Fundamenta Informaticae 73 (2006) 133–152 133
IOS Press

On the Computational Power of 1-Deterministic and Sequential P Systems*

Oscar H. Ibarra†, Sara Woodworth
Department of Computer Science, University of California
Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu; swood@cs.ucsb.edu

Hsu-Chun Yen‡
Department of Electrical Engineering, National Taiwan University
Taipei, Taiwan 106, R.O.C.
yen@cc.ee.ntu.edu.tw

Zhe Dang§
School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164, USA
zdang@eecs.wsu.edu

Abstract. The original definition of P systems calls for rules to be applied in a maximally parallel fashion. However, in some cases a sequential model may be a more reasonable assumption. Here we study the computational power of different variants of sequential P systems. Initially we look at cooperative systems operating on symbol objects and without prioritized rules, but which allow membrane dissolution and bounded creation rules. We show that they are equivalent to vector addition systems and, hence, nonuniversal. When these systems are used as language acceptors, they are equivalent to communicating P systems which, in turn, are equivalent to partially blind multicounter machines. In contrast, if such cooperative systems are allowed to create an unbounded number of

* A preliminary version of this paper was presented at the *11th International Computing and Combinatorics Conference.*
† Research supported in part by NSF Grants CCR-0208595, CCF-0430945, and CCF-0524136.
† Address for correspondence: Department of Computer Science, University of California, Santa Barbara, CA 93106, USA
‡ Research supported in part by NSC Grant 93-2213-E-002-003, Taiwan.
§ Research supported in part by NSF Grant CCF-0430531.

new membranes (i.e., with unbounded membrane creation rules) during the course of the computation, then they become universal. We then consider systems with prioritized rules operating on symbol objects. We show two types of results: there are sequential P systems that are universal and sequential P systems that are nonuniversal. In particular, both communicating and cooperative P systems are universal, even if restricted to 1-deterministic systems with one membrane. However, the reachability problem for multi-membrane catalytic P systems with prioritized rules is NP-complete and, hence, these systems are nonuniversal.

Keywords: Sequential P system, 1-deterministic P system, communicating P system, catalytic system, vector addition system, partially blind counter machine.

1. Introduction

Initiated five years ago by Gheorghe Paun [16] as a branch of molecular computing, *membrane computing* identifies an unconventional computing model, namely a P system, from natural phenomena of cell evolutions and chemical reactions. A P system 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 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.

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 [17].) 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). In fact, the Institute for Scientific Information (ISI) has recently selected membrane computing as a fast "Emerging Research Front" in Computer Science (`http://esi-topics.com/erf/october2003.html`).

In the standard definition of a P system, the computation is carried out in a maximally parallel and nondeterministic manner [16, 17]. However, an interesting class of P systems with symport/antiport rules was studied in [3] where 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 [3] 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.

The construction of the deterministic system in [3] is such that the size of the maximal multiset of rules that is applicable at every step of the computation is either 1 or 2. We refer to this system

as 2-deterministic. In general, a k-deterministic system is one in which the maximal multiset of rules applicable at each step is at most k. An interesting case is when $k = 1$, i.e., the system is 1-deterministic.

A concept, which is more general than 1-determinism, is that of sequential mode of computation in P systems; i.e., at every step, only one nondeterministically chosen rule instance is applied. Clearly, when a P system is 1-deterministic, then the system (which, by definition, is still maximally parallel) can be treated as a sequential system. So if a class of systems is nonuniversal under the sequential mode, then any 1-deterministic such system in the class is also nonuniversal. Sequential P systems (also called asynchronous P systems) have been studied in various places in the literature (see, e.g., [5, 1, 2, 10]). Here, we present results that complement these earlier results. In particular, we show the following:

1. Any sequential P system with cooperative rules (i.e., rules of the form $u \rightarrow v$, where u, v are strings of symbols) with rules for membrane creation and membrane dissolution can be simulated by a vector addition system (VAS), provided the rules are not prioritized and the number of membranes that can be created during the computation is bounded by some fixed positive integer. Hence the reachability problem (deciding if a configuration is reachable from the start configuration) is decidable. It follows that 1-deterministic such systems have a decidable reachability problem. Interestingly and somewhat surprisingly, if such cooperative systems are allowed to create an unbounded number of new membranes during the course of the computation, then they become universal.

2. A sequential communicating P system language acceptor (CPA) is equivalent to a partially blind multicounter machine (PBCM) [6]. Several interesting corollaries follow from this equivalence; for example:

   (a) The emptiness problem for CPAs is decidable.

   (b) The class of CPA languages is a proper subclass of the recursive languages.

   (c) The language $\{a^n b^n \mid n \geq 1\}^*$ cannot be accepted by a CPA.

   (d) For every r, there is an $s > r$ and a language that can be accepted by a quasirealtime CPA with s membranes that cannot be accepted by a quasirealtime CPA with r membranes. (In a CPA, we do not assume that the CPA imports an input symbol from the environment at every step. Quasirealtime means that the CPA has to import an input symbol from the environment with delay of no more than k time steps for some nonnegative integer k independent of the computation.)

   (e) A quasirealtime CPA is strictly weaker than a linear time CPA. (Here, linear time means that the CPA accepts an input of length n within cn time for some constant c.)

   (f) The class of quasirealtime CPA languages is not closed under Kleene + and complementation.

   We note that the relationship between PBCMs and sequential symport/antiport P systems (similar to communication P systems) has been studied recently in [5], but only for systems with symbol objects and not as language acceptors. Thus, the results in [5] deal only with tuples of nonnegative integers defined by P systems and counter machines. For example, it was shown in [5] that a set of tuples of nonnegative integers that is definable by a partially blind counter machine can be defined by a sequential symport/antiport system with two membranes. Our new results above cannot be derived from the results in [5].

3. The results for CPA above generalize to cooperative system acceptors with membrane dissolution and bounded creation rules. Hence, the latter are also equivalent to PBCMs.

4. Any recursively enumerable unary language can be accepted by a 1-deterministic 1-membrane CPA with prioritized rules.

5. The reachability problem for sequential catalytic systems with prioritized rules (hence, for 1-deterministic such machines as well) is NP-complete. It follows from this result that a 1-deterministic catalytic system with prioritized rules can only accept recursive languages.

Note that from items 4 and 5 above, when the rules are prioritized, there are 1-deterministic systems that are universal and 1-deterministic systems that are not universal. In contrast, from item 1, without prioritized rules, 1-deterministic systems are not universal.

2. P Systems Without Prioritized Rules Operating in Sequential Mode

In this section, we consider the general definition of a P system as given originally by G. Paun in [16, 17], but with unprioritized rules. However, we allow rules for membrane dissolution and membrane creation. We look at systems that operate in sequential mode, i.e., exactly one rule instance is applied at each step (unless the system halts).

Before proceeding further, we need the definition of a vector addition system. An n-dimensional *vector addition system* (VAS) is a pair $G = \langle x, W \rangle$, where $x \in \mathbf{N}^n$ is called the *start point* (or *start vector*) and W is a finite set of vectors in $\mathbf{Z}^n$, where $\mathbf{Z}$ is the set of all integers (positive, negative, zero). The *reachability set* of the VAS $\langle x, W \rangle$ is the set $R(G) = \{z \mid$ for some j, $z = x + v_1 + \cdots + v_j$, where, for all $1 \leq i \leq j$, each $v_i \in W$ and $x + v_1 + \cdots + v_i \geq 0\}$. The *halting reachability set* $R_h(G) = \{z \mid z \in R(G), z + v \not\geq 0$ for every v in $W\}$. An n-dimensional *vector addition system with states* (VASS) is a VAS $\langle x, W \rangle$ together with a finite set T of transitions of the form $p \rightarrow (q, v)$, where p and q are states and v is in W. The meaning is that such a transition can be applied at point y in state p and yields the point $y + v$ in state q, provided that $y + v \geq 0$. The VASS is specified by $G = \langle x, W, T, p_0 \rangle$, where p_0 is the starting state. It is known that n-dimensional VASS can be effectively simulated by $(n + 3)$-dimensional VAS [7]. The *reachability problem* for a VAS (VASS) G is to determine, given a vector y, whether y is in $R(G)$. Similarly, one can define the reachability problem for halting configurations. It is known that the reachability problem for VASS (and hence also for VAS) is decidable [12]. It is also known that VAS, VASS, and Petri net are all equivalent.

