

Characterizations of Context-Sensitive Languages and Other Language Classes in Terms of Symport/Antiport P Systems*

Oscar H. Ibarra

Department of Computer Science, University of California
Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania, and
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
george.paun@imar.ro, gpaun@us.es

Abstract

We give “syntactic” characterizations of context-sensitive languages (CSLs) in terms of some restricted models of symport/antiport P systems. These are the first such characterizations of CSLs in terms of P systems. In particular, we show the following for any language L over a binary alphabet:

1. Let m be any integer ≥ 1 . Then L is a CSL if and only if it can be accepted by a restricted symport/antiport P system with m membranes and multiple number of symbols (objects). Moreover, holding the number of membranes at m , there is an infinite hierarchy in computational power (within the class of binary CSLs) with respect to the number of symbols.
2. Let s be any integer ≥ 14 . Then L is a CSL if and only if it can be accepted by a restricted symport/antiport P system with s symbols and multiple number of membranes. Moreover, holding the number of symbols at s , there is an infinite hierarchy in computational power with respect to the number of membranes.

Thus (1) and (2) say that in order for the restricted symport/antiport P systems to accept all binary CSLs, at least one parameter (either the number of symbols or the number of membranes) must grow. These are the first results of their kind in the P systems area. They contrast a known result that (unrestricted) symport/antiport P systems with $s \geq 2$ symbols and $m \geq 1$ membranes accept (or generate) exactly the recursively enumerable

*The work of Oscar H. Ibarra was supported in part by NSF Grants CCR-0208595, CCF-0430945, and CCF-0524136.

sets of numbers even for $s + m = 6$. We also note that previous characterizations of formal languages in the membrane computing literature are mostly for the Parikh images of languages.

Variations of our model yield characterizations of regular languages, languages accepted by one-way $\log n$ space-bounded Turing machines, and recursively enumerable languages.

Keywords: Symport/antiport P system, context-sensitive language, linear-bounded automaton, one-way $\log n$ space-bounded Turing machine, two-way multihead finite automaton.

1 Introduction

Membrane computing is a branch of natural computing which abstracts computing models from the structure and the functioning of the living cell. The main ingredients of membrane systems (currently called P systems) are (i) the *membrane structure*, which consists of a hierarchical arrangement of membranes which delimit compartments where (ii) *multisets* of symbols (called *objects*) evolve according to (iii) *sets of rules* which are also localized, associated with compartments, [14]. In basic models of P systems – and this will be the case also in this paper – the rules are used in the non-deterministic maximally parallel manner. With the computations which halt, a result is associated, in general in the form of the vector of natural numbers which describes the multiset of objects from a designated membrane, in the halting configuration. Details can be found in [15], with a comprehensive information about membrane computing being available at the website [22].

Therefore, in this set-up, we can compute sets of vectors of numbers. Strings (hence languages) can be computed, for instance, when using P systems in the *accepting mode*, and this is especially suitable for so-called *symport/antiport* P systems, [13]. Such systems use rules which move objects through membranes in the way coupled chemicals pass through protein channels in cell biology; in particular, objects can be sent and imported from the environment of the system. Thus, starting from an initial configuration of a P system and recording (some of) the symbols which are brought into the system during a halting computation, we can get a string. Several classes of symport/antiport P systems working in this way (and the same is true for symport/antiport P systems working in the generative mode) were proven to be computationally universal, characterizing the family of recursively enumerable languages.

The present paper deals with a natural related problem: can we characterize other families of languages by means of classes of symport/antiport P systems? The problem is trivial for regular languages, but challenging for other families from Chomsky hierarchy, in particular, for the families of context-free and context-sensitive languages. Actually, we leave *open* the case of context-free languages and we provide here only a characterization of context-sensitive languages (and of other families), by means of some restrictions on the form of symport/antiport rules (that is why we call these characterizations “syntactic”). Several variants of the considered conditions are examined, with some combinations leading again to computational universality.

It is worth noting that the problem of characterizing Chomsky families of languages other than those of regular and of recursively enumerable languages was a challenge and it is basically open also in another area of bio-inspired natural computing, DNA computing based on splicing, see [18].

Characterizations of context-free languages in terms of P systems were obtained before, for instance, in [3], but they are not syntactic – they directly link the idea of a derivation tree to the tree describing a membrane structure (the string of the language is encoded in the labels of the membranes), hence the characterization is obtained in a completely different set-up.

2 Preliminaries

The reader is supposed to be familiar with basic elements of language and automata theory, as well as of membrane computing (from [15], [17], etc.), that is why we introduce here only some notations and the class of P systems we investigate, the symport/antiport P systems (with promoters).

As usual, we denote by V^* the set of all strings (the empty string, λ , included) over an alphabet V and by V^+ the set $V^* - \{\lambda\}$, of all non-empty strings over V . The length of $x \in V^*$ is denoted by $|x|$. The families of regular, context-free, context-sensitive, and recursively enumerable languages are denoted by REG, CF, CS, RE , respectively.

An accepting P system with symport/antiport rules (a *symport/antiport acceptor*, in short, an SAA) is a construct of the form $\Pi = (V, \Sigma, \mu, w_1, \dots, w_m, E, R_1, \dots, R_m)$, where V is the alphabet of objects, $\Sigma \subseteq V$ is the input alphabet, μ is the membrane structure (of degree $m \geq 1$, with the membranes labelled in a one-to-one manner with $1, 2, \dots, m$; as usual, we represent the membrane structures by strings of matching labelled parentheses), w_1, \dots, w_m are strings over $V - \Sigma$ representing the multisets of objects present in the m compartments of μ at the beginning of a computation, $E \subseteq O$ is the set of objects supposed to appear in the environment in arbitrarily many copies, and R_1, \dots, R_m are (finite) sets of rules associated with the m membranes of μ .

The rules from R_1, \dots, R_m can be of two types:

- *Symport rules*, of the forms (x, in) or (x, out) , where $x \in V^+$. When using such a rule, the objects specified by x enter or exit, respectively, the membrane with which the rule is associated. In this way, objects are sent to or imported from the surrounding region – which is the environment in the case of the skin membrane. (The length of x in a symport rule is called the *weight* of the rule.)
- *Antiport rules*, of the form $(x, out; y, in)$, where $x, y \in V^+$. When using such a rule for a membrane i , the objects specified by x exit the membrane and those specified by y enter from the region surrounding membrane i ; this is the environment in the case of the skin membrane. (The maximal length of x, y is called the *weight* of the rule.)

Rules as above can be applied as soon as the respective multisets x and y are present in the specified regions. A further control on the use of rules can be imposed by using *promoters*. Such rules are written in the form $(x, in)|_a, (x, out)|_a, (x, out; y, in)|_a$, where $x, y \in V^+$ and $a \in V$; when they are associated with a membrane i , these rules can be applied only if a is present in membrane i . The object a can promote at the same time several rules, it is not “consumed” by the use of the promoted rules, but a promoter cannot be used by the rule it promotes (in the previous writing, a cannot be an element of multiset x).

An SAA accepts a language in the following way. We start from the initial configuration, the one defined by μ and the multisets w_1, \dots, w_m , and we use the rules in R_1, \dots, R_m in the non-deterministic maximally parallel manner, usual in membrane computing. During the computation, objects from Σ can be brought into the system, from the environment; these objects are arranged in a sequence, corresponding to the moments when they enter the system (if several objects come at the same time, then any permutation of them is considered), and, if the computation halts, then the strings defined in this way are said to be accepted by the computation. The language of all strings accepted in this way by a system Π is denoted by $L(\Pi)$.

