Chocolate P Automata

Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Marion Oswald, Sergey Verlan

Chocolate P Automata

Artiom Alhazov[1], Rudolf Freund[2(✉)], Sergiu Ivanov[3], Marion Oswald[2], and Sergey Verlan[4]

[1] Institute of Mathematics and Computer Science, Academiei 5, 2028 Chişinău, Moldova
artiom@math.md
[2] Faculty of Informatics, TU Wien, Favoritenstraße 9–11, 1040 Vienna, Austria
{rudi,marion}@emcc.at
[3] IBISC, Université Évry, Université Paris-Saclay, 23 Boulevard de France, 91025 Évry, France
sergiu.ivanov@univ-evry.fr
[4] Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est Créteil, 61 Avenue du Général de Gaulle, 94010 Créteil, France
verlan@u-pec.fr

Abstract. We introduce several variants of input-driven tissue P automata – we also will call them *chocolate automata* – where the rules to be applied only depend on the input symbol. Both strings and multisets are considered as input objects; the strings are either read from an input tape or defined by the sequence of symbols taken in, and the multisets are given in an input cell at the beginning of a computation, enclosed in a vesicle. Additional symbols generated during a computation are stored in this vesicle, too. An input is accepted when the vesicle reaches a final cell and it is empty. The computational power of some variants of input-driven tissue P automata (*chocolate automata*) is illustrated by examples and compared with the power of the input-driven variants of other automata as register machines and counter automata.

1 Introduction

In the basic model of membrane systems as introduced at the end of the last century by Gheorghe Păun, e.g., see [9,30], the membranes are organized in a hierarchical membrane structure (i.e., the connection structure between the compartments/regions within the membranes being representable as a tree), and the multisets of objects in the membrane regions evolve in a maximally parallel way, with the resulting objects also being able to pass through the surrounding membrane to the parent membrane region or to enter an inner membrane. Many variants of membrane systems, for obvious reasons mostly called *P systems*, have been investigated during nearly two decades, most of them being computationally complete, i.e., being able to simulate the computations of register machines. If an arbitrary graph is used as the connection structure between the cells/membranes, the systems are called *tissue P systems*, see [21].

Instead of multisets of plain symbols coming from a finite alphabet, P systems quite often operate on more complex objects (e.g., strings, arrays), too. A comprehensive overview of different variants of (tissue) P systems and their expressive power is given in the handbook which appeared in 2010, see [32]. For a short view on the state of the art on the domain, we refer the reader to the P systems website [35] as well as to the Bulletin series of the International Membrane Computing Society [34].

The notion and concept of input-driven push-down automata goes back to the seminal paper [22] as well as the papers [6,10] improving the complexity measures shown in [22]. The main idea of input-driven push-down automata is that the input symbols uniquely determine whether the automaton pushes a symbol, pops a symbol, or leaves the pushdown unchanged. Input-driven push-down automata have been rediscovered at the beginning of this century under the name of visibly pushdown automata, see [3,4]. Since then, variants of input-driven push-down automata have gained growing interest, especially because closure properties and decidable questions of the language classes defined by these devices turn out to be similar to those of regular languages. Several new variants of input-driven automata have been developed, for example, using stacks or queues, see [5,19,20]. For complexity issues of input-driven push-down automata, the reader is referred to [24–27].

The so-called *point mutations*, i.e., *insertion*, *deletion*, and *substitution*, which mean inserting or deleting one symbol or replacing one symbol by another one in a string or multiset are very simple biologically motivated operations. For example, on strings graph-controlled insertion-deletion systems have been investigated in [13], and P systems using these operations at the left or right end of string objects were introduced in [16], where also a short history of using these point mutations in formal language theory can be found.

The operations of insertion and deletion in multisets show a close relation with the increment and decrement instructions in register machines. The power of changing states in connection with the increment and decrement instructions then can be mimicked by moving the whole multiset representing the configuration of a register machine from one cell to another one in the corresponding tissue system after the application of an insertion or deletion rule. Yet usually moving the whole multiset of objects in a cell to another one, besides maximal parallelism, requires *target agreement* between all applied rules, i.e., that all results are moved to the same target cell, e.g., see [15].

A different approach has been introduced in [2]: in order to guarantee that the whole multiset is moved even if only one point mutation is applied, the multiset is enclosed in a vesicle, and this vesicle is moved from one cell to another one as a whole, no matter if a rule has been applied or not. Requiring that one rule has to be applied in every derivation step, a characterization of the family of sets of (vectors of) natural numbers defined by partially blind register machines, which itself corresponds with the family of sets of (vectors of) natural numbers obtained as number (Parikh) sets of string languages generated by graph-controlled or matrix grammars without appearance checking, is obtained.

The idea of using vesicles of multisets has already been used in variants of P systems using the operations drip and mate, corresponding with the operations cut and paste well-known from the area of DNA computing, see [14]. Yet in that case, always two vesicles (one of them possibly an axiom available in an unbounded number) have to interact. In the model as introduced in [2] and also to be adapted in this paper, the rules are always applied to the same vesicle. The *point mutations*, i.e., *insertion*, *deletion*, and *substitution*, well-known from biology as operations on DNA, have also widely been used in the variants of *networks of evolutionary processors (NEPs)*, which consist of cells (processors) each of them allowing for specific operations on strings, and in each derivation step, after the application of a rule, allow the resulting string to be sent to another cell provided specific conditions (for example, random context output and input filters). A short overview on NEPs is given in [2], too.

In this paper, we now introduce input-driven tissue P automata – which we will also call *chocolate automata* – where the rules to be applied only depend on the input symbol. Taking strings as input objects, these are either read from an input tape or defined by the sequence of symbols taken in, and as a kind of additional storage we use a multiset of different symbols enclosed in a vesicle which moves from one cell of the tissue P system to another one depending on the input symbol; the input symbol at the same time also determines whether (one or more) symbols are added to the multiset in the vesicle or removed from there. The given input is accepted if the whole input has been read and the vesicle has reached a final cell and is empty at this moment. When using multisets as input objects, these are enclosed in the vesicle in the input cell at the beginning of a computation, which vesicle then will also carry the additional symbols. The given input multiset is accepted if no input symbols are present any more and the vesicle has reached a final cell and is empty at this moment.

As rules operating on the multiset enclosed in the vesicle when reading/consuming an input symbol we use insertion, deletion, and substitution of multisets, applied in the sequential derivation mode. As restricted variants, we consider systems without allowing substitution of multisets and systems only allowing symbols to be inserted or deleted (or substituted) as it is common when using point mutation rules.

