Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 320 (2004) 89–109
Theoretical Computer Science
www.elsevier.com/locate/tcs

On the computational complexity of membrane systems☆

Oscar H. Ibarra*
Department of Computer Science, University of California, Santa Barbara, CA 93106-5110, USA

Abstract

We show how techniques in machine-based complexity can be used to analyze the complexity of membrane computing systems. We focus on catalytic systems, communicating P systems, and systems with only symport/antiport rules, but our techniques are applicable to other P systems that are universal. We define space and time complexity measures and show hierarchies of complexity classes similar to well-known results concerning Turing machines and counter machines. We also show that the deterministic communicating P system simulating a deterministic counter machine in (Sosik (2002)) (Pre-Proc. of Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Arges, Romania, 2002, pp. 371–382), (Sosik and Matysek (2002)) (Unconventional Models of Computation 2002, Lecture Notes in Computer Science, vol. 2509, Springer, Berlin, 2002, pp. 264–275.) can be constructed to have a fixed number of membranes, answering positively an open question in Sosik (2002), Sosik and Matysek (2002). We prove that reachability of extended configurations for symport/antiport systems (as well as for catalytic systems and communicating P systems) can be decided in nondeterministic $\log n$ space and, hence, in deterministic $\log^2 n$ space or in polynomial time, improving the main result in Paun et al. (2002) (On the reachability problem for P systems with symport/antiport, 2002, submitted for publication.). We propose two equivalent systems that define languages (instead of multisets of objects): the first is a catalytic system language generator and the other is a communicating P system acceptor (or a symport/antiport system acceptor). These devices are universal and therefore can also be analyzed with respect to space and time complexity. Finally, we give a characterization of semilinear languages in terms of a restricted form of catalytic system language generator.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Membrane computing; Catalytic system; Communicating P system; Symport/antiport system; Space bounded; Time bounded; Reachability; Acceptor; Generator; Semilinear

☆ This research was supported in part by NSF Grants IIS-0101134 and CCR02-08595.
* Tel.: +1-805-893-4171; fax: +1-805-893-8553.
E-mail address: ibarra@cs.ucsb.edu (O.H. Ibarra).

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.03.045

1. Introduction

In recent years, there has been a flurry of research activities in the area of membrane computing [21,22,24], which identifies an unconventional computing model (namely a P system) from natural phenomena of cell evolutions and chemical reactions [1]. Due to the built-in nature of maximal parallelism inherent in the model, P systems have a great potential for implementing massively concurrent systems in an efficient way, once future bio-technology (or silicon-technology) gives way to practical bio-realization (or a chip-realization).

A large number of papers have been written concerning the "universality" of the computing power of various models of P systems. These model are all equivalent in the sense that all of them are able to simulate a Turing machine. A fundamental question concerning P systems or any new model of computation is how to quantify its computing power in terms of some complexity measures, like time and/or space (memory). Very little work has been done on this subject. In this paper, we present some results along these lines.

A P system G consists of a finite number of membranes, each of which contains a multiset of objects (symbols). The membranes are organized as a Venn diagram or a tree structure where membranes may contain other membranes. The dynamics of G is governed by a set of rules associated with each membrane. Each rule specifies how objects evolve and move into neighboring membranes. The rule set can also be associated with priority: a lower priority rule does not apply if one with a higher priority is applicable. A precise definition of G can be found in [21,22,24]. Various models of P systems have been shown to be equivalent to Turing machines in computing power. For example, recent results in [30,31,6] show that P systems with one membrane (i.e., 1-region P systems) using only catalytic rules without priority are already able to simulate a two counter machines and hence universal [18]. In this paper, we look at three P system models:

- Catalytic systems [21,22,24,30,31,6]. But here, we only allow rules of the form $Ca \rightarrow Cv$, where C is a catalyst, a is a noncatalyst, and v is a (possibly null) string of noncatalysts.
- Communicating P systems [30,32,8].
- P systems with only symport/antiport rules [19,20,16,17,8,23,9].

We show how techniques in machine-based complexity can be used to analyze the computational complexity of these systems. Our techniques are applicable to other P systems that are universal. We define space and time complexity measures and show hierarchies of complexity classes similar to well known results concerning Turing machines and counter machines. In the process, we are able to give a positive answer to an open question in [30,32]: That the deterministic communicating P system simulating a deterministic counter machine in [30,32] can be constructed to have a fixed number of membranes. We also prove that reachability of extended configurations for symport/antiport systems (as well as for catalytic systems and communicating P systems) can be decided in nondeterministic log n space and, hence, also in deterministic $\log^2 n$ space or in polynomial time, improving the main result in [23]. We propose two equivalent systems that define languages (instead of multisets of objects): the first

is a catalytic system language generator and the other is a communicating P system acceptor (or a symport/antiport system acceptor). The latter is similar to the study of P automata in [3,7]. These devices are universal and therefore can also be analyzed with respect to space and time complexity. Finally, we give a characterization of semilinear languages in terms of a restricted form of catalytic system language generator.

We should mention that computational complexity with respect to the possibilities of trading space for time in P systems has been investigated in various places, e.g., for solving NP-complete problems in polynomial time by using membrane division that generates exponential space, and by using "active membranes" (see, e.g., [22,33,27,26]).

2. Catalytic systems

In this section, we consider catalytic systems (CSs) with only one region. Such a system G operates on two types of symbols: catalytic symbols called *catalysts* (denoted by capital letters C, D, etc.) and noncatalytic symbols called *noncatalysts* (denoted by lower case letters a, b, c, d, etc.). An evolution rule in G is of the form $Ca \to Cv$, where C is a catalyst, a is a noncatalyst, and v is a (possibly null) string (an obvious representation of a multiset) of noncatalysts. There are no rules of the form $a \to v$. A CS G is specified by a finite set of rules together with an initial multiset (configuration) w_0, which is a string of catalysts and noncatalysts. As with the standard semantics of P systems [21,22,24], each evolution step of G is a result of applying all the rules in G in a maximally parallel manner. More precisely, starting from the initial configuration, w_0, the system goes through a sequence of configurations, where each configuration is derived from the directly preceding configuration in one step by the application of a subset of rules, which are chosen nondeterministically. Note that a rule $Ca \to Cv$ is applicable if there is a C and an a in the preceding configuration. The result of applying this rule is the replacement of a by v. If there is another occurrence of C and another occurrence of a, then the same rule or another rule with Ca on the left hand side can be applied. We require that the chosen subset of rules to apply must be maximally parallel in the sense that no other applicable rule can be added to the subset. Configuration w is reachable if it appears in some execution sequence; w is halting if no rule is applicable on w.

It is important to note that our definition of catalytic system is different from what is usually called catalytic system in the literature. Here, we do not allow rules without catalysts, i.e., rules of the form $a \to v$. Thus our systems use only purely catalytic rules. Also, in our definition, there is no target indication associated with the objects, i.e., we do not allow objects to exit the membrane into the environment.

We denote by $R(G)$ the Parikh map of all reachable configurations with respect to noncatalysts only. (Thus, if $a_1, \ldots, a_k$ are the noncatalysts, then $R(G) = \{(\#_{a_1}(x), \ldots, \#_{a_k}(x)) \mid x$ is a reachable configuration in $G\}$, where $\#_{a_i}(x)$ is the number of occurrences of noncatalyst a_i in x.) For convenience, when we talk about configurations, we sometimes do not include the catalysts. $R(G)$ is called the reachability set of G. $R_h(G)$ will denote the set of all halting reachable configurations. Let $\mathbf{N}$ be the set of all nonnegative integers and k be a positive integer. It is known that for any set

$Q \subseteq \mathbf{N}^k$ that can be accepted by a Turing machine, we can construct a CS G with only purely catalytic rules such that $R_h(G) = Q$ [30,31,6]. In fact, Freund et al. [6] shows that three catalysts (even when each catalyst appears exactly once in the initial configuration) are already sufficient for universality. If, however, the initial configuration w_0 of G has only *one* catalyst, then $R_h(G)$ and $R_h(S)$ are effectively computable semilinear sets [14]. The case when there are only two catalysts remain an interesting open question.

