Fundamenta Informaticae XX (2019) 1–16
DOI 10.3233/FI-2015-0000
IOS Press

On Languages of P Automata

Erzsébet Csuhaj-Varjú [C]
Department of Algorithms and Their Applications, Faculty of Informatics
Eötvös Loránd University
Pázmány Péter sétány 1/c, 1117 Budapest, Hungary
csuhaj@inf.elte.hu

György Vaszil
Department of Computer Science, Faculty of Informatics
University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
vaszil.gyorgy@inf.unideb.hu

Abstract. P automata are accepting computing devices combining features of classical automata and membrane systems. In this paper we introduce P n-stack-automata, a restricted class of P automata that mimics the behaviour of n-stack automata. We show that for $n = 1$ these constructs describe the context-free language class and for $n = 3$ the class of quasi-realtime languages.

1. Introduction

Membrane systems (also called P systems) are distributed computing devices inspired by the architecture and functioning of living cells. The basic concept was introduced by Gheorghe Păun in [1], since then membrane computing (P systems' theory) has become a vivid research area [2].

The main features of a cell-like P system are the following: a structure of hierarchically embedded membranes, each having a label and enclosing a region containing a multiset of objects and possibly other membranes. Each region has an associated set of rules working on the objects present inside

Address for correspondence: E. Csuhaj-Varjú, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary
[C]Corresponding author

the region. The unique out-most membrane is called the skin membrane. During the functioning of the membrane system, the objects inside regions may change and move across the membranes. The evolution of the objects inside the membrane structure from an initial configuration (the initial contents of the regions) to one or more pre-defined final configuration(s) (usually a halting configuration) corresponds to a computation. The result of the computation is derived from some properties of configuration; usually, it is the number of objects inside the regions (or inside a distinguished region) in the final configuration.

P systems can also be used for accepting languages; important variants of these models are P automata [3, 4, 5, 2]. These constructs are purely communicating accepting P systems which combine features of classical automata and membrane systems being in interaction with their environments. Briefly, a P automaton is a P system which receives input in each computation step from its environment that influences its functioning. The input is given as a multiset of objects. During the computation, the objects do not change, thus, the P system works only with communication rules. Among the sequences of inputs, accepted input sequences are distinguished. The language accepted by a P automaton is obtained by mapping the accepted input sequences (accepted sequences of input multisets) to words of a language. Notice that the choice of the mapping plays crucial role in this respect.

P automata have been used for describing language classes, among others the well-known language classes of the Chomsky hierarchy. For example, it has been shown that the regular, the context-sensitive, and the recursively enumerable language classes can be obtained with different variants of P automata. The interested reader is referred to [2] for more information. A characterization of the context-free language class was presented in [6], by simulating non-deterministic pushdown automata with a restricted class of P automata, called P stack-automata.

In this paper we continue our study in describing language classes in terms of P automata. To do this, we generalize the notion of P stack-automaton to P n-stack-automaton, and thus define a restricted class of P automata that is inspired by n-stack automata. We show that for $n = 3$ these computing devices describe the class of quasi-realtime languages and, obviously, for $n = 1$ the context-free language class. Note that in general, automata with two stacks are able to simulate arbitrary Turing machine, thus, accept any recursively enumerable language. In the case of P stack-automata, however, the situation is different, because they work in realtime (they read an input symbol in each computation step).

The paper is organized as follows. After presenting the necessary definitions and notations from the theory of computation and from membrane computing, we introduce the concept of P n-stack automaton and present our results, namely, the description of context-free languages and quasi-realtime languages in terms of P n-stack automata. We close the paper with conclusions.

2. Preliminaries and Definitions

We first recall the notions and the notations we use. We assume that the reader is familiar with the basics of the theory of computation, in particular automata theory. For further information consult [7].

A finite set of symbols Σ is called alphabet; the number of elements in Σ is denoted by $|\Sigma|$. The set of all words over Σ is denoted by Σ^* and $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ where ε denotes the empty word. A language L over alphabet Σ is a subset of Σ^*. The number of elements of a finite set V is denoted by $|V|$.

A context-free grammar G is denoted by $G = (N, T, P, S)$, where N is the alphabet of nonterminals, T is the alphabet of terminals, P is the set of productions (or set of rules) and S is the start symbol. It is known that for every context-free grammar G we can construct an equivalent context-free grammar in

Greibach normal form, i.e. having rules of the form $X \to a\alpha$ where X is a nonterminal, a is a terminal, and α is a possibly empty string of nonterminals.

A non-deterministic pushdown automaton (a pushdown automaton, for short) M is denoted by $M = (\Sigma, \Gamma, Q, \delta, q_0, X_1)$, where Σ is the input alphabet, Γ is the pushdown alphabet, Q is the set of states, δ is the transition mapping, $q_0 \in Q$ is the initial state, and the initial pushdown contents is X_1. It is well-known that any context-free language L can be accepted by a non-deterministic pushdown automaton by empty pushdown, and conversely, any pushdown automaton accepts a context-free language. Furthermore, for any non-empty context-free language L we can construct a so-called realtime pushdown automaton, i.e. which consumes an input symbol in any step of the computation.

In the paper we will use the notion of language classes accepted by restricted versions of Turing machines. These machines have an input tape which can only be read from left to right and several worktapes where the heads can read, write, and move in any direction. Such a Turing machine accepts a language in realtime, if it consumes one input symbol from the input tape in every computation step. The class of languages that are accepted by these types of non-deterministic multitape Turing machines in realtime is called the class of *quasi-realtime languages*. A language L can be accepted by a non-deterministic k-tape Turing machine in *linear time* if and only if there exists a k-tape non-deterministic Turing machine T and a positive integer c such that for all words w, it holds that $w \in L$ if T accepts on input w within $c \cdot |w|$ many steps. The class of languages that are accepted by non-deterministic multitape Turing machines in linear time consists of those languages that can be accepted by a non-deterministic k-tape Turing machine in linear time for some $k \geq 1$.

In the general case, multipushdown automata with two pushdown stores can simulate the computations of Turing machines, but in the case of realtime automata, the situation is different, since the simulation of Turing machines with multipushdown automata does not work in realtime. It is known, however, that the following statements are equivalent for every language L: (1) L is a quasi-realtime language, (2) L is accepted by a non-deterministic multitape Turing machine in linear time, (3) L is the length-preserving homomorphic image of the intersection of three context-free languages [8].

Finally, we recall some notions and notations concerning multisets.