Symport/antiport systems (without promoters), with a small number of membranes and with symport/antiport rules of small weights, characterize *RE* (or, in the generative case which we do not specify here, the family of Turing computable sets of numbers) – see [15], [2], [4], etc.

On the other hand, as we will see in Section 6.2, a characterization of *REG* in terms of SAAs is easy to obtain.

The present paper addresses the problem of characterizing families intermediate between *REG* and *RE*, especially *CS*, and to this aim in the next section we introduce a restricted class of SAAs. Several conditions about the form of rules will be considered, and their combinations and variants will lead to characterizations of various language families. In particular, we will also get a further characterization of *RE*, and to this aim we use counter machines, that is why we also introduce this notion here (in the deterministic accepting version).

A counter machine is a construct $M = (m, H, l_0, l_h, R)$, where m is the number of counters (one also says registers), H is the set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to instruction HALT), and R is the set of instructions; each label from H labels only one instruction from R , thus precisely identifying it. The instructions are of the following forms:

- $l_1 : (\text{ADD}(r), l_2)$ (add 1 to counter r and then go to the instruction with label l_2),
- $l_1 : (\text{SUB}(r), l_2, l_3)$ (if counter r is non-empty, then subtract 1 from it and go to the instruction with label l_2 , otherwise go to instruction with label l_3),
- $l_h : \text{HALT}$ (the halt instruction).

A counter machine M accepts a number n in the following way: we start with counter 1 containing the number n and all other counters empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of counters); if we reach the halt instruction, then the number n is accepted. The set of all numbers accepted by M is denoted by $A(M)$. Without loss of generality, we may assume that in the halting configuration, all counters are empty. It is known (see, e.g., [12]) that counter machines (even with a small number of counters, but this detail is not of interest here) accept exactly all sets of numbers which are Turing computable.

We can pass from languages to numbers (and back) in the following way. Consider an alphabet V with k elements. By interpreting the symbols in V as digits $1, 2, \dots, k$, each string in V^+ can then be thought of as a number in k -adic notation. Specifically, the value

of $x = d_n d_{n-1} \dots d_0$ in base k is $N_k(x) = \sum_{i=0}^n d_i k^i$. Then, a language $L \subseteq V^+$ is in *RE* if and only if $N_k(L) = \{N_k(x) \mid x \in L\}$ is a Turing computable set of numbers, hence if and only if there is a counter machine M such that $N_k(L) = A(M)$.

We will make below an essential use of this numerical codification of strings. (Always when the base is understood, we omit the subscript k of N_k .)

Convention: when comparing the languages generated/accepted by two devices, the empty string is ignored, two devices are equivalent if their languages differ in at most the empty string.

3 The Model

We introduce now the class of SAAs which we use in characterizing CSLs and other classes of languages.

Define an *exponential symport/antiport acceptor* (ESAA) as a 1-membrane SAA $\Pi = (V, \Sigma, []_1, w_1, V - \Sigma, R_1)$ with the components as specified in the previous section, but with the following specific properties: the input alphabet $\Sigma \subseteq V$ contains a distinguished symbol $\$$ (the end marker), the environment contains all objects from $V - \Sigma$ (and no object from Σ), and the rules are of the following four types:

1. $(u, out; v, in)$, where $u, v \in (V - \Sigma)^*$ with $|u| \geq |v|$.
2. $(u, out; va, in)$, where $u, v \in (V - \Sigma)^*$ with $|u| \geq |v|$, and $a \in \Sigma$. A rule of this type is called a *read-rule*.
3. $(u, out; v, in)|_a$, where $u, v \in (V - \Sigma)^*$, and $a \in \Sigma$ (a is a promoter). Note that there is no restriction on the relative lengths of u and v . In particular, the length of v can be greater than the length of u .
4. For every $a \in \Sigma$, there is at least one rule of the form $(a, out; v, in)$ in the set R_1 , where $v \in (V - \Sigma)^*$. Moreover, this is the only type of rules for which a can appear on the left part of the rule.

An ESAA accepts a language in the following way (note some important differences from the way a general SAA works).

An input string $x = a_1 \dots a_n \$$, where a_i is in $\Sigma - \{\$\}$ for $1 \leq i \leq n$, is provided online in the environment. The symbols are brought into the system in the order they appear in x by means of read-rules (i.e., rules of the form $(u, out; va, in)$). Maximal parallelism in the application of the rules is assumed as usual. Hence, in general, the size of the multiset of rules applicable at each step is unbounded, and, in particular, the number of instances of read-rules applicable in a step is unbounded. However, the number of read-rules in an applicable multiset cannot exceed the number of symbols of x remaining to be read, and the symbols in these read-rules, say there are s of them, must be consistent with the next s symbols of the input string x that have not yet been processed. Note that rules of types 1, 3, and 4 do not consume any input symbol from x . Note also that because of maximal parallelism, any symbol $a \in \Sigma$ that is imported from the environment by a rule of type 2 is always transported back to the environment in the following step by a rule of type 4. When a rule of type 4 is

applied, the symbol a that is exported to the environment does not get inserted to the input string, that is, it is never brought again in the system.

The input string $x = a_1 \dots a_n \$$ is accepted if, after reading *all* the input symbols, Π eventually halts.

Note the important fact, essentially used in some proofs, that if the computation halts before reading the end marker $\$,$ then the computation aborts, and no result is provided.

The language accepted is $L(\Pi) = \{a_1 \dots a_n \mid a_1 \dots a_n \$ \text{ is accepted by } \Pi\}$ (we do not include the end marker).

We have two versions of ESAAs: non-deterministic and deterministic, where in the deterministic case, the maximally parallel multiset of rules applicable at each step of the computation is unique.

4 A Characterization of Languages Accepted by ESAAs

In this section, we will show that non-deterministic (deterministic) ESAAs are equivalent to non-deterministic (deterministic) linear-bounded automata.

A non-deterministic linear-bounded automaton (NLBA) M is a generalization of a non-deterministic two-way finite automaton with input markers in that the input head can rewrite the symbols on the tape, except the end markers. An input $\$a_1 \dots a_n \$$ is accepted if M , when started in its initial state on the left end marker, eventually enters an accepting state. The language accepted by M is $L(M) = \{a_1 \dots a_n \mid \$a_1 \dots a_n \$ \text{ is accepted}\}$. It is well-known that an NLBA is equivalent to a linear space-bounded non-deterministic Turing machine. A deterministic linear-bounded automaton will be denoted by DLBA. NLBAs (DLBAs) accept exactly the context-sensitive (deterministic context-sensitive) languages.

Lemma 4.1 *Any language accepted by a non-deterministic (deterministic) ESAA can be accepted by an NLBA (DLBA).*

Proof. We first consider the deterministic case. The construction uses ideas in [9, 10]. Let Π be a deterministic ESAA with an alphabet V of m symbols (note that V contains the input alphabet Σ). Assume that $V = \{a_1, \dots, a_m\}$. We will construct a deterministic TM M with a one-way read-only input tape with a right end marker and several read-write worktapes. We describe the operation of M when given input $x \$$, where $x \in (\Sigma - \{\$\})^*$ and $\$$ is the end marker. M will simulate the computation of Π and accepts if and only if Π accepts. Moreover, M uses no more than $O(n)$ space on each worktape, where $n = |x|$.

