On Bounded Symport/Antiport P Systems*

Oscar H. Ibarra and Sara Woodworth

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

Abstract. We introduce a restricted model of a one-membrane symport/antiport system, called *bounded S/A system*. We show the following:

1. A language $L \subseteq a_1^*...a_k^*$ is accepted by a bounded S/A system if and only if it is accepted by a $log\ n$ space-bounded Turing machine. This holds for both deterministic and nondeterministic versions.
2. For every positive integer r, there is an $s > r$ and a unary language L that is accepted by a bounded S/A system with s objects that cannot be accepted by any bounded S/A system with only r objects. This holds for both deterministic and nondeterministic versions.
3. Deterministic and nondeterministic bounded S/A systems over a unary input alphabet are equivalent if and only if deterministic and nondeterministic linear-bounded automata (over an arbitrary input alphabet) are equivalent.

We also introduce a restricted model of a multi-membrane S/A system, called *special S/A system*. The restriction guarantees that the number of objects in the system at any time during the computation remains constant. We show that for every nonnegative integer t, special S/A systems with environment alphabet E of t symbols (note that other symbols are allowed in the system if they are not transported into the environment) has an infinite hierarchy in terms of the number of membranes. Again, this holds for both deterministic and nondeterministic versions. Finally, we introduce a model of a one-membrane bounded S/A system, called *bounded SA acceptor*, that accepts string languages. We show that the deterministic version is strictly weaker than the nondeterministic version.

Clearly, investigations into complexity issues (hierarchies, determinism versus nondeterminism, etc.) in membrane computing are natural and interesting from the points of view of foundations and applications, e.g., in modeling and simulating of cells. Some of the results above have been shown for other types of restricted P systems (that are not symport/antiport). However, these previous results do not easily translate for the models of S/A systems we consider here. In fact, in a recent article, "Further Twenty Six Open Problems in Membrane Computing" (January 26, 2005; see P Systems Web Page at http://psystems.disco.unimib.it), Gheorghe Paun poses the question of whether the earlier results, e.g., concerning determinism versus nondeterminism can be proved for restricted S/A systems.

Keywords: Symport/antiport system, communicating P system, Turing machine, multihead two-way finite automaton, multicounter machine, hierarchy, deterministic, nondeterministic.

* This work was supported in part by NSF Grants CCR-0208595, CCF-0430945, and CCF-0524136.

A. Carbone and N.A. Pierce (Eds.): DNA11, LNCS 3892, pp. 129–143, 2006.

1 Introduction

Membrane computing is a relatively new computing paradigm which abstracts the activities of biological cells to find a new model for computing. While humanity has learned to compute mechanically within the relatively recent past, other living processes have computed naturally for millions of years. One such natural computing process can be found within biological cells. Cells consist of membranes which are used to contain, transfer, and transform various enzymes and proteins in a naturally decentralized and parallel manner. By modeling these natural processes of cells, we can create a new model of computing which is decentralized, nondeterministic, and maximally parallel.

Using biological membranes as an inspiration for computing was first introduced by Gheorghe Paun in a seminal paper [10] (see also [11, 12]). He studied the first membrane computing model, called P system, which consists of a hierarchical set of *membranes* where each membrane contains both a multiset of *objects* and a set of *rules* which determine how these objects interact within the system. The rules are applied in a *nondeterministic* and *maximally parallel* fashion. At each step of the computation, a maximal multiset of rules is chosen nondeterministically (note that several instances of a rule may be selected) and the rules applied simultaneously (i.e., in parallel). Maximal here means that in each step, no additional rule instance not already in the multiset of rules is applicable and could be added to the multiset of rules. The system is nondeterministic because the maximal multiset may not be unique. The outermost membrane is often referred to as the *skin* membrane and the area surrounding the system is referred to as the *environment*. A membrane not containing any membrane is referred to as an *elementary* membrane. As a branch of Natural Computing which explores new models, ideas, and paradigms from the way nature computes, membrane computing has been quite successful: many models have been introduced, most of them Turing complete. (See http://psystems.disco.unimb/it for a large collection of papers in the area, and in particular the monograph [12].) Due to the maximal parallelism inherent in the model, P systems have a great potential for implementing massively concurrent systems in an efficient way that would allow us to solve currently intractable problems (in much the same way as the promise of quantum and DNA computing) once future biotechnology gives way to a practical bio-realization. Given this potential, the Institute for Scientific Information (ISI) has selected membrane computing as a fast "Emerging Research Front" in Computer Science (http://esi-topics.com/erf/october2003.html).