First, we study P systems with membrane dissolution but without rules for membrane creation. Clearly, for sequential computations, it is sufficient to only have cooperative rules of the form $u \rightarrow v$ or of the form $u \rightarrow v; \delta$, where u and v are strings of objects (symbols) where each symbol b in v has a designation or target, i.e., it is written b_x, where x can be $here$, out, or in_j. The designation $here$ means that the object b remains in the membrane containing it (we usually omit this target, when it is understood). The designation out means that the object is transported to the membrane directly enclosing the membrane that contains the object. The designation in_j means that the object is moved into a membrane, labelled j, that is directly enclosed by the membrane that contains the object. The δ attached to the rule means that after the application of the rule, the membrane and the rules it contains are dissolved and

the objects in the membrane become part of the membrane that enclosed the dissolved membrane. We will show that these types of P systems can essentially be simulated by a VASS. Hence the reachability problem is decidable.

Theorem 2.1. A sequential P system with unprioritized cooperative rules (possibly with membrane dissolution rules) can be simulated by a VASS. Hence its reachability problem is decidable.

Proof:
Let P be a system with cooperative rules which operates in sequential mode (one rule is nondeterministically chosen and applied per step). Let Σ be the set of objects in P, with $n = |\Sigma|$. The rules of P can have the form $a_1a_2 \dots a_p \to b_1b_2 \dots b_q$ or $a_1a_2 \dots a_p \to b_1b_2 \dots b_q; \delta$, where δ is the membrane dissolution indicator. We can label the membranes in P from $1, \dots, m$, where m is the skin membrane. Also, without loss of generality, label the membranes containing dissolution rules from $1, \dots, s$ (where obviously $s < m$, since the skin membrane cannot be dissolved). Label the (distinct) rules in the system from $R_1, \dots, R_k$ (note the rule number uniquely defines the membrane each rule is associated with).

We construct a VASS G to simulate P. G has dimension mn such that for $1 \leq i \leq m$ and $1 \leq j \leq n$, position ij corresponds to the multiplicity of object a_j in membrane i. The start vector of G is given by the initial configuration of P. In what follows, we let $S = \{1, 2, \dots, s\}$. The states in G are defined as follows:

- $R_0[S]$: denoting the starting state of the system. In this state all membranes are intact.
- $R_i[S']$ (for every $1 \leq i \leq k$ and $S' \subseteq S$): denoting the fact that rule R_i has been selected to be applied.
- $R_{i,j}[S']$ (for every $1 \leq j \leq$ the number of objects in rule R_i and $S' \subseteq S$): denoting that rule R_i is currently being applied to the system. At state $R_{i,j}$, the jth object in the rule is being processed.
- $F[S']$ (for every $S' \subseteq S$): denoting that the computation has halted.

Starting in state $R_0[S]$, the transitions of VASS G are created as follows:

- Create the transitions $R_0[S] \to (R_i[S], (0, \dots, 0))$ for all $1 \leq i \leq k$. This selects the next rule to apply by nondeterministically moving to state $R_i[S]$ without changing any component.
- Create the following transitions for each rule R_i in G:

  Case 1. R_i (of the form $a_1 \dots a_p \to b_1 \dots b_q$) is a nondissolving rule. For all $S' \subseteq S$:

  1. Starting in state $R_i[S']$, G (using intermediate states $R_{i,j}[S']$) sequentially subtracts 1 from the coordinates corresponding to symbols $a_1, \dots, a_p$ (in this order) in the membrane where rule R_i appears, and then sequentially adds 1 to the coordinates corresponding to $b_1, \dots, b_q$ (in this order) in the membranes targeted in the b_i's. (If there are not enough objects in the current membrane to apply rule R_i, G will become negative and hence will not lead to a valid computation.)
  2. G nondeterministically moves to either $F[S']$ or $R_{i'}[S']$ for some i' such that $R_{i'}$ is a rule not yet been dissolved. If G transitions to $F[S']$, the computation is in a halting state; otherwise, rule $R_{i'}[S']$ is applied.

Case 2. R_i (of the form $a_1 \dots a_p \to b_1 \dots b_q; \delta$) is a dissolving rule. Let R_i be in membrane r. For all $S' \subseteq S$:

1. Same as case 1 step 1. (Each a is consumed and each b is created sequentially.)
2. To dissolve membrane r, each object in the membrane must be sent out into the surrounding membrane. This is done as follows. For each symbol $a_t \in r$, G subtracts 1 from the coordinate corresponding to a_t in membrane r and adds 1 to the coordinate corresponding to a_t in the membrane enclosing r. This process is iterated a number of times, where the number of times is chosen nondeterministically (i.e., when G guesses that a_t has been exhausted from membrane r). Then G does the same thing for each symbol in r. (The goal is to expel all objects from membrane r. If done correctly, all positions in r should be 0. If the nondeterministic choices of repeats made were wrong, the vector will become negative causing an invalid computation. After we reach a final state, the symbol counts in all dissolved membranes must be checked to guarantee all objects were expelled from the dissolved membranes.)
3. G nondeterministically moves to state $F[S' - \{r\}]$ or $R_{i'}[S' - \{r\}]$ for some i' such that $R_{i'}$ is a rule which has not yet been dissolved. Again, if G transitions to $F[S']$, the computation is in a halting state; otherwise, rule $R_{i'}[S' - \{r\}]$ is applied.

At this point, the system has halted in a state $F[S']$. To determine if this state is indeed reachable, we must verify that case 2 step 2 released all objects in the dissolved membranes. For notational convenience, write $F[S - \{r\}], F[S - \{r\} - \{r'\}]$, etc. by $F[S - r], F[S - r - r']$, etc. respectively, where $r, r' \in S$. If α is a state, let $R(\alpha)$ be the set of all vectors that reach state α from the initial vector starting in state $R_0[S]$. Now the reachability set of the P system corresponds to the following: The union over *all* permutations $(r_1, \dots, r_s)$ of $(1, \dots, s)$ of the following: $R(F[S]) \cup (R(F[S - r_1]) \cap N^{mn}_{r_1}) \cup \cdots \cup (R(F[S - r_1 - \cdots - r_s]) \cap N^{mn}_{r_1 - \cdots - r_s})$, where N^{mn}_T is the set of all mn-dimensional vectors with zeros in positions corresponding to the symbols in membranes not in T.

Since it is decidable to determine, given an mn-dimensional vector, whether it is in the set above, we have shown that any sequential P system with dissolving rules and no priority rules can be simulated by a VASS. □

Another type of P systems (called *active P systems*) introduced in [14] allows rules to create new membranes during the computation. The objects of an active P system consist of *passive objects* which do not create new membranes and *active objects* which do create new membranes. A membrane creation rule is written as $u \to [_i v]_i$. If a rule of this form is applied in membrane r, this rule consumes u from r, creates a new membrane i within r, and creates the string v_{in_i}. Membrane creation changes how the degree of a system is defined. In a P system without membrane creation, its degree is the number of membranes in the initial configuration. A P system which allows membrane creation must take into consideration the membranes that may be created. Hence, the degree of a system of this type is an ordered pair where the first component consists of the number of membranes in the initial configuration and the second component consists of the maximum number of membranes at any given time in the system during computation. The concept of membrane creation in a P system has a biological basis. In biology, reproduction is a fundamental function of most cells. Including this function in the definition of a P system is a natural generalization of the model. If we allow membrane creation in our model but bound the total number of membranes that can be created (independent of the computation), we can extend our previous result to incorporate this new functionality. This gives us the following result.

