Catalytic P Systems, Semilinear Sets, and Vector Addition Systems [1]

Oscar H. Ibarra

Department of Computer Science
University of California
Santa Barbara, CA 93106, USA

Zhe Dang

School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164, USA

Omer Egecioglu

Department of Computer Science
University of California
Santa Barbara, CA 93106, USA

Abstract

We look at 1-region membrane computing systems which only use rules of the form $Ca \to Cv$, where C is a catalyst, a is a noncatalyst, and v is a (possibly null) string of noncatalysts. There are no rules of the form $a \to v$. Thus, we can think of these systems as "purely" catalytic. We consider two types: (1) when the initial configuration contains only one catalyst, and (2) when the initial configuration contains multiple catalysts. We show that systems of the first type are equivalent to communication-free Petri nets, which are also equivalent to commutative context-free grammars. They define precisely the semilinear sets. This partially answers an open question in [23,6]. Systems of the second type define exactly the recursively enumerable sets of tuples (i.e., Turing machine computable). We also study an extended model where the rules are of the form $q : \ (p,\ Ca \to Cv)$ (where q and p are states), i.e., the application of the rules is guided by a finite-state control. For this generalized model, type (1) as well as type (2) with some restriction correspond to vector addition systems. Finally, we briefly investigate the closure properties of catalytic systems.

Key words: membrane computing, catalytic system, semilinear set, vector addition system, reachability problem

[1] A short version of this paper (without proofs) was presented at the 28th International

1 Introduction

In recent years, there has been a burst of research in the area of membrane computing [19], which identifies an unconventional computing model (namely a P system) from natural phenomena of cell evolutions and chemical reactions [2]. Due to the built-in nature of maximal parallelism inherent in the model, P systems have a great potential for implementing massively concurrent systems in an efficient way, once future bio-technology (or silicon-technology) gives way to a practical bio-realization (or a chip-realization). In this sense, it is important to study the computing power of the model.

Two fundamental questions one can ask of any computing device (such as a Turing machine) are: (1) What kinds of restrictions/variations can be placed on the device without reducing its computing power? (2) What kinds of restrictions/variations can be placed on the device which will reduce its computing power? For Turing machines, the answer to (1) is that Turing machines (as well as variations like multitape, nondeterministic, etc.) accept exactly the recursively enumerable (r.e.) languages. For (2), there is a wide spectrum of well-known results concerning various sub-Turing computing models that have been introduced during the past half century – to list a few, there are finite automata, pushdown automata, linearly bounded automata, various restricted counter automata, etc. Undoubtedly, these sub-Turing models have enhanced our understanding of the computing power of Turing machines and have provided important insights into the analysis and complexity of many problems in various areas of computer science. We believe that studying the computing power of P systems would lend itself to the discovery of new results if a similar methodology is followed. Indeed, much research work has shown that P systems and their many variants are universal (i.e., equivalent to Turing machines) [4,19,20,3,7,9,23] (surveys are found in [15,21,22]; see also the comprehensive bibliography at http://psystems.disco.unimib.it). However, there is little work in addressing the sub-Turing computing power of restricted P systems. To this end, we present some new results in this paper, specifically focusing on catalytic P systems.

A P system S consists of a finite number of membranes, each of which contains a multiset of objects (symbols). The membranes are organized as a Venn diagram or a tree structure where a membrane may contain other membranes. The dynamics of S is governed by a set of rules associated with each membrane. Each rule specifies how objects evolve and move into neighboring membranes. The rule set can also be associated with priority: a lower priority rule does not apply if one with a higher priority is applicable. A precise definition of S can be found in [19]. Since, from a recent result in [23], P systems with one membrane (i.e., 1-region P systems)

Symposium on Mathematical Foundations of Computer Science (MFCS 2003). This research was supported in part by NSF Grants IIS-0101134 and CCR02-08595. Contact author: Oscar H. Ibarra, Email: ibarra@cs.ucsb.edu, Fax: 805-893-8553.

and without priority are already able to simulate two counter machines and hence universal [17], for the purposes of this paper, we focus on catalytic 1-region P Systems, or simply catalytic systems (CS's) [19,23].

A CS S operates on two types of symbols: catalytic symbols called *catalysts* (denoted by capital letters C, D, etc) and noncatalytic symbols called *noncatalysts* (denoted by lower case letters a, b, c, d, etc). An evolution rule in S is of the form $Ca \to Cv$, where C is a catalyst, a is a noncatalyst, and v is a (possibly null) string (an obvious representation of a multiset) of noncatalysts. A CS S is specified by a finite set of rules together with an initial multiset (configuration) w_0, which is a string of catalysts and noncatalysts. As with the standard semantics of P systems [19], each evolution step of S is a result of applying all the rules in S in a maximally parallel manner. More precisely, starting from the initial configuration w_0, the system goes through a sequence of configurations, where each configuration is derived from the directly preceding configuration in one step by the application of a multiset of rules, which are chosen nondeterministically. Note that a rule $Ca \to Cv$ is applicable if there is a C and an a in the preceding configuration. The result of applying this rule is the replacement of a by v. If there is another occurrence of C and another occurrence of a, then the same rule or another rule with Ca on the left hand side can be applied. We require that the chosen multiset of rules to apply must be maximally parallel in the sense that no other applicable rule can be added to the multiset. Configuration w is reachable if it appears in some execution sequence. w is halting if none of the rules is applicable on w. The set of all reachable configurations is denoted by $R(S)$. The set of all halting reachable configurations (which is a subset of $R(S)$) is denoted by $R_h(S)$.

It is important to point out that our CS has only one membrane and does not use communication. No object is sent out to the environment. Since the catalysts do not change, we do not include them in $R(S)$ and $R_h(S)$. For some of the results, we use simple projections or extensions of these reachability sets in order to select the objects which count in the the desired output. Without these projections/extensions, we might need an additional membrane for collecting the output of the computation, which could complicate the proofs. However, there is no need to try to extend the results and proofs for systems with two membranes, as we are interested mostly in showing that some 1-membrane systems are not universal.

We show that CS's whose initial configuration contains only one catalyst are equivalent to communication-free Petri nets, which are also equivalent to commutative context-free grammars [5,13]. They define precisely the semilinear sets in this case. Hence $R(S)$ and $R_h(S)$ are semilinear. This partially answers an open problem in [23], where it was shown that when the initial configuration contains multiple catalysts, S is universal, and [23] raised the question of what is the optimal number of catalysts for universality. In fact, very recently, it was shown in [6], that three catalysts (even when each catalyst appears exactly once in the initial configuration) are already sufficient for universality. It remains an interesting open question as to whether the three catalysts can be reduced to two. Our result shows that one catalyst is not enough. We also study an extended model where the rules are of the form $q : \ (p,\ Ca \rightarrow Cv)$ (where q and p are states), i.e., the application of the rules is guided by a finite-state control. For this generalized model, systems with one catalyst in its initial configuration as well as systems with multiple catalysts in its initial configuration but with some restriction correspond to vector addition systems.