For defining complexity measures, it is convenient to modify the definition of a CS G to make it an acceptor of tuples of natural numbers. Let $\mathscr{C}$ be the set of all catalysts, V the set of all noncatalysts, and $\Sigma = \{a_1, \ldots, a_k\} \subseteq V$ be a special set of "input" noncatalysts. There is a fixed string $w_0 \in (\mathscr{C} \cup (V - \Sigma))^+$ associated with G. We say that G accepts a k-tuple $(i_1, \ldots, i_k)$ of nonnegative integers if G, when started in initial configuration $w_0 a_1^{i_1}, \ldots, a_k^{i_k}$, has a computation that halts. If every computation is nonhalting, then the tuple is not accepted. We call G a CS acceptor, or simply CSA.

Let $S(n) \geqslant n$ and $T(n) \geqslant n$ be monotonic functions. We say that G is $S(n)$ space bounded (respectively, $T(n)$ time bounded) if for every tuple $(i_1, \ldots, i_k)$ that is accepted, there is a halting computation in which the maximum number of noncatalysts that G uses during the computation excluding the objects in $w_0 a_1^{i_1}, \ldots, a_k^{i_k}$ (respectively, the total number of steps in the entire computation) is at most $S(n)$ (respectively, $T(n)$), where $n = i_1 + \cdots + i_k$.

We are interested in studying the complexity of sets of tuples defined by CSAs. We denote by $CATSPACE(S(n))$ (respectively, $CATTIME(T(n))$) the class of tuples accepted by $S(n)$ space bounded (respectively, $T(n)$ time bounded) CSAs. It turns out that these classes are related to space bounded (time bounded) computations of Turing machines (TMs) and (multi)counter machines.

When we are dealing with languages that are subsets of Σ^*, we assume, without loss of generality, that Σ is an alphabet containing only two symbols, which we identify with the digits 1, 2. We interpret each string $w \in \Sigma^+$ as a number $Num(w)$ in 2-adic notation. So if $w = d_{n-1}, \ldots, d_0$, each $d_i \in \Sigma$, then $Num(w) = d_{n-1}2^{n-1} + \cdots + d_0 2^0$. Note that if $|w| = n$, then $2^n - 1 \leqslant Num(w) \leqslant 2^{n+1} - 2$. If $L \subseteq \Sigma^+$ is a language, let $Tally(L) = \{1^{Num(w)} \mid w \in L\}$.

Let M be a Turing machine (TM) with a one-way read-only input tape (with a right endmarker) and several two-way read/write worktapes. M is $S(n)$ space bounded if, on every input w of length n that is accepted, there is an accepting computation which uses no more than $S(n)$ cells on any of its worktape. It is well known that any $S(n)$ space bounded TM with multiple worktapes can be transformed to an equivalent $S(n)$ space bounded TM with only one worktape. A two-way TM is one whose input tape is two-way (with both left and right endmarkers). We will assume throughout, unless otherwise noted, that $S(n) \geqslant n$. It is then obvious that an $S(n)$ space bounded one-way TM is equivalent to an $S(n)$ space bounded two-way TM.

Convention. In the paper, when we say that a device (e.g., TM, CSA, etc.) uses $S(n)$ space (or is $S(n)$ space bounded) on an input x of length n, we mean that this is the bound for some accepting computation on x. By our definition of $S(n)$ space bounded, the bound need not hold if x is not accepted. All logarithms in this paper have base 2.

We will need the following lemma. A similar result was shown in [29,2], but is not quite applicable for our purposes. Note that the parameter n in a space bound that concerns a language L refers to the length of strings in L, but when it concerns $Tally(L)$, it refers to the length of strings in $Tally(L)$.

Lemma 1. *Let* $S(n) \geqslant n$.

1. *If L is accepted by a deterministic (respectively, nondeterministic) $S(n)$ space bounded one-way TM, then $Tally(L)$ is accepted by a deterministic (respectively, nondeterministic) $S(\log n + 1)$ space bounded one-way TM.*
2. *If $Tally(L)$ is accepted by a deterministic (respectively, nondeterministic) $S(\log n - 1)$ space bounded one-way TM, then L is accepted by a deterministic (respectively, nondeterministic) $S(n)$ space bounded one-way TM.*

Proof. Given an $S(n)$ space bounded TM M, we construct a TM M' which, when given a unary input y, first converts y to a 2-adic string w over $\{1,2\}$ on one of the worktapes. Note that M′ need only read y from left to right (i.e., one-way). If $2^n - 1 \leqslant y < 2^{n+1} - 2$, then w will have length n. M' has enough space since $S(n) \geqslant n$. Then M' simulates M on w using another worktape that uses $S(n)$ space. Hence, M' on an input y with $2^n - 1 \leqslant y \leqslant 2^{n+1} - 2$, uses $S(n)$ space. It follows that M' accepts $Tally(L)$ and is $S(\log(n+1)) \leqslant S(\log n + 1)$ space bounded.

Conversely, suppose M is an $S(\log n - 1)$ space bounded TM accepting $Tally(L)$. We construct a TM M' to accept L as follows. M', when given w of length n, first writes w on the worktape, using space n. Then M' simulates the computation of M on $Num(w)$ by using the worktape that contains w. Since $S(n) \geqslant n$, M' has enough space to write and work on w. In the simulation, M' uses another tape for simulating the worktape of M. This worktape is $S(\log\ (Num(w)) - 1) \leqslant S(\log\ (2^{n+1} - 2) - 1) \leqslant S(\log\ (2^{n+1}) - 1) = S(n)$ space bounded. Thus, M' accepts L and, on input w of length n, uses at most $S(n)$ space. □

Let $NSPACE(S(n))$ (respectively, $DSPACE(S(n))$) denote the class of languages accepted by nondeterministic (respectively, deterministic) $S(n)$ space bounded one-way TMs.

Corollary 1. *Let $S_1(n) \geqslant n$ and $S_2(n) \geqslant n$. If L is in $NSPACE(S_2(n)) - NSPACE(S_1(n))$, then $Tally(L)$ is in $NSPACE(S_2(\log n + 1)) - NSPACE(S_1(\log n - 1))$. This also holds for the deterministic classes.*

In the remainder of the paper, to simplify the notation and presentation of the results, we will assume that $S(n)$ satisfies the following property: For every $S(n)$ space bounded TM M, we can construct an $S(n-1)$ space bounded TM M' which accepts exactly the strings accepted by M whose lengths are greater than 1. This is true for many space bounds. Then the corollary above reduces to:

Corollary 2. *Let $S_1(n) \geqslant n$ and $S_2(n) \geqslant n$. If L is in $NSPACE(S_2(n)) - NSPACE(S_1(n))$, then $Tally(L)$ is in $NSPACE(S_2(\log n)) - NSPACE(S_1(\log n))$. This also holds for the deterministic classes.*

A nondeterministic (respectively, deterministic) counter machine, or CM, is a machine with a one-way read-only input (with a right endmarker) and a finite number of counters that are initially zero. During the computation, the machine can test each counter against zero, increment/decrement it by 1, or leave it unchanged. The counters can only assume nonnegative values. The machine accepts the input if there is a computation that halts, with all counters zero. The machine is $S(n)$ space bounded if for any input of length n that is accepted, there is a computation in which the sum of the counter values at any time during the computation is at most $S(n)$.

