On Model-Checking of P Systems ⋆

Zhe Dang[1]⋆⋆, Oscar H. Ibarra[2], Cheng Li[1], and Gaoyan Xie[1]

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

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

Abstract. Membrane computing is a branch of molecular computing that aims to develop models and paradigms that are biologically motivated. It identifies an unconventional computing model, namely a P system, from natural phenomena of cell evolutions and chemical reactions. Because of the 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 this paper, we look at various models of P systems and investigate their model-checking problems. We identify what is decidable (or undecidable) about model-checking these systems under extended logic formalisms of CTL. We also report on some experiments on whether existing conservative (symbolic) model-checking techniques can be practically applied to handle P systems with a reasonable size.

1 Introduction

There has been a flurry of research activities in the area of membrane computing (a branch of molecular computing) initiated five years ago by Gheorghe Paun [9]. Membrane computing identifies an unconventional computing model, namely a P system, from natural phenomena of cell evolutions and chemical reactions. It abstracts from the way living cells process chemical compounds in their compartmental structures. 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 [10].

⋆ The work by Zhe Dang, Cheng Li and Gaoyan Xie was supported in part by NSF Grant CCF-0430531. The work by Oscar H. Ibarra was supported in part by NSF Grant CCF-0430945.

⋆⋆ Corresponding author (zdang@eecs.wsu.edu).

Due to the maximal parallelism inherent in the model, P systems have a great potential for implementing massively concurrent systems in an efficient way that would allow us to solve currently intractable problems (in much the same way as the promise of quantum and DNA computing) once future bio-technology gives way to a practical bio-realization.

Designing a P system to achieve a pre-defined computational goal is difficult and extremely error-prone. This is because, unlike traditional programming languages, the inherent maximal parallelism in the model makes the P system highly nondeterministic, concurrent, and, more importantly, lack of control-flow structure (e.g., without "control states"). The difficulties naturally call for algorithmic (i.e., decidable) solutions to the verification problem: whether a designed P system does have the desired behavioral property. The solutions will also be important in the future when people implement a P system in vivo. This is because an erroneous P system will be deemed a failure in an expensive lab realization. It is highly desirable to validate the P system in advance in vitro, e.g., through digital computers. Another important application of results concerning decidable properties of P systems is in biology, where such systems are now being proposed for the modeling and simulation of cells. While previous work on modeling and simulation use continuous mathematics such as differential equations, P systems will allow us to use discrete mathematics and algorithms. As a P system models the computation that occurs in a living cell, an important problem is to develop tools for determining reachability between configurations, i.e., how the system evolves over time. Specifically, given a P system and a configuration U (a configuration is the number and distribution of the different types of objects in the various membranes in the system) and some constraints f (e.g., a linear constraint over the numbers of different types of objects), is there a configuration V satisfying f that is reachable from U? This is essentially a model-checking [4] problem: whether a transition system meets a desired temporal property.

Unfortunately, to our best knowledge, model-checking theories for P systems have never been studied so far. In our opinion, this is, probably, due to the short history of membrane computing and also due to the theoretical difficulty of handling the maximal parallelism, which is quite different from the conventional infinite state transition systems currently being studied in model-checking.

In this paper, we try to identify what is decidable about model-checking of P systems. Clearly, since a P system is Turing complete in general, we have to focus on restricted P systems in order to make the model-checking decidable. The first restriction is to focus on P systems with only one membrane. Essentially, this is more like a technical convenience than a real restriction. Since the P system model studied in this paper does not have priority rules and membrane dissolving rules, multi-membranes can be equivalently collapsed into one membrane through properly renaming symbols in a membrane. The second restriction is to focus on bounded P systems (BPS) where rules are only in the form of $u \rightarrow v$, where u and v are multisets of objects with $|u| \geq |v|$ (the size $|u|$ denotes the number of objects in u). Notice that, since we do not require that a BPS starts with a multiset whose size is bounded by a fixed constant, the BPS is essentially an infinite state system (or more precisely, a system with an unbounded number of states). An execution of a BPS can be understood as a sequence of multisets