Multiset automata have already been considered in [7], where models for finite automata, linear bounded automata, and Turing machines working on multisets are discussed. When dealing with multisets only, the tissue P automata considered in this paper can be seen as one of the variants of multiset pushdown automata as investigated in [18], where no checking for the emptiness of the multiset *memory* during the computation is possible. Various lemmas proved there then can immediately be adapted for our model. Moreover, also the input-driven variants can be defined in a similar manner, although input-driven multiset pushdown automata have not yet been considered in that paper.

We should also like to mention that the control given by the underlying communication structure of the tissue P system could also be interpreted as having a P system with only one membrane but using states instead. For a

discussion on how to use and interpret features of (tissue) P systems as states we refer to [1], where also an example only using the point mutation rules insertion and deletion is given. Moreover, we will also consider another alternative model very common in the P systems area, i.e., P systems with antiport and symport rules, which were introduced in [29]; for overviews on P automata, we refer to [32], Chapter 5, [31], and [11]. One-membrane P systems using antiport rules in a sequential manner and with specific restrictions on the rules then are an adequate model for (input-driven) P automata, yet the restrictions are less visible than in the model of input-driven tissue P automata. On the other hand, when dealing with strings instead of multisets, the way how to read or define the input string in P systems with antiport rules has already been investigated thoroughly, e.g., see [8,11,28] for an overview.

The *sweet* title "chocolate automata"[1] is motivated by the following short story, fictive, but based on long-term experiences with the fruitful and inspiring meetings in Sevilla, known as the *Brainstorming Weeks on Membrane Computing*:

> Preparing for the forthcoming week in Sevilla, expecting to meet many friends and colleagues as well as to have long nights of intensive discussions with his friends from Moldova, Artiom, Sergiu, and Sergey, Rudi thinks about how to fill his bag with a lot of chocolates. Moreover, a special birthday anniversary has to be celebrated, so some special chocolate cake is needed for this occasion. Starting to buy the cake, Rudi visits the famous Sacher in Vienna, and a big *Sacher Torte* as well as some other special Sacher sweets find their way into Rudi's *chocolate bag*.
>
> A lot more sweets are expected to be needed, so Rudi at his home town Stockerau visits several stores to buy Austrian sweets like the famous *Mozart Kugeln*. With his *chocolate bag* well filled, Rudi now is ready and *happy* to start his journey from Vienna to Sevilla together with Marion. The friendly atmosphere established by Mario's Sevillan group immediately invites the teams from Austria – Rudi and Marion – and from Moldova – Artiom, Sergiu, and Sergey – to discuss new ideas on membrane computing. During the whole *Brainstorming Week*, a lot of chocolate is needed as brain fuel for the team members.
>
> The famous churros are announced to be served in the middle of the week, during the morning coffee break; hence, to not interfere with this tradition, already on the second day the *Sacher Torte* is presented to Mario on the occasion of his special *birthday* anniversary, and he happily shares it with the participants of the meeting during the morning coffee break.
>
> At the end of the *Brainstorming Week*, special *chocolate awards* are given to some participants: as usual, Artiom has had the most questions

[1] The idea of "chocolate automata" first came up in the relaxed atmosphere of the conference dinner at *AFL 2017*, the 15th International Conference on Automata and Formal Languages, taking place in Debrecen, Hungary, at the beginning of September, 2017; the ideas initiated there then were further developed during the *Brainstorming Week on Membrane Computing* at the beginning of February, 2018.

during all the talks, so he gets the *chocolate award* for the "most active participant in discussions". From all the young researchers present in Sevilla, Sergiu has contributed the most with new ideas especially on the last day, when results obtained during the current *Brainstorming Week* have been presented; therefore, he receives the *chocolate award* as the "most innovative young P scientist". During the closing ceremony, the members of the Sevillan group of *Mario* finally get a lot of chocolates as a special thank-you gift for their outstanding friendly organization.

After the *Brainstorming Week* Rudi returns home to Vienna together with Marion, with his *chocolate bag* being empty, but with his brain full of new "P ideas" obtained based on the discussions with the participants of the meeting, especially with his friends from Moldova.

Interpreting this story in an abstract way, the different chocolate sorts correspond to the different non-terminal symbols used as intermediate symbols during the computation. The events like going to a specific store as well as the coffee breaks and the award-giving events correspond with the terminal input symbols. There is no time condition on the sequence of these events except that chocolates have to be bought before they can be given away. This perfectly corresponds with the use of a mutiset bag (vesicle) as a storage, where the sequence does not matter as it is the case when dealing with strings stored in the stack of a pushdown automaton. Finally, the acceptance condition of empty vesicle at the end of the computation corresponds with having an empty *chocolate bag* at the end of the *Brainstorming Week*. Even several variants of the input-driven automata model can be derived from this *chocolate story*: for example, it is natural to buy several pieces in one store or to give away several chocolates at the same event, which nicely corresponds with putting more than one symbol into the vesicle or deleting more than one symbol from the vesicle at the same moment when reading/consuming an input symbol.

The rest of the paper now is structured as follows: In Sect. 2 we recall some well-known definitions from formal language theory. The main definitions for the model of (input-driven) tissue P automata as well as its variants to be considered in this paper are given in Sect. 3, and there we also present the definition of the alternative model of (input-driven) one-membrane P automata with (restricted) antiport rules; moreover we also give some first examples and results. Further illustrative examples and some more results, especially for input-driven tissue P automata are exhibited in Sect. 4. As upper bound for the family of sets of vectors of natural numbers accepted by input-driven tissue P automata we get the family of sets of vectors of natural numbers generated by partially blind register machines, and as upper bound for the family of sets of strings accepted by input-driven tissue P automata we get the family of sets of strings accepted by partially blind counter automata. A summary of the results obtained in this paper and an outlook to future research are presented in Sect. 5.

2 Prerequisites

We start by recalling some basic notions of formal language theory. An alphabet is a non-empty finite set of symbols. A finite sequence of symbols from an alphabet V is called a *string* over V. The set of all strings over V is denoted by V^*; the *empty string* is denoted by λ; moreover, we define $V^+ = V^* \setminus \{\lambda\}$. The *length* of a string x is denoted by $|x|$, and by $|x|_a$ we denote the number of occurrences of the symbol a in a string x.

A *multiset* M with underlying set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping, with $\mathbb{N}$ denoting the set of natural numbers (i.e., non-negative integers). If $M = (A, f)$ is a multiset then its *support* is defined as $supp(M) = \{x \in A \mid f(x) > 0\}$. A multiset is empty (respectively finite) if its support is the empty set (respectively a finite set). If $M = (A, f)$ is a finite multiset over A and $supp(M) = \{a_1, \ldots, a_k\}$, then it can also be represented by the string $a_1^{f(a_1)} \ldots a_k^{f(a_k)}$ over the alphabet $\{a_1, \ldots, a_k\}$ (the corresponding vector $(f(a_1), \ldots, f(a_k))$ of natural numbers is called Parikh vector of the string $a_1^{f(a_1)} \ldots a_k^{f(a_k)}$), and, moreover, all permutations of this string precisely identify the same multiset M (they have the same Parikh vector). The set of all multisets over the alphabet V is denoted by V°.

