On Determinism Versus Nondeterminism in P Systems [1]

Oscar H. Ibarra

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

Abstract

An important open problem in the area of membrane computing is whether there is a model of P systems for which the nondeterministic version is strictly more powerful than the deterministic version. We resolve this problem in the following sense — we exhibit two classes of P system acceptors with only communicating rules and show:

1. For the first class, the deterministic and nondeterministic versions are equivalent if and only if deterministic and nondeterministic linear bounded automata (LBA) are equivalent. The latter problem is a long-standing open question in complexity theory.
2. For the second class, the deterministic version is strictly weaker than the nondeterministic version.

Both classes are nonuniversal, but can accept fairly complex languages.

Keywords: determinism versus nondeterminism, molecular computing, P system, communicating P system, counter machine, two-way multihead finite automaton.

1 Introduction

There has been a great deal of research activities in the area of membrane computing (a branch of molecular computing) initiated by Gheorghe Paun six years ago in his seminal paper [7] (see also [8]). Membrane computing identifies an unconventional computing model, namely a P system, from natural phenomena of cell evolutions and chemical reactions. 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 that would allow us to solve currently intractable problems (in much the same way as the promise of quantum and DNA computing) once future bio-technology (or silicon-technology) gives way to a practical bio-realization (or chip-realization).

A P system is a computing model, which abstracts from the way the living cells process chemical compounds in their compartmental structure. Thus, regions defined by a membrane structure contain objects that evolve according to given rules. The objects can be described by symbols or by strings of symbols, in such a way that multisets of objects are placed in regions of the membrane structure. The membranes themselves are organized as a Venn diagram or

[1]This research was supported in part by NSF Grants CCR-0208595 and CCF-0430945.

a tree structure where one membrane may contain other membranes. By using the rules in a nondeterministic, maximally parallel manner, transitions between the system configurations can be obtained. A sequence of transitions shows how the system is evolving. Various ways of controlling the transfer of objects from a region to another and applying the rules, as well as possibilities to dissolve, divide or create membranes have been studied. P systems were introduced with the goal to abstract a new computing model from the structure and the functioning of the living cell (as a branch of the general effort of Natural Computing – to explore new models, ideas, paradigms from the way nature computes). Membrane computing has been quite successful: many models have been introduced, most of them Turing complete and/or able to solve computationally intractable problems (NP-complete, PSPACE-complete) in a feasible time (polynomial), by trading space for time. (See the P system website at http://psystems.disco.unimib/it for a large collection of papers in the area, and in particular the monograph [9].)

In the standard semantics of P systems [8, 9, 10], each evolution step of a system G is a result of applying all the rules in G in a maximally parallel manner. More precisely, starting from the initial configuration, w, 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 multiset of rules, which are chosen nondeterministically. For example, a catalytic rule $Ca \to Cv$ in membrane m is applicable if there is a catalyst C and an object (symbol) a in the preceding configuration in membrane m. The result of applying this rule is the evolution of v from a. 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. Thus, in general, the number of times a particular rule is applied at anyone step can be unbounded. We require that the application of the rules is maximal: all objects, from all membranes, which *can be* the subject of local evolution rules *have to* evolve simultaneously. Configuration z is reachable (from the starting configuration) if it appears in some execution sequence; z is halting if no rule is applicable on z.

An interesting class of P systems with symport/antiport rules was studied in [4] – each system is *deterministic* in the sense that the computation path of the system is unique, i.e., at each step of the computation, the maximal multiset of rules that is applicable is unique. It was shown in [4] that any recursively enumerable unary language $L \subseteq o^*$ can be accepted by a deterministic 1-membrane symport/antiport system. Thus, for symport/antiport systems, the deterministic and nondeterministic versions are equivalent. It also follows from the construction in [13] that for communicating P systems, the deterministic and nondeterministic versions are equivalent as both can accept any unary recursively enumerable language. The deterministic-versus-nondeterministic question was left open in [4] for the class of catalytic systems, where the proofs of universality involve a high degree of parallelism [13, 3]. In particular, it was an open problem [1] whether there is a class of (universal or nonuniversal) P systems where the nondeterministic (maximally parallel) version is strictly more powerful than the deterministic version.