(configurations). The formalism that we choose to specify the desired behavioral property is $\mathrm{CTL}^{\mathrm{REG}}$ and $\mathrm{CTL}^{\mathrm{LIN}}$, which allow us to reason upon the executions. In short, $\mathrm{CTL}^{\mathrm{REG}}$ and $\mathrm{CTL}^{\mathrm{LIN}}$ are simply CTL [3] augmented with atomic predicates in REG and in LIN, respectively. More precisely, in REG, one can compare the multiplicity of a symbol against an integer constant, while in LIN, one can compare a linear combination of the multiplicities of all the symbols against an integer constant. Notice that basic properties like halting are expressible in $\mathrm{CTL}^{\mathrm{REG}}$. The corresponding $\mathrm{CTL}^{\mathrm{REG}}$ (as well as $\mathrm{CTL}^{\mathrm{LIN}}$) model-checking problem is to argue whether a given temporal formula is interpreted as an empty multiset of configurations.

We first look at a non-cooperative BPS M where each rule is in the form of $a \to b$, where a and b are symbols, in Section 3. Surprisingly, for such systems, the $\mathrm{CTL}^{\mathrm{REG}}$ model-checking problem is undecidable, even for a simple form of $\exists\mathcal{U}$ (exist-until) properties. When we further require that, in M, a symbol can evolve into at most one kind of symbol, we show that the $\mathrm{CTL}^{\mathrm{REG}}$ becomes decidable. On the other hand, when $\mathrm{CTL}^{\mathrm{LIN}}_{-}$ (roughly, dropping $\exists\mathcal{U}$ from $\mathrm{CTL}^{\mathrm{LIN}}$) is considered, its model-checking problem becomes decidable for non-cooperative BPS. Lastly, when some form of determinism is used to restrict a BPS, the $\mathrm{CTL}^{\mathrm{LIN}}$ is decidable. We then turn to study the model-checking problems for BPS (which is not necessarily non-cooperative, i.e., $|u|$ can be greater than 1), in Section 4. We first give an exact automata-theoretic characterization of (non)deterministic BPSs reachable and halting configurations. That is, BPS is equivalent to each of the following three classes of automata: linear-bounded multicounter machines, $log\ n$ space-bounded Turing machines, two-way multihead finite automata. From this result, one can easily conclude that even $\mathrm{CTL}^{\mathrm{REG}}_{-}$ is undecidable for (non)deterministic BPS. In the section, we also study some notions of determinism that make BPS decidable for various model-checking problems.

Given the undecidability results in model-checking P systems, finally, in Section 5, we conduct some experiments to see whether existing conservative (symbolic) model-checking techniques such as Omega (which handles infinite state space) and SPIN (which handles finite state space) for concurrent linear arithmetic programs can be practically applied to handle P systems (which are not necessarily BPS, and also with multi-membranes, and even with priority among rules) with a reasonable size. Previous experiments [1] used a model-checker of rewriting systems where, like our SPIN experiments, object multiplicities have to be restricted to a finite domain. One of the purposes of our experiments is to let us know if the maximally parallelism and "lack of control-flow structure" in P systems would cause existing symbolic encodings for concurrent systems to fail terribly. Our preliminary experiments show that additional effort is needed in studying more efficient encodings and, in particular, new techniques to extract the implicit control-flow from P system rules.

2 P Systems and Their CTL Model-checking Problems

Let $\mathbf{N}$ be the set of nonnegative integers and $\Sigma = \{a_1, \ldots, a_k\}$ be an alphabet, for some k, and u be a (finite) multiset over the alphabet. In this paper, we do not distinguish between several representations of u. That is, u can be treated as a vector in $\mathbf{N}^k$ (the components are the multiplicities of the symbols in Σ); u can be treated as a word

