P Automata: Concepts, Results, Recent Developments

Erzsébet Csuhaj-Varjú

Department of Algorithms and Their Applications,
Faculty of Informatics,
Eötvös Loránd University,
Pázmány Péter sétány 1/c, 1117 Budapest, Hungary
csuhaj@inf.elte.hu

Abstract. We summarize basic concepts, important results, and recent developments on P automata, variants of P systems combining features of classical automata and complex systems being in interaction with their environments. We also discuss topics for future research.

1 Introduction

P automata are accepting variants of P systems which combine features of classical (standard) automata and nature-motivated complex systems of agents being in interaction with their environment.

Briefly, a P automaton is a P system which receives input in each computation step from its environment. The input is given as a multiset of objects. It changes the actual configuration of the P system, thus affects its functioning. The sequences of inputs are distinguished as accepted or rejected input sequences.

The analogy between P automata and classical automata can immediately be observed, but the reader may easily notice differences between the two constructs as well: for example, standard automata have separate state sets while in the case of P automata the actual state is represented by the actual configuration of the underlying P system. Another difference between P automata and classical automata is the following: the P automaton uses for computation only the objects of the already consumed input (multisets of objects entering the system). This implies that the object of the computation and the machine which performs the computation cannot be separated as it can be done in the case of customary automata.

The first variant of P automata, introduced in [15, 16], was the so-called one-way P automaton where the underlying P system had only top-down symport rules with promoters (and implicitly inhibitors). Almost at the same time, a closely related notion, called analyzing P system was defined in [25] providing another concept of an automaton-like P system. Both models describe the class of recursively enumerable languages.

The property that automaton-like purely communicating accepting P systems may represent computationally complete classes of computing devices, have resulted in a detailed study of P automata. Several variants have been introduced

and investigated, which differ from each other in the main ingredients of these systems: the objects the P system operates with, the way of defining the acceptance, the way of communication with the environment, the types of the communication rules used by the regions, the types of the rules associated with the regions (whether or not evolution rules are allowed to be used), and whether or not the membrane structure changes during the computation.

Summaries on these constructs and their properties can be found in [36, 8, 14, 48, 9–11, 21].

Due to the power of the underlying P system, a lot of P automaton variants are able to accept any recursively enumerable language, even with size parameters bounded by a small constant. However, P automata with significantly less computational power are of special interest as well. For example, the generic variant which is based on antiport rules with promoters or inhibitors and accepts with final states, using the sequential working mode and some well-chosen mappings for defining its language, describes a language class with sub-logarithmic space complexity. In this way, a "natural description" of this particular complexity class is provided.

In the following sections we describe the most important variants of P automata and their properties.

Special emphasis is put on non-standard features of P automata, namely, that the same construct is able to operate over both finite and infinite alphabets, languages of P automata can be defined on the base of finite and infinite run (computation) of the system, and that to obtain large computational power they do not need workspace overhead.

We also discuss dP automata, i.e. distributed systems of P automata, a finite collection of P automata communicating not only with their joint environment but with each other. We demonstrate further concepts and methods for describing parallelism, cooperation, communication, and distribution in terms of purely communicating accepting P systems.

We also suggest possible new topics and problems for future research.

2 Preliminaries

The reader is assumed to be familiar with the basics of formal language and automata theory, computability, and membrane computing; for more information we refer to [35, 46, 45].

An alphabet is a finite non-empty set of symbols. Given an alphabet V, V^* denotes the set of all strings over V. If the empty string, λ, is not included, then notation V^+ is used. The length of a string $x \in V^*$ is denoted by $|x|$, the number of occurrences of symbols from a set $A \subseteq V$ in x is denoted by $|x|_A$. If A is a singleton set, $A = \{a\}$, then notation $|x|_a$ is used instead of $|x|_{\{a\}}$. The reverse (or the mirror image) x^R of a non-empty string $x = x_1 x_2 \dots x_n,\ x_i \in V,\ 1 \leq i \leq n$, is defined as $x^R = x_n x_{n-1} \dots x_1$, and $\lambda^R = \lambda$.

The class of regular, context-free, context-sensitive, and recursively enumerable languages is denoted by $\mathcal{L}REG$), $\mathcal{L}(CF)$, $\mathcal{L}(CS)$, and $\mathcal{L}(RE)$, respectively.

A finite multiset over an alphabet V is a mapping $M : V \to \mathbb{N}$ where $\mathbb{N}$ is the set of non-negative integers; $M(a)$ is said to be the multiplicity of $a \in V$ in M. A finite multiset M can also be represented by any string $x \in V^*$ where $|x|_a = M(a)$ for all $a \in M$. (Obviously, all permutations of x represent the same finite multiset). Thus, the set of all finite multisets over an alphabet V can be denoted by V^*, and we use the notation V^+ for denoting the set of non-empty (finite) multisets. The empty multiset is denoted by λ. We note that if confusion may arise, then we indicate whether we speak of a string or a finite multiset.

We recall now some notions from computability theory we will refer to in the sequel.

A register machine is a construct $M = (m, B, l_0, l_h, P)$, where m is the number of registers, B is the set of labels, l_0 is the initial label, l_h is the final label, and P is the set of instructions labelled by elements of B. The instructions have one of the following forms:

- $(l_i : ADD(r); l_j)$, where $l_i \in B \setminus \{l_h\}$, $l_j \in B$, $1 \leq r \leq m$.
  This instruction is called an increment instruction; it increases the value of (the number stored in) register r by one and then the computation continues with instruction l_j.
- $(l_i : SUB(r); l_j, l_k)$, where $l_i \in B \setminus \{l_h\}$, $l_j, l_k \in B$, $1 \leq r \leq m$.
  This instruction is called a subtract instruction. If the number stored in register r is a positive number, then this instruction decreases this number by one and then the computation continues with instruction l_j. This case is called decrement. If the number stored in the register r is zero, then the value of each register remains unchanged and the computation continues with instruction l_k. In this case we speak of a zero-test.
- $l_h : HALT$. This instruction stops the work of the register machine; it is called the halt instruction.

A configuration of the register machine is described by the current instruction label and the value of the registers. The current instruction label identifies the instruction to be executed. The register machine works with changing its configurations (a change is also called a transition). A transition sequence starting with the initial instruction l_0 and ending with the final instruction l_h is called a computation by M.

A natural number n is said to be accepted by M if there is a halting computation (a computation ending with instruction l_h) such that at the beginning of the computation n is stored in the first register and all other registers store zero.

Counter automata are extensions of register machines which are able to process strings. In this case an input alphabet T and instructions for reading symbols from the input tape T are added to the instruction set. These instructions are of the form $l_1 : (read(a), l_2)$, with $l_1 \in B \setminus \{l_h\}$, $l_2 \in B$, and $a \in T$. A counter automaton is denoted by $M = (m, B, l_0, l_h, P, T)$.

It is well-known that the family of recursively enumerable sets of natural numbers is the family of sets of numbers that can be accepted by register machines

with at most three registers. Furthermore, counter automata with two registers (counters) describe exactly the class the recursively enumerable languages.

Often, register machines (and thus counter automata) are presented in some other form. Namely, instead of labels we consider states, and the notation is changed accordingly (for more details on register machines see [35]).