In this paper, we look at two classes of nonuniversal models of P systems. For one class, we show that the deterministic and nondeterministic versions are equivalent if and only if deterministic and nondeterministic linear bounded automata (LBA) are equivalent. The latter problem is a long-standing open question in complexity theory. While it is known that a nondeterministic LBA (which is equivalent to a nondeterministic n space-bounded Turing machine) can be simulated by a deterministic n^2 space-bounded Turing machine [11], it is open whether this result is optimal. Thus, for this class of P systems, the question of whether or not the deterministic version is strictly weaker than the nondeterministic version reduces to an unresolved fundamental problem in computational complexity. For another class of P systems, we can actually show that the deterministic version is strictly weaker than the nondeterministic version.

The paper has three sections in addition to this section. Section 2 looks at a model of P systems for which the question of whether the deterministic version is strictly less powerful than the nondeterministic version is equivalent to the long-standing open problem of whether a deterministic LBA (linear-bounded automaton) is strictly less powerful than a nondeterministic LBA. Section 3 studies another model of P systems for which the deterministic version is strictly less powerful than the nondeterministic version. Section 4 is a brief conclusion.

2 A Model For Which The Question Is Equivalent To Whether Deterministic LBA = Nondeterministic LBA

The model we investigate is a restricted version of the communicating P system (CPS). A CPS, first introduced and studied in [13], has multiple membranes labeled $1, 2, ...$, where 1 is the skin membrane. The rules 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.

An RCPS [5] is a restricted CPS where the environment does not contain any object initially. The system can expel objects into the environment but only expelled objects can be retrieved from the environment. Hence, at any time during the computation, the objects in the system (including in the environment) are always the same.

Let o be a distinguished object (called the *input symbol*) in V. Assume that an RCPS G has m membranes, with a distinguished *input membrane*. We say that G accepts o^n if G, 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. Also, at any time during the computation, the number of each object $a \in V - \{o\}$ in the whole system (including the environment) remains the same, although the distribution of the a's among the membranes may change at each step. The language accepted by G is $L(G) = \{o^n \mid o^n \text{ is accepted by } G\}$.

A nondeterministic (deterministic) RCPS is one in which there may be more than one (at most one) maximally parallel multiset of rules that is applicable at each step.

It turns out that an RCPS can be characterized in terms of a two-way multihead finite automaton. The latter is a finite automaton (FA) with k two-way read-only heads (for some

k) operating on an input with left and right end markers. Here, we are only interested in two-way multihead FAs with a unary input alphabet. Thus, the input to the machine (exclusive of the end markers) is o^n for some n.

The string o^n is accepted if the automaton, when given input o^n (with left and right end markers) has a halting computation, i.e., a sequence of moves that eventually halts. Note that when the machine is deterministic, the sequence of moves is unique – either it is halting (accepting) or it is in an infinite loop (nonaccepting).

Convention: It is easy to verify that our definition of acceptance above, i.e., the existence of a halting computation is equivalent to acceptance by accepting state. We shall use the halting mode of acceptance for all machines discussed in the paper.

Theorem 2.1 *A unary language $L \subseteq 0^*$ is accepted by a deterministic (nondeterministic) RCPS if and only if it is accepted by a deterministic (nondeterministic) two-way multihead FA.*

Proof. Suppose G is a deterministic RCPS accepting a language $L(G) \subseteq o^*$. Let m be the number of membranes in G (for notational convenience, we also consider the environment as a membrane). We construct a deterministic two-way multihead FA M to accept $L(G)$. Since all objects other than o are fixed and do not change with the input, the FA's finite control can keep track of their locations in the system during the computation. However, the number of o's in the system is unbounded and dependent on the input, so we will use the multiple heads in M to keep track of the location of the o's in the system. We will need the following head to accomplish this:

- K_i for $1 \leq i \leq m$. Head K_i will keep track of the current number of o's in membrane i. Initially, if i' is the input membrane, $K_{i'}$ will point to the nth o while all other K_i heads will point to the left end marker of the input.

- $K_{i,j}$ for $1 \leq i, j \leq m$. These heads keep track of where the o's will be moving during the next step.