where we only care about the counts of symbols (i.e., its Parikh map). We now introduce formulas to define some sets of multisets. An atomic *regular* predicate is in the form of $\#(a) \sim n$, where $a \in \Sigma$, $n \in \mathbf{N}$ and $\sim \in \{>, <, =, \geq, \leq\}$. The predicate is interpreted as the subset of multisets u over Σ such that the multiplicity $\#(a)$ of symbol a satisfies the predicate. A regular formula is a Boolean combination of atomic regular predicates. We use REG to denote the set of regular formulas. An atomic *linear* predicate is in the form of $\sum_{1 \leq i \leq k} n_i \cdot \#(a_i) \sim n$, where the n_i's and n are integers (positive, 0, negative), and $\sim \in \{>, <, =, \geq, \leq, \equiv_m\}$ with $0 \neq m \in \mathbf{N}$. The predicate is interpreted as a subset of multisets over Σ accordingly. A linear formula is a Boolean combination of atomic linear predicates. We use LIN to denote the set of linear formulas. 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 $S = \{v \mid v = v_0 + a_1 v_1 + \ldots + a_t v_t,\ a_i \in \mathbf{N}\}$. A set $S \subseteq \mathbf{N}^k$ is *semilinear* if it is a finite union of linear sets. A *Presburger formula* is constructed from atomic linear predicates using quantification and Boolean operators. It is known that the following items are equivalent: (1) a set of multisets (treated as vectors) is semilinear, (2) the set is definable by a linear formula, (3) the set is definable by a Presburger formula.

In this paper, we only focus on P systems without priority rules and membrane dissolving rules. In this case, as we mentioned earlier, it suffices for us to consider P systems with one membrane since multiple membranes can be equivalently collapsed into one by properly renaming symbols within a membrane.

A (1-membrane) P system M is specified by a finite set of rules. Each rule is in the form of $u \rightarrow v$ where u and v are multisets over alphabet Σ. A configuration in M is a multiset. As with the standard semantics of P systems [9–11], each evolution step, called a *maximally parallel move*, is a result of applying all the rules in G in a maximally parallel manner. More precisely, let $u_i \rightarrow v_i$, $1 \leq i \leq m$, be all the rules in M. We use $R = (r_1, \ldots, r_m) \in \mathbf{N}^m$ to denote a multiset of rules, where there are r_i instances of rule $u_i \rightarrow v_i$, for each $1 \leq i \leq m$. Let U and V be two configurations (multisets) over Σ. The rule multiset R is *enabled* under configuration U if U contains $\sum_{1 \leq i \leq m} r_i \cdot u_i$ (i.e., U contains the multiset union of r_i copies of multiset u_i, for all $1 \leq i \leq m$). The result of applying R over U is to replace, in parallel, each of the r_i copies of u_i in U with v_i. The rule multiset R is *maximally enabled* under configuration U if it is enabled under U and, for any other rule multiset R' that strictly contains R, R' is not enabled under configuration U. Notice that, for the same U, a maximally enabled rule multiset may not be unique (i.e., M is in general nondeterministic). U can reach V through a maximally parallel move, written $U \rightarrow_M V$, if there is a maximally enabled rule multiset R such that V is the result of applying R over U. Formally, $U \rightarrow_M V$ iff $\exists r_1, \ldots, r_m \in \mathbf{N}$. $\mathrm{MaxEnable}(r_1, \ldots, r_m, U) \wedge \mathrm{Apply}(r_1, \ldots, r_m, U, V)$, where $\mathrm{MaxEnable}(r_1, \ldots, r_m, U)$, indicating that $(r_1, \ldots, r_m)$ is maximally enabled under configuration U, is the following formula:

$$U \geq \sum_{1 \leq i \leq m} r_i \cdot u_i \wedge \forall r'_1 \geq r_1, \ldots, r'_m \geq r_m . (U \geq \sum_{1 \leq i \leq m} r'_i \cdot u_i \Rightarrow r'_1 = r_1 \wedge \ldots \wedge r'_m = r_m),$$

and $\mathrm{Apply}(r_1, \ldots, r_m, U, V)$, indicating that V is the result of applying $(r_1, \ldots, r_m)$ over U, is the following formula: $V = U - \sum_{1 \leq i \leq m} r_i \cdot u_i + \sum_{1 \leq i \leq m} r_i \cdot v_i$. Notice that, in above, we treat the multisets (i.e., U, V, the u's, and the v's) as vectors in $\mathbf{N}^k$. Clearly, a maximally parallel move in M is always definable by a Presburger formula. Starting from some initial configuration, an execution of M goes through a

sequence of configurations, where each configuration is derived from the directly preceding configuration in one maximally parallel move. Formally, we use $U \rightsquigarrow_M V$ to denote the fact that V is reachable from V; i.e., for some n and $U_0, \ldots, U_n$, we have $U = U_0 \rightarrow_M \ldots \rightarrow_M U_n = V$.