We will need the following two results from [5]:

Lemma 2. *Let $S(n) \geqslant n$. The following statements hold for both deterministic and nondeterministic CMs*:

1. (*Linear space compression without increasing the number of counters.*) *Given any $S(n)$ space bounded CM with r counters, we can construct a time-equivalent* (*i.e., no loss of time*) *$cS(n)$ space bounded CM with r counters for any constant $c > 0$.*
2. (*At the cost of adding more counters.*) *For every k, given any $S^k(n)$ space bounded CM, we can construct a time-equivalent $S(n)$ space bounded CM. Thus in particular, a polynomial space bounded CM can be converted to one which is linear space bounded.*

Lemma 3. *Let $S(n) \geqslant n$. A language L is accepted by a nondeterministic* (*respectively, deterministic*) *$S(n)$ space bounded CM if and only if it is accepted by a nondeterministic* (*respectively, deterministic*) $\log(S(n))$ *space bounded TM.*

NCMSPACE$(S(n))$ denotes the class of languages accepted by $S(n)$ space bounded nondeterministic one-way counter machines. *DCMSPACE*$(S(n))$ denotes the deterministic class.

From Corollary 2 and Lemma 3, we have:

Theorem 1. *Let $S_1(n) \geqslant n$ and $S_2(n) \geqslant n$. If L is in NSPACE$(S_2(n)) - NSPACE(S_1(n))$, then Tally$(L)$ is in NCMSPACE$(2^{S_2(\log n)}) - NCMSPACE(2^{S_1(\log n)})$. This also holds for the deterministic classes.*

Suppose M is a CM whose inputs are bounded, i.e., of the form $1^{i_1}21^{i_2}2,\ldots,21^{i_k}$ (with a right endmarker) for some fixed k, where $i_1,\ldots,i_k$ are nonnegative integers. Clearly, the one-way bounded input can be simulated by k counters which initially contain the tuple $(i_1,\ldots,i_k)$. We will use this second model (but still call it CM) in the remainder of this section.

The connection between CSAs and (nondeterministic) CMs is given by the following theorem.

Theorem 2. *A set of tuples $Q \subseteq \mathbf{N}^k$ is accepted by an $S(n)$ space bounded CSA if and only if it is accepted by an $S(n)$ space bounded one-way CM.*

Proof. Clearly, an $S(n)$ space bounded CSA can be simulated by a $cS(n)$ space bounded CM for some c, where each noncatalyst is associated with a counter. We then use Lemma 2 part 1 to bring the constant c down to 1. The converse follows from the construction in [30,31]. That is, given an $S(n)$ space bounded CM M, we first use Lemma 2 part 1 to construct an equivalent CM M' that is $dS(n)$ time bounded for a constant d yet to be determined. Then from the construction in [30,31], we can construct from M' an equivalent CSA G whose space bound is linear in the space bound of M'. By using the appropriate d, the CSA G can be made $S(n)$ space bounded. □

From the above theorem and Lemma 2, we have:

Corollary 3. *Let $S(n) \geqslant n$.*

1. *Given any $S(n)$ space bounded CSA, we can construct an equivalent $cS(n)$ space bounded CSA for any constant $c > 0$.*
2. *For every k, given any $S^k(n)$ space bounded CSA, we can construct an equivalent $S(n)$ space bounded CSA. Thus in particular, a polynomial space bounded CSA can be converted to one which is linear space bounded.*

From Theorems 1 and 2, we have:

Corollary 4. *Suppose L is in $NSPACE(S_2(n)) - NSPACE(S_1(n))$. Then $Tally(L)$ is in $CATSPACE(2^{S_2(\log n)}) - CATSPACE(2^{S_1(\log n)})$.*

The above corollary shows that there is an infinite hierarchy of space bounded CSA computations. It can be used to show separation between complexity classes. For example, it is known [12] that for any positive integer k, $NSPACE(n^{k+1})$ properly contains $NSPACE(n^k)$. Hence, from Corollary 4, there is a subset Q of $\mathbf{N}$ that is accepted by a CSA in space $2^{\log^{k+1} n}$ that is not accepted by any CSA in space $2^{\log^k n}$. Thus $CATSPACE(2^{\log^{k+1} n})$ properly contains $CATSPACE(2^{\log^k n})$.

The following gives the trade-off between space bounded and time bounded computations in CSA's.

Theorem 3. *A set of tuples Q is accepted by an $S(n)$ space bounded CSA if and only if it is accepted by an $S^k(n)$ time bounded CSA for some k. In particular, polynomial space bounded CSA's are equivalent to polynomial time bounded CSAs.*

Proof. Let G be an $S(n)$ space bounded CSA. Let m be the number of noncatalysts in G. Then the number of possible configurations that G can be in is at most $(S(n)+1)^m$. (Note that the catalysts do not change during the computation and therefore are not counted in this number.) Hence, G is $(S(n)^k$ time-bounded for some k. The converse is obvious, using Corollary 3 part 2. □

3. Communicating P systems

Here we look at communicating P systems [30,32]. A communicating P system G consists of membranes organized in a tree-like structure. Each membrane has a (possibly empty) set of rules associated with it. The evolution rules are of the following forms, where V is the set of all objects that can appear in the system:

1. $a \to a_\tau$,
2. $ab \to a_{\tau_1} b_{\tau_2}$,
3. $ab \to a_{\tau_1} b_{\tau_2} c_{\text{come}}$,

where $a, b, c \in V$ and $\tau, \tau_1, \tau_2 \in (\{here, out\} \cup \{in_j \mid 1 \leqslant j \leqslant n\})$. The meaning of the subscript *out* (respectively, in_j) on an object is to transport the object from the membrane containing it into the membrane immediately outside it (respectively, into membrane labeled j, provided j is adjacent to the object). The subscript *here* on an object means that the object remains in the same membrane after the transition. A rule of the form (3) can occur only within the region enclosed by the skin membrane. When such a rule is applied, then c is imported through the skin membrane from the outer space (environment) and will become an element of this region. In one step, all rules are applied in a maximally parallel manner. There is an abundant (infinite) supply of some objects outside the skin membrane, i.e., the environment. Some objects are not available initially in the environment but may enter the environment during the computation. Thus, the number of each such object in the whole system (including the environment) remains the same. The system starts from some fixed initial configuration with objects distributed among the membranes. Some membranes can be designated input (respectively, output) membranes, to contain the initial input (respectively, final output) objects. We refer the reader to [30,32] for the details.

We know that in a communicating P system, a step of the computation is an application of a set of nondeterministically chosen rules in parallel to the current configuration to obtain the next configuration (thus, all rules must be applicable at the same time). The set of rules is maximal in the sense that no additional rule can be added to the set which still makes the resulting set of rules applicable. To emphasize this parallelism, we will also refer to this system as a *parallel communicating P system*.

Now define a *sequential communicating P system* as one in which a computation step consists of an application of a *single* rule in some membrane to the current configuration. The membrane and rule are chosen nondeterministically. If no rule in any membrane is applicable to the current configuration, the system halts. A *deterministic* communicating P system is a sequential communicating P system which is deterministic, i.e., there is a unique possible sequence of configurations from the initial configuration.