To determine the next configuration of G, the unique maximal multiset of applicable rules in each membrane must be found for all membranes and then these must be applied simultaneously. So one (parallel) step of G is simulated using two sequences of steps by M. The initial sequence of steps marks the next set of applicable rules and the second sequence of steps applies these rules. Initially M looks at each membrane individually. Hence, if we let V be the alphabet of the RCPS G and i be the current membrane whose rules are under consideration, we can use the following sequence of steps to determine the maximal set of applicable rules in membrane i:

1. Determine how many instances of rules involving the object o can be applied in membrane i during the next step. There are three types of rules which can cause the o elements in a membrane to evolve. These rules are $o \to o_x$, $oo \to o_x o_y$, and $ao \to a_x o_y$ where $x, y \in \{here, out, come\}$ and $a \in V - o$. It can be guaranteed that at most one of these rule types is located in each membrane in order to preserve the deterministic property of the computation. In fact, at most one occurrence of each of these rules can appear in a given membrane (except case 3 which can have multiple occurrences in the case $ao \to a_x o_y; bo \to b_x o_y$ such that $a \neq b$ and a, b are never both in the membrane during the same step).

Case 1. The current membrane contains a rule of form $o \to o_x$:
This rule is applied to all o's in membrane i using two-way heads to keep track of where these zeros will be after the next step. This is done by repetitively moving head K_i left one cell and head $K_{i,x}$ head right one cell (note that x is the membrane o will move to during the next step) until K_i is on the left end marker.

Case 2. Current membrane contains a rule of form $oo \to o_x o_y$:

- **Case 2a.** If the number of o's in membrane i is even, this rule is applied to all the o's in membrane i using two-way heads to keep track of where these zeros will be after the next step. This is done by repetitively moving K_i left two cells, $K_{i,x}$ right one cell, and $K_{i,y}$ head right one cell until K_i is on the left end marker.
- **Case 2b.** If the number of o's in membrane i is odd, this rule is applied to all but one of the o's in membrane i as in case 2a. In this case, K_i will move left by two until it points to the first o symbol.

Case 3. The current membrane contains a rule of form $ao \to a_x o_y$ where $a \in V - \{o\}$: This rule is applied as many times as possible based on the number of a's in i. This number is bounded by the number of a's in the initial configuration which is independent of the input. Hence, this rule is applied using the finite control.

Case 4. The current membrane contains both rules of form $oo \to o_x o_y$ and $ao \to a_x o_y$: These rules can only be in the same membrane under the following two circumstances. Any other circumstance would cause the system to become nondeterministic.

- **Case 4a.** The current membrane contains an even number of o's and no a's: In this situation, the rule $ao \to a_x o_y$ cannot be applied, so this is the same as case 2a.
- **Case 4b.** The current membrane contains an odd number of o's and at most one a.: This case uses the rule $ao \to a_x o_y$ once and follows case 2a.

2. Once the applicable rules with o symbols have been determined, the applicable rules with non-o symbols can be determined. These rules can be determined in the finite control.

After the maximal applicable multiset of rules has been found, they must be applied. This is done by setting all $K_{i,j}$ heads to zero, and updating the K_j heads accordingly. (For each $K_{i,j}$, repetitively move $K_{i,j}$ one cell left and K_j one cell right until $K_{i,j}$ is on the left end marker.) The transfer of non-o objects occurs in the finite control.

Clearly, the above procedure simulates a multi-membrane RCPS and is deterministic at each step. Hence M is a deterministic two-way multihead FA.

If G is nondeterministic, the construction of the nondeterministic two-way multihead FA M is easier, since M can just guess and apply a multiset of rules and verify that it is maximal.

The converse was essentially shown in [5]. A careful examination of the proof in [5] shows that it holds for both deterministic and nondeterministic cases. (Actually the proof in [5] was indirect, using restricted counter machines that are equivalent to multihead FAs). We omit the details. □

Theorem 2.1 can be generalized. Let $a_1, ..., a_k$ be distinct symbols. Then we have:

Corollary 2.1 *A language $L \subseteq a_1^* ... a_k^*$ is accepted by a deterministic (nondeterministic) RCPS if and only if it is accepted by a deterministic (nondeterministic) two-way multihead FA.*

The next theorem was shown in [12].