The family of all recursively enumerable sets of strings is denoted by RE, the corresponding family of recursively enumerable sets of Parikh vectors is denoted by $PsRE$. For more details of formal language theory the reader is referred to the monographs and handbooks in this area, such as [33].

2.1 Insertion, Deletion, and Substitution

For an alphabet V, let $a \to b$ be a rewriting rule with $a, b \in V \cup \{\lambda\}$, and $ab \neq \lambda$; we call such a rule a *substitution rule* if both a and b are different from λ and we also write $S(a, b)$; such a rule is called a *deletion rule* if $a \neq \lambda$ and $b = \lambda$, and it is also written as $D(a)$; $a \to b$ is called an *insertion rule* if $a = \lambda$ and $b \neq \lambda$, and we also write $I(b)$. The sets of all insertion rules, deletion rules, and substitution rules over an alphabet V are denoted by Ins_V, Del_V, and Sub_V, respectively. Whereas an insertion rule is always applicable, the applicability of a deletion and a substitution rule depends on the presence of the symbol a. We remark that insertion rules, deletion rules, and substitution rules can be applied to strings as well as to multisets. Whereas in the string case, the position of the inserted, deleted, and substituted symbol matters, in the case of a multiset this only means incrementing the number of symbols b, decrementing the number of symbols a, or decrementing the number of symbols a and at the same time incrementing the number of symbols b.

These types of rules and the corresponding notations can be extended by allowing more than one symbol on the left-hand and/or the right-hand side, i.e., $a, b \in V^*$, and $ab \neq \lambda$. The corresponding sets of all extended insertion rules, deletion rules, and substitution rules over an alphabet V are denoted by Ins_V^*, Del_V^*, and Sub_V^*, respectively.

2.2 Register Machines

Register machines are well-known universal devices for computing (generating or accepting) sets of vectors of natural numbers.

Definition 1. *A* register machine *is a construct* $M = (m, B, I, h, P)$ *where*

- m *is the number of registers,*
- B *is a set of labels bijectively labeling the instructions in the set* P*,*
- $I \subseteq B$ *is the set of initial labels, and*
- $h \in B$ *is the final label.*

The labeled instructions of M *in* P *can be of the following forms:*

- $p : (ADD(r), K)$*, with* $p \in B \setminus \{h\}$*,* $K \subseteq B$*,* $1 \leq r \leq m$*.*
  Increase the value of register r *by one, and non-deterministically jump to one of the instructions in* K*.*
- $p : (SUB(r), K, F)$*, with* $p \in B \setminus \{h\}$*,* $K, F \subseteq B$*,* $1 \leq r \leq m$*.*
  If the value of register r *is not zero then decrease the value of register* r *by one (*decrement *case) and jump to one of the instructions in* K*, otherwise jump to one of the instructions in* F *(*zero-test *case).*
- $h : HALT$*.*
  Stop the execution of the register machine.

A configuration *of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.*

In the accepting case, a computation starts with the input of a k-vector of natural numbers in its first k registers and by executing one of the initial instructions of P (labeled with $l \in I$); it terminates with reaching the $HALT$-instruction. Without loss of generality, we may assume all registers to be empty at the end of the computation.

By $\mathcal{L}(RM)$ we denote the family of sets of vectors of natural numbers accepted by register machines. It is well known (e.g., see [23]) that $PsRE = \mathcal{L}(RM)$.

Partially Blind Register Machines. In the case when a register machine cannot check whether a register is empty we say that it is partially blind: the registers are increased and decreased by one as usual, but if the machine tries to subtract from an empty register, then the computation aborts without producing any result (that is we may say that the subtract instructions are of the form $p : (SUB(r), K, abort)$; instead, we simply will write $p : (SUB(r), K)$.

Moreover, acceptance now by definition also requires all registers to be empty at the end of the computation, i.e., there is an implicit test for zero at the end of a (successful) computation, that is why we say that the device is partially blind. By $\mathcal{L}(PBRM)$ we denote the family of sets of vectors of natural numbers

accepted by partially blind register machines. It is known (e.g., see [12]) that partially blind register machines are strictly less powerful than general register machines (hence, than Turing machines); moreover, $\mathcal{L}(PBRM)$ characterizes the Parikh sets of languages generated by graph-controlled or matrix grammars without appearance checking.

2.3 Counter Automata

Register machines can also be equipped with an input tape to be able to process strings, and the registers then are only used as auxiliary storage. We then call the registers *counters* and the automaton a *counter automaton* (we mention that in the literature slightly different definitions with respect to the instructions may be found). The additional instruction needed then is a *read instruction* reading one symbol from the input tape:

$$p : (read(a), K), \text{ with } p \in B \setminus \{h\}, K \subseteq B, \text{ and } a \in T.$$

T is the input alphabet, i.e., in sum we obtain a counter automaton as a construct

$$M = (m, B, I, h, P, T).$$

A counter automaton accepts an input $w \in T^*$ if and only if it starts in some initial state and with w on its input tape, and finally M reaches h having read the whole input string w. Without loss of generality, we again may assume all registers to be empty at the end of the computation.

It is well known (e.g., see [23]) that the family of string languages accepted by counter automata equals RE (in fact, only two counters are needed).

Partially Blind Counter Automata. As in the case of register machines, a counter automaton is called partially blind if it cannot check whether a register is empty, and acceptance by definition requires the whole input to be read and all counters to be empty at the end of the computation. For basic results on partially blind counter automata we refer to the seminal paper [17]. The family of string languages accepted by partially blind counter automata is denoted by $\mathcal{L}(PBCA)$.

2.4 Input-Driven Register Machines and Counter Automata

An input-driven register machine/counter automaton (an $IDRM^*$ and $IDCA^*$, respectively, for short) can be defined in the following way: any decrement of an input register r/any reading of a terminal symbol a is followed by *fixed* sequences of instructions on the working registers/counters only depending on the input register r/the terminal symbol a. If each such sequence is of length exactly one, then we speak of a real-time input-driven register machine/counter automaton (an $IDRM$ and $IDCA$, respectively, for short).

In the case of an $IDCA$, these sequences are of the form

$$p : (read(a), K) \to q : (\alpha(r), K_q),\ q \in K,$$

with $\alpha \in \{ADD, SUB\}$, $1 \leq r \leq m$, and they could be written as *one* extended instruction

$$p : (read(a), \alpha(r), \bigcup\nolimits_{q \in K} K_q).$$

In a similar way, for an $IDCA^*$ we replace $\alpha(r)$ by the whole sequence of instructions following the reading of the input symbol a. A similar notation can be adapted for the case of a SUB-instruction on an input register instead of $read(a)$. Moreover, analogous definitions and notations hold for the partially blind variants of input-driven register machines/counter automata.