A communicating P system (parallel or sequential) computes a function $f : \mathbf{N} \to \mathbf{N}$ if, when given a^n in the input membrane (representing the nonnegative integer n) together with some fixed initial distribution of objects (different from a) in some membranes, the system halts if and only if $f(n)$ is defined, and if it halts, then $a^{f(n)}$ is in the output membrane. See [30,32] for details.

Lemma 4. *A deterministic communicating P system can be simulated by a deterministic CM* (*i.e.*, *counter machine*). *Thus any function* $f : \mathbf{N} \to \mathbf{N}$ *that can be computed by a deterministic communication P system can be computed by a deterministic CM.*

Proof. The deterministic counter machine has a counter $A_{(a,m)}$ for each object a and membrane m in the communicating P system. The machine also uses some auxiliary counters. The counter machine simulates each computation step of the P system faithfully, where each step consists of an application of the rules in a maximally parallel manner (the auxiliary counters are used in updating the contents of the counters $A_{(a,m)}$'s in the simulation of a step). Since the P system is deterministic, the counter machine is guaranteed to be deterministic. □

The converse of the above lemma is also true. It was shown in [30,32] that a deterministic CM can be simulated by a deterministic communicating P system. The number of membranes used in the simulation grows with the size (number of instructions) in the program of the CM being simulated. It was left open in [30,32] whether the number of membranes can be bounded by some fixed number. Two membranes were later shown to be sufficient for the simulation in [8]. However, the simulating system in [8] is a parallel communicating P system (it uses "trap" objects that cause the system to go into an infinite loop if the correct simulation is not taken). Below we provide an answer to the question raised in [30,32].

Theorem 4. *There is a fixed positive integer* k *such that an arbitrary recursively enumerable function* (*i.e.*, *Turing machine computable function*) $f : \mathbf{N} \to \mathbf{N}$ *can be computed by a deterministic communicating P system with* k *membranes.*

Proof. Clearly, any recursively enumerable function f can be computed by a deterministic TM, M, with a one-way unary read-only input tape (with right endmarker), one read-write worktape, and one unary output tape. M, when given n on its input tape, computes (using its worktape) and outputs $f(n)$ on its output tape and halts, if $f(n)$ is defined; otherwise, M does not halt.

It is well known that we can construct a universal deterministic TM U with two read-only input tapes, one read-write tape, and one unary output tape. U, when given n and the 2-adic description x_M of a deterministic TM M on its two input tapes, will simulate the computation of M on n and output $f(n)$ on its output tape if M halts with output $f(n)$. If M does not halt, U does not halt. Note that the unary input containing n is one-way.

First, we convert U to an equivalent universal deterministic TM U', where the description of x_M is given as $Tally(x_M)$ (i.e., in unary), instead of 2-adic. The idea is for U' to read $Tally(x_M)$ on the input and convert it into 2-adic representation x_M on the first track of the worktape. Then U' uses x_M to simulate U on n using the second track of the worktape. So now U' has two read-only unary input tapes, a worktape, and a unary output tape.

Next, we convert the worktape of U' to a finite number of counters. Three counters are sufficient to simulate the TM worktape, as shown in [4,5]. The new machine U'' will now have two-unary inputs, one unary output, and three counters.

Finally, we convert U'' to a deterministic counter machine M which will use two counters to simulate the two unary input tapes (note that the unary input tapes are one-way), one counter for the output, and three working counters, all of which are initially zero.

Thus, M has 6 counters. Initially, the first counter contains n, the second counter contains $Tally(x_M)$, and the other 4 counters are zero. If and when M halts, the output counter will contain $f(n)$. We may assume that M zero out all but the output counter before it halts.

Let s be the number of states of M. Then by the construction in [30,32], we can convert a CM M with m counters to a deterministic communicating P system G with $k = 3s + m + 1$ membranes. Hence, an arbitrary recursively enumerable function $f : N \to N$ can be computed by a $(3s + 7)$-membrane deterministic communicating P system G_f obtained from G. Note that G_f has the same program for any f. The only difference is the starting configuration: the input n is initially stored in the first membrane and the unary description $Tally(x_M)$ of the TM M computing f is initially stored in the second membrane. Thus, G is a "universal deterministic communicating P system". □

Clearly, the theorem above generalizes to computing recursively enumerable functions $f : \mathbf{N}^k \to \mathbf{N}^l$. Also one can define a communicating P system acceptor accepting a set of tuples $Q \subseteq \mathbf{N}^k$ in the obvious way—the input tuple $(i_1, \ldots, i_k)$ is represented (along with other objects) in the initial configuration of the system. The tuple is accepted if the system halts.

Remark 1. The construction in the proof of Theorem 4 can be used to show that for a universal P system of any given type, there exists a fixed universal P system of the same type. Thus, we can construct, e.g., a universal CSA, a universal symport/antiport system, etc.

We can derive results similar to those in the previous sections concerning space bounded communicating P systems. A deterministic (respectively, nondeterministic) $S(n)$ space bounded communicating P system acceptor is defined in the obvious way—$S(n)$ is the maximum number of objects imported from the environment during the computation on an input $(i_1, \ldots, i_k)$ of size $n = i_1 + \cdots + i_k$. In fact, because of the "deterministic simulation" of [30,32], we can prove a tighter hierarchy for deterministic space bounded communicating P systems because the space hierarchy theorem for deterministic TM's is tighter. For example, we can prove the following theorem by using the "deterministic version" of Theorem 1 and the fact (follows from Lemma 4 and the construction in [30,32]) that a set Q of tuples is accepted by an $S(n)$ space bounded deterministic communicating P system acceptor if and only if it is accepted by an $S(n)$ space bounded deterministic CM.

Theorem 5. *Let $S_1(n) \geqslant n$ and $S_2(n) \geqslant n$ be tape constructable functions.* [*$S(n)$ is tape constructable if there is deterministic TM which on every input of length n (accepted or not) uses exactly $S(n)$ space and halts.*] *If $\lim_{n\to\infty} S_1(n)/S_2(n) = 0$, then there is a set of tuples accepted by a $2^{S_2(\log n)}$ space bounded deterministic communicating P system acceptor that cannot be accepted by a $2^{S_1(\log n)}$ space bounded deterministic communicating P system acceptor.*

4. Systems with symport/antiport rules

The results of the previous sections also apply to space bounded systems with symport/antiport rules. Thus, the only rules allowed are of the form $(u, \text{out}; v, \text{in})$, where u and v are in V^* with $uv \neq \lambda$. See [19,20,16,17,8,23,9].

The environment has an abundant supply of some objects and finite (possibly empty) supply for some objects. The amount of objects from the environment used during the computation (not including the objects in the initial configuration of the system) is the measure of the space bound.

Theorem 6. *A set of tuples $Q \subseteq \mathbf{N}^k$ is accepted by an $S(n)$ space bounded symport/antiport system if and only if it is accepted by an $S(n)$ space bounded one-way CM.*

Proof. The proof of the "only if part" is similar to Theorem 2. Given an $S(n)$ space bounded symport/antiport system G, we construct an $S(n)$ space bounded CM M by assigning a counter $A_{(a,m)}$ for each object a and each membrane in the system. We also assign a counter B_b for each object b present in the environment that does not have infinite supply. We do not assign counters to objects with infinite supply. M simulates the computation of G using the counters to keep track of the multiplicities of distinct objects in distinct membranes. M also uses some auxiliary counters in the simulation. Clearly, M is $O(S(n))$ space bounded, but this can be reduced to $S(n)$ by the linear space compression lemma.