From above, a P system M can be treated as a transition system between multisets or vectors in $\mathbf{N}^k$. There has been an established theory, called model-checking, in algorithmically answering verification queries over a transition system's behavior. For a finite state transition system, the queries can be specified in a temporal logic like the computation tree logic (CTL) [3] and various model-checking algorithms are known [4]. For infinite state transition systems, the logic can also be interpreted in many cases (e.g., [2]). In below, we formulate the CTL formalism that we will use to specify our verification queries for P systems.

Let $\mathcal{A}$ be a given class of *atomic* predicates. The $\mathrm{CTL}^{\mathcal{A}}$ formulas f are exactly defined with the following grammar: $f ::= A \mid f \wedge f \mid f \vee f \mid \neg f \mid \exists \circ f \mid \forall \circ f \mid f \; \exists \mathcal{U} \; f \mid f \; \forall \mathcal{U} \; f$, where $A \in \mathcal{A}$ is an atomic formula (predicate), and $\circ$ stands for "next" and $\mathcal{U}$ stands for "until". As usual, the eventuality operator $\exists \diamond \; f$ is the shorthand of $true \; \exists \mathcal{U} \; f$, and, its dual $\forall \Box \; f$ is simply $\neg \; \exists \diamond \; \neg f$. We use $\mathrm{CTL}^{\mathcal{A}}_{-}$ to denote the fragment of $\mathrm{CTL}^{\mathcal{A}}$ where the formulas f are exactly defined with the following grammar: $f ::= A \mid f \wedge f \mid f_1 \vee f_2 \mid \neg f \mid \exists \circ f \mid \forall \circ f \mid \exists \diamond \; f \mid \forall \Box \; f$, where $A \in \mathcal{A}$ is an atomic formula (predicate).

Let M be a P system. We interpret each $\mathrm{CTL}^{\mathcal{A}}$ formula as a subset of configurations of M. That is, the interpretation, written $[f]^M$, is a subset of multisets of objects in M. Formally, the interpretation is recursively defined as follows [2]:

- $[A]^M$ is a *given* subset of multisets of objects in M, where $A \in \mathcal{A}$;
- $[f_1 \wedge f_2]^M$ is $[f_1]^M \cap [f_2]^M$; $[f_1 \vee f_2]^M$ is $[f_1]^M \cup [f_2]^M$;
- $[\neg f_1]^M$ is the complement of $[f_1]^M$; (the universe is the set of all multisets of objects in M)
- $[\exists \circ f_1]^M$ (resp. $[\forall \circ f_1]^M$) is the set of configurations U_1 such that, for some (resp. any) execution $U_1 \rightarrow_M U_2 \rightarrow_M \ldots$, we have $U_2 \in [f_1]^M$;
- $[f_1 \; \exists \mathcal{U} \; f_2]^M$ (resp. $[f_1 \; \forall \mathcal{U} \; f_2]^M$) is the set of configurations U_1 such that, for some (resp. any) execution $U_1 \rightarrow_M U_2 \rightarrow_M \ldots$, we have $U_1, \ldots, U_n \in [f_1]^M$ and $U_{n+1} \in [f_2]^M$, for some n.

The $\mathrm{CTL}^{\mathcal{A}}$ *model-checking problem* is to decide whether, given a P system M and a $\mathrm{CTL}^{\mathcal{A}}$ formula f, the set $[f]^M$ is empty. Notice that, in our definition of the $\mathrm{CTL}^{\mathcal{A}}$ model-checking problem shown above, we did not mention the initial configurations of M. In fact, a verification question like whether a given initial configuration U_{init} satisfies f can also be formulated in our definition as follows: is $[A_{\mathrm{init}} \wedge f]^M$ empty? where A_{init} is an atomic regular predicate where U_{init} is the only satisfying configuration.