Let U be a set - the universe - of objects, and let $\mathbb{N}$ denote the set of natural numbers. A multiset is a pair $M = (V, f)$, where $V \subseteq U$ and $f : U \to \mathbb{N}$ is a mapping which assigns to each object $a \in V$ its multiplicity, if $a \notin V$ then $f(a) = 0$. The support of $M = (V, f)$ is the set $supp(M) = \{a \in V \mid f(a) \geq 1\}$. If $supp(M)$ is a finite set, then M is called a finite multiset. The set of all finite multisets over the set V is denoted by V^*.

We say that $a \in M = (V, f)$ if $a \in supp(M)$. $M_1 = (V_1, f_1) \subseteq M_2 = (V_2, f_2)$ if $supp(M_1) \subseteq supp(M_2)$ and for all $a \in V_1$, $f_1(a) \leq f_2(a)$. The union of two multisets is defined as $(M_1 \cup M_2) = (V_1 \cup V_2, f')$ where for all $a \in V_1 \cup V_2$, $f'(a) = f_1(a) + f_2(a)$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) = (V_1 - V_2, f'')$ where $f''(a) = f_1(a) - f_2(a)$ for all $a \in V_1 - V_2$, and the intersection of two multisets is $(M_1 \cap M_2) = (V_1 \cap V_2, f''')$ where for $a \in V_1 \cap V_2$, $f'''(a) = min(f_1(a), f_2(a))$, $min(x, y)$ denoting the minimum of $x, y \in \mathbb{N}$. We say that M is empty, denoted (as in the case of sets) by $\emptyset$, if its support is empty, $supp(M) = \emptyset$.

A multiset M over the finite set of objects V can be represented as a string w over the alphabet V with $|w|_a = f(a)$ where $a \in V$ and where $|w|_a$ denotes the number of occurrences of the symbol a in the string w. Notice that any permutation of the symbols in the string represents the same multiset. In the following, if no confusion arises, the finite multiset of objects $M = (V, f)$ is denoted by word w, where w is over V and represents M.

2.1. P automata

A cell-like P system is a structure of hierarchically embedded membranes, each having a label and enclosing a region containing a multiset of objects and possibly other membranes. The out-most membrane, called the skin membrane, is unique and it is usually labeled with 1. If membrane i contains membrane j, and there is no other membrane, k, such that k contains j and i contains k, then we say that membrane i is the parent membrane of j, denoted by $parent(j) = i$.

The evolution of the contents of the regions of a P system is described by rules associated to the regions. Applying the rules synchronously in each region, the system performs a computation by passing from one configuration to some other (not necessarily different) one. The rules can be of various types, here we consider communication rules called symport or antiport rules.

A symport rule is of the form (x, in) or (x, out), where x is a finite multiset over alphabet V, i.e. $x \in V^*$. If such a rule is present in a region i, then the objects of the multiset x must enter from the parent region or must leave to the parent region, respectively. The environment is considered to be the parent region of the skin membrane. We note that if the rules are applied in the maximally parallel way, then symport rules of type (x, in) should be avoided inside the region enclosed by the skin membrane. Otherwise, if the environment contains an infinite number of copies of multiset x, then an infinite number of objects can enter the system in one step. An antiport rule is of the form $(x, in; y, out), x, y \in V^*$. In this case, objects of x enter from the parent region and in the same step objects of y leave to the parent region. All types of these rules might be equipped with a promoter or inhibitor multiset, denoted as $(x, in)|_Z, (x, out)|_Z$, or $(x, in; y, out)|_Z, x, y \in V^*, Z \in \{z, \neg z \mid z \in V^*\}$ in which case they can only be applied if region i contains the objects of multiset z, or if $Z = \neg z$, then region i must not contain the elements of z. (For more information consult [2].)

Now we introduce some notation that we would like to use later. If r is a symport rule (x, in) or (y, out), or an antiport rule $(x, in; y, out)$, then let $import(r) = x$, and $export(r) = y$. If r has a promoter multiset z (it is of the form $(x, in)|_z$, $(y, out)|_z$, or $(x, in; y, out)|_z$), then $promoter(r) = z$. If r has an inhibitor multiset z (it is of the form $(x, in)|_{\neg z}$, $(y, out)|_{\neg z}$, or $(x, in; y, out)|_{\neg z}$), then $inhibitor(r) = z$. Further, if r is a symport rule without exporting or importing any objects, then $export(r) = \varepsilon$ or $import(r) = \varepsilon$, respectively. If r does not have any promoter or inhibitor, then $promoter(r) = \varepsilon$ or $inhibitor(r) = \varepsilon$, respectively.

A *P automaton* is a $(3n + 2)$-tuple $\Pi = (V, \mu, (w_1, P_1, F_1), \ldots, (w_n, P_n, F_n))$ where $n \geq 1$ is the number of membranes, V is a finite set of objects, μ is a membrane structure of n membranes with membrane 1 being the skin membrane, and for all $i, 1 \leq i \leq n$,

- $w_i \in V^*$ is the initial contents (state) of region i, that is, it is the finite multiset of all objects contained by region i,
- P_i is a finite set of communication rules associated to membrane i, they can be symport rules or antiport rules, with or without promoters or inhibitors, as above, and
- $F_i \subseteq V^*$ is a finite set of finite multisets over V called the set of final states of region i. If $F_i = \emptyset$, then all the states of membrane i are considered to be final.

The n-tuple of finite multisets of objects present in the n regions of the P automaton Π describes a *configuration* of Π, the n-tuple $(w_1, \ldots, w_n) \in (V^*)^n$ is the initial configuration.

The application of the rules can take place in a *sequential* (seq), or in a *maximally parallel* (par) manner.

The transition mapping of a P automaton is a partial mapping $\delta_X : V^* \times (V^*)^n \to 2^{(V^*)^n}$. These mappings are defined implicitly by the rules of the rule sets P_i, $1 \leq i \leq n$. For a configuration $(u_1, \ldots, u_n)$,

$$(u'_1, \ldots, u'_n) \in \delta_X(u, (u_1, \ldots, u_n))$$

holds, that is, while reading the input $u \in V^*$ the automaton may enter the configuration $(u'_1, \ldots, u'_n) \in (V^*)^n$, if there exist rules as follows.

If $X = seq$, then there exists a rule $r_i \in P_i$ with $promoter(r_i) \subseteq u_i$ and $inhibitor(r_i) \cap u_i = \emptyset$ for all $1 \leq i \leq n$. Let $import_{seq}(i) = import(r_i)$, and let $export_{seq}(i) = export(r_i)$.