The converse follows from the construction in [30,31] and Theorem 2, since the rules in [30,31] (which can be converted to be all purely catalytic) can directly be written as symport/antiport rules, i.e., $Ca \to Cv$ becomes the rule $(Ca \text{ out}; Cv \text{ in})$. □

5. Reachability problem

The reachability problem for any P system G is defined as follows:

Given: A P system G with initial configuration w_0, and a configuration x.

Question: Is there a computation of G from w_0 which reaches x?

Clearly, if G is universal, the reachability problem is undecidable, in general. However, for restricted cases, e.g., for space bounded G, the problem is decidable. For example if G is $S(n)$ space bounded, and $S(n)$ is computable, then, for a given n, there are at most $c^{S(n)}$ possible reachable configurations of length n from w_0, for some

effectively computable constant c. Thus, given x of length n, we can examine all reachable configurations and check if x is one of these.

In [23], it was shown that the reachability problem for symport/antiport systems is decidable if the configuration x (for which we are trying to determine reachability) also *includes* the objects which are sent out of system into the environment. Thus, the configuration x will always have length proportional to the maximum number of objects brought in from the environment during the computation. In this sense, x is called an *extended configuration*. It was shown in [23] that the set of all reachable extended configurations is a context-sensitive language (and, in fact, can be generated by a matrix grammar with appearance checking with λ-free rules). The algorithm in [23] runs in time exponential in the length of the extended configuration x, where x is represented as $z_1 d z_2 d, \ldots, d z_m d z_{m+1}$, where d is a delimiter, m is the number of membranes, z_i $(i = 1, \ldots, m)$ represents the unary encodings of the multiplicities of objects (separated by markers) in membrane i, and z_{m+1} represents the unary encodings of the multiplicities of objects sent out to the environment. Here, we observe that, in fact, the set of reachable extended configurations can be accepted by a log n space bounded two-way nondeterministic TM and, hence, from Savitch's result [28], can also be accepted by a $\log^2 n$ space-bounded two-way deterministic TM. Since a log n space bounded two-way nondeterministic TM can be simulated by a deterministic TM in polynomial time, the set of extended configurations is also in PTIME (= polynomial time):

Theorem 7. *Let G be a symport/antiport system. The set of reachable extended configurations of G can be accepted by a* log *n space-bounded nondeterministic two-way TM. The same result holds if we are only interested in halting reachable extended configurations.*

Proof. The idea is the following: Given a symport/antiport system G, we construct a log n space-bounded nondeterministic TM (with a two-way read-only input with endmarkers) M which, when given the extended configuration x, first converts the input into binary representation using a finite number k of worktapes by writing the multiplicities of the objects in binary. Clearly, if the length of x is n, the space needed for the worktapes is at most $O(\log n)$. Then M simulates the nondeterministic computation of G (starting from the initial configuration) on another set of k worktapes. In the simulation, M records/updates the multiplicities of the different objects (which are in binary), making sure that at each step of the simulation, the second set of worktapes use no more space than the first set of worktapes. At some point during the computation, M guesses that it has reached the target configuration and checks that the tapes in the first set of worktapes are identical to the corresponding tapes in the second set, and accepts and halts; otherwise, M rejects and halts. Clearly, M is log n space bounded.

If we were only interested in halting reachable extended configurations, before the simulation, M first checks that x is a halting configuration by making sure that no transition rules are applicable to x. □

We note that the above construction can be used to solve the reachability problem of extended configurations for other membrane computing systems.

6. Catalytic system generator

In this section and the next, we study two models of P systems that define languages (instead of multisets of objects): the catalytic system generator and the communicating P system acceptor. Both are universal and therefore can also be analyzed with respect to space and time complexity. Language defining P systems have been studied before, e.g., P systems with external output [25], traces describing the itineraries of a given object (called "traveller") through membranes [15], and P automata [3,7]. We now define the model of a catalytic system generator.

Let G be a CS (thus the rules are of the form $Ca \to Cv$). Let V be the set of all noncatalysts, and Σ (called *language alphabet*) be a subset of V. We require that:

G has no rules of the form $Ca \to Cv$, where $a \in \Sigma$.

This means that symbols from Σ cannot be catalyzed (modified). This requirement can be made without loss of generality (see Remark 2).

If R is a rule, let $h(R)$ be the string obtained from the RHS of the rule by deleting all catalysts and noncatalysts not in Σ, preserving the order the symbols in Σ are written in the RHS. Thus, $h(R)$ is a (possibly null) string in Σ^*.

At the beginning, the initial configuration of G contains only a string of catalysts and noncatalysts not in Σ. Thus, we can think of $w_0 = \lambda$ (the null string) to be the only string in Σ^* present in the initial configuration. During the computation of G, w_0 is built up as follows. Suppose that w is the string in Σ^* in the current configuration. If in the next step $\{R_1, \ldots, R_k\}$ is a maximal set of applicable rules, then in the next configuration, $h(R_{i_1})h(R_{i_2}) \ldots h(R_{i_k})$ is appended to the right of w, where $i_1, \ldots, i_k$ is some nondeterministically chosen permutation of $1, \ldots, k$, yielding the string $w' = wh(R_{i_1})h(R_{i_2}) \ldots, h(R_{i_k}) \in \Sigma^*$ in the next configuration. We will see later (Corollary 6) that, in fact, every CSG can be converted to an equivalent CSG such that at most one symbol is appended to the string w at each step of the computation.

A string $x = a_1, \ldots, a_n \in \Sigma^*$ is generated if G when started from its initial configuration, halts after generating the symbols in x. (Note that the catalysts remain in the system, and there may be other noncatalysts not in Σ when the system halts.) The language (set of all strings) generated by G is denoted by $L(G)$. Call the CS just described a CS generator (CSG).

G is $S(n)$ space bounded if for every string x of length n that is in $L(G)$, there is a halting computation for x in which the sum of the multiplicities of all noncatalysts *not* in Σ in the system at any time during the computation is at most $S(n)$. Time boundedness is defined similarly.

In what follows, we assume without loss of generality that Σ contains only two symbols, 1 and 2. (Our results easily extend to the case when the alphabet has more than two symbols).

To simplify the proofs in this section, we use an equivalent model of a one-way counter machine (CM) (see the paragraph before Lemma 2 in Section 2). Here we assume that an atomic move of the CM consists of one of the following *labeled* instructions:

1. Read input; if it is 1 go to (k_1 or $\ldots$ or k_r) else go to (l_1 or $\ldots$ or l_s).
   (The labels need not be distinct. Thus, in general, the next instruction depends on the symbol read. Note that the use of "or" makes the instruction nondeterministic.)

2. Increment counter X by 1 and go to (k_1 or ... or k_r).
3. If counter X is positive then decrement the counter by 1 and go to (k_1 or ... or k_r) else go to (l_1 or ... or l_s).
4. Halt.

Note that an unconditional goto instruction "Go to (k_1 or ... or k_r)" can be simulated by an instruction of type 2 followed by an instruction of type 3. Hence, in the simulation proofs, we can use unconditional gotos.

A string x is accepted by M if M, when started in its initial state with all counters zero, halts after reading x.

There is an equivalent model called counter generator. Such a machine is like a counter acceptor but, instead of the Read instruction above, we have:

Generate z and go to (k_1 or ... or k_r). (Here, z is either 1 or 2.)