One very popular model of a P system is called a symport/antiport system (introduced in [9]). It is a simple system whose rules closely resemble the way membranes transport objects between themselves in a purely communicating manner. Symport/antiport systems (S/A systems) have rules of the form (u, out), (u, in), and $(u, out; v, in)$ where $u, v \in \Sigma^*$. Note that u, v are multisets that are represented as strings (the order in which the symbols are written is not important, since we are only interested in the multiplicities of each symbol). A rule of the form (u, out) in membrane i sends the elements of u from membrane i out to the membrane (directly) containing i. A rule of the form (u, in) in membrane i transports the elements of u into membrane i from the membrane enclosing i. Hence this rule can only be used when the elements of u exist in the outer membrane. A rule of the form $(u, out; v, in)$ simultaneously sends u out of the membrane i while transporting v into membrane i. Hence this rule cannot be

applied unless membrane i contains the elements in u and the membrane surrounding i contains the elements in v. Formally an S/A system is defined as

$$M = (V, H, \mu, w_1, \cdots, w_{|H|}, E, R_1, \cdots, R_{|H|}, i_o)$$

where V is the set of objects (symbols) the system uses, H is the set of membrane labels, μ is the membrane structure of the system, w_i is the initial multiset of objects within membrane i, and the rules are given in the set R_i. E is the set of objects which can be found within the environment, and i_o is the designated elementary output membrane. (When the system is used as a recognizer or acceptor, there is no need to specify i_o.) A large number of papers have been written concerning symport/antiport systems. It has been shown that "minimal" such systems (with respect to the number of membranes, the number of objects, the maximum "size" of the rules) are universal.

Initially, membrane systems were designed to be nondeterministic systems. When multiple, maximal sets of rules are applicable, nondeterminism decides which maximal set to apply. Recently, deterministic versions of some membrane models have been studied to determine whether they are as computationally powerful as the nondeterministic versions [5, 7]. Deterministic models guarantee that each step of the computation consists of only one maximal multiset of applicable rules. In some cases, both the nondeterministic and deterministic versions are equivalent in power to Turing Machines (see, e.g., [5]). In some non-universal P systems, the deterministic versus the nondeterministic question has been shown to be equivalent to the long-standing open problem of whether deterministic and nondeterministic linear-bounded automata are equivalent [7]; for another very simple class of systems, deterministic is strictly weaker than nondeterministic [7]. However, these two latter results do not easily translate for S/A systems.

In this paper, we look at restricted models of symport/antiport systems. Two models, called *bounded S/A system* and *special S/A system*, are acceptors of multisets with the restriction that the multiplicity of each object in the system does not change during the computation. These models differ in whether they also bound the number of membranes within the system or bound the number of distinct objects that can occur abundantly in the environment. Another model, called *bounded S/A acceptor*, is an acceptor of string languages. Again, this model has the property that at any time during the computation, the number of objects in the system is equal to the number of input symbols that have been read so far (in addition to a fixed number of objects given to the system at the start of the computation). We study the computing power of these models. In particular, we investigate questions concerning hierarchies (with respect to the number of distinct objects used in the system or number of membranes in the system) and whether determinism is strictly weaker than nondeterminism.

2 One-Membrane Bounded S/A System

Let M be a one-membrane symport/antiport system over an alphabet V, and let $\Sigma = \{a_1, ..., a_k\} \subseteq V$ be the *input* alphabet. M is restricted in that all rules are of the form $(u, out; v, in)$, where $u, v \in V^*$ with $|u| = |v| \geq 1$. Thus, the number of objects in the system at any time during the computation remains the same. Note that all the rules are antiport rules.

There is a fixed string (multiset) w in $(V-\Sigma)^*$ such that initially, the system is given a string $wa_1^{n_1}...a_k^{n_k}$ for some nonnegative integers $n_1, ..., n_k$ (thus, the input multiset is $a_1^{n_1}...a_k^{n_k}$). If the system halts, then we say that the string $a_1^{n_1}...a_k^{n_k}$ is accepted. The set of all such strings is the language $L(M)$ accepted by M. We call this system a *bounded S/A system*. M is *deterministic* if the maximally parallel multiset of rules applicable at each step in the computation is unique. We will show the following:

1. A language $L \subseteq a_1^*...a_k^*$ is accepted by a deterministic (nondeterministic) bounded S/A system if and only if it is accepted by a deterministic (nondeterministic) $log\ n$ space-bounded Turing machine (with a two-way read-only input with left and right end markers).
2. For every r, there is an $s > r$ and a unary language L (i.e., $L \subseteq o^*$) accepted by a bounded S/A system with an alphabet of s symbols that cannot be accepted by any bounded S/A system with an alphabet of r symbols. This result holds for both deterministic and nondeterministic versions.
3. Deterministic and nondeterministic bounded S/A systems over a unary input alphabet are equivalent if and only if deterministic and nondeterministic linear-bounded automata (over an arbitrary alphabet) are equivalent. This later problem is a long-standing open problem in complexity theory [16].

The restriction $|u| = |v| \geq 1$ in the rule $(u, out; v, in)$ can be relaxed to $|u| \geq |v| \geq 1$, but the latter is equivalent in that we can always introduce a dummy symbol d and add $d^{|u|-|v|}$ to v to make the lengths the same and not use symbol d in any rule. We note that a similar system, called bounded P system (BPS) with cooperative rules of the form $u \rightarrow v$ where $|u| \geq |v| \geq 1$, was also recently studied in [3] for their model-checking properties.

For ease in exposition, we first consider the case when the input alphabet is unary, i.e., $\Sigma = \{o\}$. Thus, the bounded S/A system M has initial configuration wo^n (for some n). The idea is to relate the computation of M to a restricted type of multicounter machine, called *linear-bounded multicounter machine*.