If $X = par$, then there exists a multiset of rules $R_i = r_{i,1} \ldots r_{i,m_i}$, for all $1 \leq i \leq n$, with $promoter(r_{i,j}) \subseteq u_i$ and $inhibitor(r_{i,j}) \cap u_i = \emptyset$, $1 \leq j \leq m_i$, satisfying the conditions below. Furthermore, there is no rule occurrence $r \in P_j$, for any j, $1 \leq j \leq n$, such that the rule multisets R'_i with $R'_i = R_i$ for $i \neq j$ and $R'_j = \{r\} \cup R_j$, also satisfy the conditions. Let $import_{par}(i) = \bigcup_{1 \leq j \leq m_1} import(r_{1,j})$, and let $export_{par}(i) = \bigcup_{1 \leq j \leq m_1} export(r_{1,j})$.

Then for $X \in \{seq, par\}$ and for all $1 \leq i \leq n$, the conditions that the chosen rules must satisfy are given by

1. $import_X(1) = u$,

2. $export_X(i) \cup \bigcup_{parent(j)=i} import_X(j) \subseteq u_i$,

and the new configuration is obtained as

$$u'_i = u_i \cup import_X(i) \cup \left(\bigcup_{parent(j)=i} export_X(j) \right) - export_X(i) - \left(\bigcup_{parent(j)=i} import_X(j) \right).$$

We define the sequence of multisets of objects accepted by the P automaton as the sequence of input multisets which appear in the environment and are consumed by the skin membrane in each computation step until the system reaches a final state, a configuration where for all j with $F_j \neq \emptyset$ it holds that $u_j \in F_j$.

We extend δ_X to $\bar{\delta}_X$, $X \in \{seq, par\}$, a function mapping the sequences of finite multisets over V and the configurations of Γ to new configurations. We define $\bar{\delta}_X$ as

1. $\bar{\delta}_X(v, (u_1, \ldots, u_n)) = \delta_X(v, (u_1, \ldots, u_n))$, $v, u_i \in V^*$, $1 \leq i \leq n$, and

2. $\bar{\delta}_X((v_1) \ldots (v_{s+1}), (u_1, \ldots, u_n)) = \bigcup \delta_X(v_{s+1}, (u'_1, \ldots, u'_n))$
   for all $(u'_1, \ldots, u'_n) \in \bar{\delta}_X((v_1) \ldots (v_s), (u_1, \ldots, u_n))$, $v_j, u_i, u'_i \in V^*$,
   $1 \leq i \leq n$, $1 \leq j \leq s + 1$.

Note that we use brackets to separate the elements of the multiset sequence $(v_1) \ldots (v_{s+1}) \in (V^*)^*$ from each other, in order to distinguish it from the multiset $v_1 \cup \ldots \cup v_{s+1} \in V^*$.

Let Π be a P automaton as above with initial configuration $(w_1, \ldots, w_n)$, let Σ be an alphabet, and let $f : V^* \longrightarrow \Sigma \cup \{\varepsilon\}$ be a mapping with $f(x) = \varepsilon$ if and only if $x = \emptyset$.

The language accepted by Π under mapping f and working mode X, for $X \in \{seq, par\}$, is defined as

$$L_X(\Pi, f) = \{f(v_1) \dots f(v_s) \in \Sigma^* \mid (u_1, \dots, u_n) \in \bar{\delta}_X((v_1) \dots (v_s), (w_1, \dots, w_n))$$
$$\text{with } u_j \in F_j \text{ for all } j \text{ with } F_j \neq \emptyset,\ 1 \leq j \leq n,\ 1 \leq s\}.$$

Notice that the choice of mapping f is crucial. It has to be "easily" computable because the power of the P automaton should be provided by the underlying membrane system and not by f itself. The notion of "easiness", however, depends on the context we are working in, so we do not provide a general definition of this concept.

The reader might easily notice that the accepting power of P automata strongly depends both on the working mode (the way of rule application) and on the mapping which is used for defining the accepted language. We briefly discuss this property by using some restricted variants of P automata as examples.

If the rule application is sequential, i.e. at each computation step at most one rule is applied in every region, then the "problem" of mapping the set of possible input multisets to a finite alphabet does not appear since the number of possible inputs is finite. If $\{v_1, \dots, v_t\}$, $v_i \in V^*$ are the different multisets appearing in rules $(v_i, in; u_i, out) \in P_1,\ 1 \leq i \leq t$, then $f(v_i) = a_i \in \Sigma$ is a natural correspondence between the two sets.

In [9], P automata with sequential rule application were shown to describe a language class which is strictly included in the class of languages accepted by Turing machines reading the separate input tape one way, and using logarithmic space for additional computations on the work-tapes. This sub-logarithmic-space language class still contains well-known non-trivial languages, $\{a^n b^n c^n \mid n \geq 1\}$, for example.

If the rule application of the P automaton is maximally parallel, the situation is more complex. A mapping between the possibly infinite set of input multisets and a finite alphabet has to be specified. In [9] it was shown that if rules of the type (v, in) are not allowed in the skin membrane, and if the mapping $f : V^* \to \Sigma \cup \{\varepsilon\}$ is specified in such a way that $f(x) = \varepsilon$ implies $x = \emptyset$ and f is linear space computable by a Turing machine, then P automata accept exactly the class of context-sensitive languages. Moreover, it is also shown that all context-sensitive languages can be accepted using an f which maps an input multiset $x \in V^*$ to some element of $\{a \in V \mid a \in supp(x)\}$ and $f(\emptyset) = \varepsilon$. That is, it is also possible to give a characterization of context-sensitive languages in terms of P automata. Furthermore, it is easy to verify if a given system conforms to this specification or not.

3. P multistack-automata

We introduce a restricted class of P automata called P multistack-automata. The concept is a generalization of the notion of a P stack-automaton, introduced in [6], to present a variant of P automata that describes context-free languages. We show that having three "stacks" and using the maximally parallel working mode, these P automata variants describe the class of quasi-realtime languages, and obviously, with one stack, the class of context-free languages can also be obtained by these constructs.