The paper has five sections in addition to this section. Section 2 recalls some definitions and fundamental results concerning semilinear sets and reversal-bounded multicounter machines needed in the paper. Section 3 considers catalytic systems whose initial configuration has only one catalyst as well as systems with multiple catalysts and establish their connection to semilinear sets and recursively enumerable sets. Section 4 characterizes several variants of catalytic systems (including systems with "matrix rules") in terms of various classes of vector addition systems. Section 5 looks at closure properties. Section 6 is a brief conclusion.

2 Semilinear Sets and Reversal-Bounded Multicounter Machines

In this section, we recall the definition of a semilinear set and its characterization in terms of a reversal-bounded multicounter machine. The characterization is useful in proving some of our results.

Let $\mathbf{N}$ be the set of nonnegative integers and k be a positive integer. A set $S \subseteq \mathbf{N}^k$ is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_t$ in $\mathbf{N}^k$ such that $S = \{v \mid v = v_0 + a_1 v_1 + \ldots + a_t v_t,\ a_i \in \mathbf{N}\}$. The vectors v_0 (referred to as the *constant vector*) and $v_1, v_2, \ldots, v_t$ (referred to as the *periods*) are called the *generators* of the linear set S. A set $S \subseteq \mathbf{N}^k$ is *semilinear* if it is a finite union of linear sets. The empty set is a trivial (semi)linear set, where the set of generators is empty. Every finite subset of $\mathbf{N}^k$ is semilinear – it is a finite union of linear sets whose generators are constant vectors. Clearly, semilinear sets are closed under union and projection. It is also known that semilinear sets are closed under intersection and complementation.

There is a simple automata characterization of semilinear sets. Let M be a nondeterministic finite-state machine *without an input tape*, but with n counters (for some $n \geq 1$). The computation of M starts with all the counters zero and the machine in the start state. The counters are *reversal-bounded* in that each counter can be tested for zero and can be incremented by one, decremented by one, or left unchanged, but the number of alternations between nondecreasing mode and nonincreasing mode in any computation is bounded by a given constant. For example, a counter whose values change according to the pattern 0 1 1 2 3 4 $\underline{4\ 3}$ 2 1 $\underline{0\ 1}$ $\underline{1\ 0}$ is 3-reversal (here the reversals are underlined). M is called a reversal-bounded multicounter machine (with n counters). Suppose $k \leq n$ counters are designated as output counters (note that all counters can be output counters) Define $G(M) = \{v \mid M$ when started from its initial configuration halts in an accepting state with v the tuple of values of the k output counters $\}$. We refer to $G(M)$ as the set generated by M.

The following result is from [14].

Theorem 2.1 *(1) Given a semilinear set S, we can effectively construct a reversal-bounded multicounter machine M such that $G(M) = S$.*

(2) Given a reversal-bounded multicounter M, $G(M)$ is an effectively computable semilinear set.

Theorem 2.1, part 2 can be strengthened. Assume that in M *one* of the counters is unrestricted (i.e., not reversal-bounded). Call this counter *free*. Then the following was also shown in [14].

Theorem 2.2 *Let M be a machine with one free counter and several reversal-bounded counters. Then $G(M)$ is an effectively computable semilinear set.*

The above result cannot be extended to allow *two* free counters, since it is known that a machine with two counters, both of which are free, can simulate the computation of a Turing machine (TM) [17].

A multicounter machine can also be used as an "acceptor" instead of a generator. Such a machine is initially given a tuple $(i_1, ..., i_k)$ in its input counters with all other counters zero. The tuple is accepted if the machine eventually halts in an accepting state. The equivalences of 1, 2, and 3 of the following theorem follows from the theorems above. The fact that every semilinear set can be accepted by a *deterministic* reversal-bounded multicounter acceptor was also shown in [14]:

Theorem 2.3 *Let $Q \subseteq \mathbf{N}^k$. Then the following statements are equivalent:*

(1) Q is a semilinear set.

(2) Q can be generated by some reversal-bounded multicounter generator.

(3) Q can be accepted by some reversal-bounded multicounter acceptor.

(4) Q can be accepted by some deterministic *reversal-bounded multicounter acceptor.*

Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be an alphabet. For each string w in Σ^*, define the *Parikh map* [18] of w to be $\psi_\Sigma(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n})$, where $|w|_{a_i}$ is the number of occurrences of a_i in w. For a language (set of strings) $L \subseteq \Sigma^*$, the *Parikh map* of L is $\psi_\Sigma(L) = \{\psi_\Sigma(w) \mid w \in L\}$.

3 1-Region Catalytic Systems

In this section, we study 1-region membrane computing systems which use only rules of the form $Ca \to Cv$, where C is a catalyst, a is a noncatalyst, and v is a (possibly null) string of noncatalysts. Note that we do not allow rules of the form $a \to v$ as in a P System. Thus, we could think of these systems as "purely" catalytic. As defined earlier, we denote such a system by CS.

Let S be a CS and w be an initial configuration (string) representing a multiset of catalysts and noncatalysts. A configuration x is a reachable configuration if S can reach x starting from the initial configuration w. Call x a halting configuration if no rule is applicable on x. Unless otherwise specified, "reachable configuration" will mean any reachable configuration, halting or not. Note that a nonhalting reachable configuration x is an intermediate configuration in a possibly infinite computation. We denote by $R(S)$ the set of Parikh maps of reachable configurations with respect to noncatalysts only. Since catalysts do not change in a computation, we do not include them in the Parikh map. Also, for convenience, when we talk about configurations, we sometimes do not include the catalysts. $R(S)$ is called the reachability set of S. $R_h(S)$ will denote the set of all halting reachable configurations.

3.1 The Initial Configuration Has Only One Catalyst

In this subsection, we assume that the initial configuration of the CS has only *one* catalyst C.

A noncatalyst a is *evolutionary* if there is a rule in the system of the form $Ca \to Cv$; otherwise, a is *non-evolutionary*. Call a CS *simple* if each rule $Ca \to Cv$ has at most one evolutionary noncatalyst in v. Our first result shows that semilinear sets and simple CS's are intimately related.

Theorem 3.1 *(1) Let $Q \subseteq \mathbf{N}^k$. If Q is semilinear, then there is a simple CS S such that Q is definable by S, i.e., Q is the projection of $R_h(S)$ on k coordinates.*
(2) Let S be a simple CS. Then $R_h(S)$ and $R(S)$ are semilinear.

Proof. Let Q be a semilinear set, where $Q = Q_1 \cup ... \cup Q_m$, each Q_i is a linear set. Define a simple CS S as follows. S will have catalyst C and noncatalysts $a_1, ..., a_k, b_1, ..., b_m, c_1, ..., c_m, d$, and initial configuration Cd. The noncatalysts b_i and c_i will be used for computing Q_i. The rules of S are:

$Cd \to Cb_1, Cd \to Cb_2, ..., Cd \to Cb_m.$

And, for each linear set Q_i, we have a set of rules. For example, for Q_1, we have the following rules:

If $(i_1, ..., i_k)$ is the constant vector in Q_1, then $Cb_1 \to Cc_1 a_1^{i_1} ... a_k^{i_k}$ is a rule in S.
If $(j_1, ..., j_k)$ is a generator in Q_1, then $Cc_1 \to Cc_1 a_1^{j_1} ... a_k^{j_k}$ is a rule in S.
We also add the rule $Cc_1 \to C$ in S.