A *deterministic* multicounter machine Z is linear-bounded if, when given an input n in one of its counters (called the input counter) and zeros in the other counters, it computes in such a way that the sum of the values of the counters at any time during the computation is at most n. One can easily normalize the computation so that every increment is preceded by a decrement (i.e., if Z wants to increment a counter C_j, it first decrements some counter C_i and then increments C_j) and every decrement is followed by an increment. Thus we can assume that every instruction of Z, which is not 'Halt', is of the form:

p : If $C_i \neq 0$, decrement C_i by 1, increment C_j by 1, and goto k else goto state l.

where p, k, l are labels (states). We do not require that the contents of the counters are zero when the machine halts.

If in the above instruction, there is a "choice" for states k and/or l, the machine is *nondeterministic*. We will show that we can construct a deterministic (nondeterministic) bounded S/A system M which uses a fixed multiset w such that, when M is started

with multiset wo^n, it simulates Z and has a halting computation if and only if Z halts on input n. Moreover, the rules of M are of the form $u \to v$, where $|u| = |v| = 1$ or 2.

It is convenient to use an intermediate P system, called SCPS, which is a restricted version of the the CPS (communicating P system) introduced in [18]. A CPS has multiple membranes, with the outermost one called the skin membrane. The rules in the membranes are of the form:

1. $a \to a_x$,
2. $ab \to a_x b_y$,
3. $ab \to a_x b_y c_{come}$,

where a, b, c are objects, x, y (which indicate the directions of movements of a and b) can be $here$, out, or in_j. The designation $here$ means that the object remains in the membrane containing it, out means that the object is transported to the membrane directly enclosing the membrane that contains the object (or to the environment if the object is in the skin membrane). The designation in_j means that the object is moved into the membrane, labeled j, that is directly enclosed by the membrane that contains the object. A rule of the form (3) can only appear in the skin membrane. When such a rule is applied, c is imported through the skin membrane from the environment (i.e., outer space) and will become an element in the skin membrane. In one step, all rules are applied in a maximally parallel manner. For notational convenience, when the target designation is not specified, we assume that the symbol remains in the membrane containing the rule.

Let V be the set of all objects (i.e., symbols) that can appear in the system, and o be a distinguished object (called the *input symbol*). A CPS M has m membranes, with a distinguished *input membrane*. We assume that only the symbol o can enter and exit the skin membrane (thus, all other symbols remain in the system during the computation). We say that M accepts o^n if M, when started with o^n in the input membrane initially (with no o's in the other membranes), eventually halts. Note that objects in $V - \{o\}$ have fixed numbers and their distributions in the different membranes are fixed initially. Moreover, their multiplicities remain the same during the computation, although their distributions among the membranes may change at each step. The language accepted by M is $L(M) = \{o^n \mid o^n \text{ is accepted by } M\}$.

It is known that a language $L \subseteq o^*$ is accepted by a deterministic (nondeterministic) CPS if and only if it is accepted by a deterministic (nondeterministic) multicounter machine. (Again, define the language accepted by a multicounter machine Z to be $L = \{o^n \mid Z \text{ when given } n \text{ has a halting computation }\}$). The "if" part was shown in [18]. The "only if" part is easily verified. Hence, every unary recursively enumerable language can be accepted by a deterministic CPS (hence, also by a nondeterministic CPS).

An SCPS ('S' for simple) is a restricted CPS which has only rules of the form $a \to a_x$ or $ab \to a_x b_y$. Moreover, if the skin membrane has these types of rules, then $x, y \neq out$ (i.e., no objects are transported to the environment).

Lemma 1. *If a language $L \subseteq o^*$ is accepted by a deterministic (nondeterministic) linear-bounded multicounter machine Z, then it is accepted by a deterministic (nondeterministic) SCPS M.*

Proof. We only prove the case when Z is deterministic, the nondeterministic case being similar. The construction of M is a simple modification of the construction in [18]. Assume Z has m counters $C_1, ..., C_m$. M has the same membrane structure as in [18]. In particular, the skin membrane contains membranes $E_1, ..., E_m$ to simulate the counters, where the multiplicity of the distinguished (input) symbol o in membrane E_i represents the value of counter C_i. There are other membranes within the skin membrane that are used to simulate the instructions of Z (see [18]). All the sets of rules $R_1, ...,$ are the same as in [18], except the instruction

p : If $C_i \neq 0$, decrement C_i by 1, increment C_j by 1, and goto l else goto k

of Z is simulated as in [18], but the symbol o is not thrown out (from the skin membrane) into the environment but added to membrane E_j. It follows from the construction in [18] that M will not have any instruction of the form $ab \rightarrow a_x b_y c_{come}$ and if instructions of the form $a \rightarrow a_x$ or $ab \rightarrow a_x b_y$ appear in the skin membrane, then $x, y \neq out$. Hence, M is a deterministic SCPS. □

Lemma 2. *If a language $L \subseteq o^*$ is accepted by a deterministic (nondeterministic) linear-bounded multicounter machine, then it is accepted by a deterministic (nondeterministic) bounded S/A system.*

Proof. We show how to convert the multi-membrane SCPS M of Lemma 1 to a (one-membrane) bounded S/A system M'. The construction is similar to the one given in [3]. Suppose that M has membranes $1, ..., m$. For each object a in V, M' will have symbols $a_1, ..., a_m$. In particular, for the distinguished input symbol o in V, M' will have $o_1, ..., o_m$. Hence the distinguished input symbol in M' is o_{i_0}, where i_0 is the index of the input membrane in M. We can convert M to a bounded S/A system M' as follows:

1. If $a \rightarrow a_x$ is a rule in membrane i of M, then $(a_i, out; a_j, in)$ is a rule in M', 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 M, then $(a_i b_i, out; a_j b_k, in)$ is a rule in M', where j and k are the indices of the membranes into which a and b are transported to, as specified by x and y.