Theorem 2.2. A sequential P system with unprioritized rules and with both membrane dissolution and creation rules, where the total number of membranes that can be created during the computation is at most t (for some positive integer t), can be simulated by a VASS. Hence its reachability problem is decidable.

Proof:
The structure of this proof follows the structure of the proof of Theorem 2.1 with a few minor changes. One change that needs to be made is the vector size we are working with. For P systems with membrane creation we need to account for the maximum number of membranes that may be needed. For a P system which uses a total of t membranes with n objects, we need a VASS G with dimension tn. The initial vector will contain all zeros for each membrane which does not yet exist.

The set associated with each state will need to be changed to represent all membranes which are currently alive. Let $S = \{1, \ldots, m\}$ and $S' \subseteq \{1, \ldots, m, m+1, \ldots, t\}$. The states of this system are $F[S']$, $R_i[S']$, and $R_{i,j}[S']$ with $1 \leq i \leq k$ and $i \leq j \leq$ the number of objects in rule R_i. The initial state of the system is $R_0[S]$. The transitions are the same as in the previous proof with one additional case:

Case 3. R_i is of the form $a_1 \ldots a_p \rightarrow [_j b_1 \ldots b_q]_j$ (R_i is a creation rule). Let R_i be in membrane r. Then for all $S' \subseteq \{\{1, \ldots, m, m+1, \ldots, t\} - \{j\}\}$:

1. Starting in state $R_i[S']$, G (using intermediate states $R_{i,j}[S']$) sequentially subtracts 1 from the coordinates corresponding to symbols $a_1, \ldots, a_p$ (in this order) in membrane r. Then G sequentially adds 1 to the coordinates corresponding to $b_1, \ldots, b_q$ (in this order) in membrane j.

2. G nondeterministically moves to either $F[S' \cup \{j\}]$ or $R_{i'}[S' \cup \{j\}]$ for some i' such that $R_{i'}$ is a rule in in membrane r' such that $r' \in S' \cup \{j\}$.

The rest of the construction follows from the proof of Theorem 2.1. □

The key reason why Theorem 2.2 works is the bound t of the total number of membranes created during any computation. What if we remove this condition? That is, suppose the number of times membrane creation rules are invoked during the computation is unbounded (i.e., it is a function of the computation). In this case, the sequential P system in Theorem 2.2 becomes universal. More precisely, we have the following.

Theorem 2.3. Sequential P systems with unprioritized rules and with both membrane dissolution and creation rules can simulate two-counter machines (and hence are universal).

Proof:
Let M be a (nondeterministic) two-counter machine with nonnegative integer counters x and y. For this proof, it is convenient to have each instruction in M to be in one of the following forms:

- $s : c++, \texttt{goto}\ s'$ (on state s, increment counter c by one and move to state s');
- $s : c--, \texttt{goto}\ s'$ (on state s, decrement counter c by one and move to state s'. The machine crashes when it attempts to decrement a counter with value 0);
- $s : c > 0?, \texttt{goto}\ s'$ (on state s, if counter c stores a positive value, move to state s'. If, however, counter c is with value 0, the machine crashes);

- $s : c == 0?, \texttt{goto}\ s'$ (on state s, if counter c stores 0, move to state s'. If, however, counter c is positive, the machine crashes),

where s, s' are states (there are only finitely many states in M) and c is one of the two counters. Initially, M starts with a designated initial state s_{init} with both counters 0. M halts when it enters a designated accepting state s_{final} with both counters 1.

We now construct a P system P to simulate M. P operates in the sequential mode and uses two membranes $\mathbf{m}_x$ and $\mathbf{m}_y$ which are directly inside the skin membrane. The membrane $\mathbf{m}_x$ (resp. $\mathbf{m}_y$) in P is to simulate the counter x (resp. y) in M. P uses symbols x and y, along with a number of other "state symbols" (each state s in M is treated a symbol s in P). In our construction, the multiplicities of symbols x (resp. y) in membrane $\mathbf{m}_x$ (resp. $\mathbf{m}_y$) correspond to the counter value x (resp. y) in M.

We now describe how each instruction in M is simulated in P. We only describe instructions I concerning x; instructions for y can be handled similarly. We have following cases to consider:

- if the instruction I is $s : x + +, \texttt{goto}\ s'$, we add the following rule to membrane $\mathbf{m}_x$: $s \to x s'_{out}$. That is, a new copy of symbol x is added in membrane $\mathbf{m}_x$, and symbol s is removed from the membrane while a copy of symbol s' is sent out into the skin membrane;

- if the instruction I is $s : x - -, \texttt{goto}\ s'$, we add the following rule to membrane $\mathbf{m}_x$: $sx \to s'_{out}$. That is, the symbol s and a copy of symbol x are removed from the membrane while a copy of symbol s' is sent out into the skin membrane;

- if the instruction I is $s : x > 0?, \texttt{goto}\ s'$, we add the following rule to membrane $\mathbf{m}_x$: $sx \to x s'_{out}$. That is, the symbol s and a copy of symbol x are removed from the membrane, and then the copy of x is added back (to ensure that $x > 0$ in M) while a copy of symbol s' is sent out into the skin membrane;

- if the instruction I is $s : x == 0?, \texttt{goto}\ s'$, we add the following rule to membrane $\mathbf{m}_x$: $s \to s'_{out} C^x_{out}; \delta$. That is, the symbol s is removed from the membrane, and then a copy of symbol s' and a copy of "membrane creation" symbol C^x are sent out into the skin membrane. As a result of this dissolving rule, the membrane $\mathbf{m}_x$ disappears and all the objects x in the membrane are now moved into the skin membrane. Notice that, according to the instruction, we expect that the number of symbols x in the membrane $\mathbf{m}_x$ before its dissolution be 0 – we will see how to handle this "zero-test" in a moment. The membrane creation symbol C^x is to create an empty $\mathbf{m}_x$ membrane using the following rule in the skin membrane: $C^x \to [\Lambda]_{\mathbf{m}_x}$.

In the skin membrane, there are the following additional rules that move a state symbol object into $\mathbf{m}_x$ (labelled with 2) or $\mathbf{m}_y$ (labelled with 3): $s \to s_{in_2}$ and $s \to s_{in_3}$, for each state s in M. Initially, P starts with three membranes: in the skin membrane, it has one copy of the symbol s_{init}, and $\mathbf{m}_x$ and $\mathbf{m}_y$ are empty membranes. Notice that, during any sequential execution of P, there is exactly one copy of a state symbol in all the three membranes. P ends when the following configuration is reached: membrane $\mathbf{m}_x$ contains exactly one x object, membrane $\mathbf{m}_y$ contains exactly one y object, and the skin membrane contains exactly one s_{final} object. The latter condition ensures that there is no x and y object in the skin membrane; i.e., all the zero-tests are handled correctly. Clearly, M has a halting computation iff P has. The result follows. □

Notice that, in the proof of Theorem 2.3, objects can move in and move out a membrane (e.g., the object of a state symbol). This move-in/out is essential in simulating the state transitions in a two-counter machine. We do not currently know whether the P systems in the theorem become nonuniversal when, in addition to membrane creation and dissolution rules, we only allow *local rules* in the form of $u \to v$ where the target of every object in v is $here$ (i.e., no move-in/out). This will be left as a topic for further investigation.

3. P Systems and Partially Blind Multicounter Machines

In this section, we show that unprioritized cooperative P systems with dissolution rules and bounded membrane creation rules used as language acceptors are equivalent to partially blind multicounter machines (PBCMs). First we show that the equivalence holds for a special class, called communicating P system acceptors.