M will need the following worktapes to keep track of the multiplicities of the symbols in the system during the computation:

1. Worktape W_i for $1 \leq i \leq m$. W_i will keep track of the current number of a_i 's.
2. Worktape $W_{i,j}$ for $1 \leq i, j \leq m$. These worktapes will keep track of how many a_i 's are replaced by a_j 's during the next step of Π .

One step of Π is simulated by a (possibly unbounded) number of steps of M . At the beginning of the simulation of every step of Π , M resets all $W_{i,j}$'s to zero. To determine the next configuration of Π , M processes the rules as follows:

Let r_1, r_2, \dots, r_s be the rules in the membrane. By using W_1, \dots, W_m (note each W_i represents the number of a_i 's in the membrane), M applies rule r_1 sequentially a maximal number of times storing the "results" (i.e., the number of a_i 's that are converted by the applications of rule r_1) to a_j in worktape $W_{i,j}$. Thus, each application of r_1 may involve decrementing the W_i 's and incrementing some of the $W_{i,j}$'s. (By definition, the sequential application of r_1 has reached its maximum at some point, if further application of the rule is no longer applicable.)

The process just described is repeated for the other rules r_2, \dots, r_s . When all the rules have been processed, M updates each worktape W_j using the values stored in $W_{i,j}$, $1 \leq i \leq m$. This completes the simulation of the unique (because Π is deterministic) maximally parallel step of Π .

Clearly, from the description above, Π can be simulated by M , and M accepts the language accepted by Π . Now let d be the smallest integer such that in every rule of type 3, $d|u| \geq |v|$. Clearly, there is a constant c such that for any input $x\$$ with $n = |x|$, the multiplicity of every symbol in the membrane at any time after reading i input symbols from input is at most cd^i . Hence each worktape holds a count of no more than cd^n . It follows that each worktape is linear (in n) space-bounded. The TM M can then be converted to a DLBA M' (with a large enough tape alphabet).

If Π is nondeterministic, the construction of the nondeterministic TM M is simpler. M just sequentially guesses the rule to include in the maximal multiset of rules (i.e., any of r_1, \dots, r_s) until no more rule is applicable. \square

Lemma 4.2 *Any language accepted by an NLBA (DLBA) can be accepted by a non-deterministic (deterministic) ESAA.*

Proof. Let M be an NLBA with input alphabet $\Sigma = \{1, 2, \dots, k, \$\}$. As in Section 2, we represent a string $x = d_n \dots d_0$ (each d_i in $\{1, \dots, k\}$) as an integer $N(x) = d_n k^n + d_{n-1} k^{n-1} + \dots + d_1 k^1 + d_0 k^0$. We show how to construct an ESAA Π equivalent to M . We describe the rules of Π in two parts.

We first describe the rules in Part 1. Let $V = \Sigma \cup \{\#, o\} \cup W$, where W consists of symbols used in Part 2; it contains o , but does not contain $\Sigma \cup \{\#\}$. The rules in Part 1 are:

1. $(\#, out; j, in)$ for every $j \in \Sigma$.
2. $(o, out; o^k, in)|_j$ for every $j \in \Sigma - \{\$\}$.
3. $(j, out; \#o^j, in)$ for every $j \in \Sigma - \{\$\}$.
4. $(\$, out; w, in)$, where $w \in (W - \{o\})^*$ is a string to be defined below.

Note that if the membrane starts with object $\#$, then when the input string is $x = d_n \dots d_0 \$$, the system will end up with $wo^{N(x)}$ in the membrane. Clearly, the computation above is deterministic.

Before proceeding further, we need to relate the computation of the NLBA (DLBA) to the computation of a non-deterministic (deterministic) two-way multihead finite automaton

over a *unary* alphabet (with end markers). We use N2MFA (D2MFA) to denote the latter machine.

In [19], it was shown that given an NLBA (DLBA) M , one can construct an N2MFA (D2MFA) M_1 such that x is accepted by M if and only if $o^{N(x)}$ is accepted by M_1 .

It was also recently shown in [9] (see also [10]) that an N2MFA (D2MFA) M_1 over a unary alphabet can be simulated by a non-deterministic (deterministic) ESAA Π_1 all of whose rules are of type 1, i.e., of the form $(u, out; v, in)$, where $|u| \geq |v|$. Thus, Π_1 when started with $o^{N(x)}$, along with a fixed multiset of symbols w not containing o , simulates M_1 and halts if M_1 accepts. (Note that this w is the string in rule 4 above.) In addition to the symbols in w , Π_1 uses other non- o symbols during the computation. Thus, Π_1 uses o and other symbols which (without loss of generality) we assume are different from those in $\Sigma = \{1, 2, \dots, k, \$\}$ and $\#$. Let W be the set of symbols in Π_1 . The number of such symbols depend on the specification of the N2MFA (D2MFA), i.e., on its transition rules and the number of heads it has.

The rules of Π_1 (which are of type 1) become Part 2 of the rules in Π . The starting multiset of Π is $\#$. It follows that Π with the combined Part 1 and Part 2 rules simulates the NLBA (DLBA) M . \square

From Lemmas 4.1 and 4.2, we have:

Theorem 4.1 *A language L is accepted by a non-deterministic (deterministic) ESAA if and only if L is accepted by an NLBA (DLBA).*

The following theorem follows from Theorem 4.1 and the fact that it holds for DLBAs and NLBAs. (Closure under complementation for NLBAs was shown in [11], [21].)

Theorem 4.2 *1. The family of languages accepted by deterministic (non-deterministic) ESAA is closed under union, intersection, complementation, concatenation, and Kleene*.*

2. The membership problem is decidable for non-deterministic ESAAs.

3. The emptiness problem is undecidable for deterministic ESAAs.

Consider now the case of the input alphabet $\Sigma = \{1, 2, \$\}$ (i.e., binary, since we ignore the end marker). We also restrict the type 3 rule so that it has the following form:

$$(u, out; v, in)|_a \text{ with } 2|u| \geq |v|.$$

Call the above a 2-ESAA. Then the following corollary follows from the proofs of Lemmas 4.1 and 4.2.

Corollary 4.1 *A language $L \subseteq \{1, 2\}^*$ is accepted by an NLBA (DLBA) if and only if it is accepted by a non-deterministic (deterministic) 2-ESAA.*

5 Hierarchy of 2-ESAA's with Respect to the Alphabet Size

In this section, we show that there is an infinite hierarchy of languages accepted by 2-ESAA's with respect to the size of the alphabet V . We will need the following result from [7]:

Lemma 5.1 *For every $n \geq 1$, there is an $m > n$ and a language $L \subseteq \{1, 2\}^*$ that is accepted by an NLBA (DLBA) with a tape alphabet of m symbols that cannot be accepted by an NLBA (DLBA) with a tape alphabet of n symbols. (Note that 1, 2, \$ are part of the tape alphabet.)*

Theorem 5.1 *For every r , there is an $s > r$ such that non-deterministic (deterministic) 2-ESAA's with an alphabet of s symbols can accept more languages than non-deterministic (deterministic) 2-ESAA's with an alphabet of r symbols.*

Proof. Suppose there is an r such that all binary languages accepted by non-deterministic (deterministic) 2-ESAA's can be accepted by non-deterministic (deterministic) 2-ESAA's with alphabet V of size at most r . Then it is easy to see (see Lemma 4.1) that there is a constant r' , which depends only on r , such that these 2-ESAA's can be simulated by NLBA's (DLBA's) that use at most r' tape symbols. Then all binary languages accepted by NLBA's (DLBA's) can be accepted by NLBA's (DLBA's) with tape alphabet of size at most r' . But this contradicts Lemma 5.1. \square

6 Variations of the ESAA Model

ESAA's have four types of rules. In this section, we look systematically, considering all possible cases, at the classes of languages generated when one or two types of rules are not allowed.