Theorem 2.2 *Nondeterministic and deterministic two-way multihead FA 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 2.1 and 2.2, we have:

Theorem 2.3 *Every unary language accepted by a nondeterministic RCPS can be accepted by a deterministic RCPS if and only if every language (over an arbitrary input alphabet) accepted by a nondeterministic linear bounded automaton can be accepted by a deterministic linear bounded automaton.*

3 A Model For Which Deterministic Is Strictly Weaker Than Nondeterministic

We begin with the definition of a restricted multicounter machine. 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 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 will not include the end marker, which is part of the input, when we talk about strings/languages accepted.)

We denote by $\mathcal{L}(restricted - 1DCM)$ ($\mathcal{L}(restricted - 1NCM)$) the class of languages accepted by restricted 1-way linear-space DCM (NCM).

Let $L = \{x\#^p \mid x$ is a binary number with leading bit 1 and $p = val(x)\}$,

where $val(x)$ is the value of the binary number x. L is similar to the language that was shown in [2] to be not acceptable by a special kind of Turing machine, called restricted 1-way linear-space nondeterministic Turing machine. The proofs of the next two lemmas use ideas in [2].

Lemma 3.1 *L is not in $\mathcal{L}(restricted - 1NCM)$.*

Proof. To see that L is not in $\mathcal{L}(restricted - 1NCM)$, suppose M is a restricted 1-way linear-space NCM accepting L. Note that M has a finite number of counters and, by definition, there is a constant c such that at any time during the computation, the amount of space used in any counter is at most ck, where k is the number of symbols of the input that has been read so far. It follows that the number of possible configurations (a configuration is a tuple consisting of the state and the values of the counters) after reading k symbols is at most k^e for some positive integer e. Now for any given k, consider the set of all strings of the form: $w\#^{val(w)}$, where w

has leading bit 1 and $|w| = k$. Clearly, $k \leq g \cdot \log n$ for some constant g, where $n = |w\#^{val(w)}|$. Since the machine is restricted 1-way linear-space, there exist an n (and therefore k) big enough and two inputs $w\#^{val(w)}$ and $w'\#^{val(w')}$ where w, w' have leading bit 1, $|w| = |w'| = k$, and $w \neq w'$ for which M will be in the same configuration after reading w and after reading w'. This is because M can be in one of at most $k^e \leq (g \cdot \log n)^e$ configurations after reading a string of length k. Now there are $2^{k-1} \geq h \cdot n$ strings w (with leading bit 1) whose lengths are k for some positive constant h. If we choose n big enough, $(g \cdot \log n)^e < h \cdot n$. It follows that the machine when given the string $w'\#^{val(w)}$ will also accept. This is a contradiction since $w'\#^{val(w)}$ is not in L. $\square$

Lemma 3.2 *$\mathcal{L}$(restricted$-1NCM$) is closed under union, intersection, concatenation, Kleene * and $^+$, inverse homomorphism, and ε-free homomorphism. It is not closed under complementation, (unrestricted) homomorphism, and reversal.*

Proof. It is straightforward to verify that $\mathcal{L}(restricted - 1NCM)$ is closed under union, intersection, concatenation, Kleene * and $^+$, inverse homomorphism, and ε-free homomorphism.

To see that $\mathcal{L}(restricted - 1NCM)$ is not closed under complementation, consider $\bar{L}$, the complement of L. $\bar{L}$ can be accepted by a restricted 1-way linear-space NCM M' as follows. Inputs that are not in $1(0+1)^*\#^+$ can easily be accepted by M'. For an input of the form $x\#^p$, M' needs to check that $val(x) \neq p$. To do this, M' scans the segment x from left to right. At some point, nondeterministically chosen, when M' is in some position i of x, M' records the symbol, say a, under the input head (i.e., the i-th symbol) and starts incrementing a counter C_1 as it scans the rest of x. When M' reaches the first $\#$ to the right of x, C_1 has value $k - i$, where $k = |x|$. M' then scans the segment $\#^p$ and stores p in another counter C_2. When M' reaches the right end marker, C_2 has value p. M' then divides p by $2^{(k-i)}$ to determine the $(k-i)$-th least significant bit, say b, of the binary representation of p. Clearly, determining b can easily be accomplished by M' by using C_1 and C_2 and another counter C_3. M' accepts if $b \neq a$. Clearly M' accepts $\bar{L}$. Since L is not in $\mathcal{L}(restricted - 1NCM)$, it follows that $\mathcal{L}(restricted - 1NCM)$ is not closed under complementation.