In this case a register machine is given as $M = (Q, R, q_0, q_f, P$, where

- Q is a finite non-empty set, called the set of states;
- $R = \{A_1, \ldots, A_k\}$, $k \geq 1$, is a set of registers;
- $q_0 \in Q$ is the initial state;
- $q_f \in Q$ is the final state;
- P is a set of instructions of the following form:
  - $(p, A+, q, s)$, where $p, q, s \in Q, p \neq q_f, A \in R$, called an increment instruction,
  - $(p, A-, q, s)$, where $p, q, s \in Q, p \neq q_f, A \in R$, called a decrement instruction.

  Furthermore, for every $p \in Q$, $(p \neq q_f)$, there is exactly one instruction of the form either $(p, A+, q, s)$ or $(p, A-, q, s)$.

The configuration, the transition, and the computation is defined in the same way as previously.

Next we recall the notion of the Arithmetical Hierarchy, a concept we will refer to in the sequel. The Arithmetical Hierarchy (see [5, 47, 23] is usually developed with the universal ($\forall$) and the existential ($\exists$) quantifiers restricted to the integers. Levels in the Arithmetical Hierarchy are labelled by Σ_n if they can be defined by expressions beginning with a sequence of at most n alternating quantifiers starting with $\exists$. Levels are labelled by Π_n if they can be defined by such expressions beginning with a sequence of at most n alternating quantifiers starting with $\forall$. Σ_0 and Π_0 have no quantifier. Σ_1 and Π_1 have only one single quantifier $\exists$ and $\forall$, respectively. The intersection of Σ_n and Π_n is denoted by Δ_n. It is known that Δ_1 and Σ_1 are the computable languages and the recursively enumerable languages.

Now we briefly recall the main features of (cell-like) P systems.

A P system is a structure of hierarchically embedded membranes where each membrane has a unique label and encloses a region containing a multiset of objects and possibly other membranes. The out-most membrane which is unique is called the skin membrane and it is usually labelled with 1. The membrane structure can be given as a sequence of matching parentheses where the matching pairs have the same label as the membranes they represent. A membrane structure can be represented by a rooted tree as well. The P system functions by (possibly) changing the objects in the different regions and moving them across the neighbouring membranes. These rules can be defined in various manners, thus making possible to create and study different variants of P systems, with different motivations (see, for example [45]).

Biologically well-motivated, particularly important variants of P systems are the P systems with symport/antiport rules (introduced in [38]) where the rules

are pure communication rules, i.e., they do not change any object under the functioning of the system.

An antiport rule is of the form $(x, out; y, in)$, where $x, y \in V^*$. In this case, the objects in y enter the region from the parent region and in the same step the objects in x leave to the parent region. The parent region of the skin region is the environment. All types of these rules might be associated with a promoter or an inhibitor multiset, denoted by $(x, in)|_Z, (x, out)|_Z$, or $(x, out; y, in)|_Z$, where $x, y \in V^*, Z \in \{z, \neg z \mid z \in V^*\}$. If $Z = z$, then the rule can only be applied if the region contains all objects of multiset z, and if $Z = \neg z$, then z must not be a sub-multiset of the multiset of objects present in the region.

Throughout the paper, symport and antiport rules with or without promoters/inhibitors are denoted by $(x, out; y, in)|_Z$, $x, y \in V^*, Z \in \{z, \neg z \mid z \in V^*\}$ where we also allow x, y, z to be the empty multiset. If $y = \lambda$ or $x = \lambda$, then the notation above denotes the symport rule $(x, in)|_Z$ or $(y, out)|_Z$, respectively, if $Z = \lambda$, then the rules above are without promoters or inhibitors.

For more information on symport/antiport P systems consult [45].

3 P Automaton - the Basic Model

In this section we provide formal details on the generic variant of P automata, with a brief summary of the recent developments. We mainly follow notations of [14].

Definition 1. *A P automaton (with n membranes or of degree n) is an* $(n+4)$*-tuple* $\Pi = (V, \mu, P_1, \ldots, P_n, c_0, \mathcal{F})$, $n \geq 1$,*where*

- V *is a finite alphabet of objects,*
- μ *is a membrane structure (a rooted tree) of n membranes with membrane* 1 *being the skin membrane,*
- P_i *is a finite set of antiport rules being associated with (possibly empty) promoters and/or inhibitors to membrane i for all* $i, 1 \leq i \leq n$,
- $c_0 = (w_1, \ldots, w_n)$ *is the initial configuration (or the initial state) of* Π *where each finite multiset* w_i, $w_i \in V^*$, *is called the initial contents of region i,* $1 \leq i \leq n$, *and*
- $\mathcal{F}$ *is a computable set of n-tuples of finite multisets* $(v_1, \ldots, v_n)$ *where* $v_i \subseteq V^*$, $1 \leq i \leq n$. $\mathcal{F}$ *is called the set of accepting configurations (or set of final states) of* Π.

An n-tuple $(u_1, \ldots, u_n)$ of finite multisets of objects over V present in the n regions of the P automaton Π is called a configuration of Π; u_i is the contents of region i in this configuration, $1 \leq i \leq n$.

A P automaton functions as a standard antiport P system with (possibly empty) promoters and/or inhibitors; it changes its configurations by applying rules according to a certain type of working mode.

Since the beginning, the most commonly used variant of rule application mode has been the (non-deterministic) maximally parallel working mode, but

the so-called sequential mode introduced in [15, 16] (also called 1-restricted minimally parallel mode in [27]) has been considered as well, due to its particular importance.

When the maximally parallel working mode is used, at every step of the computation as many rule application is performed simultaneously in each region as possible, while in case of sequential rule application exactly one rule is applied in each region where at least one rule is applicable. The rules (the multisets of rules) are non-deterministically chosen out of the applicable ones. We use notation *seq* and *maxpar* for the sequential and the maximally parallel rule application, respectively.

In the last few years, several other working modes (derivation modes) have also been considered: the asynchronous derivation mode ($asyn$), the set-maximally parallel working mode ($smax$) (it corresponds to min_1), the variant where the maximal number of rules is chosen ($max_{rules}max$) and its set-mode counterpart ($smax_{rules}max$), or $max_{objects}max$ and $smax_{objects}max$ where the maximal number of objects are taken into account. For a summary on details of these working (derivation) modes and their use in P automata theory, the reader is referred to [21].

The set of the different types of working modes is denoted by $MODE$. If only the sequential and the maximally parallel mode are considered, then we use notation $MODE_R$ (restricted set of modes).

Let $\Pi = (V, \mu, P_1, \ldots, P_n, c_0, \mathcal{F})$, $n \geq 1$, be a P automaton working in the X-mode of rule application, where $X \in MODE$. The transition mapping of Π is defined as a partial mapping $\delta_X : V^* \times (V^*)^n \to 2^{(V^*)^n}$ as follows:

For two configurations $c, c' \in (V^*)^n$, we say that c directly changes to c', denoted by $c' \in \delta_X(u, c)$, if Π enters configuration c' from configuration c by applying its rules in the X-mode, while reading the input $u \in V^*$, That is, u is the multiset of objects that enter the skin membrane from the environment while the underlying P system changes configuration c to c' by applying its rules in mode X.