In this paper, we focus on model-checking problems of $\mathrm{CTL}^{\mathrm{REG}}$ and $\mathrm{CTL}^{\mathrm{LIN}}$. Unfortunately, the maximal parallelism in P systems is too powerful to make P systems model-checkable; even in simple cases, P systems are able to be Turing complete. This leads us to study restricted forms of P systems where model-checking problems could be decidable. To this end, we focus on *bounded P systems* (BPS), in which each rule is in the form of $u \rightarrow v$ with $|u| \geq |v|$ ($|u|$ denotes the number of objects in u).

3 CTL Model-checking of Non-cooperative Bounded P Systems

Let M be a non-cooperative BPS. That is, M is a 1-membrane P system whose rules are in the form of $a \to b$ or in the form of $a \to \Lambda$ (i.e., one object evolves into at most one object), where $a, b \in \Sigma$. We first show that the $\mathrm{CTL}^{\mathrm{REG}}$ model-checking problem is undecidable for M. Clearly, as we have mentioned earlier, when M has multi-membranes, it can be collapsed into one with 1-membrane. Hence, all the results in this section can be easily generalized to non-cooperative BPSs with multiple membranes.

Theorem 1. *The* $\mathrm{CTL}^{\mathrm{REG}}$ *model-checking problem for non-cooperative BPSs is undecidable. In fact, the undecidability remains even for* $\mathrm{CTL}^{\mathrm{REG}}$ *formulas in the form of* $\mathrm{INIT} \wedge (A \exists \mathcal{U} H)$*, where* INIT, A *and* H *are regular formulas in* REG.

We should point out that in the proof of Theorem 1 we did not use rules in the form of $a \to \Lambda$. Hence, Theorem 1 still holds when only rules in the form $a \to b$ are used. Because of the theorem, we will study a restricted form of M that makes $\mathrm{CTL}^{\mathrm{REG}}$ model-checking decidable. A non-cooperative BPS M is *special* when, for any a, if $a \to b$ and $a \to c$ with $b, c \neq \Lambda$ are rules in M, then $b = c$ (i.e., a could be disappear with $a \to \Lambda$ but it can not evolve into two kinds of symbols).

Theorem 2. *The* $\mathrm{CTL}^{\mathrm{REG}}$ *model-checking problem for special and non-cooperative BPSs is decidable.*

Because of the undecidability result in Theorem 1, we would like to investigate a fragment of a CTL logic that makes the model-checking problem for non-cooperative BPSs decidable. Before we proceed further, we need an intermediate result. Let M be a non-cooperative BPS, whose alphabet is $\Sigma = \{a_1, \ldots, a_k\}$. Recall that we use $u \rightsquigarrow_M v$ to denote the fact that multiset u can reach multiset v in M through some number of maximally parallel moves. We first show a characterization on the reachability relation $\rightsquigarrow_M \subseteq \mathbf{N}^k \times \mathbf{N}^k$, which leads to Theorem 4 later.

Theorem 3. *The reachability relation* $\rightsquigarrow_M \subseteq \mathbf{N}^k \times \mathbf{N}^k$ *for a non-cooperative BPS* M *is definable by a linear formula in* LIN.

Theorem 4. *The* $\mathrm{CTL}^{\mathrm{LIN}}_{-}$ *model-checking problem for non-cooperative BPSs is decidable.*

4 Reachability in Bounded P Systems

We now consider a bounded P system (BPS) M that is not necessarily noncooperative. That is, rules in M are in the form of $u \to v$ with $|v| \leq |u|$. Clearly, from Theorem 1, the $\mathrm{CTL}^{\mathrm{REG}}$ model-checking problem remains undecidable for M. However, encouraged by the decidability results in Theorem 4 for non-cooperative bounded P systems, we would like to know whether the $\mathrm{CTL}^{\mathrm{REG}}_{-}$ model-checking problem for (not necessarily non-cooperative) BPSs would still be decidable. In this section, we will prove that this is not true, even in very simple cases. We say that, when started with some given

configuration, a BPS M has a halting computation if M has an execution that leads to a halting configuration (i.e., none of the rules is enabled).