In a communicating P system CPS (with multiple membranes) [18], each rule is of one of the following forms: (1) $a \to a_x$, (2) $ab \to a_x b_y$, and (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 (see Section 2 for their meanings). The $come$ can only occur within the outermost region (i.e., skin membrane), which brings in symbol c from the environment.

Here, we consider a variant of the CPS which is used as a language acceptor. Consider a CPS G with input alphabet $\Sigma = \{a_1, \ldots, a_k\}$. Let α be a new symbol not in Σ. We assume that only α and the symbols in Σ occur abundantly in the environment. Let $\Sigma_\alpha = \Sigma \cup \{\alpha\}$. Thus, the CPS can import an unbounded number of symbols in Σ_α from the environment. We assume that no symbol in Σ_α occurs in the initial configuration. We can view the CPS G above as a language acceptor, which we call a CPA. G accepts a string $x \in \Sigma^*$ if it constitutes all the symbols over Σ imported from the environment during the computation, in the order given in x, when the system halts. Thus, x is built up as follows. At the start of the computation, $x = \lambda$ (the null string). Symbols from Σ are appended to x as they are imported into the skin membrane during the computation. The CPA need not import a symbol (from Σ_α) at every step.

Note that in the "maximal parallelism" operating mode, an unbounded number of symbols from Σ can enter the skin membrane in one step since several rules of type (3) in the definition of G may be applicable to an unbounded number of ab pairs in the skin membrane. If the input symbols (i.e., from Σ) that enter the membrane in the step are $\sigma_1, \ldots, \sigma_k$ (note that k is not fixed), then $\sigma_{i_1} \ldots \sigma_{i_k}$ is the string appended to x, where $i_1, \ldots, i_k$ is some nondeterministically chosen permutation of $1, \ldots, k$. Actually, it can be shown [8] that, in fact, we can assume without loss of generality that $k \leq 1$ (i.e., at most one symbol enters the membrane in each step). A string $x = \sigma_1 \ldots \sigma_n \in \Sigma^*$ is accepted if G has a halting computation after importing symbols $\sigma_1, \ldots, \sigma_n$ from the environment. It follows from the result in [18] that every recursively enumerable language can be accepted by a CPA under the maximal parallelism semantics, and vice-versa.

In the rest of the paper, CPAs are assumed to operate in sequential mode, unless otherwise noted. It turns out that such a CPA is equivalent to a partially blind multicounter machine (PBCM), which is a one-way nondeterministic finite automaton augmented with blind counters [6]. At every step, each counter can be incremented/decremented by 1 or not changed, but it cannot be tested for zero. When there is an attempt to decrement a zero counter, the machine gets stuck and the computation is aborted. The machine does not have to read an input symbol at every step (i.e., it can have ϵ moves). An input string w is

accepted if, when the machine is started in a distinguished initial state with all counters zero, it processes all the input symbols in w and eventually enters an accepting state with all the counters zero. Note that if in a computation, a zero counter gets stuck (because of an attempt to decrement it), the computation is aborted and the input is not accepted. It is well known that if "testing for zero" is allowed (i.e., the counters are not partially blind), such a machine can accept every recursively enumerable language, even when there are only two counters [13]. However, PBCM's are strictly weaker than TM's – its emptiness problem (Is the accepted language empty?) and, hence, also the membership problem are decidable. This follows from the decidability of the reachability problem for VAS [12].

Clearly, we can assume that when a PBCM reads a new input symbol, it does not change the value of any counter. However, it can change the state in the transition. So the counters can only be updated during a non-reading step. We may also assume that at most one counter can be updated at each step.

Finally, we may assume that instead of "reading " an input from the one-way input tape, the PBCM has an input terminal from which it can request an input symbol, e.g., an instruction like: From state p, get 'σ' and go to state q. This means the environment delivers input 'σ' to the system. Formally, an atomic move of a PBCM consists of one of the following *labelled* instructions (the labels are states): (1) $p:$ *Get 'σ' and go to (q_1 or ... or q_k)*; (2) $p:$ *Increment counter C by 1 and go to (q_1 or ... or q_k)*; (3) $p:$ *Decrement counter C by 1 and go to (q_1 or ... or q_k)*. Note that (q_1 or ... or q_k) allows a nondeterministic choice for the next state. Also note that a move of the machine without reading an input and without updating a counter but changing only its state can be simulated by an instruction of type 2 followed by an instruction of type 3. We assume that the machine has a unique halting accepting state.

Theorem 3.1. Language L is accepted by a CPA if and only if it can be accepted by a PBCM.

Proof:
Suppose that L is accepted by a CPA G with m membranes. We construct a PBCM M from G accepting L. M has finitely many counters. For each object $a \in \Sigma_\alpha$ and membrane r of G, we associate a counter $A_{(a,r)}$. Thus, M will have $m(|\Sigma|+1)$ counters. These counters are used to keep track of the multiplicity of each symbol in each membrane.

Since no symbol in Σ_α appears in the initial configuration, all counters are initially zero. The symbols that appear in the initial configuration and their distributions are recorded in the finite-state control of M. Every step of the simulation starts with M nondeterministically selecting a rule R in some membrane. Suppose R is a read-rule. There are two cases. If R imports a symbol c in Σ from the environment, M gets symbol c from its input terminal. If R imports α, M does not execute a get instruction. Instead, it increments the counter $A_{(\alpha,skin)}$ (i.e., the counter associated with the symbol α in the skin membrane) by 1.

Simulation of a rule R of the form $a \to a_x$ or $ab \to a_x b_y$ is easy, since it only involves decrementing some counters (corresponding to a or a and b) by 1 and incrementing the target counters by 1.

At some point during the simulation (chosen nondeterministically), M guesses that G has reached a halting configuration. M verifies for each membrane r that no rule is applicable. M constructs a set Z_r for each membrane r. The purpose of Z_r is to gather all symbols that are no longer in membrane r. First M puts all symbols in the initial configuration that are no longer in membrane r to Z_r (this is easily done since there are only a finite number of such symbols, and M has kept track of their distribution during the computation). M then proceeds as follows. It looks at each rule of the form $a \to a_x$ in membrane r, and stores a in Z_r if it is not already there (thus M is "guessing" that this rule is not applicable). Next, M

looks at each rule of the form $ab \to a_x b_y$ or $ab \to a_x b_y c_{come}$. Clearly, this rule is not applicable if either there is no a or no b in membrane r. If a or b is already in Z_r, M looks at another rule. Otherwise, M nondeterministically chooses one of these, say a, and puts it in Z_r, and then proceeds to look at another rule. When M has processed all the rules in membrane r, it examines Z_r. Let $d_1, \ldots, d_s$ be the symbols in Σ_α but not in Z_r. For each d_i, M decrements the counter in membrane r corresponding to d_i (by 1) nondeterministically many times (including zero time), after which it guesses that there is no more d_i, and puts d_i in Z_r. The process above is done for all membranes. At this point, M enters an accepting state. It is easily verified that M accepts L (= language accepted by G).

The converse (i.e., constructing a CPA to simulate a PBCM) follows directly from the construction in [18]. Due to space limitations, the details are omitted here. □

In what follows, we let CPA(n) (resp., CPA(linear)) denote the class of languages accepted by CPA in quasirealtime (resp., linear time), and PBCM(n) (resp., PBCM(linear)) denote the class of languages accepted by PBCM in quasirealtime (resp., linear time). We also write COUNTER(n) to denote $\{L \mid L$ is accepted by a multicounter machine in quasirealtime$\}$.