Let us start by observing that we have two kinds of restrictions in the rules of an ESAA, those related to the length of the strings (in rules of types 1 and 2), and those related to the role of terminal symbols (rules of types 3 and 4). In particular, the obligatory existence of rules of type 4 is a very strong restriction, because we can make use of rules of type 3 (promoted, but without any restriction on the length of the strings) only one step after bringing a terminal into the system. Thus, presumably, removing rules of type 4 add power – and this will be confirmed below.

6.1 Type 2

Only rules of type 2 can bring terminals into the system, hence such rules cannot be avoided. Still, such rules are sufficient to accept at least (as we will see in the next section, *exactly*) all regular languages.

Lemma 6.1 *Any regular language $L \subseteq T^*$ can be accepted by a deterministic ESAA having only rules of type 2.*

Proof. Let $A = (K, T, s_0, F, \delta)$ be a DFA. We construct the SAA

$$\Pi = (K \cup T \cup \{X, \$\}, T \cup \{\$\}, [1]_1, s_0, R_1), \text{ where}$$

$$\begin{aligned}
R_1 &= \{(s, out; as', in) \mid \delta(s, a) = s'\} \\
&\cup \{(s, out; aX, in) \mid \delta(s, a) \in F\} \\
&\cup \{(X, out; \$, in)\}.
\end{aligned}$$

A computation in Π stops by reading $\$$ if and only if the accepted string is from $L(A)$. \square

Remark. We observe that the result above is still valid if the input string to Π does not have the end marker $\$$. We just modify R_1 to be: $R_1 = \{(s, out; as', in) \mid \delta(s, a) = s'\} \cup \{(s, out; a, in) \mid \delta(s, a) \in F\}$. We note that the same observation (about the results being valid when the input strings have no end marker) will be true for some other models discussed in this paper. We discuss this issue further in Section 8.

6.2 Types 1, 2

Let us now add rules of type 1. Thus, the only rules are of the forms $(u, out; v, in)$ and $(u, out; va, in)$, where $u, v \in (V - \Sigma)^*$ with $|u| \geq |v|$, and $a \in \Sigma$. Call this new model a *regular SAA*.

Clearly, for such systems, the input symbols that are read-in during the computation remain in the membrane (and not exported back to the environment), so there is no need to keep track of their multiplicities. Moreover, because $|u| \geq |v|$ in rules of type 1, the number of symbols in the membrane does now grow during the computation. Hence, such a system can be simulated by an NFA. Conversely, from Lemma 6.1 we know that rules of type 2 suffice in order to accept all regular languages, hence we have the following result:

Theorem 6.1 *A language L can be accepted by a non-deterministic (deterministic) regular SAA if and only if L can be accepted by an NFA (DFA).*

Corollary 6.1 *A language is regular if and only if it can be recognized by a regular SAA using only rules of type 2.*

6.3 Types 1, 2, 4

Let us continue by also adding rules of type 4 (hence no rules of type 3 are allowed). Thus, the rules are of the forms $(u, out; v, in)$ and $(u, out; va, in)$, where $u, v \in (V - \Sigma)^*$ with $|u| \geq |v|$, and $a \in \Sigma$. And, as before, for every $a \in \Sigma$, there is at least one rule of the form $(a, out; v, in)$ in the membrane, where $v \in (V - \Sigma)^*$. We call this model a *linear SAA*, or simply LSAA. We will show that this model is equivalent to a restricted form of one-way $\log n$ space-bounded Turing machine.

A non-deterministic one-way Turing machine is *restricted $S(n)$ space-bounded* if for every accepted input of length n , there is an accepting computation where the number of nonempty cells on the work-tape(s) is bounded by $S(d)$ where $d \leq n$, and d is the number of input tape cells already read, that is, the *distance* of the reading head from the left end of the one-way input tape [4]. We, are interested in the case when $S(n)$ is logarithmic.

Theorem 6.2 *A language L is accepted by a non-deterministic (deterministic) LSAA Π if and only if L is accepted by a non-deterministic (deterministic) restricted one-way $\log n$ space-bounded Turing machine M .*

Proof. Let Π be an LSAA. Clearly, every time the second type of rules is applied, the third type, i.e., of the form $(a, out; v, in)$, is applied in the following step. Let k be the maximum length of the multisets v in such rules. Then, for some constant c , the number of symbols in the system at any time after reading the first i symbols of the input is at most $c + ki$. It follows that we can construct a restricted one-way *log n* space-bounded Turing machine M equivalent to Π .

For the proof of the converse, we need a recent result from [10] concerning a restricted model of a communicating P system [20], called SCPA. An SCPA Π has *multiple* membranes, with the skin membrane labeled 1. The symbols in the initial configuration (distributed in the membranes) are *not* from Σ (the input alphabet). The rules are of the form:

1. $a \rightarrow a_x$,
2. $ab \rightarrow a_x b_y$,
3. $ab \rightarrow a_x b_y \sigma_{come}$,

where x, y in $\{here, out, in_j\}$ (see [20]). The input to Π is a string $z = a_1 \dots a_n \$$, which is provided externally online. The restrictions on the operation of Π are the following:

1. There are no rules in membrane 1 with a_{out} or b_{out} on the right-hand side of the rule (i.e., no symbol can be expelled from membrane 1 into the environment).
2. A rule of type 3 (called a read-rule) can only appear in membrane 1. This brings in σ if the next symbol in the input string z that has not yet been processed (read) is σ ; otherwise, the rule is not applicable.
3. Again, in general, the size of the maximally parallel multiset of rules applicable at each step is unbounded. In particular, the number of instances of read-rules (i.e., rules of the form $ab \rightarrow a_x b_x \sigma_{come}$) applicable in a step is unbounded. However, the number of read-rules in an applicable multiset cannot exceed the number of symbols of z remaining to be read, and the symbols in these read-rules, say there are s of them, must be consistent with the next s symbols of the input string z that have not yet been processed.

The system starts with an initial configuration w which consists of some symbols from $V - \Sigma$ distributed in the membranes. The input string $z = a_1 \dots a_n \$$ is accepted if, after reading all the input symbols, the SCPA eventually halts. The language accepted by Π is $L(\Pi) = \{a_1 \dots a_{n-1} \mid a_1 \dots a_n \text{ is accepted by } \Pi\}$ (we do not include the end marker).

It was shown in [10] that a language L is accepted by a non-deterministic (deterministic) restricted one-way *log n* space-bounded Turing machine if and only if it is accepted by a non-deterministic (deterministic) SCPA. (Actually, [10] showed that an SCPA can simulate a restricted one-way linear-space (multi) counter machine, which is equivalent to a restricted one-way *log n*-space Turing machine.) Hence, we need only show that an SCPA Π can be simulated by an LSAA.

So let Π be an SCPA. First we construct an equivalent symport/antiport system Π' that may not yet be an LSAA. Suppose Π has membranes $1, \dots, m$, with index 1 representing the skin membrane. For each object a in V , Π' will have symbols a_1, \dots, a_m . In particular, for each input symbol σ in $\Sigma \subseteq V$, Π' will have $\sigma_1, \dots, \sigma_m$. We convert Π to Π' as follows:

1. If $a \rightarrow a_x$ is a rule in membrane i of Π , then $(a_i, out; a_j, in)$ is a rule in Π' , where j is the index of the membrane into which a is transported to, as specified by x .
2. If $ab \rightarrow a_x a_y$ is a rule in membrane i of Π , then $(a_j b_i, out; a_j b_k, in)$ is a rule in Π' , where j and k are the indices of the membranes into which a and b are transported to, as specified by x and y .
3. If $ab \rightarrow a_x b_y \sigma_{come}$ in membrane 1 of Π , then $(a_1 b_1, out; a_j b_k \sigma_1, in)$ is a rule in Π' , where j and k are the indices of the membranes into which a and b are transported to, as specified by x and y .