Remark 1. We emphasize that we have chosen a very restricted variant of what it means that the actions on the working registers only depend on the input symbol just read: no matter which label the read instruction $read(a)$ has, it must always be followed by the same sequence $\alpha(r)$; only the branching to labels from $\bigcup_{q \in K} K_q$ allows for taking different actions – in fact, read-instructions followed by the corresponding sequences of instructions – afterwards. □

Remark 2. Allowing a set of initial labels as well as sets of labels in the ADD- and SUB-instructions may look quite unusual, but especially for the input-driven automata this feature turns out to be essential:

Assume we had allowed only one initial label i in any input-driven counter automaton. Now consider the finite multiset language $\{a, b\}$: assume there is an input-driven partially blind counter automaton accepting $\{a, b\}$. By definition, the instruction assigned to the initial label i must be a read instruction. With the initial label i, only *one* of the read instructions $read(a)$ or $read(b)$ can be assigned, hence, only a or only b can be accepted, a contradiction.

A similar argument holds for partially blind register machines taking the input set of two-dimensional vectors $\{(1, 0), (0, 1)\}$: the instruction assigned to i must be a SUB-instruction either on register 1 or on register 2, again leading to a contradiction.

On the other hand, with our more general definition, these sets are in $\mathcal{L}(IDCA^*)$ and $\mathcal{L}(IDRM^*)$, respectively. Still, in general we do not have closure under union, as the sequences of instructions after a read-instruction or a SUB-instruction in two different counter automata or register machines, respectively, need not be the same. □

3 Tissue P Automata as Multiset Pushdown Automata

We now define a model of a tissue P automaton and its input-driven variants, first for the case of working with multisets as input objects:

Definition 2. *A* tissue P automaton *(a* tPA* *for short) is a tuple*

$$\Pi = (L, V, \Sigma, \Gamma, R, g, I, F)$$

where

- L *is a* set of labels *identifying in a one-to-one manner the* $|L|$ *cells of the tissue P system* Π*;*
- V *is the* alphabet *of the system;*
- $\Sigma \subseteq V$ *is the (non-empty)* input alphabet *of the system;*
- $\Gamma \subseteq V$ *is the (possibly empty)* memory alphabet *of the system,* $\Gamma \cap \Sigma = \emptyset$*;*
- R *is a* set of rules *of the form* (i, p) *where* $i \in L$ *and* $p \in Ins_V^* \cup Del_V^* \cup Sub_V^*$*, i.e., p is an extended insertion, deletion or substitution rule over the alphabet* V*; we may collect all rules from cell* i *in one set and then write* $R_i = \{(i, p) \mid (i, p) \in R\}$*, so that* $R = \bigcup_{i \in L} R_i$*; moreover, for the sake of conciseness, we may simply write* $R_i = \{p \mid (i, p) \in R\}$*, too;*
- g *is a directed graph describing the underlying communication structure of* Π*,* $g = (N, E)$ *with* $N = L$ *being the set of nodes of the graph* g *and the set of edges* $E \subseteq L \times L$*;*
- $I \subseteq L$ *is the set of labels of* initial cells *one of them containing the input multiset* w *at the beginning of a computation;*
- $F \subseteq L$ *is the set of labels of* final cells*.*

If in the definition above we take $p \in Ins_V \cup Del_V \cup Sub_V$ instead of $p \in Ins_V^* \cup Del_V^* \cup Sub_V^*$, then we speak of a tPA instead of a tPA^*.

A tPA^* Π now works as follows: The computation of Π starts with a vesicle containing the input multiset w in one of the initial cells $i \in I$, and the computation proceeds with derivation steps until a specific output condition is fulfilled.

In each derivation step, with the vesicle enclosing the multiset w being in cell k, one rule from R_k is applied to w and the resulting multiset in its vesicle is moved to a cell m such that $(k, m) \in E$.

As we are dealing with membrane systems, the classic output condition is to only consider halting computations; yet in case of automata, the standard acceptance condition is reaching a final state, which in our case means reaching a final cell h, and, moreover, the vesicle to be empty. We will combine these two conditions to define acceptance in this paper, as with the vesicle being empty no decrement rule can be applied any more and, moreover, it is guaranteed that we have "read the whole input". Only requiring the vesicle to be empty or else requiring to have reached a final cell with the vesicle containing no input symbol any more, are two other variants of acceptance.

The set of multisets accepted by Π is denoted by $Ps_{acc}(\Pi)$. The families of sets of vectors of natural numbers accepted by tPA^* and tPA with at most n cells are denoted by $\mathcal{L}_n(tPA^*)$ and $\mathcal{L}_n(tPA)$, respectively. If n is not bounded, we simply omit the subscript in these notations. In order to specify which rules are allowed in the tPA^* and tPA, we may explicitly specify I^*, D^*, S^* and I, D, S, respectively, to indicate the use of (extended) insertion, deletion, and

substitution rules. For example, $\mathcal{L}(tPA, ID)$ then indicates that only insertion and deletion rules are used.

Remark 3. The model of a tPA^* comes very close to the model of a multiset pushdown automaton as introduced in [18]; in fact, the family of sets of vectors of natural numbers accepted by these multiset pushdown automata equals $\mathcal{L}(tPA^*)$. A formal proof would go far beyond the scope of this short paper, but the basic similarity of these two models becomes obvious when identifying the cells in the tPA^* with the states in the multiset pushdown automaton; moving the vesicle from one cell to another one corresponds to changing the states. As shown for the states of the multiset pushdown automata in [18], we could also restrict ourselves to only one initial as well as only one final cell in the general case, as this does not restrict the computational power of a tPA^*. On the other hand, for any of the following restricted variants this need not be true any more, especially for the input-driven variants defined later; in this context we also remind the arguments given in Remark 2. □

The following result shows that having more than one rule in a cell is not necessary:

Lemma 1. *For any tPA^* Π there exists an equivalent tPA^* Π' such that every cell contains at most one rule.*

Proof (Sketch). Let $\Pi = (L, V, \Sigma, \Gamma, R, g, I, F)$ be a tPA^*. The equivalent tPA^* $\Pi' = (L', V, \Sigma, \Gamma, R', g', I', F')$ then is constructed as follows:

For every cell k with R_k containing n_k rules, instead of cell k we take n_k copies of that cell, cells $(k, 1), \ldots, (k, n_k)$, into Π', each of it containing one of the rules from R_k, say $p_{k,l}$, $1 \leq l \leq n_k$. The connection graph g then has to be enlarged to a graph g' containing all the edges

$$\{((k, l), (j, m)) \mid (k, j) \in g, 1 \leq l \leq n_k, 1 \leq m \leq n_j\}.$$

If cell k contains no rule, we rename it to cell $(k, 1)$, and no rule is contained in this cell, too.