It is obvious from the construction described in Theorem 3.1 (see [18]) that a quasirealtime (linear time) PBCM can be simulated by a quasirealtime (linear time) CPA. However, the simulation of a quasirealtime (linear time) CPA by a PBCM M described in Theorem 3.1 does not quite yield a quasirealtime (linear time) PBCM. This is because after M guesses that G has halted, the symbols $d_1, \ldots, d_s$ which are not in Z_r must have their counters decremented to zero. Clearly, the values in the counters (the multiplicities of the d_i's) are unbounded. However, we can modify the construction of M so that at the beginning of the simulation, M nondeterministically guesses the symbols $d_1, \ldots, d_s$. If these symbols are known from the beginning, the computation can decrease the counter for each d_i sporadically throughout the computation and hence spread the decreasing steps throughout the whole computation. During the simulation, M nondeterministically chooses the times to decrement the counter corresponding to d_i (in fact, it can choose to decrement it directly after incrementing it). If M decrements each d_i too little, then at the end the counter will not be zero and hence will not accept. If M decrements each d_i too much, the counter will become negative at some point during the computation. When the computation halts, if the the instances (nondeterministically chosen) the counters are decremented are correct, the counters corresponding to each d_i will be zero and no extra processing will be done at this point. This allows M to run in quasireal time. We omit the details.

The following theorem follows from similar results for PBCM [6].

Theorem 3.2. 1. CPA has a decidable emptiness problem.

2. CPA is a proper subset of the family of recursive languages.

3. CPA does not contain the language $L = (\{a^n b^n \mid n \geq 0\})^*$ and is not closed under Kleene +.

4. CPA(n) is not closed under Kleene +.

5. CPA(n) is not closed under complementation.

6. CPA(n) is a proper subset of COUNTER(n).

7. CPA(n) is a proper subset of CPA(linear).

8. CPA(linear) – COUNTER(n) is not empty.

Lemma 3.1. (from [6]) For every k, there is a language that can be accepted accepted by a quasirealtime PBCM with $(k+1)$ counters but not by any quasirealtime PBCM with k counters.

We can now show the following result.

Theorem 3.3. For every r, there is an $s > r$ and a language L that can be accepted by a quasirealtime CPA with s membranes but not by any quasirealtime CPA with r membranes.

Proof:
Without loss of generality consider only languages over a binary alphabet Σ. Then $|\Sigma_\alpha| = |\Sigma \cup \{\alpha\}| = 3$. Suppose there is an r such that any language language accepted by a quasirealtime CPA can be accepted by a quasirealtime CPA with r membranes.

Let $k = 3r$. From Lemma 3.1, there is a language L that can be accepted by a quasirealtime PBCM with $k+1$ counters but not with k counters. By Theorem 3.3, this language can be accepted by a quasirealtime CPA. Then, by hypothesis, L can also be accepted by a quasirealtime CPA with r membranes. From the construction in the first part of the proof of Theorem 3.1, we can construct from this CPA, a quasirealtime PBCM with at most $3r$ counters. Hence, L can be accepted by a quasirealtime PBCM with k counters, a contradiction. □

A cooperative P system acceptor with dissolution rules and bounded membrane creation rules can be defined in the usual manner (as in a CPA). Clearly, a CPA is a special case of a cooperative P system acceptor. Hence, a PBCM can be simulated by a cooperative system. For the converse, the constructions in Theorems 2.1, 2.2, and 3.1 can be implemented on a PBCM. Hence, Theorems 3.1, 3.2, and 3.3 hold when CPA is replaced by PBCM. We have:

Corollary 3.1. The following are equivalent: CPA, PBCM, cooperative P system acceptors with dissolution and bounded membrane creation rules.

In Theorem 3.1, the model of the CPA G has a sequence of input symbols coming in from the environment, and this sequence constitutes the (one-way) input to the PBCM. Now suppose we modify the way the input is given to G. As before, let $\Sigma = \{a_1, \ldots, a_k\}$ and $\Sigma_\alpha = \Sigma \cup \{\alpha\}$, where α is a distinguished symbol. At the start of the computation, G is given a multiset $wa_1^{i_1} \ldots a_k^{i_k}$ in its input membrane, where w is some fixed multiset from an alphabet Δ disjoint from Σ_α, and each $i_j \geq 0, 1 \leq j \leq k$. The environment only contains an abundance of α. Initially, there are no symbols from $\Sigma \cup \Delta$ in the environment, and only symbols from this set that are exported to the environment (during the computation) can be imported from the environment. Thus, the multiplicity of each symbol in $\Sigma \cup \Delta$ in the system (including the environment) remains the same during the computation. We say that $a_1^{i_1} \ldots a_k^{i_k}$ is accepted if G, when given $wa_1^{i_1} \ldots a_k^{i_k}$ in its input membrane, halts. We denote the language accepted by $L(G)$. We call the new model NCPA. Like CPAs, the language accepted by an NCPA can also be accepted by a PBCM. Hence, the following holds.

Theorem 3.4. We can effectively construct, given an NCPA G, a PBCM M accepting $L(G)$. Hence, the emptiness problem for NCPAs is decidable.

Proof:
(*Idea*) The operation of M when given input $a_1^{i_1} \dots a_k^{i_k}$ is similar to the construction in the first part of the proof of Theorem 3.1. We just show how M makes sure that the during the simulation, the total number of a_j (which initially is i_j) in the system is always the same during computation, for $1 \leq j \leq k$.

Initially, the PBCM M reads $a_1^{i_1} \dots a_k^{i_k}$ and stores i_j in counter C_j (for $1 \leq j \leq k$). For each j, we also use another counter E_j (initially zero) to represent the number of a_j's in the environment. Whenever an a_j is exported into the environment, we increment E_j by 1. When an a_j is imported into the skin membrane from the environment, we decrement E_j by 1. Note that we do not have to worry about checking the multiplicities of the a_j's in the skin membrane and in the other membranes. □

4. 1-Deterministic P Systems with Prioritized Rules

Now let us look at P systems which allow rules to be prioritized, meaning that a rule of lower priority can only be used when rules of higher priority are no longer applicable. Previously we showed that sequential cooperative P systems without priority rules are equivalent to VASS and hence nonuniversal. Allowing priority rules increases the power of these systems causing them to be universal. In fact, even cooperative P systems with the restriction of 1-determinism with only one membrane are already universal. We have the following result.

Theorem 4.1. A prioritized 1-deterministic 1-membrane cooperative system (COS) with rules of the form $u \to v$, where $|u| = 2$, and $|v| = 1$ or 2, can simulate a 2-counter machine (hence, it is universal).

Proof:
Let M be a (deterministic) 2-counter machine. We first construct a normalized 2-counter machine M' that simulates M. The construction follows the idea in [13]. Suppose the two counters of M have values i, j. These values can be represented in one counter of M' by the number $n = 2^i 3^j$. To increment i, j by one, the number n is multiplied by 2, 3, respectively. M' uses a second counter, which is initially zero, for this purpose. The second counter is incremented by 2, 3, respectively for every decrement of 1 in the first counter. When the first counter becomes zero, the second counter has value $2n, 3n$ respectively. Similarly, decrementing i, j by one corresponds to incrementing the second counter by one for every decrement of 2, 3, in the first counter (i.e., n is divided by 2, 3, respectively), after checking that n is divisible by 2,3, respectively.

The state of M is stored in the finite control of M'. To determine the next move, M' has to determine which, if any, of i, j are zero. By passing n from one counter to the other, the finite control of M' can determine if n is divisible by 2, 3, respectively. We modify the above construction slightly by adding the factor 5 to n. Thus, $n = 2^i 3^j 5$. The purpose of the factor is so that n is always positive.