Thus, corresponding to the initial configuration wo^n of M, where o^n is in the input membrane i_0 and w represents the configuration denoting all the other symbols (different from o) in the other membranes, M' will have initial configuration $w' o_{i_0}^n$, where w' are symbols in w renamed to identify their locations in M.

Clearly, M' accepts $o_{i_0}^n$ if and only if M accepts o^n, and M' is a deterministic (nondeterministic) bounded S/A system. □

We will prove the converse of Lemma 2 indirectly. A k-head two-way finite automaton (k-2FA) is a finite automaton with k two-way read-only heads operating on an input (with left and right end markers) [8]. A multihead 2FA is a k-2FA for some k.

Lemma 3. *If M is a deterministic (nondeterministic) bounded S/A system with an alphabet V of m symbols (note that V contains the distinguished input symbol o), then M can be simulated by a deterministic (nondeterministic) $m(m+1)$-2FA Z.*

Proof. Suppose M is a deterministic bounded S/A system accepting a language $L(M) \subseteq o^*$. Assume that its alphabet is $V = \{a_1, ..., a_m\}$, where $a_1 = o$ (the input symbol). We construct a deterministic multihead FA Z to accept $L(G)$. The input to Z (not including the left and right end markers) is o^n for some n. We will need the following heads to keep track of the multiplicities of the symbols in the membrane during the computation (note that the bounded S/A system M is given wo^n initially):

1. K_i for $1 \leq i \leq m$. Head K_i will keep track of the current number of a_i's. Initially, K_1 will point to the right end marker (indicating that there are n o's in the input) while all other K_i will point to the appropriate position on the input corresponding to the multiplicity of symbol a_i in the fixed string w.
2. $K_{i,j}$ for $1 \leq i, j \leq m$. These heads keep track of how many a_i's are replaced by a_j's during the next step of M.

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

Let $R_1, R_2, ..., R_s$ be the rules in the membrane. By using $K_1, ..., K_m$ (note each K_i represents the number of a_i's in the membrane), Z 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 head $K_{i,j}$. Thus, each application of R_1 may involve decrementing the K_i's and incrementing some of the $K_{i,j}$'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 $R_2, ..., R_s$. When all the rules have been processed, Z updates each head K_j using the values stored in $K_{i,j}$, $1 \leq i \leq m$. This completes the simulation of the unique (because M is deterministic) maximally parallel step of M.

It follows from the above description that a deterministic bounded S/A system can be simulated by a deterministic $m(m+1)$-2FA.

If M is nondeterministic, the construction of the nondeterministic multihead 2FA M is simpler. M just sequentially guesses the rule to apply each time (i.e., any of $R_1, R_2, ..., R_s$) until no more rule is applicable. Note that Z does not need the heads $K_{i,j}$'s. □

For the proof of the next theorem, we need a definition. Define a generalized linear-bounded multicounter machine as follows. As before, at the start of the computation, the input counter is set to a value n (for some n), and all other counters are set to zero. Now we only require that there is a positive integer c such that at any time during the computation, the value of any counter is at most cn. (Thus, we no longer require that the sum of the values of the counters is at most n.) In [3], it was shown that a generalized linear-bounded multicounter machine can be converted to a linear-bounded multicounter machine. For completeness, we describe the construction.

Suppose that Z is a generalized linear-bounded multicounter machine with counters $C_1, ..., C_m$, where C_1 is the input counter. Construct another machine Z' with counters $D, C_1, ..., C_m$, where D is now the input counter. Z' with input n in counter D, first moves n from D to C_1 (by decrementing D and incrementing C_1.) Then Z' simulates Z on counters $C_1, ..., C_m$ (counter D is no longer active).

Let d be any positive integer. We modify Z' to another machine Z'' which uses, for each counter C_i, a buffer of size d in its finite control to simulate Z', and Z'' increments and decrements each counter modulo d. Z'' does not alter the action of Z' on counter D.

By choosing a large enough D, it follows that the computation of Z'' is such that when given input n in counter D and zeros in counters $C_1, ..., C_m$, the sum of the values of counters $D, C_1, ..., C_m$ at any time is at most n. It follows that, given a generalized linear-bounded multicounter, we can construct an equivalent linear-bounded multicounter machine.

The next theorem is similar to a result in [3] concerning BPS.

Theorem 1. *Let $L \subseteq o^*$. Then the following statements are equivalent:*

(1) L is accepted by a bounded S/A system.
(2) L is accepted by a linear-bounded multicounter machine,
(3) L is accepted by a $\log n$ space-bounded Turing machine.
(4) L is accepted by a multihead 2FA

These equivalences hold for both the deterministic and nondeterministic versions.