Similar rules can be constructed for $Q_2, ..., Q_m$. Clearly, the configuration when the system halts is $Ca_1^{t_1} ... a_k^{t_k}$ if and only if $(t_1, ..., t_k)$ is in Q. Thus, the Parikh map of any halting configuration is $(t_1, ..., t_k, 0, 0, ..., 0, 0)$ if and only if $(t_1, ..., t_k)$ is in Q.

We now prove the second part of the theorem. Suppose we are given a simple CS S with an initial configuration w. We will construct a multicounter machine M which simulates the computation of S. We associate a counter with every noncatalyst. By definition of a simple CS, the multiplicity of catalyst C during the computation is always 1, and the multiplicity of any evolutionary noncatalyst during the computation is always bounded by the number of evolutionary noncatalysts in the initial configuration w. Therefore, M can use the finite control to keep track of the multiplicities. The counters associated with the non-evolutionary noncatalysts are nondecreasing. The finite control of M first creates the initial configuration w of S and then simulates the computation of S faithfully, using the counters to keep track of the multiplicities of the non-evolutionary noncatalysts. When S halts, M halts in an accepting state. Since M's counters are nondecreasing (i.e., 0-reversal bounded), the set of tuples it generates (when it halts) is exactly $R_h(S)$, and by Theorem 2.1, it is semilinear.

To see that $R(S)$ is semilinear, we modify M so that at any time during the simulation, it has a choice of nondeterministically halting in an accepting state. Hence, any reachable tuple in $R(S)$ is accepted by M and therefore semilinear. ∎

Later, in Section 4, we will see that, in fact, the above theorem holds for any CS whose initial configuration has only one catalyst.

Suppose that we extend the model of a CS so that the rules are now of the form $q : (p,\ Ca \to Cv)$, i.e., the application of the rules is guided by a finite-state control. The rule means that if the system is in state q, application of $Ca \to Cv$ will land the system in state p. We call this system a CS with states or CSS. In addition, we allow the rules to be prioritized, i.e., there is a partial order on the rules: A rule r' of lower priority than r cannot be applied if r is applicable. We refer to such a system as a CSSP. For both systems, the computation starts at (q_0, w), where q_0 is a designated start state, and w is the initial configuration consisting of catalyst C and noncatalysts. In Section 4, we will see that a CSS can define only a recursive set of tuples. In contrast, the following result shows that a CSSP can simulate a Turing machine.

Theorem 3.2 *Let S be a CSSP with one catalyst and two noncatalysts. Then S can simulate a Turing machine.*

Proof. It is known that a two-counter machine can simulate a Turing machine [17].

Our construction of the system follows from the proof of this result. We briefly describe the construction in [12]. In [12] it is first shown that a Turing machine can be simulated by a four-counter machine. Then a two-counter machine is constructed as follows. Suppose the four counters have values i, j, k, m. These values can be represented in one counter by the number $n = 2^i 3^j 5^k 7^m$. To increment i, j, k, m by one, the number n is multiplied by 2, 3, 5, 7, respectively. We use a second counter, which is initially zero, for this purpose. The second counter is incremented by 2, 3, 5, 7, respectively for every decrement of 1 in the first counter. When the first counter becomes zero, the second counter has value $2n, 3n, 5n, 7n$, respectively. Similarly, decrementing i, j, k, m by one corresponds to incrementing the second counter by one for every decrement of 2, 3, 5, 7 in the first counter.

The state of the four-counter machine is stored in the finite control of the two-counter machine. To determine the move of the two-counter machine, it has to determine which, if any, of i, j, k, m are zero. By passing n from one counter to the other, the finite control of the two-counter machine can determine if n is divisible by 2, 3, 5, 7.

We modify the above construction slightly by adding the factor 11 to n. Thus, $n = 2^i 3^j 5^k 7^m (11)$. The purpose of the factor 11, which is a dummy, will be seen later.

From the above description of how the two-counter machine operates, we see that the counters behave in a regular pattern. The two-counter machine operates in phases in the following way. Let A and B be its counters. M's operation can be divided into phases $P_1, P_2, P_3, \ldots$, 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 decreasing by one at every step 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 11). Then in the next phase the modes of the counters are interchanged. Thus, a sequence of configurations corresponding to the phases above will be of the form:

$$(q_1, x_1, 0), (q_2, 0, x_2), (q_3, x_3, 0), (q_4, 0, x_4), ...$$

where the q_i are states and $x_1 = 11, x_2, x_3, \ldots$ 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.

We may assume that the state names of the two counters when it is operating in (odd) phases $P_1, P_3, ...$ are different from the state names when it is operating in (even) phases $P_2, P_4, ...$ Clearly, an instruction of the two-counter machine has one of the following forms:

(1) $\delta(q, positive, na) = (p, -1, d)$.

(2) $\delta(p, na, positive) = (q, d, -1)$.
(3) $\delta(q, zero, na) = (p, d, -1)$.
(4) $\delta(p, na, zero) = (q, -1, d)$.

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, p (respectively, q) is the next state, -1 means decrementing the first (respectively, second) counter by 1, and $d \in \{0, 1, 2, 3, 5, 7\}$ means incrementing the second (respectively, the first) counter by d. (Note that $d =$ 0 or 1 when division by 2, 3, 5, or 7 is being simulated.)

Let $q_1, q_2, ...$ be the states the counter machine uses in the odd phases and $p_1, p_2, ...$ be the states it uses in the even phases. 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 $2^0 3^0 5^0 7^0 (11)$ and is about to begin phase P_1. We construct a system S with catalyst C and states $q_1, q_1', q_2, q_2', ..., p_1, p_1', p_2, p_2', ...$ The noncatalysts are a and b, representing the two counters. The initial configuration is (q_1, Ca^{11}). (This means that the first counter has value 11 and the second counter is zero.) The catalytic rules are defined according to the type of rules above.

(1) For a rule of form 1, define the rule $q: \ (p, Ca \to Cb^d)$.
(2) For a rule of form 2, define the rule $p: \ (q, Cb \to Ca^d)$.
(3) For a rule of form 3, define the rule $q: \ (p, Cb \to Ca^d)$.
(4) For a rule of form 4, define the rule $p: \ (q, Ca \to Cb^d)$.

The rules of the form $q: \ (p, Cb \to Ca^d)$ and $p: \ (q, Ca \to Cb^d)$ are of lower priority than the rules of the form $q: \ (p, Ca \to Cb^d)$ and $p: \ (q, Cb \to Ca^d)$. It is clear that the system described above simulates the computation of the two-counter machine. ∎

Directly from Theorem 3.2, we have:

Corollary 3.3 Let S be a CSSP with one catalyst and two noncatalysts. Then $R(S) \subseteq \mathbf{N}^2$ need not be a semilinear set.

