwmc15.pdf`.
11. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 1998.
12. Gh. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.