Before presenting the formal definition, we briefly describe the idea behind P multistack-automata. The goal is to give a syntactic characterization of the rules and object alphabets that are necessary to enable a stack-automaton-like functioning by imitating a stack type of information storage mechanism using the P automata framework. Stacks can be simulated by multisets similarly to the way they are simulated by counters: the multiplicity of an object represents a number, which in turn corresponds to the contents of a stack. The correspondence is based on the positional numeral notation: stack contents correspond to strings, and having an alphabet Σ of k symbols, a string over Σ can naturally be interpreted as a $(k+1)$-ary number. (We take $k+1$ as the base of the positional notation in order to exclude the zero digit, so we do not have to deal with the problem of leading zeroes.) If the stack contents is represented by strings where the right end of the string corresponds to the top of the stack, then a pop operation corresponds to a division of the numerical representation by $k+1$. Similarly, a push operation pushing l symbols to the stack corresponds to a multiplication by $(k+1)^l$ and the addition of a number consisting of l digits. The goal of the following definition of P multistack-automata is to provide a minimal set of features that are necessary to maintain such numerical stack representations in a P automaton.

Definition 3.1. A P multistack-automaton $\Pi = (V, \mu, (w_1, P_1, F_1), (w_2, P_2, F_2))$ is a P automaton where $\mu = [_1[_2]_2]_1$, V can be partitioned into disjoint subsets as

$$V = \Sigma \cup B \cup C \cup D \cup \{I\},$$

the initial multisets are of the form

$$\begin{aligned} w_1 &= b_1 \dots b_m, \text{ where } B = \{b_1, \dots, b_m\}, \\ w_2 &= c_1 \dots c_l I u, \text{ where } u \in D^*, c_i \in C, 1 \le i \le l, \end{aligned}$$

the sets of final states are finite sets of finite multisets over D,

$$F_i \subset D^* \text{ for } 1 \le i \le 2,$$

and there is a fixed positive integer k, such that all the rules are in one of the following forms.

For P_1:

- (a, out) where $a \in \Sigma \cup C$,
- $(b^{k^i}, in; b^k, out)|_c$ where $i \ge 0,\ b \in B,\ c \in C$, and
- $(ab_1^{i_1} \dots b_m^{i_m} c_1, in; b_1^{j_1} \dots b_m^{j_m} c_2, out)$ where $a \in \Sigma,\ B = \{b_1, \dots, b_m\},\ c_1, c_2 \in C,\ 1 \le j_i < k,$ $1 \le i \le m$, and

moreover, if for a given $c_2 \in C$ and $b_s \in B$ we have the rule $(b_s^{k^j}, in; b_s^k, out)|_{c_2}$ in a region together with the rule $(ab_1^{i_1} \dots b_m^{i_m} c_1, in; b_1^{j_1} \dots b_m^{j_m} c_2, out)$, then the multiplicity of the objects b_s brought in by the second type of rule is less than the number of objects b_s brought in by one copy of the first type of rule, that is, $i_s < k^j$.

For P_2:

- $(b^j c^l, in; u, out)$ where $j \ge 1, b \in B,\ c \in C,\ l \ge 0, u \in D^+$, and
- (Ic, out) where $c \in C$.

A P multistack-automaton is a *P m-stack-automaton* for some $m \in \mathbb{N}$, if $|B| = m$, that is, if the cardinality of the set B is m. We also call P 1-stack automata simply *P stack-automata*.

Definition 3.2. The *language accepted* by the P multistack-automaton Π over the finite alphabet Σ and in working mode X, where $X \in \{seq, par\}$ is the language $L_X(\Pi, f)$ accepted by the P automaton Π working in mode X and with a function f defined in the usual manner, i.e. the empty multiset, and only the empty multiset is mapped to ε, and for non-empty input multisets, the image depends only on the occurrence or non-occurrence of certain symbols, but not on their multiplicity. That is,

$$f : V^* \to \Sigma \cup \{\varepsilon\}, \text{ where for any } x \neq \emptyset,\ f(x) \neq \varepsilon,$$

and moreover, for any $x, y \in V^*$, $supp(x) = supp(y)$ implies that $f(x) = f(y)$.

As we have seen above, a P multistack-automaton is a P automaton having rules of a restricted form and a special, very simple mapping. Initially, the skin membrane contains a number of copies of objects from the set B, and the other membrane contain finite multisets of objects over the set D plus single occurrences of some objects from the set C. The rules are applied until a final configuration, i.e. a certain combination of a finite number of elements from the set D, is reached.

Now we show that P 1-stack-automata accept exactly the class of context-free languages. We note that the proof is a revised version of the corresponding proof in [6].

Theorem 3.3. A language is context-free if and only if it is accepted by a P stack-automaton working in the maximally parallel working mode.

Proof:
First we prove that every context-free language can be accepted by a P stack-automaton working in the maximally parallel manner. For simplicity, we will not indicate the working mode in the sequel.

Let L be a context-free language and $M = (\Sigma, \Gamma, Q, \delta, q, X_1)$ be a (non-deterministic) pushdown automaton with input alphabet Σ, pushdown alphabet Γ, set of states Q, transition mapping δ, initial state $q \in Q$, and initial pushdown contents X_1, accepting $L \setminus \{\varepsilon\}$ with empty pushdown. Let $k = |\Gamma| + 1$, and let us assume, without the loss of generality, that $\Gamma = \{X_1, X_2, \ldots, X_{k-1}\}$, $Q = \{q\}$, and the transitions of M are of the form

$$(q, \alpha) \in \delta(q, a, X) \text{ where } q \in Q,\ \alpha \in \Gamma^*,\ a \in \Sigma,\ X \in \Gamma.$$

(Such a pushdown automaton can be obtained from a context-free grammar in Greibach normal form, i.e. having rules $X \to a\alpha$ where X is a nonterminal, a is a terminal, and α is a possibly empty string of nonterminals.)

Before turning to the details of the proof, we provide an auxiliary notations. For a string w, where w is considered as an integer given in the k-ary notation, $val(w)$ denotes the numerical value of w.