We will see later that in contrast to the above result, when the rules are not prioritized, i.e., we have a CSS S with one catalyst and two noncatalysts, $R(S)$ is semilinear.

3.2 The Initial Configuration Has Multiple Catalysts

In this subsection, we assume that initial configuration of the CS can have *multiple* catalysts.

In general, we say that a noncatalyst is k-bounded if it appears at most k times in any reachable configuration. It is bounded if it is k-bounded for some k.

Consider a CSSP whose initial configuration has multiple catalysts. Assume that except for one noncatalyst, all other noncatalysts are bounded or make at most r (for some fixed r) alternations between nondecreasing and nonincreasing multiplicity in any computation. Call this a reversal-bounded CSSP. As in the proof of part 2 of Theorem 3.1 we can construct a machine with one unrestricted counter and several reversal-bounded counters simulating such a system. Note that the final multiplicities of the bounded noncatalysts are stored into their corresponding counters when the CSSP being simulated halts. Hence we have:

Corollary 3.4 If S is a reversal-bounded CSSP, then $R_h(S)$ and $R(S)$ are semilinear.

Without the reversal-bounded restriction, a CSSP can simulate a TM. In fact, a CS with multiple catalysts in its initial configuration can simulate a TM. It was shown in [23] that a CS augmented with noncooperating rules of the form $a \to v$, where a is a noncatalyst and v is a (possibly null) string of noncatalysts is universal in the sense that such an augmented system with 6 catalysts can define any recursively enumerable set of tuples. A close analysis of the proof in [23] shows that all the rules can be made purely catalytic (i.e., of the form $Ca \to Cv$) using at most 8 catalysts. Actually, this number has been reduced substantially [8,6]. In fact, in [6], the following was shown:

Corollary 3.5 A CS with 3 catalysts (even when each catalyst appears exactly once in the initial configuration) can define any recursively enumerable set of tuples.

It is an open problem whether the 3 catalysts in the corollary above can be reduced to 2. We show in the next section that 1 catalyst is not enough.

There is another restriction on a CSSP S that makes it define only a semiliner set. Let T be a sequence of configurations corresponding to some computation of S starting from a given initial configuration w (which contains multiple catalysts). A noncatalyst a is *positive* on T if the following holds: if a occurs in the initial configuration or does not occur in the initial configuration but later appears as a result of some catalytic rule, then the number of occurrences (multiplicity) of a in any configuration after the first time it appears is at least 1. (There is no bound on the number of times the number of a's alternate between nondecreasing and nonincreasing, as long there is at least 1.) We say that a is *negative* on T if it is not positive on T, i.e., the number of occurrences of a in configurations in T can be zero.

Any sequence T of configurations for which every noncatalyst is bounded or is positive is called a *positive computation*.

Corollary 3.6 Any semilinear set is definable by a CSSP where every computation path is positive.

Proof. This follows from the proof of Theorem 3.1, part 1 since every computation in a simple CS is positive. ∎

Conversely, we have,

Corollary 3.7 Let S be a CSSP. Suppose that every computation path of S is positive. Then $R_h(S)$ and $R(S)$ are semilinear.

Proof. As in part 2 of Theorem 3.1, we construct a multicounter machine M to simulate S. For each positive noncatalyst a, we associate two counters A_a^+ and A_a^-. M simulates the computation of S, where the finite control keeps track of the multiplicities of the bounded noncatalysts. Increments (respectively, decrements) in the multiplicity of a positive noncatalyst a is recorded in A_a^+ (respectively, A_a^-). When the computation of S halts, the value of counter A_a^+ is reduced by the value of counter A_a^- and the final values of the bounded noncatalysts are stored in the appropriate counters. Clearly, M's counters are 1-reversal bounded. Hence the set $R_h(S)$ of tuples corresponding to the final values of the counters is semilinear. But the tuples in $R_h(S)$ correspond exactly to halting reachable configurations in S. We can easily modify M to accept $R(S)$. ∎

The previous corollary can further be strengthened.

Corollary 3.8 Let S be a CSSP. Suppose we allow one (and only one) noncatalyst, say a, to be negative. This means that a configuration with a positive occurrence (multiplicity) of a can lead to a configuration with no occurrence of a. Suppose that every computation path of S is positive, except for a. Then $R_h(S)$ and $R(S)$ are semilinear.

Proof. The proof is similar to the previous corollary, except that now, we associate one counter to the negative catalyst a, and this counter can make an unbounded number of reversals. The result then follows from the fact that a multicounter machine with one unrestricted counter and several reversal-bounded counters still only defines a semilinear set. ∎

4 Characterizations in Terms of Vector Addition Systems

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, $z = x + v_1 + ... + v_j$, where, for all $1 \leq i \leq j$, each $v_i \in W$ and $x + v_1 + ... + v_i \geq 0\}$. The *halting reachability set* $R_h(G) = \{z \mid z \in R(G), z + v \ngeq 0$ 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 $p \to (q, v)$, where q and p 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 $G = \langle x, T, p_0 \rangle$, where p_0 is the starting state.

The *reachability problem* for a VASS (respectively, VAS) G is to determine, given a vector y, whether y is in $R(G)$. The *equivalence problem* is to determine given two VASS (respectively, VAS) G and G', whether $R(G) = R(G')$. Similarly, one can define the reachability problem and equivalence problem for halting configurations.

We summarize the following known results concerning VAS and VASS [24,10,1,11,16]:

Theorem 4.1 *(1) Let G be an n-dimensional VASS. We can effectively construct an $(n+3)$-dimensional VAS G' that simulates G.*
(2) If G is a 2-dimensional VASS G, then $R(G)$ is an effectively computable semilinear set.
(3) There is a 3-dimensional VASS G such that $R(G)$ is not semilinear.
(4) If G is a 5-dimensional VAS G, then $R(G)$ is an effectively computable semilinear set.
(5) There is a 6-dimensional VAS G such that $R(G)$ is not semilinear.
(6) The reachability problem for VASS (and hence also for VAS) is decidable.
(7) The equivalence problem for VAS (and hence also for VASS) is undecidable.

Clearly, it follows from part 6 of the theorem above that the halting reachability problem for VASS (respectively, VAS) is decidable.

4.1 The Initial Configuration Has Only One Catalyst

We first consider CSS (i.e., CS with states) whose initial configuration has only one catalyst.

We will need an example in [11] showing that there is a 3-dimensional VASS G such that $R(G)$ is not semilinear: $G =< x, T, p >$, where $x = (0, 0, 1)$, and the transitions in T are:

$p \to (p, (0, 1, -1))$,

$p \to (q, (0, 0, 0))$,

$q \to (q, (0, -1, 2))$,

$q \rightarrow (p, (1, 0, 0))$.