Corresponding to the initial configuration w of Π , where w represents the configuration denoting the initial multisets the membranes, Π' will have initial configuration w' , where w' are symbols in w renamed to identify their locations in Π .

Now define the input alphabet of Π' to be $\Sigma' = \{\sigma_1 \mid \sigma \in \Sigma\}$. Clearly, Π' accepts $z' \in \Sigma'^*$ if and only if Π accepts $z \in \Sigma^*$, where z' is the string z where each symbol is subscripted with 1.

Finally replace each read-rule $(a_1 b_1, out; a_j b_k \sigma_1, in)$ in Π' by three rules:

$$\begin{aligned} & (a_1 b_1, out; a'_j b'_k \sigma, in), \\ & (a'_j b'_k, out; a_j b_k, in), \\ & (\sigma, out; \sigma_1, in), \end{aligned}$$

where a'_j, b'_k are new symbols. Call the resulting system Π'' , and let Σ be its input alphabet. The purpose of the transformation is to make sure that rules of the form $(u, out; v, in)$ and $(u, out; v\sigma, in)$ in Π'' satisfy $u, v \in (V - \Sigma)^*$. Note that the computation of Π'' when reading a new symbol is delayed by one step. However, one can check that this does not create any problem, and Π'' accepts $L(\Pi)$. Moreover, Π'' is non-deterministic (deterministic) if Π is non-deterministic (deterministic). \square

Corollary 6.2 *Special SAAs and LSAs are equivalent. This holds for both non-deterministic and deterministic cases.*

The next result follows from Corollary 6.2 and the fact that it holds for special SAAs [9, 10].

Corollary 6.3 *Deterministic LSAs are strictly weaker than non-deterministic LSAs.*

6.4 Types 2, 4, and 2, 3, 4

By adding rules $(a, out, v, in), a \in \Sigma = T \cup \{\$\}, v \in (V - \Sigma)^*$, to the system in the proof of Lemma 6.1, the language accepted is not changed, but the system becomes of type 2, 4 (or 2, 3, 4), hence such systems accept all regular languages.

Because of the rules of type 4, all computations in a system of types 2, 4 (or 2, 3 4) halt after (or before) reading the string, irrespective which is the string; the only control about

the string of symbols is provided by the finite sequence of antiport rules used, and by the fact that we have to stop with reading the end marker.

However, these two controls, weak as they seem to be, are sufficient in order to accept non-semilinear languages.

Here is an example: take $\Sigma = \{a, b, \$\}$ (all other symbols used below are in $V - \Sigma$), the initial multiset cf , and the following rules:

- $(c, out; a, in),$
- $(a, out; dd, in),$
- $(d, out; b, in),$
- $(b, out; cc, in),$
- $(cf, out; b, in),$
- $(df, out; a, in),$
- $(f, out; \$, in),$
- $(\$, out; c, in).$

The end marker can be read only by the rule $(f, out; \$, in)$; if f is sent out by any of the rules $(cf, out; b, in), (df, out; a, in)$, then reading $\$$ is no longer possible. However, a rule $(cf, out; b, in), (df, out; a, in)$ should be used if the sequence of input symbols is not of the form $ab^2a^4 \dots a^{2^{2i}}b^{2^{2i+1}} \dots \$$. Before the end marker we can have a substring of symbols a or b which is not a power of 2. Thus, if the string ends with symbols b , then it contains a number of symbols a of the form $\sum_{i=0}^n a^{2^{2i}}$; if the string ends with symbols a , then it contains a number of symbols b of the form $\sum_{i=0}^n b^{2^{2i+1}}$. In both cases, the language is not semilinear.

A more precise characterization of the power of ESAA's with rules of types 2, 4, or of types 2, 3, 4 remains to be found.

6.5 Types 1, 2, 3', 4

Suppose we replace the type 3 rule by the following (type 3') rule:

$$(u, out; v, in)|_a, \text{ where } u, v \in (V - \Sigma)^* \text{ with } |u| \geq |v|, \text{ and } a \in \Sigma \text{ (} a \text{ is a promoter).}$$

Clearly, because of the restriction that $|u| \geq |v|$, a restricted $\log n$ space-bounded Turing machine can simulate the computation of this system. Hence, such systems are equivalent to LSAA's (in both the non-deterministic and deterministic cases).

6.6 Types 2, 3, and 1, 2, 3

If we do not have to send back to the environment the terminal symbols immediately after reading them, then this gives us the opportunity to use them as promoters of rules of type 3 at any time. As these rules have no restriction on the length of the strings, this freedom leads to Turing computational power.

Theorem 6.3 *A language is accepted by an ESAA with rules of types 2, 3 if and only if it is a recursively enumerable language.*

Proof. Let us consider a language $L \subseteq \{1, 2, \dots, k\}^*$. As pointed out in Section 2, L is recursively enumerable if and only if $N(L) = A(M)$ for some (deterministic) counter machine $M = (m, H, l_0, l_h, R)$. Starting from this observation, for the “if” part of the statement we construct the following ESAA:

$$\begin{aligned}\Pi &= (V, \Sigma, [1]_1, w_1, V - \Sigma, R_1), \text{ where} \\ V &= \Sigma \cup \{l, l', l'', l''', l^{iv} \mid l \in H\} \cup \{a_r \mid 1 \leq r \leq m\} \\ &\quad \cup \{b, b_0, b_1, b_2, b_3, c, d, e, e', e'', f, g, t\}, \\ \Sigma &= \{1, 2, \dots, k\} \cup \{\$\}, \\ w_1 &= c^k d e',\end{aligned}$$

and the set of rules, R_1 , is constructed as follows.

In all these rules, α is any symbol from Σ , hence when we write $(u, out; v, in)|_\alpha$ it is understood that we have $k + 1$ rules of this form, one for each $\alpha \in \{1, 2, \dots, k, \$\}$.

The computations of Π will consists of two phases, and we present the rules separately for these phases.

Phase 1: reading a string $x\$$ and producing $N_k(x)$.

Reading a symbol from Σ is carried out in three steps, using the following rules:

$$\begin{aligned}\text{Step 1: } & (c^k, out; j b_2^j, in), \quad (e, out; e', in)|_\alpha, \quad (b, out; b_0^k g^k, in), \\ \text{Step 2: } & (e', out; e'', in)|_\alpha, \quad (b_2, out; b_3, in)|_\alpha, \quad (b_0, out; b_1, in)|_\alpha, \\ \text{Step 3: } & (e'', out; e c^k, in)|_\alpha, \quad (b_3, out; b, in)|_\alpha, \quad (b_1 g, out; b, in)|_\alpha.\end{aligned}$$

It is easy to see that in this way we compute the value of x in base k and in the system we get $N(x)$ copies of the object b . Object g is just a witness of this process; it is important to note that in step 1 we introduce the same number of copies of b_0 and of g , the copies of g wait unchanged in step 2, and all of them are sent to the environment in Step 3, together with the copies of b_1 . During these steps, object d waits in the system. (Only the reading rule is of type 2, all other rules are of type 3; the reading rule has to use c^k in the left hand string in order to fulfill the length restriction for the two multisets involved in it. No restriction is imposed to the length of the strings used in the other rules, and this makes possible bringing back k copies of c in the system, in the third step of the computation.)