We construct a P stack-automaton Π accepting L. Π simulates M and if $\varepsilon \in L$, then it accepts ε as well. Let

$$\Pi = (V, [_1\ [_2\ ]_2]_1, (w_1, P_1, \{E\}), (w_2, P_2, \emptyset))$$

be a P stack-automaton with

$$V = \Sigma \cup B \cup C \cup D \cup \{I\},$$

where $B = \{b\}$, $C = \{(a, X, \alpha) \mid (q, \alpha) \in \delta(q, a, X)$ is a transition of $M\}$, and $D = \{T, E\}$. Let

$$\begin{aligned}
w_1 &= b,\\
P_1 &= \{(b^{k^{|\alpha|}}, in; b^k, out)|_{(a,X,\alpha)} \mid (a, X, \alpha) \in C\} \cup\\
&\quad \{(a(a', X', \alpha')b^{val(\alpha)}, in; b^i(a, X_i, \alpha), out) \mid (a, X_i, \alpha), (a', X', \alpha') \in C\} \cup\\
&\quad \{(a, out) \mid a \in \Sigma\} \cup \{(I, out)\},
\end{aligned}$$

$$\begin{aligned}
w_2 &= TEI \cup \{(a, X_1, \alpha) \mid a \in \Sigma, \alpha \in \Gamma^*\},\\
P_2 &= \{(b, in; T, out)\} \cup \{(c, in; E, out), (Ic, out) \mid c \in C\} \cup P_2'.
\end{aligned}$$

where

$$P_2' = \begin{cases} \{(b \cup \{(a, X_1, \alpha) \mid a \in \Sigma, \alpha \in \Gamma^*\}, in; E, out)\} & \text{if } \varepsilon \in L,\\ \emptyset & \text{otherwise.}\end{cases}$$

A configuration (w, γ) of M corresponds to the configuration (u_1, u_2) where the k-ary notation of $|u_1|_b$ is the string $i_1 i_2 \dots i_s$ where $X_{i_1} X_{i_2} \dots X_{i_s} = \gamma$, X_{i_s} being the topmost symbol. In the following, $string_k(n)$ will denote the sequence of pushdown symbols corresponding to integer n.

Let us assume that

$$u_1 = a'(a, X_i, \alpha)b^n \tag{1}$$

where $a' \in \Sigma$, $(a, X_i, \alpha) \in C$, and b^n corresponds to the string $\gamma \in \Gamma^*$, $string(n) = \gamma$. Since only one occurrence of an object from C is present, the rules which could be used in the skin region are of the form

$$(b^{k^{|\alpha|}}, in; b^k, out)|_{(a,X_i\alpha)} \text{ and } (a(a', X', \alpha')b^{val(\alpha)}, in; b^i(a, X_i, \alpha), out) \tag{2}$$

where there exists a transition among the transitions of the pushdown automaton M

$$(q, \alpha) \in \delta(q, a, X_i).$$

If the application of two such rules consumes all the objects b of the skin region, then $string(n) = \gamma = \gamma' X_i$, and after the application of the rules we obtain a configuration (u_1', u_2) where $u_1' = acb^p$ for some $c \in C$ with $p = ((n - i)/k)k^{|\alpha|+1} + val(\alpha)$, that is, an object $a \in \Sigma$ (and a number of b-s) has entered the system, the new string corresponding to the new number of objects b is $string(p) = \gamma'\alpha$, where $\gamma'\alpha$ is the new contents of the pushdown of M after executing the transition above.

If no rule-pair of the form (2) can be applied in the skin region or a rule-pair is applied in a way which does not consume all b objects, that is, when no transition of M is simulated, then, due to the maximal parallelism of rule application, a rule $(b, in; T, out)$ is applied in the second region sending the trap object T to the skin region which makes it impossible for Π to reach the final configuration.

As we have seen, a configuration $cf = (u_1, u_2)$ with u_1 of the form (1) of Π corresponds to a configuration of M, and any configuration of Π following cf is either corresponding to a possible next configuration of M, or the computation of Π cannot produce any result.

Now we explain how the computation of Π is started and finished. The initial configuration of Π is

$$(b, TEI \cup \{(a, X_1, \alpha) \mid a \in \Sigma, \alpha \in \Gamma^*\}).$$

In the first computation step of Π, an object from C and the symbol I move from the second region to the first one, and then the initial transition of M, $(q,\alpha) \in \delta(q,a,X_1)$ is simulated by the rules

$$(b^{k^{|\alpha|}}, in; b^k, out)|_{(a,X_1,\alpha)} \text{ and } (a(a',X',\alpha')b^{val(\alpha)}, in; b(a,X_1,\alpha), out).$$

Besides these rules, we also have in the first region the rule (I, out), thus, after the first two computation steps, we either obtain a configuration with the trap symbol in the first region, or we get

$$(b^{val(\alpha)}c, TE) \text{ for some } c \in C.$$

This is a configuration corresponding to a configuration of M after the initial transition.

Let us assume now that Π is in a configuration

$$(vb^n(a,X_m,\varepsilon), u_2)$$

where $string(n) = X_m$ and a possible transition of M is

$$(q,\varepsilon) \in \delta(q,a,X_m)$$

which finishes the computation by emptying the pushdown. This implies that out of the rule pair

$$(b^{k^0}, in; b^k, out)|_{(a,X_m,\varepsilon)} \text{ and } (a(a',X',\alpha), in; b^n(a,X_m,\varepsilon), out),$$

associated to the first region, only $(a(a',X',\alpha), in; b^n(a,X_m,\varepsilon), out)$ is applicable (since X_m denotes a positive integer less than k). The new configuration will be of the form

$$(a(a',X',\alpha), u_2).$$

Now the computation can be finished by applying the rule (a, out) in the first region, and the rule $((a',X',\alpha), in; E, out)$ in the second region, producing the final configuration

$$(E, (a',X',\alpha)u_2')$$

where $Eu_2' = u_2$.

As we have seen, Π either simulates a computation of M (or accepts the empty word, if $\varepsilon \in L$), or does not reach the final configuration.

To show that all languages accepted by P stack-automata are context-free, we present the following, rather informal argument.

Looking at the restrictions which define a P stack-automaton, we can notice that the region other than the skin region, can only contain a limited number of objects which means that they can be in a finite number of different configurations. The skin region can also contain a finite number of objects other than b, thus, to record a configuration of the P stack-automaton, we need to record the potentially unbounded number of b-s in the skin membrane, plus a finite set of states to record the distribution of the objects in the two regions.

If we look at the rules of the skin region, we can also notice that the number of b-s can be recorded with a pushdown, thus, that a usual pushdown automaton is able to simulate the work of the P stack-automaton. To see this, consider the rules

$$(b^{k^i}, in; b^k, out)|_c \text{ where } i \geq 0,\ c \in C, \tag{3}$$

and

$$(ab^l c_1, in; b^m c_2, out) \text{ where } a \in \Sigma,\ c_1, c_2 \in C,\ l, m \geq 0. \tag{4}$$

Since at most one occurrence of one object from C is present in the skin region, the rules of type (3) multiply the number of b-s by k^{i-1}, thus, if we consider the k-ary number denoting the number of b-s, the rules of type (3) append $i-1$ zeroes to the end of the string of k-ary digits (or, if $i = 0$ add the value expressed by the last digit to the value obtained by erasing it from the end of the string). Since the length of the manipulated suffix of the digit string is bounded, the string of digits can be stored in a pushdown by introducing $k-1$ symbols to denote the digits. Moreover, since only one copy of a rule of type (4) can be used (since at most one copy of $c \in C$ is present), the changes of the number of b-s caused by these rules can be simulated using the finite control by modifying a bounded number of topmost symbols of the pushdown. □

Before we continue, we demonstrate the above construction with an example.