Every counter generator can be simulated by a counter acceptor, and conversely. Going from generator to acceptor is obvious. To simulate the Generate instruction, the acceptor Reads the input and if the input is z, it goes to instruction labeled (k_1 or ... or k_r); otherwise, the acceptor goes into an infinite loop by cycling through an increment$(+1)$/decrement(-1) of some fixed counter. To see the converse, suppose M is an acceptor. We construct a generator M' as follows. We introduce a new counter, DUMMY, which is initially zero. Suppose there is a Read instruction in M labeled h,

h: Read input; if it is 1 go to (k_1 or ... or k_r) else go to (l_1 or ... or l_s)

We replace this instruction by the following instructions:

h: Increment counter DUMMY by 1 and go to (k' or l')

k': If counter DUMMY is positive then decrement the counter by 1 and go to k else go to k

l': If counter DUMMY is positive then decrement the counter by 1 and go to l else go to l

k: Generate 1 and go to (k_1 or ... or k_r)

l: Generate 2 and go to (l_1 or ... or l_s)

The labels k', l', k, l are new and only used for instruction h. We use the above procedure to transform all Read instructions to Generate instructions.

We denote the language accepted (generated) by a counter acceptor (generator) M by $L(M)$.

Corollary 5. *A language L is accepted by an $S(n)$ space bounded counter acceptor if and only if it is generated by an $S(n)$ space bounded counter generator.*

We now show that a counter generator is equivalent to a CSG.

Theorem 8. 1. *Let G be an $S(n)$ space bounded CSG that runs in $T(n)$ time. We can construct an $S(n)$ space bounded counter generator M that runs in $O(T(n))$ time such that $L(M) = L(G)$.*

2. *Let M be an $S(n)$ space bounded counter generator that runs in $T(n)$ time. We can construct an $S(n)$ space bounded CSG G that runs in $O(T(n))$ time such that $L(G)=L(M)$.*

Proof. The first part follows from a straightforward simulation of the CSG by a counter generator, using one counter for each noncatalyst not in Σ to keep track of its multiplicity noting that the counter generator takes at most $O(k)$ steps to simulate 1 step of the CSG, where k is the number of catalysts. We need to use Lemma 2 part 1 on the counter machine so that it uses no more than $S(n)$ space.

The second part is an easy generalization of the construction in [30,31]. First we note that the construction in [30,31] involves the simulation of a deterministic counter machine. However, the construction easily generalizes to the case when the counter machine is nondeterministic (i.e., the use of "or" in the go to instructions). Clearly, the simulation of the Generate instruction is handled like incrementing a counter by 1. Again, we need to use Lemma 2 part 1 on the counter machine before applying the construction so that the resulting CSG remains $S(n)$ space bounded. □

From part 2 of Theorem 8, the definition of a counter generator, and the fact that the simulation described in [30,31] is faithful, we have:

Corollary 6. *Every CSG can be converted to an equivalent CSG such that at most one symbol in Σ is appended to the string $w\in\Sigma^*$ at each step of the computation.*

Remark 2. The construction of the counter generator in part 1 of Theorem 8 still works even if we allow the CSG to have rules of the form $Ca\rightarrow Cv$, where $a\in\Sigma$. Then we can use part 2 to construct (from the counter generator) an equivalent CSG without such rules.

The next corollary follows from Lemma 3, Corollary 5, Theorem 8, and the fact that for $S(n)\geqslant n$, an $S(n)$ space bounded TM with a one-way input tape is equivalent to an $S(n)$ space bounded TM with a two-way input tape. In what follows, TM means a two-way TM.

Corollary 7. *Let L be a language.*

1. *L is generated by a CSG if and only if it is accepted by a TM. Hence, a CSG can generate any recursively enumerable set.*
2. *Let $S(n)\geqslant n$. L is generated by a $c^{S(n)}$ space bounded CSG (for some c) if and only if it is accepted by a nondeterministic $S(n)$ space bounded TM.*
3. *L is generated by a c^n space bounded CSG (for some c) if and only if it is context-sensitive.*
4. *L is generated by an n^k space bounded CSG (for some k) if and only if it is accepted bu a nondeterministic $\log n$ space bounded one-way TM.*

From Theorem 8 and Lemma 2, we have:

Corollary 8. *Let* $S(n) \geqslant n$.

1. *Given any* $S(n)$ *space bounded CSG, we can construct an equivalent* $cS(n)$ *space bounded CSG for any constant* $c > 0$.
2. *For every* k, *given any* $S^k(n)$ *space bounded CSG, we can construct an equivalent* $S(n)$ *space bounded CSG. Thus in particular, a polynomial space bounded CSG can be converted to one which is linear space bounded.*

Because of the equivalence in Theorem 8, we can talk about space bounded and time bounded CSGs. We can use the techniques in the previous sections to prove results concerning complexity classes of languages defined by CSGs. For example, the following follows from Lemma 3:

Theorem 9. *If* L *is in* $NSPACE(S_2(n)) - NSPACE(S_1(n))$, *then* L *can be generated by a* $2^{S_2(n)}$ *space bounded CSG but not by a* $2^{S_1(n)}$ *spaced bounded CSG.*

So, e.g., since for any integer $k > 0$, $NSPACE(n^{k+1}) - NSPACE(n^k)$ is not empty [12], it follows that there is a language generated by a $2^{n^{k+1}}$ space bounded CSG that cannot be generated by a 2^{n^k} space bounded CSG.

The next result says that linear space is a lower bound for CSG to generate a nonregular language.

Theorem 10. *If a CSG* G *generates a language* L *that is not regular, then* G *must use linear space* (*i.e., linear number of objects*) *for infinitely many strings in* L.

Proof. It is known that if L is accepted by a nondeterministic one-way TM and L is not regular, then the TM uses $\log n$ space for infinitely many inputs of length n [11]. The result follows from Theorem 8 and Lemma 3. □

Consider the languages $L_1 = \{y \mid y \in \{1,2\}^+, |y| \text{ is even and } y \text{ is not a palindrome}\}$ and $L_2 = \{1^i 2^i \mid i \geqslant 1\}$. Clearly, L_1 and L_2 can be accepted by $\log n$ space-bounded (one-way) TMs. Hence by Lemma 3, they can be accepted by CM acceptors (or, equivalently, by CM generators) in linear space and, therefore, can also be generated by CSGs in linear space. By the theorem above, L_1 and L_2 cannot be generated by CSGs in sublinear space.

Now consider $L_3 = \{xx^R \mid x \in \{1,2\}^+\}$, which is essentially the complement of L_1. Clearly, L_3 can be accepted by a one-way TM in linear space. It is known that any one-way TM accepting L_3 must use linear space for infinitely many strings in L_3. Hence, L_3 can be generated by an exponential space bounded CSG, and any CSG generating L_3 must use exponential space for infinitely many strings in L_3.

7. Communicating P system acceptor

Using the ideas in the previous section, we can define a communicating P system acceptor (rather than a generator). Our approach is similar to the study of P automata in [3,7].

Let G be a communicating P system [30,32,8]. Recall that the rules in G are of the following forms, where V is the set of all objects:

1. $a \to a_\tau$,
2. $ab \to a_{\tau_1} b_{\tau_2}$,
3. $ab \to a_{\tau_1} b_{\tau_2} c_{\text{come}}$,

where $a, b, c \in V$ and $\tau_1, \tau_2 \in (\{here, out\} \cup \{in_j \mid 1 \leqslant j \leqslant n\})$. A rule of form (3) can occur only within the region enclosed by the skin membrane. When such a rule is applied, then c is imported through the skin membrane from the outer space (environment) and will become an element of this region.

We can make G a language acceptor, called a CPA. Let $\Sigma = \{1, 2\}$ be a distinguished subset of V, called the *language alphabet*. We impose the following requirements:

1. Initially, G does not contain any object in Σ.
2. No object in Σ appears on the left hand side of any rule. This means that when a symbol c in Σ enters the skin membrane from the environment (using a rule of the form $ab \to a_{\tau_1} b_{\tau_2} c_{\text{come}}$), it remains in the skin region and is not exported to any region. Thus c is "read-only".