The new sets of labels of initial and final cells are obtained by taking all copies of the original cell labels, i.e., we take

$$I' = \{(k, l) \mid (k \in I, 1 \leq l \leq n_k\},$$
$$F' = \{(k, l) \mid (k \in F, 1 \leq l \leq n_k\}.$$

We now immediately infer $Ps(\Pi) = Ps(\Pi')$. □

Remark 4. Continuing the construction from Lemma 1, it is easy to show how to avoid having more than one final cell: we introduce a new final cell f', i.e., we take $F' = \{f'\}$, with this new cell not containing any rule; moreover, we add all edges

$$\{((k, l), f') \mid ((k, l), (j, m)) \in g', j \in F\}.$$

This new cell corresponds to the label of the final HALT instruction in a register machine or a counter automaton. As f' does not contain a rule, the computation will stop there in any case. □

Remark 5. Having only one initial cell cannot be shown by only using a new structure: we may add two new cells i', i'' containing the rules $I(a)$ and $D(a)$, respectively, for some $a \in V$; the first one i' is used as the only new initial cell having one arc to the second one i'', i.e., (i', i''), from where to branch to the original initial cells as constructed in the proof of Lemma 1, i.e. we add all edges

$$\{(i'', (k, l)) \mid (k, l) \in I'\}.$$

Continuing the discussions from Remarks 2 and 3 we mention that this construction is not feasible for the input-driven variants to be defined in Subsect. 3.2. □

The following result is based on the fact that the insertion, deletion, or substitution of a multiset over V can easily be simulated by a sequence of insertions and deletions:

Lemma 2. *For any tPA^* Π there exists an equivalent tPA Π' even not using substitution rules.*

Now let $\mathcal{L}(mARB)$ denote the family of sets of multisets generated by arbitrary multiset grammars.

Corollary 1. $\mathcal{L}(tPA^*, IDS) = \mathcal{L}(tPA, ID) = \mathcal{L}(mARB) = \mathcal{L}(PBRM)$.

Proof (*Sketch*). The equality $\mathcal{L}(tPA^*, IDS) = \mathcal{L}(tPA, ID)$ follows from the definitions and Lemma 2.

The equality $\mathcal{L}(tPA^*, IDS) = \mathcal{L}(mARB)$ is a consequence of the observation discussed above in Remark 3 that $\mathcal{L}(tPA^*, IDS)$ corresponds to the family of sets of multisets accepted by multiset pushdwon automata as defined in [18]. In a similar way, interpreting the cells in a tissue P automaton as the states of a partially blind register machine and seeing the correspondence of the acceptance conditions, we also infer the equality $\mathcal{L}(tPA^*, IDS) = \mathcal{L}(PBRM)$. The details are left to the reader. □

3.1 Accepting Strings

The tissue P automata defined above can also be used to accept sets of strings by assuming the input string to be given on a separate input tape, from where the symbols of the input string are read from left to right. As when going from register machines to counter automata, we use the additional instruction (*read instruction*) $read(a)$ with $a \in \Sigma$, Σ being the input alphabet. The corresponding automata then are defined as follows:

Definition 3. *A* tissue P automaton for strings *(a* tPAL* *for short) is a tuple*

$$\Pi = (L, V, \Sigma, \Gamma, R, g, I, F)$$

where L, V, Σ, Γ, R, g, I, F are defined as for a tPA^*, except that besides insertion, deletion, and substitution rules we also allow rules of the form $read(a)$ with $a \in \Sigma$, i.e., read instructions.

If we only take rules from $Ins_V \cup Del_V \cup Sub_V$ instead of $Ins_V^* \cup Del_V^* \cup Sub_V^*$, then we speak of a $tPAL$ instead of a $tPAL^*$.

A $tPAL^*$ Π works as follows: The computation of Π starts with the input string on the input tape as well as an empty vesicle in one of the initial cells $i \in I$, and the computation proceeds with derivation steps until the whole input string has been read and the vesicle has reached a final cell, again being empty at the end of the computation.

In each derivation step, with the vesicle enclosing the multiset w being in cell k, one rule from R_k is applied, either reading a symbol from the input tape or affecting w, and the resulting multiset in its vesicle then is moved to a cell m such that $(k, m) \in E$.

The set of strings accepted by Π is denoted by $L(\Pi)$. The families of sets of strings accepted by $tPAL^*$ and $tPAL$ with at most n cells are denoted by $\mathcal{L}_n(tPAL^*)$ and $\mathcal{L}_n(tPAL)$, respectively. If n is not bounded, we simply omit the subscript in these notations. In order to specify which rules are allowed in the $tPAL^*$ and $tPAL$, we again may explicitly specify I^*, D^*, S^* and I, D, S, respectively, to indicate the use of (extended) insertion, deletion, and substitution rules.

As for tissue P automata accepting multisets, also for the ones accepting strings we obtain some similar results as shown above:

Lemma 3. *For any $tPAL^*$ Π there exists an equivalent $tPAL^*$ Π' such that every cell contains at most one rule.*

Lemma 4. *For any $tPAL^*$ Π there exists an equivalent $tPAL$ Π' even not using substitution rules.*

Corollary 2. $\mathcal{L}(tPAL^*, IDS) = \mathcal{L}(tPAL, ID) = \mathcal{L}(PBCA)$.

3.2 Input-Driven Tissue P Automata

We now define the input-driven variants of tPA^* and tPA as well as $tPAL^*$ and $tPAL$:

Definition 4. *A tPA^* $\Pi = (L, V, \Sigma, \Gamma, R, g, I, F)$ is called* input-driven *(and called an* IDtPA* *for short) if the following conditions hold true:*

- *to each cell, (at most) one rule is assigned;*
- *any decrement of an input register r is followed by some* fixed *sequence of instructions on the working registers only depending on the input register r before a cell with the next decrement instruction on an input register is reached. Such a sequence of instructions may even be of length zero.*

If each such sequence is of length exactly one, then we speak of a real-time input-driven tPA^* (a $rtIDtPA^*$ for short).

Definition 5. *A $tPAL^*$ $\Pi = (L, V, \Sigma, \Gamma, R, g, I, F)$ is called* input-driven *(and called an* IDtPAL* *for short) if the following conditions hold true:*

- *to each cell, (at most) one rule is assigned;*
- *any reading of a terminal symbol a by a read instruction $read(a)$ is followed by some* fixed *sequence of instructions on the working registers only depending on the terminal symbol a before a cell with the next read instruction is reached. Such a sequence of instructions may even be of length zero.*

If each such sequence is of length exactly one, then we speak of a real-time input-driven $tPAL^*$ (a $rtIDtPAL^*$ for short).