Example 3.4. Let $G = (\{a, b\}, \{X_2, X_3\}, X_2, P)$ be a context-free grammar where the set of productions is $P = \{X_2 \to aX_2X_3, X_2 \to aX_3, X_3 \to c\}$. Note that G is in Greibach normal form, and it generates the language $L(G) = \{a^n c^n \mid n \geq 1\}$.

Let us consider the pushdown automaton $M = (\{a, c\}, \{X_1, X_2, X_3\}, \{q\}, \delta, q, X_1)$ accepting $L(G)$, with $\delta : Q \times \Sigma \times \Gamma \to 2^{\{q\}\times\Gamma^*}$ defined as

$$\delta(q, a, X_1) = \{(q, X_2X_3), (q, X_3)\}, \quad \delta(q, a, X_2) = \{(q, X_2X_3), (q, X_3)\}, \quad \delta(q, c, X_3) = \{(q, \varepsilon)\}.$$

Since $|\Gamma| = 3$, let $k = 4$, and construct $\Pi = (V, [\ [\ ]_2\ ]_1,)(w_1, P_1, F_1), (w_2, P_2, F_2))$, the P stack-automaton corresponding to M. Let

$$\begin{aligned} V \ = \ & \{a, c\} \cup \{b\} \cup \{(a, X_1, X_2X_3), (a, X_1, X_3), (a, X_2, X_2X_3), (a, X_2, X_3),\ (c, X_3, \varepsilon)\} \cup \\ & \{T, E\} \cup \{I\}, \end{aligned}$$

and let

$$\begin{aligned} w_1 \ &= \ \emptyset, \\ P_1 \ &= \ \{(b^{k^2}, in; b^k, out)|_{(a,X_2,X_2X_3)}, (b^k, in; b^k, out)|_{(a,X_2,X_3)}, (b^k, out)|_{(c,X_3,\varepsilon)}\} \cup \\ & \quad \{(ab^{3k+2}(a, X_2, X_2X_3), in; b(a, X_1, X_2X_3), out), \\ & \quad\ (ab^3(c, X_3, \varepsilon), in; b(a, X_1, X_3), out), \\ & \quad\ (ab^{3k+2}(a, X_2, X_2X_3), in; b^2(a, X_2, X_2X_3), out), \\ & \quad\ (ab^3(c, X_3, \varepsilon), in; b^2(a, X_2, X_3), out), \\ & \quad\ (c(c, X_3, \varepsilon), in; b^3(c, X_3, \varepsilon), out)\} \cup \\ & \quad \{(a, out), (c, out)\} \cup \{(I, out)\}, \\ F_1 \ &= \ \{E\}, \\ \\ w_2 \ &= \ \{b, T, E, I, (a, X_1, X_3), (a, X_1, X_2X_3)\}, \\ P_2 \ &= \ \{(b, in; T, out), ((c, X_3, \varepsilon), in; E, out), (bI(a, X_1, X_2X_3), out), (bI(a, X_1, X_3), out)\}, \\ F_2 \ &= \ \emptyset. \end{aligned}$$

Notice that for the sake of brevity, we indicated only those rules which can actually used by the P stack-automaton. By rigorously following the construction of the previous theorem, we also obtain a few additional rules that can never be applied in a successful computation.

To see how the P stack-automaton Π works, consider the following. After the initial configuration

$$(\emptyset, bTEI(a, X_1, X_2X_3)),$$

we get

$$(u_1bI, u_2TE)$$

with either $u_1 = (a, X_1, X_3)$ and $u_2 = (a, X_1, X_2X_3)$, or $u_1 = (a, X_1, X_2X_3)$ and $u_2 = (a, X_1, X_3)$, the single b in first region corresponds to the fact that the pushdown automaton has symbol X_1 in its stack. In the first case, we must proceed with

$$(bI(a, X_1, X_3), u_2TE) \Rightarrow (ab^3(c, X_3, \varepsilon), u_2TE),$$

simulating the exchange of X_1 to X_3 in the stack, while reading an input letter a, and indicating that the transition corresponding to the object (c, X_3, ε) will be simulated in the next step, and then the computation will be finished as

$$(ab^3(c, X_3, \varepsilon), u_2TE) \Rightarrow (c(c, X_3, \varepsilon), u_2TE) \Rightarrow (E, u_2(c, X_3, \varepsilon)T).$$

In the other case, we have

$$(bI(a, X_1, X_2X_3), u_2TE) \Rightarrow (ab^{3k+2}u_3, u_2TE),$$

where u_3 is either (a, X_2, X_2X_3) or (a, X_2, X_3) depending on how the computation will continue. The number of bs, $3k+2$ corresponds to the fact that the simulated pushdown automaton has X_2X_3 in its stack. Now, if $u_3 = (a, X_2, X_3)$, then we get

$$(ab^{3k+2}(a, X_2, X_3), TE) \Rightarrow (ab^{3k+3}(c, X_3, \varepsilon), u_2TE),$$

and then

$$(ab^{3k+3}(c, X_3, \varepsilon), u_2TE) \Rightarrow (cb^3(c, X_3, \varepsilon), u_2TE) \Rightarrow (c(c, X_3, \varepsilon), u_2TE).$$

Now, the number of bs in the first region is zero, corresponding to the empty stack of the simulated pushdown automaton. The computation can be finished by

$$(c(c, X_3, \varepsilon), u_2TE) \Rightarrow (E, u_2T(c, X_3, \varepsilon))$$

reaching a final configuration.