Using the above description of how M' operates, we see that the counters behave in a regular pattern. M' operates in phases in the following way. Let A and B be its counters. M's operation can be divided into phases $P_1, P_2, P_3, \dots$, where each P_i starts with one of the counters equal to some positive integer d_i and the other counter equal to zero. During the phase, the first counter is nonincreasing and the other counter is nondecreasing. The phase ends with the first having value zero and the second counter having a positive value (note that the positiveness is guaranteed by the factor 5). This value is equal to n (i.e., no change), $2n$, $3n$, $n/2$, $n/3$, where n is the value of the first counter before the start of the phase. Thus, a sequence of configurations corresponding to the phases above will be of the form:

$(q_1, x_1, 0), (q_2, 0, x_2), (q_3, x_3, 0), (q_4, 0, x_4), \ldots$, where the q_i are states and $x_1 = 5, x_2, x_3, \ldots$ are positive integers. Note that the second component of the configuration refers to the value of counter A, while the third component refers to the value of counter B. Moreover, the state determines which of the five case $(n, 2n, 3n, n/2, n/3)$ applies.

We may assume that the state names of the 2-counter machine M' when it is operating in odd phases $P_1, P_3, \ldots$ are different from the state names when it is operating in even phases $P_2, P_4, \ldots$. We use q's for states in the odd phases p's for states in the even phases.

Clearly, an instruction of the 2-counter machine M' has one of the following forms:

1. $\delta(q, positive, na) = (q', -1, d)$ (i.e., in the odd phases).
2. $\delta(p, na, positive) = (p', d, -1)$ (i.e., in the even phases).
3. $\delta(q, zero, positive) = (p, d, -1)$ (i.e., switching from odd phase to even phase).
4. $\delta(p, positive, zero) = (q, -1, d)$ (i.e., switching from even phase to odd phase).

where q (respectively, p) is the current state, *positive* means that the first (respectively, second) counter is positive, *na* means that the value of the second (respectively, first) counter does not matter, q' (respectively, p') is the next state, -1 means decrementing the first (respectively, second) counter by 1, and $d \in \{0, 1, 2, 3\}$ means incrementing the second (respectively, the first) counter by d. (Note that $d = 0$ or 1 when division by 2, 3, is being simulated.)

Let $q_1, q_2, \ldots$ be the states the counter machine uses in the odd phases and $p_1, p_2, \ldots$ be the states it uses in the even phases. As already stated, we assume that the q_i's are different from the p_i's. Assume that q_1 is the state of the machine when the first counter has value $2^0 3^0 5) = 5$ and is about to begin phase P_1. We construct a prioritized 1-deterministic 1-membrane cooperative P system with symbols $a, b, q_1, q_2, \ldots, p_1, p_2, \ldots$ to simulate the normalized counter machine. The symbols a and b represent the two counters. The initial configuration is $[_1 q_1 a^5]_1$. (This means that the first counter has value 5 and the second counter is zero.) The rules are defined according to the type of rules above.

1. For a transition of form 1, define the rule $qa \to q'b^d$.
2. For a transition of form 2, define the rule $pb \to p'a^d$.
3. For a rule of form 3, define the rule $qb \to pa^d$.
4. For a rule of form 4, define the rule $pa \to qb^d$.

Rules of the form $qa \to q'b^d$ and $pb \to p'a^d$ are of higher priority than rules of the form $qb \to pa^d$ and $pa \to qb^d$. It is clear that the system described above simulates the computation of the two-counter machine. Note that the rules above can be simplified, e.g., a rule of the form $qa \to q'bbb$ can be simulated by the rules: $qa \to [q'bbb], [q'bbb] \to [q'bb]b, [q'bb] \to [q'b]b, [q'b] \to q'b$, where $[q'bbb], [q'bb], [q'b]$ are new symbols. □

The above result applies to a 1-deterministic 1-membrane symport/antiport system (SAS) [11, 15]. This system has rules of the following forms: $(x, out; y, in)$ which is an antiport rule, and (x, out) or (x, in) which is an symport rule, where x, y are strings of symbols. The *radius* of an antiport rule is $(|x|, |y|)$. For a symport rule, the radius is $|x|$.

Corollary 4.1. A prioritized 1-deterministic 1-membrane symport/antiport system (SAS) whose rules are antiport of radius (2,1) or (2,2) can simulate a 2-counter machine.

Looking at the class of communicating P systems, we find similar results. The class of 1-deterministic 1-membrane CPS with priority rules can also simulate a 2-counter machine using a technique similar to the proof of Theorem 4.1.

Theorem 4.2. A prioritized 1-deterministic 1-membrane CPS can simulate a 2-counter machine.

Proof:
We need only show that the rules in the COS constructed above can be implemented in a 1-membrane CPS with priority rules.

We construct a CPS C with priority rules which simulates a COS. C has one membrane (the skin membrane) and a distinguished symbol X. Initially, C contains symbols Xq_1a^5. We describe how the rules of C are constructed.

Note that in the COS constructed above, its rules are of the form $s\alpha \to tv$, where s, t are states, α is a symbol (either a or b), and v is (a unary string) of the form $\epsilon, \beta, \beta\beta, \beta\beta\beta$, where β is a symbol (either a or b). The rules of the CPS are defined by cases:

Case 1. If the rule is $s\alpha \to t$, then the following rules are in C:

$$R_1 : s\alpha \to s_{out}\alpha_{out}t_{come}$$

Case 2. If the rule is $s\alpha \to t\beta$, then the following rules are in C:

$$R_1 : s\alpha \to s_{out}\alpha_{out}[t\beta]_{come}$$
$$R_2 : X[t\beta] \to X_{here}[t\beta]_{here}[t\beta_1]_{come}$$
$$R_3 : [t\beta][t\beta_1] \to [t\beta]_{out}[t\beta_1]_{here}\beta_{come}$$
$$R_4 : X[t\beta_1] \to X_{here}[t\beta_1]_{out}t_{come}$$

Priority: $R_2 \leq R_3$

Case 3. If the rule is $s\alpha \to t\beta\beta$, then the following rules are in C:

$$R_1 : s\alpha \to s_{out}\alpha_{out}[t\beta\beta]_{come}$$
$$R_2 : X[t\beta\beta] \to X_{here}[t\beta\beta]_{here}[t\beta\beta_1]_{come}$$
$$R_3 : [t\beta\beta][t\beta\beta_1] \to [t\beta\beta]_{out}[t\beta\beta_1]_{here}\beta_{come}$$
$$R_4 : X[t\beta\beta_1] \to X_{here}[t\beta\beta_1]_{here}[t\beta\beta_2]_{come}$$
$$R_5 : [t\beta\beta_1][t\beta\beta_2] \to [t\beta\beta_1]_{out}[t\beta\beta_2]_{here}\beta_{come}$$
$$R_6 : X[t\beta\beta_2] \to X_{here}[t\beta\beta_2]_{out}t_{come}$$

Priority: $R_2 \leq R_3; R_4 \leq R_5$

Case 4. If the rule is $s\alpha \to t\beta\beta\beta$, then the following rules are in C:

$$R_1 : s\alpha \to s_{out}\alpha_{out}[t\beta\beta\beta]_{come}$$
$$R_2 : X[t\beta\beta\beta] \to X_{here}[t\beta\beta\beta]_{here}[t\beta\beta\beta_1]_{come}$$
$$R_3 : [t\beta\beta\beta][t\beta\beta\beta_1] \to [t\beta\beta\beta]_{out}[t\beta\beta\beta_1]_{here}\beta_{come}$$
$$R_4 : X[t\beta\beta\beta_1] \to X_{here}[t\beta\beta\beta_1]_{here}[t\beta\beta\beta_2]_{come}$$
$$R_5 : [t\beta\beta\beta_1][t\beta\beta\beta_2] \to [t\beta\beta\beta_1]_{out}[t\beta\beta\beta_2]_{here}\beta_{come}$$

$R_6 : X[t\beta\beta\beta_2] \to X_{here}[t\beta\beta\beta_2]_{here}[t\beta\beta\beta_3]_{come}$
$R_7 : [t\beta\beta\beta_2][t\beta\beta\beta_3] \to [t\beta\beta\beta_2]_{out}[t\beta\beta\beta_3]_{here}\beta_{come}$
$R_8 : X[t\beta\beta\beta_3] \to X_{here}[t\beta\beta\beta_3]_{out}t_{come}$

Priority: $R_2 \leq R_3; R_4 \leq R_5; R_6 \leq R_7$

Clearly these rules use a 1-membrane CPS with priority rules to simulate a 2-counter machine. Hence a 1-membrane CPS with priority rules is universal. □

The above theorem is in contrast to a result in [10] that a sequential multi-membrane CPS whose rules are not prioritized is equivalent to a VAS.