Now let $L' = \{\alpha^p x\#^p \mid x$ is a binary number with leading bit 1 and $p = val(x)\}$ (where α is a new symbol). Like in the construction above, it is easy to show that L' is in $\mathcal{L}(restricted - 1NCM)$. However, erasing the α's produces L from L'. Hence, $\mathcal{L}(restricted - 1NCM)$ is not closed under unrestricted homomorphism. It is also not closed under reversal since the reverse of L is in $\mathcal{L}(restricted - 1NCM)$, but as we have seen, L is not. $\square$

The next lemma concerns $\mathcal{L}(restricted - 1DCM)$.

Lemma 3.3 *$\mathcal{L}$(restricted$-1DCM$) is closed under union, intersection, and complementation.*

Proof. Closure under union and intersection is obvious. To see closure under complementation, let M be a restricted 1-way linear-space DCM. We construct a restricted 1-way linear-space DCM M' to accept the complement of $L(M)$. M' simulates M and accepts if and only if M rejects. In the simulation, M' needs to keep track of the number of steps M has made since it last read an input symbol but before reading the next symbol to make sure M is not looping. Clearly, since M has only a finite number of counters, there exists a positive integer e such that if M has read k symbols and does not read the next input symbol after k^e steps, then it is in an infinite loop. Thus, we equip M' with additional counters (i.e., a clock) to keep track of the number of steps between readings of input symbols. When the clock overflows, M is in an infinite loop, and M' rejects the input. $\square$

From Lemmas 3.1-3.3, we have:

Lemma 3.4 *A restricted 1-way linear-space DCM is strictly weaker than a restricted 1-way linear-space NCM.*

Corollary 3.1 *Let* $L = \{x\#^p \mid x \text{ is a binary number with leading bit 1 and } p = val(x)\}$. *Then* $\bar{L}$ *(the complement of L) cannot be accepted by a restricted 1-way linear-space DCM.*

Now by definition, in a restricted 1-way linear-space DCM (NCM) M, there is a positive integer c such that at any time during the computation, the amount of space used in any counter is at most ck, where k is the number of symbols of the input that has been read so far. We refer to M as a ck-DCM (NCM). Similarly, we use the notation k/c-DCM (NCM) for the case when the space used in each counter after reading k symbols is at most k/c (thus, the space used is smaller than k).

Let M be a restricted 1-way linear-space DCM (NCM). For any positive integer d, we can effectively construct a k/d-DCM (NCM) M' equivalent to M. The construction is straightforward. If M is a ck-DCM, M' uses, for each counter, a buffer of size cd in its finite control to simulate M and it increments/decrements each counter modulo cd.

It follows that any restricted 1-way linear-space DCM(NCM) can be converted to one in which at any time, the sum of the spaces (counts) used in *all* counters is at most k, where k is the number of symbols that has been read so far. Without loss of generality (by adding a dummy counter), we may assume that every time a new symbol is read, the machine increments exactly one counter. Thus, at any time, the sum of the values of the counters is exactly the number of symbols that have been read so far. We also assume that exactly one counter is updated (increment or decremented) at each step, because all (nonreading) steps which only change the state and no counter can be removed. We shall refer to this kind of machine as a special 1-way linear-space DCM (NCM). Hence, we have:

Lemma 3.5 *A language is accepted by a special 1-way linear-space DCM (NCM) if and only if it is accepted by a restricted 1-way linear-space DCM (NCM).*

3.1 The model

We now define another restricted model of a P system, called SCPA, whose deterministic and nondeterministic versions are equivalent to restricted 1-way linear-space DCM and NCM, respectively. An SCPA is a language acceptor over an input alphabet Σ containing a distinguished symbol \$ (the right end marker for the input). An input to the SCPA is a string $a_1...a_n$, where $a_1, ..., a_{n-1}$ are in $\Sigma - \{\$\}$ and $a_n = \$$. We impose the following conditions on the system:

1. The system has at least one membrane: the skin membrane labeled 1. There may be other membranes contained in membrane 1.