If in the configuration

$$(ab^{3k+2}u_3, u_2TE)$$

we have $u_3 = (a, X_2, X_2X_3)$, then we get

$$(ab^{3k+2}(a, X_2, X_2X_3), u_2TE) \Rightarrow (ab^{3k^2+3k+2}u_3, u_2TE)$$

where u_3 is again either (a, X_2, X_2X_3) or (a, X_2, X_3) depending on how the computation will continue. The number of bs, $3k^2 + 3k + 2$ corresponds to the fact that the simulated pushdown automaton has $X_2X_3X_3$ in its stack. Now, if $u_3 = (a, X_2, X_3)$, then computation finishes by emptying the X_3 symbols from the stack and reading a c in each step, as we have seen above, or if $u_3 = (a, X_2, X_2X_3)$, then we put on more X_2X_3 pair on the pushdown as

$$(ab^{3k^2+3k+2}(a, X_2, X_3), u_2TE) \Rightarrow (aB^{3k^3+3k^2+3k+2}u_3, u_2TE),$$

where u_3 ia again the same as above.

Note that $\{a\}^i\{c\}^i,\ i \geq 1$ are the sequences of multisets entering Π during its computations, so having $f(\{x\}) = x$ as the input function, Π accepts strings of the form $a^ic^i,\ i \geq 1$.

In the following we show that (with a suitable mapping f and using the maximally parallel working mode) any quasi-realtime language can be accepted by a P 3-stack- automaton and conversely. The proof is based on the fact that any quasi-realtime language is the length-preserving homomorphic image of the intersection of three context-free languages, and that any quasi-realtime language can be accepted by a non-deterministic multitape Turing machine working in linear time [8].

Theorem 3.5. A language is quasi-realtime if and only if it is accepted by a P 3-stack-automaton working in the maximally parallel working mode.

Proof:
We first prove that for arbitrary context-free languages L_1, L_2, and L_3 it holds that their intersection, $L = L_1 \cap L_2 \cap L_3$, can be accepted by a P 3-stack-automaton. Let us consider the non-deterministic pushdown automata $M_i = (\Sigma, \Gamma_i, Q, q, X_{i,1}, \delta_i)$ accepting languages $L_i,\ 1 \leq i \leq 3$. Without the loss of generality we assume that $Q = \{q\}$, $\delta_i : Q \times \Sigma \times \Gamma_i \to Q \times \Gamma_i^*,\ 1 \leq i \leq 3$, i.e., the pushdown automata have one internal state, and they read one symbol of the input string in each computational step.

Let us also assume that the three pushdown alphabets contain the same number of symbols, and let k be a natural number such that $|\Gamma_1| = |\Gamma_2| = |\Gamma_3| = k - 1$. (If $|\Gamma_i| < |\Gamma_j|$ for some $1 \leq i, j \leq 3$, we add "dummy" elements to Γ_i to make the cardinality of the two alphabets equal.) Now the pushdown symbols can be denoted as $\Gamma_i = \{X_{i,1}, X_{i,2}, \ldots, X_{i,k-1}\},\ 1 \leq i \leq 3$ (with $X_{i,1}$ being the initial pushdown symbol). (Recall that for $\alpha_i \in \Gamma_i^*$, $val(\alpha_i)$ denotes the numerical value of the string α_i when it is considered as an integer given in k-ary notation with $X_{i,j}$ denoting digit $j,\ 1 \leq j \leq k - 1$.)

We construct the following P 3-stack automaton. $\Pi = (V, [\ [\ ]_2\ ]_1, (w_1, P_1, F_1), (w_2, P_2, F_2))$ with $V = \Sigma \cup B \cup C \cup D \cup \{I\}$, where $B = \{b_1, b_2, b_3\}$, $D = \{T, E\}$, and

$$C = \{(a, X_1, X_2, X_3, \alpha_1, \alpha_2, \alpha_3) \mid (q, \alpha_i) \in \delta_i(q, a, X_i) \text{ is a transition of } M_i\}.$$

Let

$$\begin{aligned}
w_1 &= b_1b_2b_3,\\
P_1 &= \{(b_i^{k^{|\alpha_i|}}, in; b_i^k, out)|_{(a,X_1,X_2,X_3,\alpha_1,\alpha_2,\alpha_3)} \mid (a,X_1,X_2,X_3,\alpha_1,\alpha_2,\alpha_3) \in C,\ 1 \le i \le 3\} \cup\\
&\quad \{(acb_1^{val(\alpha_1)}b_2^{val(\alpha_2)}b_3^{val(\alpha_3)}, in; b_1^{i_1}b_2^{i_2}b_3^{i_3}(a, X_{1,i_1}X_{2,i_2}X_{3,i_3}, \alpha_1,\alpha_2,\alpha_3), out) \mid\\
&\quad\ \text{where } (a, X_{1,i_1}X_{2,i_2}X_{3,i_3}, \alpha_1,\alpha_2,\alpha_3), c \in C\} \cup\\
&\quad \{(a, out), (c, out) \mid a \in \Sigma, c \in C\},
\end{aligned}$$

$$\begin{aligned}
w_2 &= TEI \cup \{(a, X_{1,1}, X_{2,1}, X_{3,1}, \alpha_1,\alpha_2,\alpha_3) \mid a \in \Sigma, \alpha_i \in \Gamma_i^*,\ 1 \le i \le 3\},\\
P_2 &= \{(b_i, in; T, out) \mid 1 \le i \le 3\} \cup \{(c, in; E, out), (Ic, out) \mid c \in C\} \cup P_2'.
\end{aligned}$$

where

$$P_2' = \begin{cases} \{(b_1b_2b_3c, in; E, out) \mid c \in C\} & \text{if } \varepsilon \in L,\\ \emptyset & \text{otherwise.}\end{cases}$$

The three pushdown automata are realtime, so if they are started with the same input string, then after a given number of computation steps, their configurations can be denoted by (w,γ_1), (w,γ_2), and (w,γ_3), or by $(w,\gamma_1,\gamma_2,\gamma_3)$ in short. This corresponds to the configuration (u_1,u_2) of the P automaton where the k-ary notation of $|u_1|_{b_i}$ is the string $i_1i_2\ldots i_{s_i}$ where $X_{i,i_1}X_{i,i_2}\ldots X_{i,i_{s_i}} = \gamma_i$, X_{i,s_i} being the topmost symbol, $1 \le i \le 3$.

Now, using similar considerations as in the proof of the previous theorem, we can see that the P 3-stack-automaton works by simultaneously executing the computation steps of the three pushdown automata when they all read the same input symbol. Since the pushdown automata are realtime, their computations have the same length when started with the same input, so an input string belongs to the intersection of their accepted languages if they all enter an accepting configuration (they all empty their pushdown store) in the same computation step, right after reading the last symbol of the string. If such a configuration can be reached by the three pushdown automata, the simulating P 3-stack automaton can also reach an accepting configuration by sending all terminal symbols out to the environment, and exchanging the symbol $c \in C$ from the first region to the symbol E from the second region, entering the accepting configuration (E, cT) for some $c \in C$.