The restriction that c is "read-only" can be made without loss of generality (see Remark 3).

At the start of the computation, the only string from Σ^* imported so far from the environment is $w = \lambda$. Symbols from Σ are appended to w as they are imported from the environment during the computation. Note that in general, because of "maximal parallelism", an unbounded number of symbols from Σ can enter the skin membrane in one step since several rules of type 3 may be applicable to an unbounded number of ab pairs in the skin membrane. If the symbols that enter the membrane in the step are $c_1, \ldots, c_k$ (note that k is not fixed), then $c_{i_1}, \ldots, c_{i_k}$ is the string appended to w, where $i_1, \ldots, i_k$ is some nondeterministically chosen permutation of $1, \ldots, k$. Later, we will see that, in fact, we can assume without loss of generality that $k \leqslant 1$ (Corollary 9).

We say that a string $x = a_1, \ldots, a_n \in \Sigma^*$ is accepted by a CPA G if G has a halting computation after importing symbols $a_1, \ldots, a_n$ from the environment. The language accepted by G is denoted by $L(G)$. G is $S(n)$ space bounded if for every string x of length n that is in $L(G)$, there is a halting computation for x in which the sum of the multiplicities of all symbols *not* in Σ in the system at any time during the computation is at most $S(n)$.

Theorem 11. 1. *Let G be an $S(n)$ space bounded CPA. We can construct an $S(n)$ space bounded counter generator M such that $L(M) = L(G)$.*
2. *Let M be an $S(n)$ space bounded counter generator. We can construct an $S(n)$ space bounded CPA G such that $L(G) = L(M)$.*

Proof. The proof of the first part is similar to the proof of Lemma 4. Again, auxiliary counters are used in the simulation. Note that if in one step, $c_1, \ldots, c_k$ (k is not fixed) from Σ enter the skin membrane, the counter generator needs to generate $c_{i_1}, \ldots, c_{i_k}$, where $i_1, \ldots, i_k$ is some nondeterministically chosen permutation of $1, \ldots, k$. This can be done since each c_i comes from a rule of the form $ab \to a_{\tau_1} b_{\tau_2} c_{\text{come}}$, and the multiplicities of a and b are recorded in the counters.

The second part follows from the construction in [30,32]. The only modification is when an object in Σ is imported to the skin membrane (region 1) from the environment, the symbol remains in the skin membrane. □

The next result follows from the above theorem, the definition of a counter generator, and the fact that the simulation described in [30,32] is faithful.

Corollary 9. *Every CPA can be converted to an equivalent CPA such that at most one symbol from Σ is imported from the environment (i.e., at most one symbol is appended to the string $w \in \Sigma^*$) at each step of the computation.*

Remark 3. The construction of the counter generator in part 1 of Theorem 11 still works even if the CPA has rules that allow symbols in Σ to be exported from the skin membrane to other regions. Then we can use part 2 to construct (from the counter generator) an equivalent CPA without such rules.

From Theorems 8 and 11, we obtain:

Corollary 10. *A language L is generated by an $S(n)$ space bounded CSG if and only if it is accepted by an $S(n)$ space bounded CPA.*

As in Corollary 7, we have:

Corollary 11. *Let L be a language.*
1. *L is accepted a CPA if and only if it is accepted by a TM. Hence, a CPA can accept any recursively enumerable set.*
2. *Let $S(n) \geqslant n$. L is accepted by a $c^{S(n)}$ space bounded CPA (for some c) if and only if it is accepted by a nondeterministic $S(n)$ space bounded TM.*
3. *L is accepted by a c^n space bounded CPA (for some c) if and only if it is context-sensitive.*
4. *L is accepted by an n^k space bounded CPA (for some k) if and only if it is accepted by a nondeterministic* log *n space bounded one-way TM.*

Again, we can prove complexity results (hierarchies, etc.) concerning CPAs similar to those in the previous section. One can also study symport/antiport acceptors [3,7] that are space bounded and obtain similar hierarchy results.

8. Semilinear languages

In this section, we give a characterization of semilinear languages in terms of a restricted form of CSG (catalytic system generator).

First we recall the definition of a semilinear set. Let $\mathbf{N}$ be the set of all nonnegative integers and n be a positive integer. A set $S \subseteq \mathbf{N}^n$ is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_t$ in $\mathbf{N}^n$ such that $S = \{v \mid v = v_0 + a_1 v_1 + \cdots + a_t v_t,\ a_i \in \mathbf{N}\}$. The vectors

v_0 (referred to as the *constant vector*) and $v_1, v_2, \ldots, v_t$ (referred to as the *periods*) are called the *generators* of the linear set S. A set $S \subseteq \mathbf{N}^n$ is *semilinear* if it is a finite union of linear sets. The empty set is a trivial (semi)linear set, where the set of generators is empty. Every finite subset of $\mathbf{N}^n$ is semilinear—it is a finite union of linear sets whose generators are constant vectors. Clearly, semilinear sets are closed under union and projection. It is also known that semilinear sets are closed under intersection and complementation.

Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be an alphabet. For each string w in Σ^*, define the *Parikh map* of w to be $\psi(w) = (\#_{a_1}(w), \ldots, \#_{a_n}(w))$, where $\#_{a_i}(x)$ is the number of occurrences of a_i in w. For a language $L \subseteq \Sigma^*$, the *Parikh map* of L is $\psi(L) = \{\psi(w) \mid w \in L\}$.

We say that a class $\mathscr{L}$ of languages is a *semilinear class of languages* if (a) for every language L in $\mathscr{L}$, $\psi(L)$ is a semilinear set, and (b) for every semilinear set S, the language $L(S) = \{a_1^{i_1}, \ldots, a_n^{i_n} \mid (i_1, \ldots, i_n) \in S\}$ is in $\mathscr{L}$.

Let G be a CSG with language alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$, and k be a positive integer. Define the set $L_k(G) = \{x \mid$ there is a halting computation on x such that for each object in G, the number of times its multiplicity changes mode from nondecreasing to nonincreasing and vice-versa during the computation is at most $k\}$. We call $L_k(G)$ the k reversal bounded language generated by G. If every string generated by G has a k reversal bounded computation, then we say that G is k reversal bounded. G is reversal bounded if it is k reversal bounded for some k. Let $\mathscr{L}_{\text{rev}}(CSG) = \{L_k(G) \mid G$ is a CSG and k is a positive integer $\}$. Similar notions of k reversal boundedness, etc. apply to counter acceptors (which are equivalent to counter generators). In particular, $\mathscr{L}_{\text{rev}}(CM) = \{L_k(M) \mid M$ is a CM and k is a positive integer $\}$, where CM denotes counter acceptor.