The set of input sequences accepted by P automaton $\Pi = (V, \mu, P_1, \ldots, P_n, c_0, \mathcal{F})$, $n \geq 1$, with X-mode of rule application, $X \in MODE$, is defined as the set of sequences of input multisets which enter the skin membrane during an accepting computation.

Formally, the set of accepted input sequences, $A_X(\Pi)$ is defined as

$$A_X(\Pi) = \{v_1 \ldots v_s \mid v_i \in V^*, \text{ there are } c_0, c_1, \ldots, c_s \in (V^*)^n \text{ such that } c_i \in \delta_X(v_i, c_{i-1}), 1 \leq i \leq s, \text{ and } c_s \in \mathcal{F}\}.$$

A P automaton Π (of degree n) is said to be accepting by final states if $\mathcal{F} = E_1 \times \cdots \times E_n$ for some $E_i \subseteq V^*$, $1 \leq i \leq n$, where E_i is either a finite set of finite multisets or $E_i = V^*$. If Π accepts by halting, then $\mathcal{F}$ contains all configurations c with no $c' \in (V^*)^n$ such that $c' \in \delta_X(v, c)$ for some $v \in V^*$, $X \in MODE$, i.e., no rule can be applied in any of the regions.

Recently, motivated by studies of P automata with infinite runs (infinite sequence of configurations) on finite inputs, new variants of acceptance have been

considered [2, 23]. In this case special conditions are imposed on the infinite configuration sequence of the P automaton which correspond to the existence/non-existence of a recursive feature in the infinite configuration sequence. If from some moment on during the infinite run, the configuration of the P automaton does not change any more, then the P automaton is said to be accepting by adult halting; it accepts with partial adult halting if a specific pre-defined part of it does not change any more. This moment is defined by the occurrence/non-occurrence of the recursive feature.

By encoding the accepted multiset sequences of a P automaton to strings, a language can be associated to the P automaton. While in the case of sequential rule application, during the computation the set of multisets that enter the system is finite, thus the input multisets can obviously be encoded by a finite alphabet. In the case of maximally parallel rule application, the number of objects which enter the system in one step may not be bounded by a constant. This implies, that in this case the accepted input sequences may correspond to strings over infinite alphabets.

To restrict the languages to be defined over finite alphabets, we apply a mapping to produce a finite set of symbols from a possibly infinite set of input multisets.

Let V and Σ be two alphabets and let $MAP_{V,\Sigma}$ denote the family of computable mappings $f : V^* \to 2^{\Sigma^*}$ such that f orders to any finite multiset u over V a finite set of strings U from Σ^* ($f(u) = U$ is a finite set) and the empty multiset is mapped to λ. If $f(u) = \lambda$ if and only if u is the empty multiset, then we say that f is non-erasing.

Definition 2. *For a P automaton $\Pi = (V, \mu, P_1, \ldots, P_n, c_0, \mathcal{F})$, $n \geq 1$, a finite alphabet Σ, and a computable mapping $f \in MAP_{V,\Sigma}$, we define the language accepted by Π with respect to f and using the X-mode of rule application, where $X \in MODE$, by*

$$L_{X,f}(\Pi) = \{f(v_1) \ldots f(v_s) \in \Sigma^* \mid v_1 \ldots v_s \in A_X(\Pi)\}.$$

The family of languages accepted by P automata with X-mode of rule application, where $X \in MODE$, with respect to a mapping f satisfying the above conditions is denoted by $\mathcal{L}_{X,f}(PA)$. If a family $\mathcal{C}$ of such mappings is considered, then we use notation $\mathcal{L}_{X,\mathcal{C}}(PA)$.

Notice that it is reasonable to consider mapping f of low complexity, since the accepted language depends on f.

4 Discussion of the Model

P automata combine features of classical automata and natural complex systems. For comparisons to classical automata, the reader is referred to [9] and for comparing P automata to natural complex systems to [11].

In the following we briefly discuss parameters and features of P automata that make them non-standard accepting computing devices.

In case of classical language accepting devices (automata, Turing machine, etc.), the whole input string is available at the beginning of the computation, but in case of P automata the input will be available step-by-step, determined by its actual configuration. Furthermore, the processed input will also be part of the computing device and it is not separated from the configuration of the machine. This characteristics resembles a feature of natural systems: the behaviour (the work) of the system is determined by its existing constituents and their interaction with the environment, there is no abstract component (or workspace) for influencing the functioning of the system. We note, however, that the concept of a region designated for storing the possible input (a bounded local environment) has been introduced and examined [24].

Notice that the input objects have no direct influence on the rules to be applied at the step when they enter the system, they will affect the work of the P automaton only in the coming computation steps. Interesting variants are P automata where the input is pre-defined, and furthermore, that case where the input decides which rules are applied at that computation step. The idea of pre-defined input was raised by György Vaszil (see [8]); different variants and developments of the concept were realized as input-driven tissue P automata [1], P colony automata, PCol automata [7], and active P automata [3].

Since standard (generic) P automata are antiport P systems working in the maximally parallel working mode, there may be P automata which have no constant bound on the number of objects entering the skin membrane during a successful computation.

Due to this property, P automata can be used for describing languages over infinite alphabets without any extension or additional component added to the construct. P finite automaton [20] is based on this property. The main characteristics of this variant is that it has a distinguished object which during the computation (in the maximally parallel working mode) may appear in the skin membrane in a number of copies not bounded by some constant and the other objects appear only in a number of copies bounded by some constant. Thus, when we define the language, we may use a mapping f such that its domain is infinite. Namely, if the distinguished object is a, depending on the number of as, say k, we can consider $f(a^k) = a_k$ for any $k \geq 1$, and $f(\emptyset) = \lambda$. In this way we obtain an infinite alphabet $\{a_1, a_2, \ldots, \}$.

In [20] it was shown that for any language $L \subseteq \Sigma^*$ over a finite alphabet Σ, L is regular if and only if L is accepted by some P finite automaton Π. Thus, the languages which are defined over infinite alphabets and accepted by P finite automata can be considered as extensions of the class of regular languages to infinite alphabets. The above construction significantly differs from other infinite alphabet extensions of regular languages defined by, for example, the finite memory automata from [32] or the infinite alphabet regular expressions introduced in [37], as is shown in [20].

By definition, a P automaton may perform infinite runs (may have infinite configuration sequences), thus their languages can also be defined similarly to languages of ω-Turing machines. Counterparts of ω-Turing machines, called ω-

P automata, introduced in [26], were inspired by the above considerations. In [26], it was shown that for any well-known variant of acceptance mode of ω-Turing machines one can construct an ω-P automaton with two membranes which simulates the computations of the corresponding ω-Turing machine.

Recently, the topic of infinite runs has been significantly developed. First, the notion of a red-green P automaton was introduced [2], on the analogy of red-green register machines (red-green counter automata), variants of red-green Turing machines. For information on red-green Turing machines, consult [33]. In case of red-green machines (red-green automata) the set of states is divided into two disjoint sets, the set of so-called red states and that of green states. Then infinite runs on finite inputs of the automaton are considered. If the number of changes from red state to green state and vice versa is greater than one, than red-green automata is able to recognize more than the family of recursively enumerable languages. On the analogy of red-green counter automaton, red-green P automaton (with one membrane) was defined and shown to be able to "go beyond Turing". For details, the reader is referred to [2].