Finally, consider the model of a sequential multi-membrane CS where the rules are prioritized. Specifically, there is a priority relation on the rules: A catalytic rule R' of lower priority than R cannot be applied if R is applicable. We refer to this system as prioritized CS. We know that the reachability set of a sequential multi-membrane CS is semilinear and, hence, its reachability problem is NP-complete. In [9], the status of the reachability problem for systems with prioritized rules was left open. Here we show that the reachability problem is also NP-complete.

Taking advantage of the equivalence between sequential multi-membrane CS and communication-free VAS[1] [9], we first show the reachability problem for prioritized communication-free VAS (which will be defined in detail below) to be NP-complete, which immediately yields the mentioned complexity result for prioritized sequential multi-membrane CS.

Given a communication-free VAS $G = \langle x, W\rangle$, a *priority relation* ρ over W is an irreflexive, asymmetric, and transitive relation such that v_2 takes precedence over v_1 if $(v_1, v_2) \in \rho$, meaning that v_1 cannot be applied if v_2 is applicable. Due to the nature of communication-freeness, we further assume ρ to satisfy a property that if v and v' subtract from the same coordinate, neither (v, v') nor (v', v) is in ρ; otherwise, one of the two could never be applied. Let $\bar{\rho}$ denote $\{(v, v')|(v, v') \notin \rho \text{ and } (v', v) \notin \rho\}$. In this paper, $\bar{\rho}$ is assumed to be an equivalence relation. Given a vector $z \in \mathbf{N}^n$, an addition vector v can be applied at z under priority relation ρ if $z + v \geq 0$ and no other $v' \in W$ of higher priority has $z + v' \geq 0$. Let $z \xrightarrow{\sigma} z'$ (resp., $z \xrightarrow{\sigma}_\rho z'$) denote the reachability of z' from z through transition sequence σ under the unprioritized (resp., prioritized with priority relation ρ) semantics.

The following result plays a key role in our analysis.

Lemma 4.1. Given a communication-free VAS $G = \langle x, W\rangle$ and a priority relation ρ, if $z \xrightarrow{\sigma} z'$ ($\sigma \in W^*$) and for every v applicable at z', v is in the lowest priority class induced by $\bar{\rho}$, then $z \xrightarrow{\sigma'}_\rho z'$, for some permutation σ' of σ.

Proof:
Along $z \xrightarrow{\sigma} z'$, let z'' be the leftmost vector at which the priority requirement is violated. Let v_1 be the addition vector applied at z'', and v_2 be one of the highest priority applicable at z''. We claim that there exists a v_2', $(v_2, v_2') \in \bar{\rho}$ and v_2' is present in the segment from z'' to z'; otherwise, v_2 would still be applicable at z' (due to the communication-freeness nature of G) – violating the assumption of the

[1] A communication-free VAS is a VAS where in every transition, at most one component is negative, and if negative, its value is -1.

lemma. Since v_2' and v_1 do not subtract from the same coordinate, applying v_2' at z'' followed by v_1 remains a valid computation. By repeatedly applying such a rearrangement to the remaining sequence, a computation meeting the priority requirement can be constructed. □

For related results concerning other types of prioritized concurrent models, the reader is referred to, e.g., [20]. Based upon the above lemma and the fact that the binary reachability relation of communication-free VAS can be characterized by integer linear programming [4, 19], we have the following result.

Theorem 4.3. The reachability problem for prioritized communication-free VAS is NP-complete.

Proof:
The lower bound follows immediately from the NP-hardness of checking reachability for the basic model of communication-free VAS, as they are special cases of their prioritized counterparts (with an empty priority relation).

First we recall the following result (referred to as 'Fact A' in our subsequent discussion) from [4, 19], indicating that the binary reachability relation of communication-free VAS can be characterized by integer linear programming:

- Given a communication-free VAS $G = \langle x, W \rangle$, there exists a system of linear inequalities $\mathcal{L}(G, x')$ of polynomial size such that $x' \in R(G)$ iff $\mathcal{L}(G, x')$ has an integer solution. Furthermore, $\mathcal{L}(G, x')$ remains linear even if x and x' are replaced by variables.

Recall that $\bar{\rho}$ is an equivalence relation, which partitions W into a number of *equivalence classes* $\Omega_1, \ldots, \Omega_d$, for some d ($d \leq |W|$). Intuitively, each $\Omega_i, 1 \leq i \leq d$, represents a set of vectors having the same priority. For every $v \in \Omega_i$ and $v' \in \Omega_j$ ($i \neq j$), either $(v, v') \in \rho$ or $(v', v) \in \rho$ (but not both); we write $\Omega_i < \Omega_j$ (resp., $\Omega_j < \Omega_i$) if $(v, v') \in \rho$ (resp., $(v', v) \in \rho$). Without loss of generality, we assume $\Omega_1, \ldots, \Omega_d$ to be enumerated in increasing priority. Given a $v \in W$ and a ρ, we let $class(v) = i$ if $v \in \Omega_i$ (i.e., $class(v)$ is the index of the equivalence class containing v).

To show the upper bound, suppose $x \xrightarrow{\sigma}_\rho z$ is a computation reaching z in a prioritized communication-free VAS $G = \langle x, W \rangle$ with priority relation ρ. Listing in increasing priority, let the equivalence classes induced by $\bar{\rho}$ be $\Omega_1, \ldots, \Omega_d$, for some d ($\leq |W|$). We let $z_1, \ldots, z_s$, $z_1', \ldots, z_s'$ be vectors and $v_1, \ldots, v_s$ be addition vectors along σ satisfying the following: (We assume $z_0' = x$, $z_s' = z$, and $v_i \in \Omega j_i, 1 \leq j_i \leq d$.)

1. $v_i \in W$ is the vector applied at z_i in σ, and $z_i \xrightarrow{v_i}_\rho z_i'$, $\forall 1 \leq i \leq s$.
2. $\forall 1 \leq i \leq s$,
   (a) $class(v_1) < class(v_2) < \cdots < class(v_s)$,
   (b) $\forall v$ applied in $(z_{i-1}' \xrightarrow{*}_\rho z)$, $class(v) \geq j_i$, and $\forall v$ applied in $(z_i' \xrightarrow{*}_\rho z)$, $class(v) > j_i$. (In words, v_i is the rightmost occurrence among the lowest priority transitions in $z_{i-1}' \xrightarrow{*}_\rho z$.)

The crux of our subsequent analysis lies in the fact that in $z_{i-1}' \xrightarrow{*}_\rho z_i, 1 \leq i \leq s$, v_i is of the lowest priority in VAS G_i with its set of addition vectors restricted to $W_i = W - (\bigcup_{l=1,\ldots,j_{i-1}} \Omega_l)$ (The start

vector of G_i is not important here.) . By Lemma 4.1, $z'_{i-1} \stackrel{*}{\rightarrow} z_i$ iff $z'_{i-1} \stackrel{*}{\rightarrow}_\rho z_i$ in G'. This, in conjunction with Fact A, enables us to set up a system of linear inequalities to capture the reachability of z from x:

We begin by guessing the following:

1. transitions $v_1, \ldots, v_s$ with $class(v_1) < \cdots < class(v_s)$,

2. $\forall 1 \leq i \leq s$, a set of coordinates P_i such that v subtracts from some coordinate in P_i, for every v with $class(v) > class(v_i)$. (P_i is the set of coordinates that are zero so that no vector of higher priority than v_i is applicable in z_i.