Thus, there are only two states p and q. The following was shown in [11]:

(1) (x_1, x_2, x_3) is reachable in state p if and only if $0 < x_2 + x_3 \leq 2^{x_1}$.
(2) (x_1, x_2, x_3) is reachable in state q if and only if $0 < 2x_2 + x_3 \leq 2^{x_1+1}$.

Hence $R(G)$ is not semilinear. We use the above example to show that there is a CSS S such that $R(S)$ is not semilinear. Let D be the catalyst, a, b, c the noncatalysts, and p, q the states. The initial configuration of S is $(p, Dbcc)$. The rules are:

$p : \ (p, Dc \rightarrow Db)$,

$q : \ (q, Db \rightarrow Dcc)$,

$p : \ (q, Dc \rightarrow Dc)$,

$q : \ (p, Db \rightarrow Dab)$.

Clearly, the reachable configurations of S at state p precisely satisfy $0 < x_2 + x_3 \leq 2^{x_1} + 2$, and the reachable configurations of S at state q precisely satisfy $0 < x_2 + x_3 \leq 2^{x_1+1} + 3$, where, in a configuration, x_1, x_2, x_3 are the multiplicities of noncatalysts a, b, c, respectively. Thus, the set of all reachable configurations is not semilinear; hence we have:

Corollary 4.2 There is CSS S with 1 catalyst, 3 noncatalysts, and two states such that $R(S)$ is not semilinear.

In fact, as shown in the following two lemmas, each CSS corresponds to a VASS and vice versa.

Lemma 4.3 *Let S be a CSS. We can effectively construct a VASS G such that $R(G) = R(S)$.*

Proof. We construct a VASS G as follows:

(1) Let C be the catalyst of S and $a_1, ..., a_k$ be the noncatalysts. Number the rules of S by $1, 2, ...$ Let $q_1, ..., q_n$ be the states of S, with q_1 the initial state. The states of G will consist of $q_1, ..., q_n$, and (q_i, r) for every $1 \leq i \leq n$ and every rule number r.
(2) If (q_1, Cw) is the initial configuration of S, let q_1 be the starting state of G and $(s_1, ..., s_k)$ = Parikh map of w (w.r.t symbols $a_1, ..., a_k$) be the start vector of G.
(3) The generating vectors of G are defined as follows:
Suppose $q : \ (p, Ca_m \rightarrow Ca_1^{j_1} a_2^{j_2} ... a_k^{j_k})$ is rule number r in S. Then G will have the following rules depending on the case:

(a) Case 1: If $j_m = 0$ (i.e., a_m does not appear on the RHS of the rule), then $q \to (p, (j_1, j_2, ..., j_{(m-1)}, -1, j_{(m+1)}, ..., j_k))$ is a transition in G.
(b) Case 2: If $j_m = t > 0$ (i.e., a_m appears on the RHS of the rule), then the following transitions are in G:
$q \to ((q, r), (0, 0, ..., 0, -1, 0, ..., 0))$ and
$(q, r) \to (p, (j_1, j_2, ..., j_{(m-1)}, t, j_{(m+1)}, ..., j_k))$.

Clearly, the reachability set $R(G)$ of G equals $R(S)$. ∎

Conversely, we have:

Lemma 4.4 *Every VASS can be simulated by a CSS.*

Proof. Suppose G is a VASS. The construction of the CSS S is essentially the "reverse" of the construction in the above proof. Number the transitions of G by $1, 2, ...$ For every state q of G and transition r of G, let $q, (q, r, 1), (q, r, 2), ...$ be states in S. Thus, S will have multiple (but finite) copies of (q, r). For convenience, we also take q to be the same as $(q, r, 0)$.

Suppose $q \to (p, (j_1, ..., j_k))$ is transition number r in G. We consider two cases:

Case 1: Some j_i's are negative. We illustrate, by an example, how the corresponding rules in S are defined. Suppose that j_m and j_n are negative and the rest of the j_i's are nonnegative. Then the following rules are in S:

$$q: \ ((q, r, 1), Ca_m \to C),$$

$$(q, r, 1): \ ((q, r, 2), Ca_m \to C),$$

$$\vdots$$

$$(q, r, j_m - 1): \ ((q, r, j_m), Ca_n \to C),$$

$$(q, r, j_m): \ ((q, r, j_m + 1), Ca_n \to C),$$

$$\vdots$$

$$(q, r, j_m + j_n - 1): \ (p, Ca_n \to Ca_1^{j_1} ... a_{m-1}^{j_{m-1}} a_{m+1}^{j_{m+1}} ... a_{n-1}^{j_{n-1}} a_{n+1}^{j_{n+1}} ... a_k^{j_k}).$$

Case 2: All the j_i's are nonnegative. Then the following rule is in S:

$q: \ (p, C\# \rightarrow C\#a_1^{j_1}...a_k^{j_k})$.

Clearly, G reaches a vector in state q if and only if S reaches the same vector in state q. ∎

From Theorem 4.1 part 6, we have:

Corollary 4.5 The reachability problem for CSS is decidable.

Clearly a reachable configuration is halting if no rule is applicable on the configuration. It follows from the above result that the halting reachability problem (i.e., determining if a configuration is in $R_h(S)$) is also decidable.

A VASS is *communication-free* if for each transition $q \rightarrow (p, (j_1, ..., j_k))$ in the VASS, at most one j_i is negative, and if negative its value is -1. From Lemmas 4.3 and 4.4 and the observation that the VASS constructed in the proof of Lemma 4.3 is communication-free, we have:

Theorem 4.6 *The following systems are equivalent in the sense that each system can simulate the others: CSS, VASS, communication-free VASS.*

Now consider a communication-free VASS without states, i.e., a VAS where in every transition, at most one component is negative, and if negative, its value is -1. Call this a *communication-free VAS*. Communication-free VAS's are equivalent to communication-free Petri nets, which are also equivalent to commutative context-free grammars [5,13]. It is known that they have effectively computable semilinear reachability sets [5]. It turns out that communication-free VAS's characterize CS's.

Theorem 4.7 *Every communication-free VAS G can be simulated by a CS, and vice versa.*

Proof. Let G be a communication-free VAS. We construct a CS S which has one catalyst C, noncatalysts $\#, a_1, ..., a_k$, and starting configuration $C\#w$, where w corresponds to the starting vector of G.

Suppose $(j_1, ..., j_{m-1}, j_m, j_{m+1}, ..., j_k)$ is a transition in G.

Case 1: $j_m = -1$ and all other j_i's are nonnegative. Then the following is in S:

$Ca_m \rightarrow Ca_1^{j_1}...a_{m-1}^{j_{m-1}}a_{m+1}^{j_{m+1}}...a_k^{j_k}$.

Case 2: All the j_i's are nonnegative. Then the following rule is in S:

$C\# \rightarrow C\#a_1^{j_1}...a_k^{j_k}$.

Clearly, S simulates G. In fact, $R(G) = R(S) \times \{1\}$.