An important step further has been made in [23]. The authors assigned so-called observer languages to infinite computations of the P automaton. The observer language is an ω-language over alphabet $\{0, 1\}$ where 1 indicates that a pre-defined specific feature of the configuration in the infinite computation sequence is fulfilled and 0 appears in the infinite word if this feature is not fulfilled. These observer languages extend the recognition power of P automata. For example, particular variants of regular observer languages are sufficient to describe infinite runs of red-green P automata.

The generic variant of P automata is given with static membrane structure, that is, the membrane structure does not change during the work of the system. This condition is rather restrictive, since the architecture of natural systems (a P automata models a natural system, namely, a living cell) may change during their functioning. An example for P automaton with dynamically changing structure is the P automaton with marked membranes ([17]); the notion was motivated in part by brane calculi [6]. One other such model is the so-called active P automaton that was proposed for parsing sentences of natural languages [3]. An active P automaton starts the computation with one membrane containing the string to be analyzed, together with some additional information assisting the computation. It computes with the membrane structure, using operations as membrane creation, division, and dissolution. There are also rules for extracting a symbol from the left-hand end of the input string and for processing assistant objects. The computation is successful (accepting) if all symbols from the string are consumed and all membranes are dissolved. It was shown that the model is suitable for recognizing any recursively enumerable language, and if some well-chosen restrictions are imposed on the types of its rules, then other well-known language classes (the regular language class andthe class of context-sensitive languages) can be described by this model. We note that this special variant of P automata has the whole input at the beginning of the computation.

5 Power of P Automata

The P automaton uses for computation its actual input multiset and the objects of the already consumed input multisets (obviously, the objects already available at the beginning are considered as well). Although this fact appears to bound the accepting power, since the P automaton may input an exponentially growing number of objects (using the maximally parallel working mode), the obtained computational power can be rather large.

For the maximally parallel and the sequential working modes, a description of the accepted language classes was presented in [12, 13]).

A non-deterministic one-way Turing machine is said to be restricted $S(n)$ space bounded if for every accepted input of length n, there exists an accepting computation where the number of non-empty cells on the work-tape(s) is bounded by $S(d)$ where $d \leq n$, and d is the number of input tape cells already read, that is, the distance of the reading head from the left end of the one-way input tape.

Let $\mathcal{L}(1LOG)$, $\mathcal{L}(1LIN)$, $\mathcal{L}(restricted-1LOG)$, and $\mathcal{L}(restricted-1LIN)$ denote the class of languages accepted by one-way nondeterministic Turing machines with logarithmic space bound, linear space bound, restricted logarithmic space bound, and restricted linear space bound, respectively.

If we consider the class of non-erasing linear-space computable mappings, $\mathcal{C}$, then by [12, 13] we obtain

$$\mathcal{L}_{seq,\mathcal{C}}(PA) = \mathcal{L}(restricted-1LOG) \text{ and}$$

$$\mathcal{L}_{maxpar,\mathcal{C}}(PA) = \mathcal{L}(restricted-1LIN) = \mathcal{L}(CS).$$

The second statement was proved by simulating counter machines (counter automata).

The idea of describing language accepting power of P automata in terms of one-way non-deterministic Turing machines with restricted space complexity above, appeared in [29] and [30]. In these papers, so-called symport/antiport P system acceptors were examined. These accepting membrane systems are similar to P automata, the main difference in the two models is the fact that the alphabet of symport/antiport acceptors is divided into a set of terminal and nonterminal objects. Both terminals and nonterminals may leave or enter the P system but only terminals form the part of the input sequence which is accepted in a successful computation. The nonterminal objects serve for providing additional workspace for the computation. This feature motivated the introduction of $S(n)$ space bounded symport/antiport acceptors, systems where the total number of objects used in an accepting computation on a sequence of length n is bounded by a function $S(n)$. Notice that P automata do not distinguish between terminal and nonterminal objects; if we introduce such distinction, then we speak of extended P automata.

If we use arbitrary (possibly erasing) linear space computable mappings for the input multisets of the P automaton to obtain the alphabet of the accepted

language, then we obtain a description of the class of recursively enumerable languages. That is,

for any recursively enumerable language $L \subseteq \Sigma^*$ *there exists a P automaton* $\Pi = (V, \mu, P_1, \ldots, P_n, c_0, \mathcal{F})$, $n \geq 1$, *and a linear space computable mapping* $f : V^* \to \Sigma^*$ *such that* $L = L_{maxpar,f}(\Pi)$ *holds.*

In the following we will discuss mapping f_{perm} that is widely used in P automata theory for assigning language to the P automaton.

Let $f_{perm} \in MAP_{\Sigma,\Sigma}$ be defined in such a way that every multiset over Σ is mapped by f_{perm} to the set of strings which consists of all permutations of the elements of the multiset. This mapping was first considered in [25].

If f_{perm} is composed with a special homomorphism, i.e., when nonterminal and terminal objects are distinguished, then P automata using the maximally parallel working mode describe the class of recursively enumerable languages . However, mapping f_{perm} itself does not provide the necessary power to obtain any context-sensitive language [43].

In [25] it is shown that *any recursively enumerable language* $L \subseteq \Sigma^*$ *can be obtained by as* $L = h(L_{maxpar,f_{perm}}(\Pi))$, *where* Π *is given over object set* Σ, f_{perm} *is defined as above. Furthermore,* $\Sigma = N \cup T$, *where* N *and* T *are disjoint sets of nonterminals and terminals, and* h *is a homomorphism over* Σ^* *onto itself which orders* λ *to any element of* N *and to any element of* T *it orders itself.*

By [22], it holds that for an arbitrary alphabet Σ and any injective mapping $g : \Sigma^* \to \Sigma^*$, $L_g = \{wg(w) \mid w \in \Sigma^*\}$ is not in $L_{maxpar,f_{perm}}(\Pi)$ for any P automaton Π.

Furthermore, it is shown that all families of languages which properly include the family of regular languages and closed under λ-free morphisms contain languages which cannot be obtained as the language of any P automaton working in the maximally parallel mode and using mapping f_{perm} for defining words of the language. This fact implies that there exist context-sensitive languages which cannot be obtained as languages of a P automaton working in the maximally parallel mode and using f_{perm} for defining words of the language, although, any language $L \subseteq \Sigma^*$, where $L = L_{maxpar,f_{perm}}(\Pi)$, where Π has object set Σ, is a context-sensitive language [43].

A precise description of the accepting power of P automata with respect to f_{perm} can be found in [19]. The result was obtained by introducing and using special variants of counter machine acceptors.

A restricted k-counter machine acceptor M (an RCMA) is a (non-deterministic) counter machine (counter automata) with k counters (storing non-negative integers) and a one-way read only input tape. Thus, $M = (Q, \Sigma, k, \delta, q_0, F)$ for some $k \geq 1$, where Q is the set of internal states, Σ is the input alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma^* \times C^k \to 2^{Q \times D^k}$, where $C = \{zero, nonzero\}$, denoting the two types of observations the machine can make on its counters, $D = \{increment, decrement, none\}$ denoting the operations (instructions) the machine can perform on its counters. Furthermore,

– the transition relation is defined in such a way that $\delta(q, x, \alpha) = \delta(q, y, \alpha)$ for each $x, y \in \Sigma^*$ which represent the same multiset, that is, the reading head can read more than one input symbol in one computation step. Moreover,
– the sum of the numbers stored in the counters can only increase as much in one computation step as the number of symbols read in that same step, that is, for all $(q', \beta) \in \delta(q, x, \alpha)$ we have $|\beta|_{increment} - |\beta|_{decrement} \leq |x|$.