The *shuffle* $u \amalg v$ of two words $u, v \in \Sigma^*$ is a finite set consisting of the words $u_1v_1, \ldots, u_kv_k$, where $u = u_1u_2, \ldots, u_k$ and $v = v_1v_2, \ldots, v_k$ for some $u_i, v_i \in \Sigma^*$. If L_1 and L_2 are two languages, their *shuffle* is the language

$$L_1 \amalg L_2 = \bigcup_{u \in L_1, v \in L_2} u \amalg v.$$

It is known that $\mathscr{L}_{\text{rev}}(CM)$ is a semilinear class of languages [13]. It is also known [10] that $\mathscr{L}_{\text{rev}}(CM)$ is the smallest class containing the regular sets that is closed under homomorphism, intersection, and the shuffle operation. Thus, we have:

Theorem 12. 1. *$\mathscr{L}_{\text{rev}}(CSG)$ is a semilinear class of languages.*
2. *$\mathscr{L}_{\text{rev}}(CSG)$ is the smallest class containing the regular sets that is closed under homomorphism, intersection, and the shuffle operation.*

9. Conclusion

We showed how techniques in machine-based complexity can be used to analyze the space and time complexity of membrane computing systems. Our focus was on catalytic systems, communicating P systems, and systems with only symport/antiport rules, but our techniques are applicable to other P systems that are universal. In particular, the

space hierarchy results apply to any universal P system, since any such system that is $S(n)$ space bounded can be simulated by an $S(n)$ space bounded counter machine, and vice-versa. Our results on time complexity was only for catalytic systems where the only rules are of the form $Ca \to Cv$. We will look at time complexity issues for other types of systems (e.g., allowing rules of the form $a \to v$ in the catalytic system; communicating P system; symport/antiport system; etc.) in the future. We also showed that there is a fixed k such that any deterministic counter machine can be simulated by a deterministic communicating P system with k membranes, answering positively an open question in [30,32]. We improved the main result in [23]—that reachability of extended configurations for symport/antiport systems (as well as catalytic systems and communicating P systems) can be decided in nondeterministic log n space and, hence, in deterministic $\log^2 n$ space or in polynomial time. We introduced two equivalent systems that define languages (instead of multisets of objects): the first is a catalytic system language generator and the other is a communicating P system acceptor (or a symport/antiport system acceptor). These devices are universal and therefore can also be analyzed with respect to space and time complexity. Finally, gave a characterization of semilinear languages in terms of a restricted form of catalytic system generator.

Acknowledgements

I would like to thank Andrei Paun, Gheorghe Paun, and Petr Sosik for their comments and for providing some of the references concerning communicating P systems and systems with symport/antiport rules. I would also like to thank the anonymous referee for suggestions which improve the presentation of our results.

References

[1] G. Berry, G. Boudol, The chemical abstract machine, in: POPL'90, ACM Press, New York, 1990, pp. 81–94.
[2] R. Book, Tally languages and complexity classes, Inform. and Control 26 (1974) 186–193.
[3] E. Csuhaj-Varju, G. Vaszil, P automata or purely communicating accepting P systems, in: WMC-CdeA2002, Lecture Notes in Computer Science, vol. 2597, Springer, Berlin, 2003, pp. 219–233.
[4] P.C. Fischer, Turing machines with restricted memory access, Inform. and Control 9 (1966) 364–379.
[5] P.C. Fischer, A.R. Meyer, A.L. Rosenberg, Counter machines and counter languages, Math. Systems Theory 2 (1968) 265–283.
[6] R. Freund, L. Kari, M. Oswald, P. Sosik, Computationally universal P systems without priorities: two catalysts are sufficient, available at http://psystems.disco.unimib.it, 2003.
[7] R. Freund, M. Oswald, A short note on analyzing P systems with antiport rules, Bull. EATCS 78 (2002) 231–236.
[8] R. Freund, A. Paun, Membrane systems with symport/antiport rules: universality results, in: WMC-CdeA2002, Lecture Notes in Computer Science, vol. 2597, Springer, Berlin, 2003, pp. 270–287.
[9] P. Frisco, H. Jan Hoogeboom, Simulating counter automata by P systems with symport/antiport, in: WMC-CdeA2002, Lecture Notes in Computer Science, vol. 2597, Springer, Berlin, 2003, pp. 288–301.
[10] T. Harju, O.H. Ibarra, J. Karhumaki, A. Salomaa, Some decision problems concerning semilinearity and commutation, J. Comput. System Sci. 65 (2002) 278–294.

[11] J. Hartmanis, P.M. Lewis II, R.E. Stearns, Hierarchies of memory limited computations, in: IEEE, Conf. Record on Switching Circuit Theory and Logical Design, IEEE, New York, 1965, pp. 179–190.
[12] O.H. Ibarra, A note concerning nondeterminstic tape complexities, J. ACM 19 (1972) 608–612.
[13] O.H. Ibarra, Reversal-bounded multicounter machines and their decision problems, J. ACM 25 (1) (1978) 116–133.
[14] O.H. Ibarra, Z. Dang, O. Egecioglu, G. Saxena, Characterizations of catalytic membrane computing systems, in: Mathematical Foundations of Computer Science 2003. Lecture Notes in Computer Science, vol. 2747, Springer, Berlin, 2003, pp. 480–489.
[15] M. Ionescu, C. Martin-Vide, Gh. Paun, P systems with symport/antiport rules: the traces of objects, Grammars 5 (2002) 65–79.
[16] C. Martin-Vide, A. Paun, Gh. Paun, On the power of P systems with symport rules, J. Universal Comput. Sci. 8 (2) (2002) 317–331.
[17] C. Martin-Vide, A. Paun, Gh. Paun, G. Rozenberg, Membrane systems with coupled transport: universality and normal forms, Fund. Inform. 49 (2002) 1–15.
[18] M. Minsky, Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines, Ann. of Math. 74 (1961) 437–455.
[19] A. Paun, Gh. Paun, The power of communication: P systems with symport/antiport, New Generation Comput. 20 (3) (2002) 295–306.
[20] A. Paun, Gh. Paun, G. Rozenberg, Computing by communication in networks of membranes, Internat. J. Found. Comput. Sci. 13 (6) (2002) 779–798.
[21] Gh. Paun, Computing with membranes, J. Comput. System Sci. 61 (1) (2000) 108–143.
[22] Gh. Paun, Membrane Computing: An Introduction, Springer, Berlin, 2002.
[23] Gh. Paun, M. Perez-Jimenez, F. Sancho-Caparrini, On the reachability problem for P systems with symport/antiport, Proceedings of the 10th International Conference on Automata and Formal Languages, Debrecen, Hungary, 2002.
[24] Gh. Paun, G. Rozenberg, A guide to membrane computing, Theoret. Comput. Sci. 287 (1) (2002) 73–100.
[25] Gh. Paun, G. Rozenberg, A. Salomaa, Membrane computing with an external output, Fund. Inform. 41 (3) (2000) 259–366.
[26] M.J. Perez-Jimenez, A. Riscos-Nunez, A linear-time solution to the knapsack problem using active membranes, in: Pre-Proc. Workshop on Membrane Computing (WMC-CdeA2002), Tarragona, Spain, 2003, pp. 326–343.
[27] M.J. Perez-Jimenez, A. Riscos-Nunez, Solving the subset-sum problem by active membranes, New Generation Computing, Springer, Berlin, to appear in 2004.
[28] W. Savitch, Relationships between nondeterministic and deterministic tape complexities, J. Comput. System Sci. 4 (1970) 177–192.
[29] W. Savitch, A note on multihead automata and context-sensitive languages, Acta Inform. 2 (1973) 249–252.
[30] P. Sosik, P systems versus register machines: two universality proofs, in: Pre-Proc. Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Arges, Romania, 2002, pp. 371–382.
[31] P. Sosik, R. Freund, P systems without priorities are computationally universal, in: WMC-CdeA2002, Lecture Notes in Computer Science, vol. 2597, Springer, Berlin, 2003, pp. 400–409.
[32] P. Sosik, J. Matysek, Membrane computing: when communication is enough, in: Unconventional Models of Computation 2002, Lecture Notes in Computer Science, vol. 2509, Springer, Berlin, 2002, pp. 264–275.
[33] C. Zandron, C. Ferretti, G. Mauri, Solving NP complete problems using P systems with active membranes, available at http://psystems.disco.unimib.it. 2000.