2. The symbols in the initial configuration (distributed in the membranes) are *not* from Σ.

3. No symbols are expelled into the environment.

4. The rules (similar to those of a CPS) are of the form:

   (a) $a \to a_x$

(b) $ab \to a_x b_y$

(c) $ab \to a_x b_y c_{come}$

The restrictions are the following:

As before, 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 $w = a_1...a_n$ that has not yet been processed (read) is c ; otherwise, the rule is not applicable. Also, there are no rules in membrane 1 with c_{out} on the right-hand side of the rule for any symbol c (i.e., no symbol can be expelled from membrane 1 into the environment). It follows that the at any time after reading the j-th symbol of the input string but before reading the $j+1$-st symbol, the system will have exactly j symbols from Σ.

(d) Maximal parallelism in the application of the rules is assumed as usual. However, we assume that the parallelism is bounded, i.e., there exists a k (independent of the computation) such that at each step, the size of the maximally parallel multiset of rules applicable is at most k. In particular, it follows that j ($\leq k$) symbols can be imported into the skin membrane from the environment at each step, and the choice of these symbols must be consistent with the next j symbols of the input string that have not yet been processed (by the semantics of the read-rule described in the previous item).

(e) The input string $w = a_1...a_n$ (note that a_n is the right end marker \$) is accepted if, after reading all the input symbols, the SCPA eventually halts.

The language accepted by G is $L(G) = \{a_1...a_{n-1} \mid a_1...a_n \text{ is accepted by } G\}$ (we do not include the end marker).

We have two versions of the system described above: deterministic SCPA and nondeterministic SCPA. Again, in the deterministic case, the maximally parallel multiset of rules applicable at each step of the computation is unique.

Before we state the next result, we note again that in an SCPA, at every step, maximal parallelism is applied in the usual sense (i.e., a multiset of rules can be applied – thus the same rule can be applied many times), except that we limit the size of maximal parallelism to at most k (for some k independent of the computation). This requirement is not really crucial, since in many constructions involving maximal parallelism in the literature, the size of a maximally parallel multiset of rules applied at each step is usually at most a small number, like 2 or 3.

Note: Later, we will consider the general case where maximal parallelism in the model is not bounded.

Lemma 3.6 *Let L be a language accepted by deterministic SCPA (nondeterministic SCPA) G. Then we can effectively construct a restricted 1-way linear-space DCM (NCM) M accepting L.*

Proof. The construction of a restricted 1-way linear-space DCM (NCM) M from a deterministic (nondeterministic) SCPA G is straightforward and uses the ideas in the proof Theorem 2.1. (There, "multiple heads" were used, but clearly, counters can be used.) Note that because the size of a maximally parallel multiset of rules that can be applied at each step is limited to at most a fixed number k, M in the case of deterministic SCPA, can systematically try all possible multiset of rules of size at most k to determine the unique multiset of rules to apply. Note also that at most k read-rules can be involved in the step. Hence, M need only read (i.e., look ahead at) the next k symbols in determining the maximally parallel multiset of rules to apply. If the number of read-rules involved in the step is $j < k$, M uses only the first j symbols and records the remaining $k - j$ symbols in its finite control for the simulation of the next multiset of rules. Note that if the SCPA is nondeterministic, M will also be nondeterministic. In fact, the construction of M is easier, since M can just guess and apply a multiset of rules and verify that it is maximal. Note also that because M is making the selection nondeterministically, the assumption that a maximally parallel multiset of rules must be bounded is *no longer needed.*

□

For the converse, it is convenient to define a new accepting device M that is able to simulate a special 1-way linear-space DCM (NCM). A special 1-way linear-space DSUM (NUSM) has a one-way (read-only) input with end marker and a finite number of storage units, $C_1, ..., C_m$. The storage units are similar to counters but are operated in a different way. At the start of the computation, all the storage units are empty. At any time during the computation, a storage unit contains a multiset of symbols from the input alphabet Σ. An atomic move of the machine is either a read-step or a non-read-step. In a read-step, the machine reads the next input symbol, places the input symbol in one of the storage units, and changes state. In a non-read-step, the machine deletes a specified symbol from a storage unit if the storage unit contains the symbol, adds the symbol to another storage unit, and changes state; else (i.e., if the storage unit does not contain the specified symbol), it only changes state. Thus, at any time during the computation, the multiplicity of each symbol in all the storage units is exactly the multiplicity of that symbol in the input segment that has been read so far. More precisely, the computation of a special 1-way DSUM (NSUM) can be described by using the following types of instructions (where C_i and C_j are storage units, and l, k are states):

1. j : Read input symbol, add it to C_i, and goto l.
2. j : If C_i contains σ then delete it from C_i, add it to C_j, and go to l; otherwise, goto k
3. j : Halt

When the machine is nondeterministic, there may be more than one choice for l and k.