Let the class of languages accepted by restricted counter machine acceptors be denoted by $\mathcal{L}$(RCMA).

In [19], it is shown that for $X \in \{seq, maxpar\}$

$$\mathcal{L}_{X,f_{perm}}(PA) \subset \mathcal{L}(restricted - 1LOG) \text{ and}$$

$$\mathcal{L}(\text{RCMA}) = \mathcal{L}(restricted - 1LOG).$$

The properness of the inclusion of $\mathcal{L}_{X,f_{perm}}(PA)$ in $\mathcal{L}(restricted - 1LOG)$, $X \in \{seq, maxpar\}$, is shown by $L = \{(ab)^n \# w \mid w \in \{1\}\{0,1\}^*, val(w) = n > 1\}$ where $val(w)$ denotes the value of w as a binary number.

Furthermore, in [19] it is also shown that *there exists an infinite hierarchy of language classes of P automata using mapping f_{perm} for defining the accepted languages.*

To prove that statement, a further restriction on RCMA is introduced, and properties of this special restricted counter machine acceptor are used.

A special restricted k-counter machine acceptor (an SRCMA) is a restricted k-counter machine acceptor $M = (Q, \Sigma, k, \delta, q_0, F)$ where δ is defined in such a way that if the length of the string x read in one computation step is l, then the sum of the numbers stored in the counters can only increase at most as much as $l - 1$ in the same computation step.

It is shown that *a language L is accepted by a P automaton with input mapping f_{perm}, working in any of the sequential or maximally parallel working modes, if and only if L can be accepted by an SRCMA.*

Furthermore, for P automata working in any of the sequential or the maximally parallel modes it holds that for every natural number r, there is an $s > r$ and a unary language L which can be accepted by a P automaton with input mapping f_{perm} and s membranes, but not by any such P automaton with r membranes.

So far we have recalled results about P automata accepting by final states, mainly by halting. Recently, P automata with infinite runs over finite inputs have obtained increased interest and found to be able to "go beyond Turing".

Such variants of P automata are the so-called red-green P automata, introduced in [2].

The concept is constructed on the analogy of red-green Turing machines [33]. In case of these machines, the set of states is divided into two disjoint parts, the set of green states and the set of red states. The machines are special types of ω-Turing machines with a recognition criterion based on some property of these sets of states. An infinite run of a red-green Turing machine M on input word w is called recognizing if and only if no red state is visited infinitely often and

one or more green states are visited infinitely often. The run starts from a red state. A set of strings L over the input alphabet Σ is said to be accepted by M if and only if every string in L is recognized by M and for every string $u \notin L$ the infinite computation (infinite run) of M on input u eventually stabilizes in red state; we say that u is rejected. When the "color" of the state changes, i.e. from red to green or vice versa, then we speak of a "mind change".

Red-green Turing machine recognize any recursively enumerable language with one mind change and vice versa. Furthermore, red-green Turing machines recognize exactly the Σ_2-sets of the Arithmetical Hierarchy, and they accept exactly those sets which are in $\Sigma_2 \cap \Pi_2$ (Π_2-sets are in the Arithmetical Hierarchy as well).

On the analogy of red-green Turing machines, red-green register machines and red-green counter automata have been defined in [2], and similar results to that of red-green Turing machines have been obtained. In that paper, the concept of a red-green P automata (and its certain variants) have also been introduced and studied, motivated by the fact that P automata (with antiport rules) are able to simulate register machines and counter automata.

Thus, in [2] it was shown that *a set of (finite) multisets L is recognized by a red-green P automaton with one mind change if and only if L is recursively enumerable. Furthermore, red-green P automata recognize exactly the Σ_2-sets and they accept exactly those sets which are in $\Sigma_2 \cap \Pi_2$.*

An important development of the description of the accepting power of P automata with infinite run can be found in [23], where the authors assigned observer languages to infinite computations of the P automaton.

It was shown that *if L is a language recognized by a red-green P automaton using an observer language $F \subseteq \{0,1\}^\omega$, then the following hold: if $F \in \Sigma_2$, then $L \in \Sigma_2$; if $F \in \Pi_2$, then $L \in \Pi_2$; and if F is regular and $F \in \Sigma_2 \cap \Pi_2$, then $L = \cup_{i=0}^{k}(K_i \setminus L_i)$, where K_i, L_i are recursively enumerable languages for i, $0 \leq i \leq k$.*

Furthermore, *if F is a Boolean combination of ω-languages $F_i, E_i \in \Sigma_2$, where $0 \leq i \leq k$, then $L = \cup_{i=0}^{k}(K_i \setminus L_i)$, where $K_i, L_i \in \Sigma_2$ for i, $0 \leq i \leq k$.*

Moreover, *if language L is a Boolean combination of languages in Σ_2, then it is recognized by a red-green P automaton using a regular observer ω-language F over $\{0,1\}$.*

These results provide a nice characterization of the acceptance condition partial adult halting.

6 dP automata

A particularly important development in P automata theory is the concept of the dP automaton [41], the distributed P automaton. Although P automata are distributed systems, dP automata provide further possibilities to understand parallelism, cooperation, communication, and distribution in terms of purely communicating accepting P systems.

A distributed P automaton (a dP automaton for short) is finite collection of P automata communicating with each other. The notion was introduced [41] with the aim of formulating a model for distributed problem solving in terms of cooperating P automata and to define measures and provide tools for parallelizability of languages.

A dP automaton consists of a finite number of P automaton which have their separate inputs and communicate from skin to skin membranes by means of special antiport-like rules. The input accepted by the dP automaton is the concatenation of the inputs accepted by the component P automata at the halting of the system, namely when no rule of any component or no inter-component communication rule can be performed.

In the following we recall the notion of a dP automaton in a slightly modified form as it was defined in [41], in order to make it conform with the notations used for P automata in the previous sections.

A dP automaton (of degree $n \geq 1$) is a construct $\Delta = (V, \Pi_1, \ldots, \Pi_n, R, \mathcal{F})$, where V is an alphabet, the alphabet of objects; $\Pi_i = (V, \mu_i, P_{i,1}, \ldots, P_{i,k_i}, c_{i,0}, \mathcal{F}_i)$ is a P automaton of degree $k_i \geq 1$, $1 \leq i \leq n$, called the ith component of the system; R is a finite set of rules of the form $(s_i, u/v, s_j)$, $1 \leq i, j \leq n$, $i \neq j$, $uv \in O^+$, called the set of inter-component communication (shortly, communication) rules of Δ; s_k, $1 \leq k \leq n$ denotes the skin membrane of Π_k; $\mathcal{F} \subseteq \mathcal{F}_1 \times \cdots \times \mathcal{F}_n$, is called the set of accepting configurations of Δ.