Conversely, suppose S is a CS. First assume that each rule in S has the form $Ca \rightarrow Cv$, where a is not contained in v. If $Ca_m \rightarrow Ca_1^{i_1}...a_{m-1}^{j_{m-1}}a_{m+1}^{j_{m+1}}...a_k^{j_k}$ is a rule in S, then the following transition is in G: $(j_1, ..., j_{m-1}, -1, j_{m+1}, ..., j_k)$. It is easy to see that G simulates S.

Now we show how to convert a CS S to another system S' which satisfies the property above. The system S' has many catalysts and simulates S. Let C be the catalyst of S and $a_1, ..., a_k$ be its noncatalysts and Cw its initial configuration. Number the rules of S by $1, ..., s$.

CS S' will have catalysts $C, Q_1, ..., Q_s$ and noncatalysts $a_1, ..., a_k, d_1, ..., d_s$. Its initial configuration is Cwz, where $z = Q_1...Q_sd_1...d_s$. (Note that Q_i and d_i are associated with rule number i of S.) The rules of S' are defined as follows:

Case 1: Suppose $Ca_i \rightarrow Cv$ is a rule in S, and v does not contain a_i. Then this rule is in S'.

Case 2: Suppose $Ca_i \rightarrow Ca_i^jv$ is rule number r in S, where $j \geq 1$ and v does not contain a_i. Then we have the following rules in S':

$Ca_i \rightarrow Cd_r^jv$ and

$Q_rd_r \rightarrow Q_ra_i$.

We say that d_r is a_i-related since it is associated with a rule with a_i on the LHS (left hand side). It is easily verified that any reachable configuration in S' with the property that for each i, the number of occurrences of a_i + the number of occurrences of a_i-related d_r's (over all instruction number r) – call the sum m – uniquely corresponds to a reachable configuration in S where the number of occurrences of a_i is equal to m. (Note that reachable configurations in S are over $a_1, ..., a_k$). Thus, S' simulates S.

Since in S' every rule $Xb \rightarrow Xv$ (where X is a catalyst, b is a noncatalyst, and v a string of noncatalysts) has the property that v does not contain b, S' can be converted to a communication-free VAS G. ∎

Corollary 4.8 If S is a CS, then $R(S)$ and $R_h(S)$ are effectively computable semilinear sets.

Proof. Let G be the communication-free VAS in the proof above. Then $R(G) \subseteq \mathbf{N}^{k+s}$ is semilinear (since a communication-free VAS has a semilinear reachability set). Now the first k components of each vector in $R(G)$ correspond to $a_1, ..., a_k$, and the remaining s components correspond to $d_1, ..., d_s$. Clearly, from the proof above, $R(S) = proj_k(R(G) \cap (\mathbf{N}^k \times \{0\}^s))$, where $proj_k$ is the projection of the tuples on the first k coordinates. Since $\mathbf{N}^k \times \{0\}^s$ is semilinear and semilinear sets are closed under intersection and projection, it follows that $R(S)$ is semilinear.

For $R_h(S)$, without loss of generality (by simple relabeling), assume that $a_t, ..., a_k$ ($t \geq 1$) be all the noncatalysts for which there is some rule with a_i on the LHS, $t \leq i \leq k$. Clearly, a reachable configuration in $R(S)$ is halting if coordinates $t, ..., k$ are zero. Hence $R_h(S) = R(S) \cap (\mathbf{N}^{t-1} \times \{0\}^{k-t+1})$, which is semilinear. ∎

The following is obvious, as we can easily construct a VAS from the specification of the linear set.

Corollary 4.9 If Q is a linear set, then we can effectively construct a communication-free VAS G such that $R(G) = Q$. Hence, every semilinear set is a union of the reachability sets of communication-free VAS's.

From the NP-completeness of the reachability problem for communication-free Petri nets (which are equivalent to commutative context-free grammars) [13,5], we have:

Corollary 4.10 The reachability problem for CS is NP-complete.

We have already seen that a CSS S with prioritized rules (CSSP) with two noncatalysts can simulate a TM (Theorem 3.2); hence $R(S)$ need not be semilinear. Interestingly, if we drop the requirement that the rules are prioritized, such a system has a semilinear reachable set.

Corollary 4.11 Let S be a CSS with two noncatalysts. Then $R(S)$ and $R_h(S)$ are effectively computable semilinear sets.

Proof. As in Lemma 4.3, given a CSS S, we can construct a VASS G such that $R(S) = R(G)$. But since S has two noncatalysts the VASS G will be over $\mathbf{N}^2$ (i.e., dimension 2). By Theorem 4.1, parts 1 and 4, $R(G) = R(S)$ is semilinear. By the same construction as in Corollary 4.8, $R_h(S)$ can be constructed from $R(S)$, which preserves semilinearity. ∎

Open Problem: Suppose S has only rules of the form $Ca \to Cv$ whose initial configuration has exactly one catalyst. Suppose the rules are prioritized. How is $R(S)$ related to VASS?

We have seen that a CS with multiple catalysts can simulate a TM. Consider the following restricted version: Instead of "maximal parallelism" in the application of the rules at each step of the computation, we only allow "limited parallelism" by organizing the rules to apply in one step to be in the following form (called a matrix rule):

$(D_1b_1 \to D_1v_1, ..., D_sb_s \to D_sv_s)$

where the D_i's are catalysts (need not be distinct), the b_i's are noncatalysts (need not be distinct), the v_i's are strings of noncatalysts (need not be distinct), and s is the degree of the matrix. The matrix rules in a given system may have different degrees. The meaning of a matrix rule is that it is applicable if and only if each component of the matrix is applicable. The system halts if no matrix rule is applicable. Call this system a *matrix CS*, or MCS for short. We shall also consider MCS with states (called MCSS), where now the matrix rules have states and are of the form:

$p: \ (q, (D_1b_1 \to D_1v_1, ..., D_sb_s \to D_sv_s))$

Now the matrix is applicable if the system is in state p and all the matrix components are applicable. After the application of the matrix, the system enters state q.

Lemma 4.12 *Given a VAS (VASS) G, we can effectively construct an MCS (MCSS)S such that* $R(S) = R(G) \times \{1\}$.

Proof. Suppose that G is a VAS over $\mathbf{N}^n$ with start vector $(t_1, ..., t_n)$. We define an MCS with catalysts $C, C_1, ..., C_n$, and noncatalysts $a_1, ..., a_n, \#$. Let d_i be the largest integer such that $-d_i$ is the i-th component of some matrix rule. (Note that this means that if the i-th component in all matrix rules is nonnegative, $d_i = 0$.) The initial configuration of S is $CC_1^{d_1}...C_n^{d_n}\#a_1^{t_1}...a_n^{t_n}$. (If $d_i = 0$, C_i does not appear in the initial configuration.) The matrix rules are defined as follows.