When we read the marker $\$$, we use the following rules:

$$(c^k d e, out; \$ l_0 f, in), \quad (b, out; a_1, in)|_\alpha.$$

In this way, the copies of b present in the system are replaced by a_1 . (At each moment, the number of copies of a_r , $1 \leq r \leq m$, present in the system represents the contents of counter r of M .) Simultaneously, the start label of M , l_0 , is introduced, thus triggering the simulation of a computation in M (starting with the first counter holding the number $N(x)$), and the “carrier” e is sent out, thus not bringing again copies of c in the system.

Besides these rules, for the first phase we also introduce the following “control” rules:

$$(f g, out; t, in)|_\alpha, \quad (d a_1, out; t, in)|_\alpha, \quad (t, out; t, in)|_\alpha.$$

These rules ensure the correctness of the use of the other rules specified above in the following way.

If in the moment when we encounter the marker \$, together with the rule $(c^k de, out; \$l_0 f, in)$ we do not also use the rule $(b, out; a_1, in)|_\alpha$ for all copies of b present in the system, but, instead, we use at least once the rule $(b, out; b_0^k g^k, in)|_\alpha$, then copies of g are brought into the system, and in the next step the rule $(fg, out; t, in)|_\alpha$ must be used (remember that objects g cannot be processed in the next step by any rule). In this way, the trap-object t is introduced, and the computation will never stop because of the rule $(t, out; t, in)|_\alpha$.

Similarly, if the rule $(b, out; a_1, in)|_\alpha$ is used prematurely, during the reading of x , instead of $(b, out; b_0^k g^k, in)$, then the rule $(da_1, out; t, in)$ is used in the next step, and again the computation never stops.

Thus, the encoding of x into $N(x)$ should be completed exactly in the moment when the reading of x is terminated.

If the rules above are used in such a way that the trap-object is not introduced, then the system contains inside $N(x)$ copies of a_1 , as well as the objects l_0, f ; the copy of f will play no role in the rest of the computation.

Phase 2: the simulation of a computation in M . This is done exactly as in [6], but all the rules being promoted; for the sake of completeness, we briefly recall here the construction.

An ADD instruction $l_1 : (\text{ADD}(r), l_2)$ is simply simulated by an antiport rule

$$(l_1, out; l_2 a_r, in)|_\alpha.$$

A SUB instruction $l_1 : (\text{SUB}(r), l_2, l_3)$ is simulated in three steps, using the next rules:

$$\begin{aligned} \text{Step 1: } & (l_1, out; l_1' l_1'', in)|_\alpha, \\ \text{Step 2: } & (l_1' a_r, out; l_1''', in)|_\alpha, \quad (l_1'', out; l_1^{iv}, in)|_\alpha, \\ \text{Step 3: } & (l_1^{iv} l_1', out; l_3, in)|_\alpha, \quad (l_1^{iv} l_1''', out; l_2, in)|_\alpha. \end{aligned}$$

In the second step, l_1' tries to subtract one from counter r , while l_1'' brings inside the “checker” l_1^{iv} ; depending on what it finds in the system, in the third step this object will introduce one of the labels l_2 and l_3 . The computation in Π continues correctly simulating the computation in M . If the computation in M stops, then also the computation in Π stops (there is no rule in R_1 for l_h).

Consequently, $x \in L(\Pi)$ if and only if $N(x) \in A(M)$.

The “only if” implication can be proved in a straightforward way (or we can invoke for it the Turing-Church thesis). \square

Corollary 6.4 *A language is accepted by an ESAA with rules of types 1, 2, 3 if and only if it is a recursively enumerable language.*

6.7 Types 1, 2, 3'', 4

Finally, let us bring back rules of type 4, but, in order to balance the restriction they impose, let us modify the type 3 of rules by removing their promoter (we say that the modified rules are of type 3''). Thus, the rules of type 3'' are of the form $(u, out; v, in)$ (again, there is no restriction on the lengths of u and v , so $|v|$ can be greater than $|u|$). Hence, in this model,

which we call *RESAA*, type 1 is already covered by this type 3'' of rules, so the only rules are of the forms:

1. $(u, out; v, in)$, where $u, v \in (V - \Sigma)^*$ (there is no restriction on the lengths of u, v).
2. $(u, out; va, in)$, where $u, v \in (V - \Sigma)^*$ with $|u| \geq |v|$, and $a \in \Sigma$.
3. For every $a \in \Sigma$, there is at least one rule of the form $(a, out; v, in)$ in the membrane, where $v \in (V - \Sigma)^*$. This is the only form of rules for which a can appear on the left part of the rule.

As expected, we get again computational completeness:

Theorem 6.4 *A language L is accepted by an RESAA if and only if L is a recursively enumerable language.*

Proof. Let us consider a language $L \subseteq \{1, 2, \dots, k\}^*$. We proceed as in the proof of Theorem 6.3: starting from a (deterministic) counter machine $M = (m, H, l_0, l_h, R)$ which accepts $N(L)$, we construct the RESAA Π with $\Sigma = \{1, 2, \dots, k\} \cup \{\$\}$, $w_1 = cd$, and the set of rules much similar to the rules in the proof of Theorem 6.3 (so that we present them without giving details about their use).

Phase 1: reading a string $x\$\$ and producing $N(x)$.

Reading a symbol from Σ :

- Step 1: $(c, out; c'j, in), (b, out; b_0^k g^k, in),$
 Step 2: $(c', out; c'', in), (j, out; b_2^j, in), (b_0, out, b_1, in),$
 Step 3: $(c'', out; c, in), (b_2, out; b, in), (b_1 g, out; b, in).$

Reading the end marker $\$$:

- Step 1: $(cd, out; \$c''f, in), (b, out; a_1, in),$
 Step 2: $(c'', out; l_0, in), (\$, out; f, in).$

Besides these rules, for the first phase we also introduce the “control” rules used also in the proof of Theorem 6.3:

$$(fg, out; t, in), (da_1, out; t, in), (t, out; t, in).$$

After reading the input string (in such a way that the trap-object is not introduced), the system contains inside $N(x)$ copies of a_1 , as well as the objects l_0, f, f ; the two copies of f will play no role in the rest of the computation.

The second phase, of simulating a computation in M , is done exactly as in the proof of Theorem 6.3 (with the rules not having promoters), so that we omit the details. \square

7 Multi-Membrane ESAAAs

We can generalize the ESAAAs to have multiple membranes. In the skin membrane, the rules are the four types used for ESSA. The rules in the other membranes can be of the forms: (u, out) , (v, in) , and $(u, out; v, in)$, where $u, v \in (V - \Sigma)^*$, but there is no restriction on the relative lengths of u and v . Call this generalized model MESAA. Obviously, a non-deterministic (deterministic) MESSA can simulate an NLBA (DLBA). The converse is also easy to see.