An inter-component communication rule $(s_i, u/v, s_j)$, $1 \leq i, j \leq n$, $i \neq j$, serves for direct communication between components Π_i and Π_j: a multiset u in the skin region of Π_i is exchanged with a multiset v in the skin region of Π_j.

A configuration of Δ is $((\mu_1, u_{1,1}, \ldots, u_{1,k_1}), \ldots, (\mu_n, u_{n,1}, \ldots, u_{n,k_n}))$, where $u_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, is a multiset over V.

The initial configuration of Δ is the n-tuple

$((\mu_1, w_{1,1}, \ldots, w_{1,k_1}), \ldots, (\mu_n, w_{n,1}, \ldots, w_{n,k_n})) = (c_{1,0}, \ldots, c_{n,0})$ where $c_{i,0}$, $1 \leq i \leq n$, is the initial configuration of component Π_i.

Analogously to P automaton, dP automaton functions by changing its configurations. The components work synchronously, governed by a global clock, using the rules from their own rule sets and the corresponding inter-component communication rules R. The generic variant of dP automata uses the nondeterministic maximally parallel working mode. Each component Π_i, $1 \leq i \leq n$, takes an input (may be the empty multiset) from the environment, works on it by using the rules in its rule sets $P_{i,1}, \ldots, P_{i,k_i}$ and possibly communicates with the other components by means of rules in R.

A configuration C changes to configuration C' by taking the n-tuple of multisets $(u_1, \ldots, u_n)$ from the environment, denoted by $(u_1, \ldots, u_n, C) \Longrightarrow C'$, if C' can be obtained from C by applying the rule sets of Δ (including R) such that u_i enters the skin region of Π_i from the environment, $1 \leq i \leq n$.

A computation in Δ is a sequence of configurations directly following each other, starting from the initial configuration; it is accepting if it enters one of the accepting configurations of $\mathcal{F} \subseteq \mathcal{F}_1 \times \ldots \times \mathcal{F}_n$. If the components accept by final states, then $\mathcal{F} = \mathcal{F}_1 \times \cdots \times \mathcal{F}_n$, or if Δ accepts by halting, then $\mathcal{F} \subseteq \mathcal{F}_1 \times \cdots \times \mathcal{F}_n$,

it contains the direct product of those halting configurations of the components which are also halting configurations of Δ.

Δ accepts the n-tuple $(\alpha_1, \ldots, \alpha_n)$, where α_i, $1 \leq i \leq n$, is a sequence of multisets over V, if the component Π_i, starting from its initial configuration, performing computation steps in the non-deterministic maximally parallel working mode, takes from the environment the multiset sequence α_i, $1 \leq i \leq n$, and Δ eventually enters an accepting configuration.

As in the case of P automata, we may associate languages to the dP automaton $\Delta = (V, \Pi_1, \ldots, \Pi_n, R, \mathcal{F})$, $n \geq 1$.

The *(concatenated) language* of Δ over an alphabet Σ with respect to the mapping $f = (f_1, \ldots, f_n)$ for $f_i \in MAP_{V,\Sigma}$, $1 \leq i \leq n$, is defined as

$$L_{concat,f}(\Delta) = \{w_1 \ldots w_n \in \Sigma^* \mid w_i = f_i(v_{i,1}) \ldots f_i(v_{i,s_i}) \text{ and } \alpha_i = v_{i,1} \ldots v_{i,s_i},\ 1 \leq i \leq n, \text{ for an } n\text{-tuple of accepted multiset sequences } (\alpha_1, \ldots, \alpha_n)\}.$$

The notion was introduced in [41] with mapping f_{perm}, defined above; i.e., for every mapping f_i, $f_{perm} = f_i$ holds. For the sake of brevity, in case of dP automata we use/we refer to f_{perm} in the previously given meaning.

As for P automata, the choice of f essentially influences the power of the components, and thus, the power of the whole dP automaton. Although most of the investigations concerning dP automata uses f_{perm} for defining the language(s), other mappings would also be interesting.

In the following we denote by $\mathcal{L}_{concat,f,n}(dP)$ the family of all languages recognized by dP automata with n components, $n \geq 1$, where the dP automaton uses the non-deterministic maximally parallel working mode. If its language is defined by f_{perm}, then we may write $\mathcal{L}_{concat,n}(dP)$.

If the number of components is irrelevant, then we use notation $\mathcal{L}_{concat,f}(dP)$. To simplify the notation, in case of mapping f_{perm} we write $\mathcal{L}_{concat}(dP)$.

Observing dP automaton, it is easy to notice the similarity with multitape (multihead) automaton: the current configuration of the n-tuple of membranes (supposed that the system consists of n components) corresponds to the state of the automaton, the strings (multisets) that already have been processed represent the part of the input string on the corresponding tape that has already been read. However, since the number of configurations of a dP automaton can be arbitrarily large, to find direct correspondence between different types of multitape (multihead) automata and dP automata, new definitions of the accepted languages of dP automata and restrictions for their configurations have to be given.

To this aim, one reasonable candidate is the so-called finite dP automaton: a dP automaton Δ is called finite, if the number of configurations reachable from its initial configuration is finite [41]. Notice in this case the set of configurations may correspond to states of a finite state control.

To describe strings scanned/accepted by a multitape (multihead) automaton, two variants of languages based on agreement of the components of a dP automaton were introduced in [18].

The weak agreement language of a dP automaton Δ over an alphabet Σ with respect to a mapping $f = (f_1, \ldots, f_n)$ for $f_i \in MAP_{V,\Sigma}$, $1 \leq i \leq n$, is defined as

$$L_{w,agree,f}(\Delta) = \{w \in \Sigma^* \mid w = f_i(v_{i,1}) \ldots f_i(v_{i,s_i}) = f_j(v_{j,1}) \ldots f_j(v_{j,s_j}) \text{ for all } 1 \leq i, j \leq n, \text{ where } \alpha_i = v_{i,1} \ldots v_{i,s_i},\ 1 \leq i \leq n, \text{ and } (\alpha_1, \ldots, \alpha_n) \text{ is an } n\text{-tuple of accepted multiset sequences of } \Delta\}.$$

The *strong agreement language* of Δ over an alphabet Σ with respect to a mapping $f = (g, \ldots, g)$ for $g \in MAP_{v,\Sigma}$, is defined as

$$L_{s,agree,f}(\Delta) = \{w \in \Sigma^* \mid\ w = g(v_1) \ldots g(v_s) \text{ and } \alpha = v_1 \ldots v_s, \text{ for an } n\text{-tuple of accepted multiset sequences } (\alpha, \ldots, \alpha) \text{ of } \Delta\}.$$

The strong agreement language consists of all words which can be accepted in such a way that all components accept the same sequence of multisets and their languages are defined with the same mapping. In the case of weak agreement languages, the accepted multiset sequences can be different, only the equality of their images should hold.

7 Accepting Power of dP Automata

Since P automata, i.e., dP automata with only one component can be as powerful as Turing machines, due to their ability of working with an exponential amount of workspace (in polynomial time), the large accepting power of dP automata is not surprising.

In [22], [43] it is shown that

$\mathcal{L}(REG) \subset \mathcal{L}_{concat,1}(dP)$ and $\mathcal{L}_{concat}(dP) \subset \mathcal{L}(CS)$.