The corresponding families of sets of vectors of natural numbers and of sets of strings accepted by tissue P automata of type X with X being one of the types $IDtPA^*$, $IDtPA$, $rtIDtPA^*$, $rtIDtPA$ as well as $IDtPAL^*$, $IDtPAL$, $rtIDtPAL^*$, $rtIDtPAL$, are denoted by $\mathcal{L}(X)$.

Remark 6. As already discussed in Remark 1 for input-driven register machines and counter automata, we emphasize that we have chosen a very restricted variant of what it means that the actions on the multiset in the vesicle only depend on the input symbol just read: no matter in which cell we have the read instruction $read(a)$, it must always be followed by the same finite sequence of instructions not including read instructions. □

Remark 7. If we only have SUB-instructions on input registers/read instructions, i.e., if the $tPA^*/tPAL^*$ does not use the vesicle at all for storing any intermediate information of working registers, then such a $tPA^*/tPAL^*$ can be interpreted as a finite automaton accepting a regular multiset/string language. In this case, the condition of not having rules on the vesicle for symbols representing working registers, already subsumes the condition of the P automaton being input-driven. In fact, P systems of that kind exactly characterize the regular multiset/string languages. □

3.3 One-Membrane Antiport P Automata

The idea of using states instead of cells can also be "implemented" by using a well-investigated model of membrane systems using antiport rules:

Definition 6. *A* one-membrane antiport P automaton *(a* 1APA* *for short) is a tuple $\Pi = (V, \Sigma, \Gamma, Q, R, I, F)$ where*

- *V is the* alphabet *of the system;*
- *$\Sigma \subseteq V$ is the (non-empty)* input alphabet *of the system;*
- *$\Gamma \subseteq V$ is the (possibly empty)* memory alphabet *of the system, $\Gamma \cap \Sigma = \emptyset$;*
- *$Q \subseteq V$, $Q \cap (\Gamma \cup \Sigma) = \emptyset$, is the set of* states*;*
- *R is a* set of rules *of the form $pu \to qv$, $p, q \in Q$, $u \in (\Gamma \cup \Sigma)^*$, $v \in \Sigma^*$;*
- *$I \subseteq Q$ is the set of* initial states*;*
- *$F \subseteq Q$ is the set of* final states*.*

The $1APA^*$ can be seen as a membrane system consisting of only one membrane with the rules $pu \to qv$ interpreted as antiport rules $(pu, out; qv, in)$, i.e., the multiset pu leaves the membrane region and the multiset qv enters the membrane region.

Π starts with an input multiset w_0 together with one of the initial states p_0, i.e., with w_0p_0 in its single membrane region, and then applies rules from R until a configuration with only $p_f \in F$ in the membrane region is reached, thus accepting the input multiset w_0.

For antiport P automata the acceptance of strings can be defined without needing an input tape as follows, e.g., see [28]: the rules in R now are of the form $pu \to qv$, $p, q \in Q$, $u \in \Gamma^*$ and $v \in (\Gamma \cup \Sigma)^*$, i.e., the input symbols are now taken from outside the membrane (from the environment); the sequence how the input symbols are taken in defines the input string (we may assume v to contain only one symbol from Σ; otherwise, if in one step several symbols are taken in, we have to take any permutation of these symbols, in which way several input strings are defined).

Using such rules and the interpretation of the input string as defined above, we obtain the model of a *one-membrane antiport P automaton for strings* (a $1APAL^*$ for short).

As in the preceding subsections we now can define specific variants of $1APA^*$ and $1APAL^*$, e.g., the corresponding input-driven automata. Yet as we have introduced these models especially to show the correspondence with an automaton model well-known in the area of P systems, we leave the technical details to the interested reader.

4 Examples and Results

The concepts of $IDtPA^*$ and $IDPBRM^*$ are closely related:

Theorem 1. $\mathcal{L}(IDtPA^*) \subseteq \mathcal{L}(PBRM^*)$ *and*
$\mathcal{L}(IDtPA^*) = \mathcal{L}(IDtPA^*, ID) = \mathcal{L}(IDPBRM^*)$.

Proof (*Sketch*). The inclusion $\mathcal{L}(IDPBRM^*) \subseteq \mathcal{L}(PBRM^*)$ is obvious from the definitions.

The equality $\mathcal{L}(IDtPA^*, ID) = \mathcal{L}(IDPBRM^*)$ follows from the definitions of these types of input-driven automata: as already mentioned earlier, the cells in a tPA^* correspond to the states in a $PBRM$. The acceptance conditions – the vesicle being empty in a final cell in a tPA^* and all registers being empty in a $PBRM$ when reaching the final label – directly correspond to each other, too. Moreover, insertion and deletion rules directly correspond to ADD- and SUB-instructions. Finally, the conditions for the input-driven variants requiring the same actions for a consumed input symbol and the decrement of the corresponding register are equivalent, too.

The equality $\mathcal{L}(IDtPA^*) = \mathcal{L}(IDtPA^*, ID)$ follows from the possibility to simulate substitution rules by a sequence of insertion and deletion rules. This observation completes the proof. □

Using similar arguments as in the preceding proof, now considering read instructions instead of decrements on input registers, we obtain the corresponding result for the string case:

Theorem 2. $\mathcal{L}(IDtPAL^*) \subseteq \mathcal{L}(PBCA^*)$ *and*
$\mathcal{L}(IDtPAL^*) = \mathcal{L}(IDtPAL^*, ID) = \mathcal{L}(IDPBCA^*)$.

In the real-time variants, we cannot use substitution rules in the input-driven tissue P automata, as the simulation by deletion and insertion rules takes more than one step:

Theorem 3. $\mathcal{L}(rtIDtPA, ID) = \mathcal{L}(rtIDPBRM)$ *and*
$\mathcal{L}(rtIDtPAL, ID) = \mathcal{L}(rtIDPBCA)$.

We now illustrate the computational power of input-driven tissue P automata accepting strings by showing how well-known string languages can be accepted. We remark that in all cases the automaton has only one initial label and one final label.