Now define a 2-MESAA as in a 2-ESAA. Thus, a 2-MESSA has input alphabet $\{1, 2, \$\}$, and in the skin membrane the rules of type 3 satisfy $2|u| \geq |v|$. Then, from Theorem 4.1, a non-deterministic (deterministic) 2-MESSA is equivalent to an NLBA (DLBA) over a binary alphabet. Also, as in Theorem 5.1, fixing the number of membranes to $m \geq 1$, will induce an infinite hierarchy with respect to the number of symbols. Thus, we have:

Corollary 7.1 *Let m be any integer ≥ 1 . Then L is a binary context-sensitive language if and only if it can be accepted by a 2-MESAA with m membranes and multiple number of symbols. Moreover, holding the number of membranes at m , there is an infinite hierarchy in computational power (within the class of binary context-sensitive languages) with respect to the number of symbols.*

Let us now suppose we fix the alphabet of the 2-MESSA to a given size s ; then, the question is whether by allowing the 2-MESSA to have multiple membranes, it can simulate any LBA over a binary alphabet. We will prove that the answer is affirmative.

To this aim we define a *restricted counter machine* M as a counter machine with m counters which is restricted in its operation (see [8]) in the sense that a move of the machine consists of executing an instruction of the following form:

$$h : \text{ If } C \neq 0 \text{ then [decrement } C \text{ by 1, increment } D \text{ by 1, and go to } (r_1 \text{ or } \dots \text{ or } r_k)] \\ \text{ else go to } (s_1 \text{ or } \dots \text{ or } s_l)$$

where C, D represent counters (which need not be distinct) and $h, r_1, \dots, r_k, s_1, \dots, s_l$ represent instruction labels (which need not be distinct). The use of “or” makes the instruction non-deterministic. It becomes deterministic if $r_1 = \dots = r_k$ and $s_1 = \dots = s_l$. Note that by setting $C = D$, one can simulate an unconditional “go to” instruction (thus the machine can change states without altering any counter).

Clearly, in a restricted counter machine, the sum of the values of all the counters at any time during the computation is always equal to the initial value of the input counter.

The following theorem shows that two-way multihead finite automata over a unary alphabet are equivalent to restricted counter machines.

Theorem 7.1 *Let $Q \subseteq \mathbf{N}$. Then Q is accepted by a non-deterministic (deterministic) restricted counter machine if and only if $\{o^n \mid n \in Q\}$ is accepted by an N2MFA (D2MFA).*

Proof. Clearly, if M is a non-deterministic (deterministic) restricted counter machine with $k \geq 1$ counters (the first being the input counter), we can easily construct an N2MFA (D2MFA) M' with k heads to simulate M .

Conversely, suppose M is an N2MFA (D2MFA) with k heads. We construct a non-deterministic (deterministic) restricted counter machine M' with $2k + 1$ counters: $c_0, c_1, c_2, \dots, c_{2k-1}, c_{2k}$. Given a nonnegative integer n in counter c_0 and zero in the other counters, M' simulates the computation of M on input o^n . Each head h_i of M will be simulated by counters c_i, c_{i+1} . The simulation is as follows:

1. M decrements counter c_0 and increments the counters so that when counter c_0 becomes zero, counter c_i holds the number n/k for $i = 1, 3, \dots, 2k - 1$. Thus, M' “distributes” n (in c_0) equally to every odd-indexed counter. The remainder r_i after division (by k) for each counter c_i , $i = 1, 3, \dots, 2k - 1$, is recorded in the finite control of M' . (Note that initially, they have the same values.) At this point, counter c_i is empty for $i = 2, 4, \dots, 2k$. The finite control also associates a remainder r_i with counter c_i , $i = 2, 4, \dots, 2k$, and it is zero initially.
2. From this point on, counter c_0 is no longer used. M' now simulates head h_i using counters c_i and c_{i+1} . Moving h_i to the right (left) is simulated by M' by decrementing c_i and incrementing c_{i+1} (incrementing c_i and decrementing c_{i+1}). However, the decrementing/incrementing of the counters is done modulo k . Clearly, since M' keeps tracks of the remainders, the simulation can be carried out correctly. Note that head h_i is on the right end marker (left end marker) if and only if counter c_i (c_{i+1}) is zero and remainder r_i (r_{i+1}) is zero.

Clearly, M' is equivalent to M . □

We will need the following result.

Theorem 7.2 *Let M be an restricted counter machine with m counters (the first being the input counter). We can construct a non-deterministic 2-MESAA Π_1 with $m + 1$ membranes and alphabet o, a, c, d whose membrane structure has the form*

$$[0[1]_1[2]_2 \dots [m]_m]_0 \quad (0 \text{ is the label of the skin membrane}),$$

*such that when Π_1 is started with some fixed $w \in a^*c^*d^*$ in membrane 0, o^n in membrane 1 for some nonnegative integer n , and no symbols in the other membranes, Π_1 halts if and only if n is accepted by M . Hence, Π_1 accepts exactly the set of numbers accepted by M . (It is understood that the membranes have symport/antiport rules.) Moreover, because of the restriction on the way the restricted counter machine operates (i.e., a counter can be decremented if and only if another counter is incremented), Π_1 has the property that at any time during the computation, the number of copies of o in the system, including the environment, is always equal to n . Also, the numbers of objects a, c , and d in the system during the computation (including the environment) is at most some fixed number t , which only depends on the specification of M (and not on n).*

Proof. The construction of Π_1 is a modification of the construction in [16]. There, three symbols were used: o, a, c (actually, b was used instead of o). The construction in [16] was such that the number of copies of a during the computation is bounded by some fixed number

t . However, the number of copies of c during the computation becomes unbounded when the system goes into an infinite loop. By using an additional symbol d as a trap-symbol (instead of growing unboundedly the number of copies of c we make d oscillate across membranes), we can make the numbers of these symbols bounded during the computation. Clearly, because of the property of a restricted counter machine, Π_1 can be constructed satisfying the property above. \square

Now modify Π_1 of Theorem 7.2 by enclosing Π_1 in another membrane s ; the resulting system, call it Π_2 , has $m + 2$ membranes with structure

$$[s[0[1]_1[2]_2 \cdots [m]_m]0]_s \quad (s \text{ is the label of the skin membrane}).$$

Moreover, the initial multiset in the skin membrane s is $a^i c^t d^t$ (where t is as defined in Theorem 7.2). There are no rules in this new skin membrane. Thus, Π_2 does not communicate with the environment. Then, clearly, we have:

Corollary 7.2 Π_2 is equivalent to Π_1 .

We now prove the main result of this section.

Theorem 7.3 A language $L \subseteq \{1, 2\}^*$ is accepted by an NLBA if and only if it can be accepted by a non-deterministic 2-MESSA with an alphabet of size 14.

Proof. The “if” part is obvious. Now suppose $L \subseteq \{1, 2\}^*$ is accepted by an NLBA. We construct a 2-MESAA Π accepting L . From [19], $U = \{o^{N(x)} \mid x \in L\}$ is accepted by an M2MFA. Then from Theorem 7.1, U can be accepted by a restricted counter machine M . Suppose M has m counters (with the first counter being the input counter). The 2-MESAA Π has the same membrane-structure as Π_2 above. It has alphabet of symbols $V = \{1, 2, \$, \#, o, \alpha, \beta, \gamma, \delta, f, g, a, c, d\}$. We describe the initial multisets and rules in Π in two parts: Part 1 and Part 2.

Part 1 (Computes $N(x)$ into membrane 1):

Membrane s has initial initial multiset $\#g^2$ and the following rules:

1. $(\#, out; j, in)$, for every $j \in \{1, 2, \$\}$.
2. $(o, out; o^2, in)|_j$, for every $j \in \{1, 2\}$.
3. $(j, out; \#o^j, in)$, for every $j \in \{1, 2\}$.
4. $(\$, out; a^t c^t d^t \alpha \beta^2 \gamma^2 w, in)$, where t is as defined in Theorem 7.2 and w is as defined in Lemma 4.2,
5. $(\beta o, out; f, in)$
6. $(f, out; f, in)$,

Membrane 0 has initial multiset α and the following rules:

1. $(\alpha, out; \alpha o, in)$,
2. $(\alpha, out; \beta, in)$,
3. $(\beta, out; \gamma^2 in)$,
4. $(\gamma o, out; g, in)$,
5. $(g, out; g, in)$,
6. $(\delta, out; w, in)$.