In [22] *it is shown that for every recursively enumerable language $L \subseteq V^*$, there is a language $L' \in \mathcal{L}_{concat}(dP)$ and an alphabet U disjoint from V such that $L' \subseteq LU^*$ and for each $w \in L$ there is an $y \in U^*$ such that $wy \in L'$.*

Furthermore, $\mathcal{L}_{concat,n}(dP)$, $n \geq 1$, *forms a proper hierarchy according to inclusion* [44].

In [18] a direct correspondence between the language family of one-way multihead finite automata and that of finite dP automata is presented.

A multihead finite automaton as a usual finite automaton has a finite state control and an input tape. But, unlike usual finite automaton, it may have more than one heads reading the same input word; the heads may scan the input symbol and move when the state of the automaton changes. Acceptance is defined as in the one-head case: an input string is accepted if starting from the beginning of the word with all heads (that never leave the input word), the automaton enters an accepting state. Analogously to the one-head case, deterministic and non-deterministic, one-way and two-way variants are considered. (If the heads are allowed to move in both directions, the automaton is called two-way, if only from left to right, then one-way.) The class of languages accepted by one-way

k-head finite automata is denoted by $\mathcal{L}(1\text{NFA}(k))$ and the class of languages accepted by two-way k-head finite automata with $\mathcal{L}(2\text{NFA}(k))$. For a survey of results on these constructs consult [28].

In [18] it was shown that *the weak agreement language of any finite dP automaton is equal to the language of a one-way multi-head automaton, and furthermore, the language of any one-way finite multi-head automaton can be obtained as the strong or weak agreement language of a finite dP automaton.*

Analyzing the way of establishing correspondence between finite dP automata and one-way multi-head finite automata, it was proved that using so-called double alphabets, two-way multi-head finite automata can be represented in terms of dP automata as well [18].

An alphabet of the form $V \cup \bar{V}$, where V is an alphabet itself and $\bar{V} = \{\bar{a} \mid a \in V\}$ is called a double alphabet [4].

A dP automaton $\Delta = (V', \Pi_1, \ldots, \Pi_k, R, \mathcal{F})$ where $V' = V \cup \bar{V}$ is a double alphabet is called a two-way dP automaton if any multiset u_i which enters component Π_i, $1 \leq i \leq k$, in a computation consists of either objects of V, or objects of $\bar{V}$, or it is the empty multiset.

Following the approach of [4], to describe the two-way motion of a head of a two-way multi-head finite automaton in terms of two-way dP automata, so-called two-way trails and two-way multiset trails, i.e., strings and multisets over double alphabets were defined in [18]. The notions of the strong agreement language and the weak agreement language of a two-way dP automaton were obtained from the corresponding notions of (one-way) dP automaton by the obvious modifications.

In [18] it was shown that *any language which is the weak agreement language of a two-way finite dP automaton is equal to the language of a two-way multi-head finite automaton, furthermore, the language of any two-way multi-head finite automaton for is equal to the strong or weak agreement language of a two-way finite dP automaton.*

Since $\mathcal{L}(1LOG)$ is equal to the class of languages of two-way multi-head automata, the above statements provide characterizations of this important complexity class in terms of finite dP automata.

8 Discussion

The theory of P automata as can be developed in several directions. One possible way is to study their relation to classical automata variants, standard and non-standard features.

The following two research topics were raised in [10] but they have not been elaborated yet. However, they might give new launch to studies of P automata.

The P automaton with dynamically varying structure combines properties of self-configurating systems and systems re-configurating theirselves under control coming from outside, since both the objects inside the regions and entering the system from the environment can launch a re-configuration in the membrane structure. It would be interesting to examine the decidability of whether or not

re-configuration takes place during the functioning of the system and if it is the case to what extent the membrane structure changes.

One other problem is the following. P automaton is suitable for modelling variants of weighted systems in a natural manner: the multiplicity of the object in a finite multiset may represent its weight in the multiset. In this way, we may order weights to rules and to objects as well, thus we can build a bridge between special variants of weighted automata and P automata.

The following research topics were raised in [9], but they are also not elaborated yet. Since membrane systems are nested architectures, investigations in relations of P automata theory to the theory of data languages, a theory mainly motivated by applications in XML databases and parametrized verification, are of particular importance. In particularly interesting would be to study their relation to words with nested data and to high-order multi-counter automata.

Another promising research direction would be to study P automata as models for complex, natural systems, since they can be considered as collections of agents which are in communication (interaction) with their environments. (Initial ideas and proposals were presented in [11]). In this research direction several interesting open problems are waiting for future investigations: how to interpret emergent phenomena, non-linearity, interaction complexity, behavioural complexity, etc. in terms of P automata.

9 Acknowledgement

This work was supported by NKFIH (National Research, Development, and Innovation Office), Hungary, Grant no. K 120558.

References

1. Alhazov, A., Freund, R., Ivanov, S., Oswald, M., Verlan, S.: Chocolate P Automata. In: Graciani, G. et al. (eds): Pérez-Jiménez Festschrift, LNCS 11270, 1-20. Springer (2018)
2. Aman, B., Csuhaj-Varjú, E., Freund, R.: Red-green P Automata. In: Gheorghe, M. et al (eds), Proc. CMC15, LNCS 8961, 139-157. Springer (2014)
3. Bel-Enguix, G., Gramatovici, R.: Parsing with P automata. In: Ciobanu,G., Pérez-Jiménez, M., Păun, G. (eds.), Applications of Membrane Computing. Natural Computing Series, pp. 389-410. Springer, Berlin (2006)
4. Birget, J.-C.: Two-way automaton computations, RAIRO Informatique Théorique et Application 24, 44-66 (1990)
5. Budnik, P.: What is and What Will be. Mountain Math Software, (2006)
6. Cardelli, L.: Brane Calculi. Interactions of biological membranes. In: Danos, V., Schacter, V. (eds.) Computational Methods in Systems Biology. International Conference CMSB 2004, Paris, France, May 2004, Revised Selected Papers. LNCS 3082, 257-280. Springer-Verlag, Berlin (2005)
7. Ciencialová L., Csuhaj-Varjú, E., Cienciala, L., Sosík, P.: P Colonies. Bulletin of the of the International Membrane Computing Society 2, 129-156. (2016)