If $(i_1, ..., i_n)$ is a transition in G, then the corresponding matrix rule in S has the following components:

(1) If at least one of the i_j's is positive, then $C\# \to C\#v$ is in the matrix, where v contains $a_j^{i_j}$ if $i_j > 0$, $1 \leq j \leq n$.
(2) If $i_j = -k$, then $C_ja_j \to C_j$ appears k times in the matrix

E.g., if $n = 5$ and $(0, -2, 7, -1, 2)$ is a transition, the corresponding matrix rule is

$$(C\# \to C\#a_3^7a_5^2, C_2a_2 \to C_2, C_2a_2 \to C_2, C_4a_4 \to C_4).$$

For transition $(0, -2, 0, -1, -2)$, the corresponding matrix rule is

$$(C_2a_2 \to C_2, C_2a_2 \to C_2, C_4a_4 \to C_4, C_5a_5 \to C_5, C_5a_5 \to C_5).$$

Now $R(S)$ is $(n+1)$-dimensional, where the first n coordinates correspond to noncatalysts $a_1, .., a_n$, and the n+1st coordinate corresponds to the noncatalyst $\#$ which occurs exactly once in any reachable configuration. Clearly, $R(S) = R(G) \times \{1\}$.

It is obvious that if G is a VASS (i.e., a VAS with states), then a similar construction would yield an MCSS (i.e., an MCS with states) S. ∎

Lemma 4.13 *Given an MCSS S over n noncatalysts, we can effectively construct an $(n+1)$-dimensional VASS G such that $R(S) = proj_n(R(G) \cap (\mathbf{N}^n \times \{1\}))$.*

Proof. Suppose that S is an MCSS with noncatalysts $a_1, ..., a_n$, catalysts $C_1, C_2, ...$, initial configuration w consisting of catalysts and noncatalyst, and states $q_1, q_2, ...$ with q_1 the start state. Number the matrix rules $1, 2, ...$

We construct an $(n+1)$-dimensional VASS G to simulate S. G will have states $p_0, q_1, q_2, ...$ and states of the form (q_i, r) for every instruction number r. The start state is p_0, and the start vector is $(t_1, ..., t_n, 1)$, where t_j is the number of occurrences of a_j in w. The purpose of the n+1st coordinate will become clear later. G has the following transition rules:

$$p_0 \to (q_1, (t_1, ..., t_n, 1)).$$

Suppose $q : \ (q', D_1b_1 \to D_1v_1, ..., D_sb_s \to D_sv_s))$ is rule number r in S. Let i_j = the number of occurrences of a_j in $v_1, ..., v_s$. Then the following rules are in G:

$q \to ((q', r), (i'_1, ..., i'_n, -1))$ and

$(q', r) \to (q', (i_1, ..., i_n, 1))$.

where i'_j is obtained as follows:

Case 1: None of $b_1, ..., b_s$ is a_j. Then $i'_j = 0$.

Case 2: There are $m \geq 1$ of $b_1, ..., b_s$ that are a_j. Then $i'_j = -m$.

Clearly, the configurations reachable in S are exactly the configurations reachable in G when the n+1st component of the configuration is 1. Hence $R(S) = proj_n(R(G) \cap (\mathbf{N}^n \times \{1\}))$. ∎

The VASS in Lemma 4.13 can be converted to a VAS. It was shown in [11] that if G is an n-dimensional VASS with states $q_1, ..., q_k$, then we can construct an $(n+3)$-dimensional VAS G' with the following property: If the VASS G is at $(i_1, ..., i_n)$ in state q_j, then the VAS G' will be at $(i_1, ..., i_n, a_j, b_j, 0)$, where $a_j = j$ for $j = 1$ to k, $b_k = k + 1$ and $b_j = b_{j+1} + k + 1$ for $j = 1$ to $k - 1$. The last three coordinates keep track of the state changes, and G' has additional transitions for updating these coordinates. However, these additional transitions only modify the last three coordinates. Define the finite set of tuples $F_k = \{(j, (k - j + 1)(k + 1)) \mid j = 1, ..., k\}$ (note that k is the number of states of G). Then we have:

Corollary 4.14 Given an MCSS S over n noncatalysts, we can effectively construct an $(n + 4)$-dimensional VAS G' such that $R(S) = proj_n(R(G') \cap (\mathbf{N}^n \times \{1\} \times F_k \times \{0\}))$, for some effectively computable k (which depends only on the number of states and number of rules in G).

From Theorem 4.6, Lemmas 4.12 and 4.13, and the above corollary, we have:

Theorem 4.15 *The following systems are equivalent in the sense that each system can simulate the others: CSS, MCS, MCSS, VAS, VASS, communication-free VASS.*

Corollary 4.16 It is decidable to determine, given an MCSS S and a configuration α, whether α is a reachable configuration (halting or not).

Proof. Given an MCSS S, construct the VASS G as described in the proof of Lemma 4.13. Then check if $(\alpha, 1)$ is reachable in G. The result follows since the reachability problem for VASS is decidable. ∎

Corollary 4.17 It is decidable to determine, given an MCSS S and a configuration α, whether α is a halting reachable configuration.

Proof. If there is some matrix rule in S that is applicable to α, then α is not a halting reachable configuration; otherwise, construct the VASS G of Lemma 4.13 and check if $(\alpha, 1)$ is reachable in G. ∎

From Lemma 4.12 and Theorem 4.1 part 7, we have:

Corollary 4.18 The equivalence and containment problems for MCSS are undecidable.

5 Closure Properties

In this section, we briefly look at some closure properties of catalytic systems.

Let S be a catalytic system of any type introduced in the previous sections. For the purposes of investigating closure properties, we will say that S *defines* a set $Q \subseteq \mathbf{N}^k$ (or Q is *definable* by S) if $R_h(S) = Q \times \{0\}^r$ for some given r. Thus, the last r coordinates of the $(k+r)$-tuples in $R_h(S)$ are zero, and the first k-components are exactly the tuples in Q.

Fixed the noncatalysts to be $a_1, a_2, a_3, \ldots$. Thus, any system S has noncatalysts $a_1, \ldots, a_t$ for some t. We say that a class of catalytic systems of a given type is closed under:

(1) *Intersection* if given two systems S_1 and S_2, which define sets $Q_1 \subseteq \mathbf{N}^k$ and $Q_2 \subseteq \mathbf{N}^k$, respectively, there exists a system S which defines $Q = Q_1 \cap Q_2$.
(2) *Union* if given two systems S_1 and S_2, which define sets $Q_1 \subseteq \mathbf{N}^k$ and $Q_2 \subseteq \mathbf{N}^k$, respectively, there exists a system S which defines $Q = Q_1 \cup Q_2$
(3) *Complementation* if given a system S which defines a set $Q \subseteq \mathbf{N}^k$, there exists a system S' which defines $Q' = \mathbf{N}^k - Q$.
(4) *Concatenation* if given two systems S_1 and S_2, which define sets $Q_1 \subseteq \mathbf{N}^k$ and $Q_2 \subseteq \mathbf{N}^k$, respectively, there exists a system S which defines $Q = Q_1Q_2$, where $Q_1Q_2 = \{(i_1 + j_1, \ldots, i_k + j_k) \mid (i_1, \ldots, i_k) \in Q_1, (j_1, \ldots, j_k) \in Q_2\}$.
(5) *Kleene* + if given a system S which defines a set $Q \subseteq \mathbf{N}^k$, there exists a system S' which defines $Q' = \bigcup_{n \geq 1} Q^n$.
(6) *Kleene* * if given a system S which defines a set $Q \subseteq \mathbf{N}^k$, there exists a system S' which defines $Q' = \bigcup_{n \geq 0} Q^n$.