Membrane 1 has initial multiset $\beta\delta$ and the following rules:

1. $(\beta, out; \beta o, in)$,
2. $(\delta, out; \gamma^2, in)$.

Membranes 2 to m have no initial multisets and no rules.

Clearly, for any binary string x , when Π is given the string $x\$$, Π halts with the following in the membranes:

Membrane s : $a^t c^t d^t \alpha^2 \beta^2 g^2$,
 Membrane 0: gw ,
 Membrane 1: $o^{N(x)} \gamma^2$,
 Each of membranes 2 to m : empty.

Part 2:

Now let Π_2 be the $(m + 2)$ -membrane MESAA accepting $\{o^n \mid n \text{ accepted by the restricted counter machine } M\}$ in Corollary 7.2. Note that Π_2 has alphabet $\{o, a, c, d\}$.

We now combine the rules of Π_2 to the rules described in Part 1. It is easy to see that the resulting Π accepts the binary language L . \square

As in Corollary 7.1, we have:

Corollary 7.3 *Let s be any integer ≥ 14 . Then a binary language is context-sensitive if and only if it can be accepted by a 2-MESAA with s symbols and multiple number of membranes. Moreover, holding the number of symbols at s , there is an infinite hierarchy in computational power with respect to the number of membranes.*

Thus Corollaries 7.1 and 7.3 say that in order for the restricted 2-MESSAs to accept all binary CSLs, at least one parameter (either the number of symbols or the number of membranes) must grow. As far as we know, these are the first results of their kind in the P systems area. They contrast the result from [1] that (unrestricted) symport/antiport systems with $s \geq 2$ symbols and $m \geq 1$ membranes accept (or generate) exactly the recursively enumerable sets of numbers even for $s + m = 6$.

8 Another Variant

The model of ESAA we have studied (including its variants and generalizations) had an online input, $x = a_1 \dots a_n \$$, and only these symbols (in the order given) can be read into the membrane. We defined the model with the input given this way because it allowed us to study both the deterministic and non-deterministic versions.

We now look at a new variant of the model of a non-deterministic ESAA. The system, which we call ESAA+, has alphabet V containing the input alphabet Σ , but we no longer assume that Σ contains an end marker symbol. It has the same four types of rules as in an ESAA, but it accepts a language in the following way. We start from the initial configuration, and (as usual) we use the rules in the non-deterministic maximally parallel manner. During the computation, symbols from Σ can be brought into the system, from the environment; these symbols are arranged in a sequence, corresponding to the moments when they enter the system (if several objects come at the same time, then any permutation of them is considered), and, if the computation halts, then the strings defined in this way are said to be accepted by the computation.

Clearly, an ESAA+ can be simulated by an NLBA. From the observation that an NLBA can be simulated by an ESAA+ without end markers, one can easily check that the proof of Lemma 4.2 still holds for ESAA+. Hence, we have:

Theorem 8.1 *A language L is accepted by a non-deterministic ESAA+ if and only if L is accepted by an NLBA.*

Finally, we remark that the characterizations for the non-deterministic versions of some of the variants of ESAA in Section 6 remain valid when the input and acceptance are defined as in ESAA+.

9 Conclusion

We have addressed here a problem which proved to be difficult in natural computing: characterizing families of languages (other than those of regular and recursively enumerable languages) in terms of computing devices inspired from biology – in our case, by means of symport/antiport P systems. Besides a characterization of context-sensitive languages and of other families of languages, such as of languages accepted by one-way $\log n$ space-bounded Turing machines and of recursively enumerable languages, we have also found a rather interesting double infinite hierarchy for binary context-sensitive languages, in terms of the number of membranes and of the objects used.

As an intriguing open problem, it remains to find similar (syntactic, that is, based on restrictions on the form of rules of the P systems used) for other families of languages, in particular, for context-free languages.

A natural research topic is also to find characterizations of known families of languages by using other types of P systems. For instance, it is known that symport/antiport systems are universal also when using only symport rules. Can results as above be proved also for such systems?

References

- [1] A. Alhazov, R. Freund, M. Oswald: Symbol/membrane complexity of P systems with symport/antiport rules. In [5], 123–146.
- [2] A. Alhazov, R. Freund, Y. Rogozhin: Computational power of symport/antiport: history, advances, and open problems. In [5], 44–78.
- [3] G. Bel-Enguix, M. Cavaliere, R. Ceterchi, C. Martin-Vide: An application of dynamic P systems: Generating context-free languages. In *Membrane Computing. International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer, Berlin, 2003, 90–106.
- [4] E. Csuhaj-Varju, O.H. Ibarra, G. Vaszil: On the computational complexity of P automata. In *Proceedings of DNA10 Conference, Milano, 2004* (C. Ferretti, G. Mauri, C. Zandron, eds.), LNCS 3384, Springer, Berlin, 2005, 76–89.
- [5] R. Freund, G. Lojka, M. Oswald, Gh. Păun, eds.: *Pre-Proceedings of the 6th International Workshop on Membrane Computing, WMC6*. Vienna Technological Univ., 2005.
- [6] R. Freund, Gh. Păun: On deterministic P systems. Available at [22], 2003.
- [7] O.H. Ibarra: A hierarchy theorem for polynomial space recognition. *SIAM Journal on Computing*, 3: 184–187, 1974.
- [8] O.H. Ibarra: The number of membranes matters. In *Membrane Computing. International Workshop, WMC2003, Tarragona*, LNCS 2933, Springer, Berlin, 2004, 218–231. Journal version to appear in *Theoretical Computer Science*, 2005.
- [9] O.H. Ibarra, S. Woodworth: On bounded symport/antiport P systems. *Pre-Proceedings of 11th International Meeting on DNA Computing*, UWO, London, Ontario, 2005, 37–48.
- [10] O.H. Ibarra: On determinism versus non-determinism in P systems. *Theoretical Computer Science*, to appear, 2005.
- [11] N. Immerman: Non-deterministic space is closed under complementation. *SIAM J. Computing*, 17(5): 935–938, 1988.
- [12] M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
- [13] A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20(3): 295–306, 2002.
- [14] Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61(1): 108–143, 2000.
- [15] Gh. Păun: *Membrane Computing: An Introduction*. Springer-Verlag, Berlin, 2002.

- [16] Gh. Păun, J. Pazos, M.J. Pérez-Jiménez, A. Rodríguez-Paton: Symport/antiport P systems with three objects are universal. *Fundamenta Informaticae*, 64(1-4): 353–367, 2005.
- [17] Gh. Păun, G. Rozenberg: A guide to membrane computing. *Theoretical Computer Science*, 287(1): 73–100, 2002.
- [18] Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New Computing Paradigms*. Springer-Verlag, Berlin, 1998.
- [19] W. Savitch: A note on multihead automata and context-sensitive languages. *Acta Informatica*, 2: 249–252, 1973.
- [20] P. Sosik: P systems versus register machines: two universality proofs. In *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Arges, Romania*, pages 371–382, 2002.
- [21] R. Szelepcsényi: The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3): 279–284, 1988.
- [22] The P Systems Web Page: <http://psystems.disco.unimib.it>.