Now, since in each step of the computation exactly one symbol from Σ enters the P automaton, we can define $f : V^* \to \Sigma$ for an $u \in V^*$ as $f(u) = a,\ a \in \Sigma$ for $|u|_a \neq 0$, thus, with such an f, we have $L(\Pi, f) = L_1 \cap L_2 \cap L_3 = L$. To accept the length preserving homomorphic image $h(L)$ of the intersection, we can modify the input mapping by $f'(u) = h(f(u))$, and consider the language $L(\Pi, f')$ of the P 3-stack-automaton.

To see that languages of P 3-stack-automata are quasi-realtime, that is, that they can be accepted by non-deterministic Turing machines in linear time, consider the following.

Looking at the construction of the P 3-stack-automaton, we can notice that the region other than the skin region can only contain a limited number of objects which means that they can be in a finite number of different configurations. The skin region can also contain a finite number of objects other than b_1, b_2, b_3, thus, to record a configuration of the P stack-automaton, we need to record the potentially unbounded number of b_1, b_2, b_3 in the skin membrane, plus a finite set of states to record the distribution of the objects in the other region.

If we look at the rules of the skin region, we can also notice that a record of the number of b_1, b_2, b_3 can be easily maintained. The number of b_is are changed by the rules

$$(b_1^{k^{m_1}}, in; b_1^k, out)|_{c_2},\ (b_2^{k^{m_2}}, in; b_2^k, out)|_{c_2},\ (b_3^{k^{m_3}}, in; b_3^k, out)|_{c_2} \tag{5}$$

and

$$(ab_1^{i_1}b_2^{i_2}b_3^{i_3}c_1, in; b_1^{j_1}b_2^{j_2}b_3^{j_3}c_2, out) \tag{6}$$

where $0 \leq j \leq t,\ a \in \Sigma,\ c_1, c_2 \in C,\ 0 \leq i_l, j_l \leq k^t,\ 1 \leq l \leq 3$, for some $t \in \mathbb{N}$, and $m_j = \log_k i_j,\ 1 \leq j \leq 3$.

Since at most one occurrence of one object from C is present in the skin region, the rules of type (5) multiply the number of b_is by at most k^t, thus, if we consider the k-ary number denoting the number of b_is, the rules of type (5) append at most t zeros to the end of the string of k-ary digits (or, if $j = 0$, then they erase the last digit from the end of the string). Moreover, since only one copy of a rule of type (6) can be used (because there is at most one copy of $c \in C$ present), the changes of the number of b_is caused by these rules can be recorded by replacing the zeros at the end of the number strings by non-zero digits. Since the length of the manipulated suffix of the digit string is bounded by t, the strings can be modified in a constant number of Turing machine steps, which means that computations of the P 3-stack automaton can be simulated by a non-deterministic Turing machine in linear time. □

4. Conclusion

In this paper we introduced the notion of a P n-stack-automaton, a generalization of P stack-automaton, introduced in [6] for describing the class of context-free languages in terms of symport/antiport P systems. Just like P stack-automaton was inspired by the concept of pushdown automaton, P n-stack-automata are motivated by classical automata having n, possibly more than one stack or pushdown store. Pushdown automata have crucial importance in automata theory since these machines accept exactly the class of context-free languages. Context-free languages are widely used in computer science, both in theory and in practical applications, but they do not have all properties which the applications need (for example, modelling natural languages non-context-free structures are needed as well). One possible way for having larger, more complex language classes is to extend the concept of a pushdown automaton. In the literature, we find several machine models with possibly more than one pushdown stores or stacks, like multi-pushdown automata, multi-stack automata and their variants, which have a significantly increased the accepting capability, even computational completeness can be obtained with these types of constructs. For example, interesting variants are the so-called shrinking multi-pushdown automata that with two pushdown stores describe the class of growing context-sensitive languages and with three pushdown stores the class of quasi-realtime languages [10]. (We note that in the case of these multi-pushdown automata models there is no separate one-way input tape, the input is provided as the initial contents of the first pushdown store.) Notice that P n-stack automata for $n = 3$ describe the class of quasi-realtime languages.

One challenging research direction for the future is to find restricted P automata classes that provide descriptions of language classes of other well-known machine models with possibly more than one pushdown stores.

5. Acknowledgment

This work was supported by Grant No. K 120558 of the National Research, Development, and Innovation Office - NKFIH, Hungary.

References

[1] Păun G. Computing with Membranes. Journal of Computer and Systems Sciences. 2000;6(1):108–143.

[2] Păun G, Rozenberg G, Salomaa A, editors. The Oxford Handbook of Membrane Computing. Oxford: Oxford University Press; 2010.

[3] Csuhaj-Varjú E, Vaszil G. P Automata. In: Pre-Proceedings of the Workshop on Membrane Computing WMC-CdeA 2002. vol. 1 of MolCoNet-IST-2001-32008; 2002. p. 177–192.

[4] Csuhaj-Varjú E, Vaszil G. P Automata or Purely Communicating P systems. In: Păun G, Rozenberg G, Salomaa A, Zandron C, editors. WMC2002. vol. 2597 of Lecture Notes in Computer Science. Springer, Berlin; 2003. p. 2019–233.

[5] Freund R, Oswald M. A Short Note on Analysing P Systems. EATCS Bulletin. 2002;78:231–236.

[6] Vaszil G. A Class of P Automata Characterizing Context-free Languages. In: Gutiérrez-Naranjo MA, et al, editors. Proc. BWMC4. vol. II of RGNC Report 03/2006. Fénix Editora; 2006. p. 267–276.

[7] Rozenberg G, Salomaa A, editors. Handbook of Formal Languages. Berlin Heidelberg: Springer-Verlag; 1997.

[8] Book RV, Greibach SA. Quasi-realtime Languages. Mathematical Systems Theory. 1970;4:97–111.

[9] Csuhaj-Varjú E, Ibarra OH, Vaszil G. On the Computational Complexity of P automata. Natural Computing. 2006;5(2):109–126.

[10] Holzer M, Otto F. Shrinking Multi-pushdown Automata. In: Liskiewicz M, Reischuk R, editors. FCT 2005. vol. 3623 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg; 2005. p. 305–316.