Proof. The equivalence of (3) and (4) is well known. By Lemmas 2 and 3, we need only show the equivalence of (2) and (4). That a linear-bounded multicounter machine can be simulated by a multihead 2FA is obvious. Thus (2) implies (4). We now show the converse. Let M be a two-way multihead FA M with m heads $H_1, ..., H_m$. From the discussion above, it is sufficient to construct a generalized multicounter machine Z equivalent to M. Z has $2m$+1 counters, $D, C_1, ..., C_m, E_1, ..., E_m$. Z with input n in counter D, and zero in the other counters first decrements D and stores n in counters $C_1, .., C_m$. Then Z simulates the actions of head H_i of M using the counters C_i and E_i. □

Lemmas 2 and 3 and Theorem 1 can be generalized to non-unary inputs, i.e., inputs of the form $a_1^{n_1}...a_k^{n_k}$, where $a_1, ..., a_k$ are distinct symbols. The constructions are straightforward generalizations of the ideas above. Thus, we have:

Corollary 1. *Let $L \subseteq a_1^*...a_k^*$. Then the following statements are equivalent:*

(1) L is accepted by a bounded S/A system.
(2) L is accepted by a linear-bounded multicounter machine,
(3) L is accepted by a $\log n$ space-bounded Turing machine.
(4) L is accepted by a multihead 2FA.

These equivalences hold for both the deterministic and nondeterministic versions.

We now proceed to show that the number of symbols in the alphabet V of a bounded S/A system induces an infinite hierarchy. This is an interesting contrast to a result in [14] that an unbounded S/A system with three objects is universal. The proof follows the ideas in [6], which showed an infinite hierarchy for a variant of SPCS, called RCPS.

We will need the following result from [8]:

Theorem 2. *For every k, there is a unary language L that can be accepted by a $(k+1)$-2FA but not by any k-2FA. The result holds for both deterministic and nondeterministic versions.*

Theorem 3. *For every r, there exist an $s > r$ and a unary language L (i.e., $L \subseteq o^*$) accepted by a bounded S/A system with an alphabet of s symbols that cannot be accepted by any bounded S/A system with an alphabet of r symbols. This result holds for both deterministic and nondeterministic versions.*

Proof. Suppose there is an r such that any unary language language accepted by any bounded S/A system with an arbitrary alphabet can be accepted by a bounded S/A system with an alphabet of r symbols. Let $k = r(r+1)$. From Theorem 2, there is a unary language L that can be accepted by a $(k+1)$-2FA but not by any k-2FA. By Theorem 1, this language can be accepted by a bounded S/A system. Then, by hypothesis, L can also be accepted by a bounded S/A system with an alphabet of r symbols. Then, from Lemma 3, we can construct from this bounded S/A system an $r(r+1)$-2FA accepting L. Hence, L can be accepted by a k-2FA, a contradiction. □

For our next result, we need the following theorem from [17].

Theorem 4. *Nondeterministic and deterministic multihead 2FAs over a unary input alphabet are equivalent if and only if nondeterministic and deterministic linear bounded automata (over an arbitrary input alphabet) are equivalent.*

From Theorems 1 and 4, we have:

Theorem 5. *Nondeterministic and deterministic bounded S/A systems over a unary input alphabet are equivalent if and only if nondeterministic and deterministic linear bounded automata (over an arbitrary input alphabet) are equivalent.*

3 Multi-membrane Special S/A Systems

Let M be a multi-membrane S/A system, which is restricted in that only rules of the form $(u, out; v, in)$, where $|u| = |v| \geq 1$, can appear in the skin membrane. There are no restrictions on the weights of the rules in the other membranes. Clearly, the number of objects in the system at any time during the computation remains the same. We denote by E_t the alphabet of t symbols (for some t) in the environment. There may be other symbols in the membranes that remain in the system during the computation and are not transported to/from the environment, and they are not part of E_t. Note that E_0 means that the environment alphabet is empty (i.e., there are no symbols in the environment at any time). As before, we consider the case where the input alphabet is unary (i.e. $\Sigma = \{o\}$). M's initial configuration contains o^n in the input membrane (for some n) and a *fixed* distribution of some *non-o* symbols in the membranes. The string o^n is accepted if the system eventually halts. We call the system just described a *special S/A system*.

Theorem 6. *Let $L \subseteq o^*$. Then the following statements are equivalent:*

(1) L is accepted by a multi-membrane special S/A system with **no** *symbols in the environment, i.e., has environment alphabet E_0 (= empty set).*

(2) L is accepted by a bounded S/A system.

(3) L is accepted by a linear-bounded multicounter machine.

(4) L is accepted by a $\log n$ space-bounded Turing machine.
(5) L is accepted by a multihead 2FA.

These equivalences hold for both the deterministic and nondeterministic versions.

Proof. As in Lemma 3, it is easy to show that a deterministic (nondeterministic) m-membrane special S/A system with no symbols in the environment can be simulated by a deterministic (nondeterministic) two-way FA with $2m$ heads.

By Theorem 1, to complete the proof, we need only show that a linear-space bounded multicounter machine can be simulated by a multi-membrane special S/A with no symbols in the environment. For notational convenience, we will assume the multicounter machine is controlled by a program with instructions of the type $l_i : (ADD(r), l_j)$, $l_i : (SUB(r), l_j, l_k)$, and $l_i : (HALT)$ where l_i is the label for the current instruction being executed and r is the counter which is either being incremented or decremented. If the current instruction is an add instruction, the next instruction to execute will be l_j. If the current instruction is a subtract instruction the next instruction depends on the value of r. If $r \neq 0$, the next instruction is denoted by l_j otherwise the next instruction is denoted by l_k.