Then the system of linear inequalities is set up as follows:

(A0) $x'_0 = x$,

(A1) $x_i(j) = 0, \forall j \in P_i$,

(A2) $\forall 1 \leq i \leq s$, $\mathcal{L}(\langle x'_{i-1}, W_i \rangle, x_i)$ — a system of linear inequalities guaranteed by Fact A,

(A3) $x_i + v_i \geq 0$ and $x'_i = x_i + v_i$,

(A4) $x'_s = z$.

In the above inequalities, $x'_0, x_1, x'_1, \ldots x_s, x'_s$ are vector variables representing the values of markings x, z_1, z'_1, $\ldots$ z_s, z'_s, respectively, mentioned in our earlier discussion. (A0) is trivial. What (A1) says is that at x_i, no vectors with priorities higher than v_i are applicable. By Fact A, (A2) is sufficient to imply $x'_{i-1} \stackrel{*}{\rightarrow} x_i$. Lemma 4.1, in conjunction with (A1) and (A2), further implies $x'_{i-1} \stackrel{*}{\rightarrow}_\rho x_i$. (A3) and (A4) are again trivial.

Based on our earlier discussion, it is then straightforward that the above system of linear inequalities has an integer solution iff $z \in R_\rho(G)$. Hence, the reachability problem is in NP. □

We now have the following.

Corollary 4.2. The reachability problem for sequential multi-membrane catalytic systems with prioritized rules is NP-complete.

Proof:
(Sketch) As shown in [10], every sequential multi-membrane CS S can be simulated by a communication-free VAS G, and vice-versa. Consider a priority relation for multi-membrane CS such that the prioritizing is only between rules in the same membrane, and if $Ca \rightarrow v$ and $C'a \rightarrow v'$ are rules in the same membrane, then neither the first rule takes precedence over the second rule nor the second rule takes precedence over the first rule. Examining the proof of Theorem 4.3 shows that every sequential multi-membrane CS S with prioritized rules can be simulated by a communication-free VAS G with prioritized rules, and vice-versa. □

5. 1-Deterministic Language Acceptors with Prioritized Rules

It is known that any recursively enumerable unary language $L \subseteq o^*$ can be accepted by a (deterministic) 2-counter machine M with a one-way input tape which operates as follows: M, starting in its start state with both counters zero, when given a string $o^n\$$, reads the input symbols left-to-right and halts if and only if $o^n \in L$. Note that M need not read a new input symbol at every step. We may assume that after the right delimiter is read by the counter machine, it can continue with the computation (without attempting to read another symbol on the tape). The input is accepted if M eventually halts. The input is not accepted if the machine does not halt.

The 2-counter machine can be normalized, with each instruction taking one of the following forms: (1) $\delta(q, \epsilon, positive, na) = (q', -1, d)$; (2) $\delta(q, a, na, na) = (q', 0, 0)$, where $a = o$ or \$ (meaning that q is a reading state and the machine reads an input symbol and changes state without altering the counter contents); (3) $\delta(p, \epsilon, na, positive) = (p', d, -1)$; (4) $\delta(p, a, na, na) = (p', 0, 0)$, where $a = o$ or \$; (5) $\delta(q, \epsilon, zero, positive) = (p, d, -1)$; (6) $\delta(p, \epsilon, positive, zero) = (q, -1, d)$. See the proof of Theorem 4.1 in the appendix for more details. Note that the state can also be "tagged" whether they are in a reading mode or nonreading mode. Transitions of the form 1, 3, 5, 6 are non-reading (indicated by ϵ in the second parameter) and are handled as before. Transitions 2 and 4 are reading and are translated to cooperative rules $qa \to q'$ and $pa \to p'$, respectively.

It follows that all the results in the previous section can be reformulated for P systems that are acceptors when the 2-counter machine is an acceptor. For example, Theorem 4.2 can be written to read:

Corollary 5.1. Every recursively enumerable unary language L can be accepted by a deterministic 1-membrane CPS with prioritized rules.

6. Conclusion

In this paper we investigated the computational power of different types of sequential and 1-deterministic P systems. We showed that without prioritized rules, sequential P systems are equivalent to VAS, even if they are allowed to have dissolution rules and bounded creation rules. We also showed that these systems when used as language acceptors are equivalent to communicating P system acceptors which, in turn, are equivalent to to partially blind counter machines. When the rules are prioritized, there are two types of results: there are sequential P systems that are universal and sequential P systems that are nonuniversal. In particular, both communicating and cooperative P systems are universal, even if restricted to 1-deterministic systems with one membrane. However, catalytic P systems with prioritized rules have NP-complete reachability problem and, hence, nonuniversal.

References

[1] E. Csuhaj-Varju, O.H. Ibarra, G. Vaszil: On the computational complexity of P automata. *Proc DNA10*, Milano, 2004 (C. Ferretti, G. Mauri, C. Zandron, eds.), LNCS 3384, Springer, Berlin, 2005, 76–89.

[2] Z. Dang, O.H. Ibarra: On P systems operating in sequential and limited parallel modes. *Pre-Proceedings of 6th Workshop on Descriptional Complexity of Formal Systems*, London, Ontario, 2004, 164–177.

[3] R. Freund, Gh. Păun: *On deterministic P systems*, See P Systems Web Page at `http://psystems.disco.unimib.it`, 2003.

[4] L. Fribourg, H. Olsen: Proving safety properties of infinite state systems by compilation into Presburger Arithmetic. *CONCUR'97*, LNCS 1243, Springer, Berlin, 1997, 213–227.

[5] P. Frisco: About P systems with symport/antiport. *Second Brainstorming Week on Membrane Computing, Sevilla, Spain*, 2004, 224–236.

[6] S.A. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7, 3 (1978), 311–324.

[7] J. Hopcroft, J.-J. Pansiot: On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8, 2 (1979), 135–159.

[8] O.H. Ibarra: On the computational complexity of membrane systems. *Theoretical Computer Science*, 320, 1 (2004), 89–109.

[9] O.H. Ibarra, Z. Dang, O. Egecioglu: Catalytic P systems, semilinear sets, and vector addition systems. *Theoretical Computer Science*, 11, 1 (2004), 167–181.

[10] O.H. Ibarra, H. Yen, Z. Dang: On the power of maximal parallelism in P systems. *Developments in Language Theory 2004*, LNCS 3340, Springer, Berlin, 2004, 212–224.

[11] C. Martin-Vide, A. Păun, Gh. Păun: On the power of P systems with symport rules. *Journal of Universal Computer Science*, 8, 2 (2002), 317–331.

[12] E. Mayr: An algorithm for the general Petri net reachability problem. *SIAM J. Computing*, 13 (1984), 441–460.

[13] 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.

[14] M. Mutyam, K. Kriithivasan: P systems with active objects: Universality and efficiency. *Proceedings of the Third International Conference on Machines, Computations, and Universality*, Chişinău, Moldova, May 23–27 2001, 276 – 287.

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

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

[17] Gh. Păun: *Membrane Computing: An Introduction*. Springer, Berlin, 2002.

[18] P. Sosik: P systems versus register machines: two universality proofs. *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Argeş, Romania*, 2002.

[19] H. Yen: On reachability equivalence for BPP-nets, *Theoretical Computer Science*, 179 (1997), 301–317.

[20] H. Yen: Priority conflict-free Petri nets. *Acta Informatica*, 35, 8 (1998), 673–688.