We can show the following:

Lemma 3.7 *A special 1-way linear-space DCM (NCM) can be simulated by a special 1-way linear-space DSUM (NSUM).*

We can now prove the converse of Lemma 3.6.

Lemma 3.8 *Let L be a language accepted by a restricted 1-way linear-space DCM (NCM) M. Then we can effectively construct a deterministic SCPA (nondeterministic SCPA) G accepting L.*

Proof. From Lemmas 3.5 and 3.7, it is enough to show that a deterministic SCPA (nondeterministic SCPA) G can simulate a special 1-way linear-space DSUM (NSUM) M.

We describe the construction which is a generalization of the ideas in [13]. Assume M has m storage units, $C_1, ..., C_m$. G has the same membrane structure as in [13], with some modifications. As in [13], membrane 1 contains membranes $E_1, ..., E_m$ to simulate the storage units. All the sets of rules $R_1, ...$ are the same as in [13], except with modifications and additions we describe below.

1. The first type of instruction (i.e., read-instruction) is simulated like the Increment instruction in [13] with the input extracted from the environment, i.e., in R_1, for this type of rule, add

   $cd_j \to c_{in_{I_j}} d_j \sigma_{come}$

   for every σ in Σ.

   Note that at the end of the simulation of this rule, we want symbol σ to be in C_i.

2. The second type of instruction is simulated like the instruction "If counter is nonzero then decrement and goto l else goto k" in [13], but decrement is interpreted as delete symbol from membrane E_i and the symbol is not thrown out into the environment but added to membrane E_j.

3. Halt is handled as in [13].

□

From Lemmas 3.6 and 3.8,

Theorem 3.1 *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).*

From Theorem 3.1 and Lemma 3.4, we have:

Theorem 3.2 *A deterministic SCPA is strictly weaker than a nondeterministic SCPA.*

Denote by $\mathcal{L}(det\ SCPA)$ ($\mathcal{L}(nondet\ SCPA)$) the class of languages accepted by deterministic SCPAs (nondeterministic SCPAs). Then from Lemmas 3.1- 3.3, we have:

Corollary 3.2 *1. $\mathcal{L}(nondet\ SCPA)$ is closed under union, intersection, concatenation, Kleene * and $^+$, inverse homomorphism, and ε-free homomorphism. It is not closed under complementation, (unrestricted) homomorphism, and reversal.*

2. $\mathcal{L}(det\ SCPA)$ is closed under union, intersection, and complementation.

One might wonder why in Theorem 2.3, we are only able to show that the question of determinism versus nondeterminism for RCPS is equivalent to the question of deterministic LBA versus nondeterministic LBA, whereas in Theorem 3.2 we are able to prove that deterministic SCPA is strictly weaker than nondeterministic SCPA). The reason for this is that in the first model, the entire unary input o^n is available at the start of the computation; hence, space n is available for computation at the beginning. However, in an SCPA, the space that can be used for the computation only increases as more input symbols are read.

Now consider only deterministic (nondeterministic) SCPAs over a unary input alphabet (excluding the right delimiter \$). Clearly, over a unary input alphabet, a restricted 1-way linear-space DCM (NCM) is equivalent to a two-way multihead FA: The former can simulate the latter by first scanning the input and storing the length of the unary input in a counter. Then using the value stored in this counter, it employs additional counters to simulate the two-way heads of the latter. We omit the details. Hence, from Theorem 2.2, we have:

Corollary 3.3 *Deterministic and nondeterministic SCPAs over a unary alphabet are equivalent if and only if deterministic and nondeterministic LBAs (over arbitrary alphabet) are equivalent.*

3.2 Generalization

In our definition of SCPA, we assume that maximal parallelism is bounded, i.e., there is a k (independent of the computation) such that every maximal multiset of rules applicable at each step has size at most k. Call an SCPA without a bound on maximal parallelism a generalized SCPA. Clearly, as indicated in the proof of Lemma 3.6, a nondeterministic generalized SCPA is still equivalent to a restricted 1-way linear-space NCM and, hence, also equivalent to an SCPA. We are not able to show that a deterministic generalized SCPA is equivalent to a restricted 1-way linear-space DCM. However, we can show that it is strictly weaker than a nondeterministic generalized SCPA.