Example 1. The Dyck language L_D over the alphabet of brackets $\{\,[\,,\,]\,\}$ can easily be accepted by the $rtIDtPBCA$ M_D:

$$\begin{aligned} M_D = (1, B = \{1,2,3,4,5\}, l_0 = 1, l_h = 5, P, T = \{\,[\,,\,]\,\}), \\ P = \{1 : (read\,(\,[\,)\,, \{2\}), 2 : (ADD\,(1)\,, \{1,3\}), \\ 3 : (read\,(\,]\,)\,, \{4\}), 4 : (SUB\,(1)\,, \{1,3,5\}), 5 : HALT\}. \end{aligned}$$

L_D can also be accepted by the corresponding $rtIDtPAL$ Π_D:

$$\begin{aligned} \Pi_D = (L = \{1,2,3,4,5\}, V, \Sigma, \Gamma, R, g = (L, E), I = \{1\}, F = \{5\}), \\ V = \{a_1, [\,,\,]\,\}, \\ \Sigma = \{\,[\,,\,]\,\}, \\ \Gamma = \{a_1\}, \\ R = \{(1, read\,(\,[\,)), (2, I\,(a_1)), (3, read\,(\,]\,)), (4, D\,(a_1))\}, \\ E = \{(1,2), (2,1), (2,3), (3,4), (4,1), (4,3), (4,5)\}. \end{aligned}$$

The two constructions elaborated above implement the following definition of a well-formed bracket expression w over the alphabet of brackets $\{\,[\,,\,]\,\}$:

- for every prefix of w, the number of closing brackets] must not exceed the number of opening brackets [;
- the number of closing brackets] in w equals the number of opening brackets [.

Hence, during the whole computation, the (non-negative) difference between the number of opening and the number of closing brackets is stored as the number of symbols a_1; at the end, this number must be zero, which is guaranteed by the acceptance conditions. □

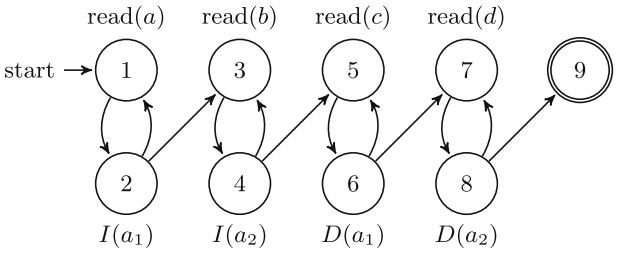

Fig. 1. Graphic representation of the $rtIDtPAL$ Π_{il}.

$\mathcal{L}(rtIDPBCA)$ even contains a non-context-free language:

Example 2. The language $L_{il} = \{a^n b^m c^n d^m \mid m, n \geq 1\}$ is not context-free, but accepted by the following $rtIDtPAL$ Π_{il}:

$$
\begin{aligned}
\Pi_{il} &= (L = \{1, \ldots, 9\}, V, \Sigma, \Gamma, R, g = (L, E), I = \{1\}, F = \{9\}), \\
V &= \{a_1, a_2, a, b, c, d\}, \\
\Sigma &= \{a, b, c, d\}, \\
\Gamma &= \{a_1, a_2\}, \\
R &= \{(1, read(a)), (2, I(a_1)), (3, read(b)), (4, I(a_2)), \\
&\qquad (5, read(c)), (6, D(a_1)), (7, read(d)), (8, D(a_2))\}, \\
E &= \{(1,2), (2,1), (2,3), (3,4), (4,3), \\
&\qquad (4,5), (5,6), (6,5), (6,7), (7,8), (8,7), (8,9)\}.
\end{aligned}
$$

By this construction, we conclude $L_{il} \in \mathcal{L}(rtIDtPAL, ID)$. □

For the language considered in the next example we show that it is in $\mathcal{L}(rtIDtPAL^*)$, but we claim that it is not in $\mathcal{L}(rtIDtPAL)$:

Example 3. Let $k > 2$ and consider the string language $L_k = \{{b_1}^n \ldots {b_k}^n \mid n \geq 1\}$, which is not context-free, but accepted by the following $rtIDtPAL^*$ Π:

$$
\begin{aligned}
\Pi_k &= (L = \{1, \ldots, 2k+1\}, V, \Sigma, \Gamma, R, g = (L, E), I = \{1\}, F = \{2k+1\}), \\
V &= \{a_i, b_i \mid 1 \leq i \leq k\}, \\
\Sigma &= \{b_i \mid 1 \leq i \leq k\}, \\
\Gamma &= \{a_i \mid 1 \leq i \leq k\}, \\
R &= \{(1, read(b_1)), (2, I(a_2 \ldots a_k))\} \\
&\quad \cup \{(2j-1, read(b_j)), (2j, D(a_j)) \mid 1 < j \leq k\}, \\
E &= \{(2j-1, 2j), (2j, 2j-1), (2j, 2j+1) \mid 1 \leq j \leq k\}.
\end{aligned}
$$

Without proof we claim that $L_k \notin \mathcal{L}(rtIDtPAL)$. □

5 Conclusion and Future Research

In this paper, we have introduced tissue P automata as a specific model of multiset automata as well as input-driven tissue P automata – which we also called *chocolate automata* – where the rules to be applied depend on the input symbol. Taking strings as input objects, these are either read from an input tape or defined by the sequence of symbols taken in, and as an additional storage of a multiset of different symbols we use a vesicle which moves from one cell of the tissue P system to another one depending on the input symbol; the input symbol at the same time determines whether (one or more) symbols are added to the multiset in the vesicle or removed from there and where the vesicle moves afterwards. The given input is accepted if it has been read completely and the vesicle has reached a final cell and/or is empty at this moment. When using multisets as input objects, these are enclosed in the vesicle in the input cell at the beginning of a computation, which vesicle then will also take the additional symbols. The given input multiset is accepted if no input symbols are present any more and the vesicle has reached a final cell and is empty at this moment.

As rules operating on the multiset enclosed in the vesicle when reading/consuming an input symbol we have used insertion, deletion, and substitution of multisets, working in the sequential derivation mode. As restricted variants, we have considered systems without allowing substitution of multisets and systems only allowing symbols to be inserted or deleted (or substituted).

We have shown how *chocolate automata* with multisets and strings can be characterized by input-driven register machines and input-driven counter automata, respectively. Moreover, we have exhibited some illustrative examples, for example, how the Dyck language or even some non-context-free languages can be accepted by simple variants of *chocolate automata*.

Several challenging topics remain for future research: for example, a characterization of the language classes accepted by several variants of tissue P automata accepting multisets or strings, especially for the input-driven variants (*chocolate automata*), introduced in this paper is still open.

As acceptance condition we have only considered reaching the final cell h with an empty vesicle. The other variants of acceptance, i.e., only requiring the vesicle to be empty or else requiring to have reached the final cell with the vesicle containing no input symbol any more, are to be investigated in the future in more detail.

Acknowledgements. The authors appreciate the helpful comments of the unknown referees.