8. Csuhaj-Varjú, E.: P automata. In: Mauri, G., Păun, G., Pérez-Jiménez, M., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing: 5th International Workshop, WMC 2004, Milan, Italy, June, 14-16, 2004. Revised Selected and Invited Papers. LNCS 3365, pp. 19-35. Springer, Berlin (2005)
9. Csuhaj-Varjú, E.: P automata: Concepts, Results, and New Aspects. In: Păun, G. et. al. (eds.) WMC 2009. LNCS 5957, 1-15. Springer, Berlin (2010)
10. Csuhaj-Varjú, E.: P and dP Automata: Unconventional versus Classical Automata. In: H-Ch. Yen, O. H. Ibarra (eds.): Developments in Language Theory - 16th International Conference, DLT 2012, Taipei, Taiwan, August 14-17, 2012. Proceedings. LNCS 7410, 7-22. Springer (2012)
11. Csuhaj-Varjú, E. : P automata: Automata-like constructs modeling complex natural systems. In: Bensch, S. et al. (eds.), NCMA 2013, Proceedings, books@ocg.at 294, 13-30. OCG, Vienna (2013)
12. Csuhaj-Varjú, E., Ibarra, O.H., Vaszil, G.: On the computational complexity of P automata. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA Computing, 10th International Workshop on DNA Computing, DNA10, Milan, Italy, June 7-10, Revised Selected Papers. LNCS 3384, 77-90. Springer (2005)
13. Csuhaj-Varjú, E., Ibarra, O.H., Vaszil, G.: On the computational complexity of P automata. Natural Computing 5(2), 109-126 (2006)
14. Csuhaj-Varjú, E., Oswald, M., Vaszil, G.: P automata. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing. Chapter 6, 144-167. Oxford University Press, Oxford (2010)
15. Csuhaj-Varjú, E., Vaszil, G.: P automata. In: Păun, G., Zandron, C. (eds.), Pre-Proceedings of the Workshop on Membrane Computing WMC-CdeA 2002, Curtea de Argeş, Romania, August 19-23, 2002. Pub. No. 1 of MolCoNet-IST-2001-32008, 177-192. (2002)
16. Csuhaj-Varjú, E., Vaszil, G.: P automata or purely communicating accepting P systems. In: Păun, G., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing. International Worskhop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers. LNCS 2597, 219-233. Springer, Berlin (2003)
17. Csuhaj-Varjú, E., Vaszil, G.: (Mem)brane automata. Theoretical Computer Science 404(1-2), 52-60 (2008)
18. Csuhaj-Varjú, E., Vaszil, G.: Finite dP Automata versus Multi-head Finite Automata In: Gheorghe, M. et. al. (eds.) CMC 2011, LNCS 7184, 120-138. Springer-Verlag, Berlin Heidelberg (2012)
19. Csuhaj-Varjú, E., Vaszil, G.: P automata with restricted power. International Journal of Foundations of Computer Science 25(4), 391-408 (2014)
20. Dassow, J., Vaszil, G. : P finite automata and regular languages over countably infinite alphabets. In: Hoogeboom, H-J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing. 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006, Revised, Selected, and Invited Papers. LNCS 4361,367-381. Springer-Verlag, Berlin (2006)
21. Freund, R.: P Automata: New ideas and results. In: Bordihn, H. et al. (eds). Proceedings NCMA 2016, 13-40. OCG, Vienna (2016)
22. Freund, R., Kogler, M., Păun, G., Pérez-Jiménez, M. J.: On the power of P and dP automata. Annals of Bucharest University. Mathematics-Informatics Series 63, 5-22 (2009)
23. Freund, R., Ivanov, S, Verlan S.: Going Beyond Turing with P Automata: Partial Adult Halting and Regular Observer ω-Languages. In: Calude, S. and Dinneen, M.J. (eds): UNCN 2015, LNCS 9252, 169-180, Springer (2015)

24. Freund, R., Martín-Vide, C., Obtułowicz, A., Păun, G.: On three classes of automata-like P systems. In: Ésik, Z., Fülöp, Z. (eds.) Developments in Language Theory. 7th International Conference, DLT 2003, Szeged, Hungary, July 7-11, 2003. Proceedings. LNCS 2710, 292-303. Springer, Berlin (2003)
25. Freund, R., Oswald, M.: A short note on analysing P systems. Bulletin of the EATCS 78, 231-236 (2002)
26. Freund, R., Oswald, M., Staiger, L.: ω-P automata with communication rules. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing. International Workshop, WMC 2003, Tarragona, Spain, July 17-22, 2003. Revised Papers. LNCS 2933, 203-217. Springer, Berlin (2004)
27. Freund, R., Verlan, S.: (Tissue) P systems working in the k-restricted minimally parallel derivation mode. In: Csuhaj-Varjú, E., Freund, R., Oswald, M., Salomaa, K. (eds.) International Workshop on Computing with Biomolecules, August 27, 2008, Wien, Austria, 43-52. OCG, Vienna (2008)
28. Holzer, M., Kutrib, M., Malcher, A.: Complexity of multi-head finite automata: Origins and directions. Theoretical Computer Science 412, 83-96 (2011)
29. Ibarra, O. H.: The Number of Membranes Matters. In: Alhazov, A., Martín-Vide, C., Păun, Gh. (eds.) Workshop on Membrane Computing, WMC-2003, Tarragona, July 17-22, 2003, pp. 273-285. Technical Report 28/03 of the Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain (2003)
30. Ibarra, O. H. On the Computational Complexity of Membrane Systems. Theoretical Computer Science 320(1), 89-109 (2004)
31. Ibarra, O.H., Păun G.: Characterization of context-sensitive languages and other language classes in terms of symport/antiport P systems. Theoretical Computer Science 358(1), 88-103 (2006)
32. Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science 134, 329-363 (1994)
33. van Leeuwen, J., Wiedermann, J. : Computation as an Unbounded Process. Theoretical Computer Science 429 (2012), 202–212.
34. Martín-Vide, C., Păun, A., Păun, G.: On the power of P systems with symport rules. Journal of Universal Computer Science 8, 317-331 (2002)
35. Minsky, M. L. : Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, New Yersey (1967)
36. Oswald, M.: P Automata. PhD dissertation, Vienna University of Technology, Vienna (2003)
37. Otto, F.: Classes of regular and context-free languages over countably infinite alphabets. Discrete Applied Mathematics 12, 41-56 (1985)
38. Păun, A., Păun, G.: The power of communication: P systems with symport/antiport. New Generation Computing 20(3), 295-305 (2002)
39. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108-143 (2000)
40. Păun, G.: Membrane Computing. An Introduction. Springer Verlag, Berlin-Heidelberg (2002)
41. Păun, G., Pérez-Jiménez, M. J.: Solving Problems in a Distributed Way in Membrane Computing: dP Systems. Int. J. of Computers, Communication & Control, V(2), 238-250 (2010)
42. Păun, G., Pérez-Jiménez, M. J.: P and dP automata: A survey. In: Calude, C.S., Rozenberg, G., Salomaa, A. (eds). Rainbow of Computer Science. LNCS 6570, 102-115. Springer, Berlin (2011)
43. Păun, G., Pérez-Jiménez, M. J.: P automata revisited. Theoretical Computer Science 454, 222-230 (2012)

44. Păun, G., Pérez-Jiménez, M. J.: An Infinite Hierarchy of Languages Defined by dP Systems. Theoretical Computer Science, 431, 4-12 (2012)
45. Păun, G., Rozenberg, G., Salomaa, A.(eds.) The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
46. Rozenberg, G., Salomaa, A. (eds.), Handbook of Formal Languages, Vol.I-III, Springer-Verlag, Berlin, Heidelberg (1997)
47. Staiger, L.: ω-languages. In: G. Rozenberg, A. Salomaa (eds.): Handbook of Formal Languages, 339-387. Springer, (1997)
48. Vaszil, G.: Automata-like membrane systems - A natural way to describe complex phenomena. In: Campeanu, C., Pighizzini, G. (eds.) 10th International Workshop on Descriptional Complexity of Formal Systems, July 16-18, Charlottetown, PE, Canada. Proceedings. University of Prince Edwards Island, 26-37. Charlottetown (2008)