The special S/A system simulating a linear-space bounded multicounter machine will use one membrane to simulate each counter of the multicounter machine. These membranes will be placed within a 'program' membrane where the current instruction is brought in, implemented, and then expelled. This entire system is enclosed within a dummy membrane (the skin membrane) containing no rules and a single copy of each instruction object along with a a few auxiliary objects. So the overall system uses $m + 2$ membranes. Obviously, if the skin membrane of the special S/A system contains no rules, no object can ever be brought into the system or expelled from the system. Hence, since the system initially contains $|wo^n|$ symbols, the system will continue to contain $|wo^n|$ symbols after each step of the computation.

To show how any linear-space bounded multicounter machine can be simulated, we give a formal transformation to a special S/A system. Our transformation is similar to the transformation in [14] except that our transformation yields a deterministic (nondeterministic) special S/A system if the original linear-space bounded multicounter machine is deterministic (nondeterministic). (The transformation in [14] only produces a nondeterministic S/A system.) The transformation is done as follows. Consider a multicounter machine Z with m counters. Construct a symport / antiport system M which simulates Z as follows:

$$M = (V, H, \mu, w_1, w_2, \cdots, w_{m+2}, E_0, R_1, R_2, \cdots, R_{m+2}, i_o)$$

where $H = \{1, 2, \cdots, m + 2\}$; $\mu = [_1[_2[_3]_3[_4]_4 \cdots [_{m+2}]_{m+2}]_2]_1$; $w_1 =$ one copy of each element in V except o and l_{01} (we assume Z's program begins with the instruction l_0); $w_2 = l_{01}$; $w_3 = o^n$; $w_i = \lambda$, for all $i = 4, \cdots, m + 2$; $E_0 = \emptyset$ (the environment, E_t, is empty because $t = 0$); No need to specify i_0, since our system is an acceptor.

The elements of V are as follows:

1. o — The symbol o is used as the counting object for the system. The multiplicity of o's in each counter membrane signifies the count of that counter.

2. $d_1, d_2, d_3, d_4, d_5, d_6$ — These objects are used to delay various objects from being used for a number of steps. The objects d_1 and d_2 are used to delay an action for 1 step. The remaining objects are used to delay an action for 3 steps.
3. c_1, c_2, c_3 — These objects are called check objects and are used to guarantee a subtract instruction expels at most one o object from the appropriate counter membrane.
4. l_{i1}, l_{i2} for each instruction $l_i : (ADD(r), l_j)$.
   The object l_{i1} signifies that the next instruction we will execute is l_i. The object l_{i2} is used in executing instruction l_i.
5. $l_{i1}, l_{i2}, l_{i3}, l_{i4}$ for each instruction $l_i : (SUB(r), l_j, l_k)$.
   The object l_{i1} signifies that the next instruction we will execute is l_i. The objects l_{i2}, l_{i3}, and l_{i4} are used in executing instruction l_i and are used to signify which branch of l_i will determine the next instruction.
6. l_{i1} for each instruction $l_i : (HALT)$.

The sets of rules for the R_i's are created as follows:

1. The set $R_1 = \emptyset$.
2. The set R_2 contains the following delay rules: $(d_1, out; d_2, in)$; $(d_3, out; d_4, in)$; $(d_4, out; d_5, in)$;
   $(d_5, out; d_6, in)$.
3. For each instruction $l_i : (ADD(r), l_j)$ in Z:
   The set R_2 contains the following rules: $(l_{i1}, out; l_{i2}d_1o, in)$; $(l_{i2}d_2, out; l_{j1}, in)$.
   The set R_{r+2} contains the following rules: $(l_{i2}o, in)$; (l_{i2}, out).
4. For each instruction $l_i : (SUB(r), l_j, l_k)$ in Z:
   The set R_2 contains the following rules: $(l_{i1}, out; l_{i2}c_1d_3, in)$; $(l_{i2}o, out; c_2l_{i3}, in)$; $(c_1c_2d_6l_{i3}, out; l_{j1}, in)$; $(l_{i2}d_6, out; c_3l_{i4}, in)$; $(c_1c_3l_{i4}, out; l_{k1}, in)$.
   The set R_{r+2} contains the following rules: $(l_{i2}c_1, in)$; $(l_{i2}o, out)$; $(c_1, out; c_2, in)$; (c_2, out);
   $(l_{i2}, out; d_6, in)$; (d_6, out); $(c_1, out; c_3, in)$; (c_3, out).
5. For each instruction $l_i : (HALT)$ no rules are added.

Informally, these special S/A system rules work using the following ideas. Initially the system is started with the first instruction label object l_{01} in the program membrane and the input o^n within membrane 3 (corresponding to counter 1). To execute an add instruction, the initial instruction object is replaced with the objects needed to execute the instruction - l_{i2}, d_1, and o. If the instruction is a subtract instruction the instruction l_{i1} is replaced with l_{i2} along with a delay object d_3 and a check object c_1. Once the appropriate objects are in the program membrane, a o object is appropriately moved into or out of the counter membrane corresponding to the current instruction. In the case where the current instruction tries to decrement a zero counter, the check objects cooperate with the delay objects to detect the situation and bring the appropriate objects into and out of the active membranes. Finally, the instruction executing objects are expelled from the program membrane and the correct next instruction object is brought into the program membrane.