References

1. Alhazov, A., Freund, R., Heikenwälder, H., Oswald, M., Rogozhin, Yu., Verlan, S.: Sequential P systems with regular control. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, Gy. (eds.) CMC 2012. LNCS, vol. 7762, pp. 112–127. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36751-9_9

2. Alhazov, A., Freund, R., Ivanov, S., Verlan, S.: (Tissue) P systems with vesicles of multisets. In: Csuhaj-Varjú, E., Dömösi, P., Vaszil, Gy. (eds.) Proceedings 15th International Conference on Automata and Formal Languages. AFL 2017, 4–6 September 2017, Debrecen, Hungary, vol. 252, pp. 11–25. EPTCS (2017). https://doi.org/10.4204/EPTCS.252.6
3. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Babai, L. (ed.) Proceedings of the 36th Annual ACM Symposium on Theory of Computing, 13–16 June 2004, Chicago, IL, USA, pp. 202–211. ACM (2004). https://doi.org/10.1145/1007352.1007390
4. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM **56**(3), 16:1–16:43 (2009). https://doi.org/10.1145/1516512.1516518
5. Bensch, S., Holzer, M., Kutrib, M., Malcher, A.: Input-driven stack automata. In: Baeten, J.C.M., Ball, T., de Boer, F.S. (eds.) TCS 2012. LNCS, vol. 7604, pp. 28–42. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33475-7_3
6. von Braunmühl, B., Verbeek, R.: Input-driven languages are recognized in log n space. In: Karpinski, M. (ed.) FCT 1983. LNCS, vol. 158, pp. 40–51. Springer, Heidelberg (1983). https://doi.org/10.1007/3-540-12689-9_92
7. Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V.: Multiset automata. In: Calude, C.S., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) WMC 2000. LNCS, vol. 2235, pp. 69–83. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45523-X_4
8. Csuhaj-Varjú, E., Vaszil, Gy.: P automata or purely communicating accepting P systems. In: Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) WMC 2002. LNCS, vol. 2597, pp. 219–233. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36490-0_14
9. Dassow, J., Păun, Gh.: On the power of membrane computing. J. UCS **5**(2), 33–49 (1999). https://doi.org/10.3217/jucs-005-02-0033
10. Dymond, P.W.: Input-driven languages are in log n depth. Inf. Process. Lett. **26**(5), 247–250 (1988). https://doi.org/10.1016/0020-0190(88)90148-2
11. Freund, R.: P automata: new ideas and results. In: Bordihn, H., Freund, R., Nagy, B., Vaszil, Gy. (eds.) Proceedings of Eighth Workshop on Non-Classical Models of Automata and Applications. NCMA 2016, 29–30 August 2016, Debrecen, Hungary, vol. 321, pp. 13–40. Österreichische Computer Gesellschaft (2016). https://shop.ocg.at/de/books.html
12. Freund, R., Ibarra, O., Păun, Gh., Yen, H.C.: Matrix languages, register machines, vector addition systems. In: Third Brainstorming Week on Membrane Computing, pp. 155–167 (2005). https://www.gcn.us.es/3BWMC/bravolpdf/bravol155.pdf
13. Freund, R., Kogler, M., Rogozhin, Yu., Verlan, S.: Graph-controlled insertion-deletion systems. In: Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems. DCFS 2010, 8–10 August 2010, Saskatoon, Canada, pp. 88–98 (2010). https://doi.org/10.4204/EPTCS.31.11
14. Freund, R., Oswald, M.: Tissue P systems and (mem)brane systems with mate and drip operations working on strings. Electron. Notes Theor. Comput. Sci. **171**(2), 105–115 (2007). https://doi.org/10.1016/j.entcs.2007.05.011
15. Freund, R., Păun, Gh.: How to obtain computational completeness in P systems with one catalyst. In: Proceedings Machines, Computations and Universality 2013. MCU 2013, 9–11 September 2013, Zürich, Switzerland, pp. 47–61 (2013). https://doi.org/10.4204/EPTCS.128.13
16. Freund, R., Rogozhin, Yu., Verlan, S.: Generating and accepting P systems with minimal left and right insertion and deletion. Nat. Comput. **13**(2), 257–268 (2014). https://doi.org/10.1007/s11047-013-9396-3

17. Greibach, S.A.: Remarks on blind and partially blind one-way multicounter machines. Theor. Comput. Sci. **7**, 311–324 (1978). https://doi.org/10.1016/0304-3975(78)90020-8
18. Kudlek, M., Totzke, P., Zetzsche, G.: Multiset pushdown automata. Fundam. Inform. **93**(1–3), 221–233 (2009). https://doi.org/10.3233/FI-2009-0098
19. Kutrib, M., Malcher, A., Wendlandt, M.: Tinput-driven pushdown, counter, and stack automata. Fundam. Inform. **155**(1–2), 59–88 (2017). https://doi.org/10.3233/FI-2017-1576
20. Kutrib, M., Malcher, A., Wendlandt, M.: Queue automata: foundations and developments. In: Adamatzky, A. (ed.) Reversibility and Universality. ECC, vol. 30, pp. 385–431. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73216-9_19
21. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: tissue P systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 290–299. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45655-4_32
22. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL-recognition. In: de Bakker, J., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 422–435. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10003-2_89
23. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs (1967)
24. Okhotin, A., Salomaa, K.: Input-driven pushdown automata: nondeterminism and unambiguity. In: Bensch, S., Drewes, F., Freund, R., Otto, F. (eds.) Proceedings of Fifth Workshop on Non-Classical Models for Automata and Applications. NCMA 2013, 13–14 August 2013, Umeå, Sweden, vol. 294, pp. 31–33. Österreichische Computer Gesellschaft (2013). https://shop.ocg.at/de/books.html
25. Okhotin, A., Salomaa, K.: Input-driven pushdown automata with limited nondeterminism. In: Shur, A.M., Volkov, M.V. (eds.) DLT 2014. LNCS, vol. 8633, pp. 84–102. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09698-8_9
26. Okhotin, A., Salomaa, K.: Descriptional complexity of unambiguous input-driven pushdown automata. Theor. Comput. Sci. **566**, 1–11 (2015). https://doi.org/10.1016/j.tcs.2014.11.015
27. Okhotin, A., Salomaa, K.: State complexity of operations on input-driven pushdown automata. J. Comput. Syst. Sci. **86**, 207–228 (2017). https://doi.org/10.1016/j.jcss.2017.02.001
28. Oswald, M.: P automata. Ph.D. thesis, Faculty of Computer Science, TU Wien (2003)
29. Păun, A., Păun, Gh.: The power of communication: P systems with symport/antiport. New Gener. Comput. **20**(3), 295–306 (2002). https://doi.org/10.1007/BF03037362
30. Păun, Gh.: Computing with membranes. J. Comput. Syst. Sci. **61**(1), 108–143 (2000). https://doi.org/10.1006/jcss.1999.1693
31. Păun, Gh., Pérez-Jiménez, M.J.: P automata revisited. Theor. Comput. Sci. **454**, 222–230 (2012). https://doi.org/10.1016/j.tcs.2012.01.036
32. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
33. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1–3. Springer, Heidelberg (1997)
34. Bulletin of the International Membrane Computing Society (IMCS). http://membranecomputing.net/IMCSBulletin/index.php
35. The P Systems Website. http://ppage.psystems.eu/