Theorem 3.3 *A deterministic generalized SCPA is strictly weaker than a nondeterministic generalized SCPA.*

Proof. Let $L' = \{x\#^p \mid x$ is a binary number with leading bit 1 and $p \neq 2val(x)\}$. As already remarked above, a nondeterministic generalized SCPA is equivalent to a restricted 1-way linear-space NCM (which is equivalent to an SCPA). Thus, we only need to show that L' can be accepted by a restricted 1-way linear-space NCM. The construction of such a machine is similar to the construction of M' to accept $\bar{L}$ in the proof of Lemma 3.2.

To see that L' cannot be accepted by a deterministic generalized SCPA, assume that G is such an SCPA accepting L'. Suppose G has m membranes (not including the environment). Now, at any time during the computation, each membrane of G will contain some multiple (possibly zero) copies of each symbol in $\Sigma = \{0, 1, \#\}$. Note that there are only a fixed number t of symbols not in Σ that are in the initial configuration of the system.

In what follows, by "observable configuration", we mean the total state (i.e., the multiplicities of the symbols in the different membranes) that the system enters at the end of a step. Let x be any binary string of length $n \geq t$ and consider the computation of G on string $x\#^{2n}$. Clearly, although G can read any number of symbols at each step, there is an $0 \leq i \leq 2n$ such that G will be in an observable configuration after reading the symbols in $x\#^i$, and such a configuration is one of at most $(2n^3)^m$ configurations. The upper bound follows from the fact that at any time, each of the m membranes will have at most n 0's, n 1's, and $2n$ #'s. Consider the set of all n-bit binary strings with 1 in the most significant position and $n \geq t$. Clearly, there are 2^{n-1} binary such strings, and for n large enough, $2^{n-1} > (2n^3)^m$. It follows that

(*) there are two strings x and y of length n (with 1 in the most significant position) and $0 \leq i \leq 2n$ with $x \neq y$ such that M is in the same observable configuration after reading $x\#^i$ and after reading $y\#^i$.

By definition of L', M will reject both $x\#^{i+2val(x)-i} = x\#^{2val(x)}$ and $y\#^{i+2val(y)-i} = y\#^{2val(y)}$. Then from (*), both $x\#^{2val(y)}$ and $y\#^{2val(x)}$ will also be rejected. This is a contradiction, since these strings are in L'. $\square$

4 Conclusion

We investigated the question of deterministic versus nondeterministic computation in P systems and showed that for some classes of systems, the question is equivalent to the open problem of whether deterministic LBA is equivalent to nondeterministic LBA. For some classes, the deterministic version is strictly weaker than the nondeterministic version. Finally, we note that our techniques can be used to obtain similar results for suitably restricted models of P systems with symport/antiport rules [6].

Acknowledgment:

I would like to thank Sara Woodworth for help in preparing the manuscript. I would also like to thank Petr Sosik for his comments on an earlier version of this paper.

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*, 2004.

[3] R. Freund, L. Kari, M. Oswald, and P. Sosik. Computationally universal P systems without priorities: two catalysts are sufficient. Aailable at *http://psystems.disco.unimib.it*, 2003.

[4] R. Freund and Gh. Paun. On deterministic P systems. See P Systems Web Page at http://psystems.disco.unimib.it, 2003.

[5] O. H. Ibarra. The number of membranes matters. In *Proc. 4th Workshop on Membrane Computing*, 218-231, 2003.

[6] A. Paun and Gh. Paun. The power of communication: P systems with symport/antiport. *New Generation Computers* 20(3): 295–306, 2002.

[7] Gh. Paun. Computing with membranes. *Turku University Computer Science Research Report No. 208*, 1998.

[8] Gh. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[9] Gh. Paun. *Membrane Computing: An Introduction.* Springer-Verlag, 2002.

[10] Gh. Paun and G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287(1):73–100, 2002.

[11] W. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2): 177–192, 1970.

[12] W. Savitch. A note on multihead automata and context-sensitive languages. *Acta Informatica*, 2:249–252, 1973.

[13] P. Sosik. P systems ersus register machines: two universality proofs. In *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Arges, Romania*, pages 371–382, 2002.