Note that when a counter is decremented, an o object is removed from the corresponding membrane and moved into the skin membrane. When a counter is incremented, an o

object is brought into the corresponding membrane from the skin membrane. Since the multicounter machine being simulated is, by definition, guaranteed to always decrement before incrementing, we are guaranteed to have thrown an o object into membrane 1 before we ever try bringing an o object from membrane 1 to membrane 2. This guarantees that the special S/A system will operate through the multicounter machine's program instructions correctly. □

Corollary 2. *Let t be any positive integer. Then multi-membrane special S/A systems with an environment alphabet of t symbols are equivalent to multi-membrane special S/A systems with no symbols in the environment. This holds for deterministic and nondeterministic versions.*

Proof. This follows from the above theorem and the observation that a system with an environment of t symbols can be simulated by a two-way FA with $2m(t+1)$ heads. □

The proof of the next result is similar to that of Theorem 3.

Theorem 7. *For every r, there exist an $s > r$ and a unary language L (i.e., subset of o^*) accepted by an s-membrane special S/A system that cannot be accepted by any r-membrane special S/A system. This result holds for both deterministic and nondeterministic versions.*

4 One-Membrane Bounded S/A Systems Accepting String Languages

Let M be a (one-membrane) S/A system with alphabet V and input alphabet $\Sigma \subseteq V$. We assume that Σ contains a distinguished symbol \$, called the (right) end marker. The rules are restricted to be of the form:

1. $(u, out; v, in)$,
2. $(u, out; vc, in)$

where both u and v are in V^+, $|u| = |v| \geq 1$, and c is in Σ. Note that because of the requirement that $|u| = |v|$, the only way that the number of symbols in the membrane can grow is when a rule of type 2 is used. The second type of of rule is called a *read-rule*. We call M a *bounded S/A acceptor*. There is an abundance of symbols from V in the environment. The symbol c in a rule of type 2 can only come from the input string $z = a_1...a_n$ (where a_i is in $\Sigma - \{\$\}$ for $1 \leq i < n$, and $a_n = \$$), which is provided online externally; none of the symbols in v in the rules come from z.

There is a fixed string w in $(V - \Sigma)^*$, which is the initial configuration of M. 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. In particular, the number of instances of read-rules (i.e., rules of the form $(u, out; vc, in)$) applicable in a step is unbounded. However, if a step calls for reading k input symbols (for some k), these symbols must be consistent with the next k symbols of the input string z that have not yet been processed. Note that rules of type 1 do not consume any input symbol from z.

The input string $z = a_1...a_n$ (with $a_n = \$$) is accepted if, after reading all the input symbols, M eventually halts. The language accepted is $L(M) = \{a_1...a_{n-1} \mid a_1...a_n$ is accepted by $M\}$ (we do not include the end marker).

We have two versions of the system described above: deterministic and nondeterministic bounded S/A acceptors. Again, in the deterministic case, the maximally parallel multiset of rules applicable at each step of the computation is unique. We will show that the deterministic version is strictly weaker than the nondeterministic version. The proof uses some recent results in [7] concerning a simple model of a CPS, called SCPA.

An SCPA M 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 (similar to those of a CPS) are of the form:

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

The input to M is a string $z = a_1...a_n$ (with $a_n = \$$, the end marker), which is provided externally online. The restrictions on the operation of M 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 c if the next symbol in the input string $z = a_1...a_n$ that has not yet been processed (read) is c ; 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 $ab \rightarrow a_x b_x c_{come}$) applicable in a step is unbounded. However, if a step calls for reading k input symbols (for some k), these symbols must be consistent with the next k symbols of the input string z that have not yet been processed (by the semantics of the read-rule described in the previous item).

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

A *restricted 1-way linear-space DCM (NCM)* M is a deterministic (nondeterministic) finite automaton with a one-way read-only input tape with right delimiter (end marker) \$ and a number of counters. As usual, each counter can be tested for zero and can be incremented/decremented by 1 or unchanged. The counters are restricted in that there is a positive integer c such that at any time during the computation, the amount of space used in any counter (i.e., the count) is at most ck, where k is the number of symbols of the input that have been read so far. Note that the machine need not read an input symbol at every step. An input $w = a_1...a_n$ (where a_n is the end marker, \$, which only occurs at the end) is accepted if, when M is started in its initial state with all counters zero, it eventually enters an accepting state while on \$.

We note that although the machines are restricted, they can accept fairly complex languages. For example, $\{a^n b^n c^n \mid n \geq 1\}$ and $\{a^{2^n} \mid n \geq 0\}$ can both be accepted by

restricted 1-way linear-space DCMs. (We usually do not include the end marker, which is part of the input, when we talk about strings/languages accepted.) It can be shown that a restricted 1-way linear-space DCM (NCM) is equivalent to a restricted 1-way $log\ n$-space deterministic (nondeterministic) Turing machine that was studied in [2].