Other unary and binary operations can be defined similarly.

Theorem 5.1 *The class CS with only one catalyst in the initial configuration is closed under intersection, union, complementation, concatenation, and Kleene$^+$ (or Kleene*).*

Proof. For intersection, let S_1 and S_2 be CS defining sets $Q_1 \subseteq \mathbf{N}^k$ and $Q_2 \subseteq \mathbf{N}^k$, respectively. From Corollary 4.8, $R_h(S_1) = Q_1 \times \{0\}^{r_1}$ and $R_h(S_2) = Q_2 \times \{0\}^{r_2}$ (for some r_1 and r_2) are semilinear sets. Since semilinear sets are closed under projection, Q_1 and Q_2 are semilinear. From Theorem 3.1, part 2, we can construct multicounter machines M_1 and M_2 generating Q_1 and Q_2, respectively. Note that each of M_1 and M_2 has k 0-reversal counters. We construct a reversal-bounded multicounter machine M which simulates M_1 using one set of k counters. When M_1 accepts and halts, M then simulates M_2 using another set of k counters. When M_2 halts and accepts, M verifies that the k counters of M_1 have the same values as the corresponding k counters of M_2 (by decrementing the counters simultaneously and checking that corresponding counters reach zero at the same time) while making copies of their values onto a third set of k counters. Clearly, the third set of counters correspond to $Q = Q_1 \cap Q_2$, which is semilinear by Theorem 2.1. Then from Theorem 3.1, we can construct a CS S that defines Q. The proof for closure under union is similar.

For complementation, let S be a CS defining a set $Q \subseteq \mathbf{N}^k$. Then $R_h(S) = Q \times \{0\}^r$ is semilinear. From the fact that semilinear sets are closed under projection and Theorem 2.3, Q can be accepted by a deterministic reversal-bounded multicounter machine. It follows that $\mathbf{N}^k - Q$ is semilinear and, hence, definable by a CS.

For concatenation, let S_1 and S_2 define sets $Q_1 \subseteq \mathbf{N}^k$ and $Q_2 \subseteq \mathbf{N}^k$, respectively. Then Q_1 and Q_2 are semilinear, and we can construct multicounter machines M_1 and M_2 generating Q_1 and Q_2, respectively. Each machine has k 0-reversal counters. We construct a 0-reversal multicounter machine M with k counters which operates as follows: (1) M simulates M_1 (note that the counters are nondecreasing). (2) When M_1 accepts and halts, M simulates M_2 (on the same counters). (3) When M_2 accepts and halts, M accepts and halts. Clearly, M generates $Q_1 Q_2$.

Closure under Kleene + (or Kleene *) follows from the construction for concatenation, since $M_2 = M_1$. The simulating machine M iterates step (1) above a nondeterministically chosen number of times (including zero time for the case of Kleene *) before accepting and halting. ∎

6 Conclusion

A large number of papers has been written in the area of membrane computing. For the most part, the results reported are universality results, i.e., various membrane systems have been shown to have the computational power of Turing machines. Not much work has been reported on sub-Turing models of membrane systems. In this paper, we studied various classes of catalytic systems that are not universal and showed that they can be characterized in terms of semilinear sets, reversal-bounded multicounter machines, and vector addition systems. We hope to study other restricted (non-universal) models of membrane systems in the future.

Acknowledgment

We would like to thank Dung Huynh and Hsu-Chun Yen for their comments and for pointing out some of the references concerning vector addition systems. We also appreciate the comments and encouragement of Gheorghe Paun and Petr Sosik on this work. Additionally, we thank Gaurav Saxena for reading an earlier version of this paper.

References

[1] H. G. Baker. Rabin's proof of the undecidability of the reachability set inclusion problem for vector addition systems. In *C.S.C. Memo 79, Project MAC, MIT*, 1973.

[2] G. Berry and G. Boudol. The chemical abstract machine. In *POPL'90*, pages 81–94. ACM Press, 1990.

[3] P. Bottoni, C. Martin-Vide, Gh. Paun, and G. Rozenberg. Membrane systems with promoters/inhibitors. *Acta Informatica*, 38(10):695–720, 2002.

[4] J. Dassow and Gh. Paun. On the power of membrane computing. *Journal of Universal Computer Science*, 5(2):33–49, 1999.

[5] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. In *Proc. Fundamentals of Computer Theory*, volume 965 of *Lecture Notes in Computer Science*, pages 221–232. Springer, 1995.

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

[7] R. Freund and M. Oswald. P Systems with activated/prohibited membrane channels. In *WMC-CdeA'02*, volume 2597 of *Lecture Notes in Computer Science*, pages 261–269. Springer, 2003.

[8] R. Freund, M. Oswald, and P. Sosik. Reducing the number of catalysts needed in computationally universal P systems without priorities. In *5th Descriptional Complexity of Formal Systems Workshop (DFCS)*, 2003.

[9] P. Frisco and H. Jan Hoogeboom. Simulating counter automata by P Systems with symport/antiport. In *WMC-CdeA'02*, volume 2597 of *Lecture Notes in Computer Science*, pages 288–301. Springer, 2003.

[10] M. H. Hack. The equality problem for vector addition systems is undecidable. In *C.S.C. Memo 121, Project MAC, MIT*, 1975.

[11] J. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comp. Sci.*, 8(2):135–159, 1979.

[12] J. Hopcroft and J. Ullman. *Introduction to Automata theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[13] D.T. Huynh. Commutative grammars: The complexity of uniform word problems. *Information and Control*, 57:21–39, 1983.

[14] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.

[15] C. Martin-Vide and Gh. Paun. Computing with membranes (P Systems): Universality results. In *MCU*, volume 2055 of *Lecture Notes in Computer Science*, pages 82–101. Springer, 2001.

[16] E. Mayr. Persistence of vector replacement systems is decidable. *Acta Informat.*, 15:309–318, 1981.

[17] M. Minsky. Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines. *Ann. of Math.*, 74:437–455, 1961.

[18] R. Parikh. On context-free languages. *Journal of the ACM*, 13:570–581, 1966.

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

[20] Gh. Paun. Computing with membranes (P Systems): A variant. *International Journal of Foundations of Computer Science*, 11(1):167–181, 2000.

[21] Gh. Paun. *Membrane Computing: an Introduction*. Springer-Verlag, Berlin, 2002.

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

[23] P. Sosik and R. Freund. P Systems without priorities are computationally universal. In *WMC-CdeA'02*, volume 2597 of *Lecture Notes in Computer Science*, pages 400–409. Springer, 2003.

[24] J. van Leeuwen. A partial solution to the reachability problem for vector addition systems. In *Proceedings of STOC'74*, pages 303–309.