We will need the following result that was recently shown in [7]:

Theorem 8. *A language L is accepted by a restricted 1-way linear-space DCM (NCM) if and only if it is accepted by a deterministic SCPA (nondeterministic SCPA).*

Theorem 9. *Deterministic (nondeterministic) bounded S/A acceptors are equivalent to deterministic (nondeterministic) SCPAs.*

Proof. First we show that a deterministic (nondeterministic) SCPA M can be simulated by a deterministic (nondeterministic) bounded S/A acceptor M', which has only one membrane. Suppose M has membranes $1, ..., m$, with index 1 representing the skin membrane. For every symbol a in the system and membrane i, create a new symbol a_i. We construct M' by converting the rules to one-membrane rules as described in the proof of Lemma 2, except that now we have to handle rules of the form $ab \rightarrow a_x b_y c_{come}$ in membrane 1. We transform such a rule to $(a_1 b_1, out; a_j b_k c_1, in)$, where j and k are the indices of the membranes into which a and b are transported to, as specified by x and y. After we have constructed M', modify it slightly by deleting the subscripts of all symbols with subscript 1 (in the rules and initial configuration). Thus unsubscripted symbols are associated with symbols in membrane 1 of the SCPA M.

For the converse, we need only show (by Theorem 8) that a deterministic (nondeterministic) bounded S/A acceptor M can be simulated by a restricted 1-way linear-space DCM (NCM) Z. The construction of Z is like in Lemma 3, except that now, Z uses counters (instead of heads), and in the maximally parallel step, the read-rules are the first ones to be processed. Define an atomic read-rule process as follows: Z systematically cycles through the read-rules and finds (if it exists) the first one that is applicable (note that for a read-rule $(u, out; vc, in)$ to be applicable, the next input symbol that has yet to be processed must be c). Z applies a sequence of these read-rules until no more read-rule is applicable. Then all the other rules are processed. We omit the details. If M is a nondeterministic SCPA, the construction of a nondeterministic Z is similar, in fact, easier. □

From Theorem 8 and the fact that deterministic SCPAs are strictly weaker than nondeterministic SCPAs [7], we have:

Theorem 10. *Deterministic bounded S/A acceptors are strictly weaker than nondeterministic bounded S/A acceptors.*

Let $L = \{x\#^p \mid x \text{ is a binary number with leading bit 1 and } p \neq 2val(x)\}$, where $val(x)$ is the value of x. It was shown in [7] that L can be accepted by a nondeterministic SCPA but not by any deterministic SCPA. Hence, L is an example of a language that can be accepted by a nondeterministic bounded S/A acceptor that cannot be accepted by any deterministic bounded S/A acceptor.

The following follows from Theorem 9 and the fact that similar results hold for SCPAs [7].

Theorem 11. *Let NBSA (DBSA) be the class of languages accepted by nondeterministic (deterministic) bounded S/A acceptors. Then:*

1. *NBSA is closed under union and intersection but not under complementation.*
2. *DBSA is closed under union, intersection, and complementation.*

References

1. C. S. Calude and Gh. Paun. *Computing with Cells and Atoms: After Five Years (new text added to Russian edition of the book with the same title first published by Taylor and Francis Publishers, London, 2001)*. To be published by Pushchino Publishing House, 2004.
2. E. Csuhaj-Varju, O. H. Ibarra, and G. Vaszil. On the computational complexity of P automata. In *Proc. DNA 10* (C. Ferretti, G. Mauri, C. Zandron, eds.), Univ. Milano-Bicocca, 97–106,2004.
3. Z. Dang, O. H. Ibarra, C. Li, and G. Xie. On model-checking of P systems. *Proc. 4th International Conference on Unconventional Computation*, to appear, 2005.
4. R. Freund, L. Kari, M. Oswald, and P. Sosik. Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330(2): 251–266, 2005.
5. R. Freund and Gh. Paun. On deterministic P systems. See P Systems Web Page at http://psystems.disco.unimib.it, 2003.
6. O. H. Ibarra. The number of membranes matters. In *Proc. 4th Workshop on Membrane Computing*, Lecture Notes in Computer Science 2933, Springer-Verlag, 218-231, 2004. Journal version to appear in *Theoretical Computer Science*, 2005.
7. O. H. Ibarra. On determinism versus nondeterminism in P systems. Theoretical Computer Science, to appear, 2005.
8. B. Monien, Two-way multihead automata over a one-letter alphabet, *RAIRO Informatique theorique*, 14(1):67–82, 1980.
9. A. Paun and Gh. Paun. The power of communication: P systems with symport/antiport. *New Generation Computing* 20(3): 295–306, 2002.
10. Gh. Paun. Computing with membranes. *Turku University Computer Science Research Report No. 208*, 1998.
11. Gh. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
12. Gh. Paun. *Membrane Computing: An Introduction*. Springer-Verlag, 2002.
13. Gh. Paun. Further twenty six open problems in membrane computing. See P Systems Web Page at http://psystems.disco.unimib.it, 1-12, January 20, 2005.
14. Gh. Paun, J. Pazos, M. J. Perez-Jimenez, and A. Rodriguez-Paton. Symport/antiport P systems with three objects are universal. *Fundamenta Informaticae*, 64(1-4): 353–367, 2005.
15. Gh. Paun and G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287(1):73–100, 2002.
16. W. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2): 177–192, 1970.
17. W. Savitch. A note on multihead automata and context-sensitive languages. *Acta Informatica*, 2:249–252, 1973.
18. 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.