We first consider the following problem: Given a bounded P system M with rules of the form $u \rightarrow v$, where $|u| = |v| = 1$ or 2 and a fixed multiset w and a distinct symbol o not in w, is there an n such that when M is started with multiset wo^n (the multiset union of w and n copies of o), it eventually halts? We shall refer to this as the emptiness problem for bounded P systems. We will show that this problem is undecidable. In fact, this result holds even when the system is *deterministic* in the sense that the maximally parallel multiset of rules applicable at each step in the computation is unique. We only sketch the proof in this paper. The idea is to relate the computation of M to a restricted type of multicounter machine, called linear-bounded multicounter machine, whose emptiness is known undecidable.

Consider a deterministic (nondeterministic) multicounter machine Z that is linear-bounded in the sense that when given an input n in one of the counters (called the input counter) and zeros in the other counters, computes in such a way that the sum of the values of the counters at any time during the computation is at most n. One can normalize the computation so that every increment is preceded by a decrement (i.e., if Z wants to increment a counter C, it first decrements some counter D and then increments C) and every decrement is followed by an increment. We do not require that the contents of the counters are zero when the machine halts.

We will show that we can construct a deterministic (nondeterministic) bounded P system M which uses a fixed multiset w such that, when M is started with multiset wo^n, it simulates Z and has a halting computation if and only if Z halts on input n. (Again, we do not assume that the halting configuration of M to be in any special form.) Moreover, the rules of M are of the form $u \rightarrow v$, where $|u| = |v| = 1$ or 2. Clearly, it follows that the computation of M is linear-bounded in the sense that any reachable configuration has length exactly $|w| + n$ (i.e., the size of the computation space is always the same).

It is convenient to use an intermediate P system, which we shall call RCPS, a restricted version of the CPS (communicating P system) introduced in [13]. A CPS has multiple membranes labeled $1, 2, ...$, where 1 is the skin membrane. The rules in any membrane are of the forms: (1). $a \rightarrow a_x$, (2). $ab \rightarrow a_x b_y$, (3). $ab \rightarrow a_x b_y c_{come}$, where a, b, c are objects, x, y (which indicate the directions of movements of a and b) can be $here$, out, or in_j. The designation $here$ means that the object remains in the membrane containing it, out means that the object is transported to the membrane directly enclosing the membrane that contains the object (or to the environment if the object is in the skin membrane). The designation in_j means that the object is moved into the membrane, labeled j, that is directly enclosed by the membrane that contains the object. A rule of the form (3) can only appear in the skin membrane. When such a rule is applied, c is imported through the skin membrane from the environment (i.e., outer space) and will become an element in the skin membrane. In one step, all rules are applied in a maximally parallel manner. For notational convenience, when the target designation is not specified, we assume that the symbol remains in the membrane containing the rule.

Let V be the set of all objects (i.e., symbols) that can appear in the system, and o be a distinguished object (called the *input symbol*). A CPS M has m membranes, with a

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

It is known that a language $L \subseteq o^*$ is accepted by a deterministic (nondeterministic) CPS if and only if it is accepted by a deterministic (nondeterministic) multicounter machine. (Again, define the language accepted by a multicounter machine Z to be $L = \{o^n \mid Z \text{ when given } n \text{ has a halting computation }\}$). The "if" part was shown in [13]. The 'only if" part is easily verified. Hence, every unary recursively enumerable language can be accepted by a deterministic CPS (hence, also by a nondeterministic CPS).

In a recent paper [8], it was shown that $L \subseteq o^*$ is accepted by a deterministic (nondeterministic) linear-bounded multicounter machine if and only if it is accepted by a deterministic (nondeterministic) CPS which is restricted in that the environment does not contain any object initially. The system can expel objects into the environment but only expelled objects can be retrieved from the environment. The restricted system is called deterministic (nondeterministic) RCPS.

We can now modify the construction in [8] by introducing a new membrane in the skin membrane which would simulate the environment. This is possible since, in an RCPS, the environment does not contain any object initially and only o can be expelled into the environment and can be retrieved from the environment. It follows that the modified RCPS need only use rules of the form (1) and (2). But the modified RCPS, call it M, has multiple membranes. We will convert this to a 1-membrane system M'. Suppose that M has membranes $1, ..., m$. For each object a in V, M' will have symbols $a_1, ..., a_m$. In particular, for the distinguished input symbol o in V, M' will have $o_1, ..., o_m$. Hence the distinguished input symbol in M' is o_{i_0}, where i_0 is the index of the input membrane in M. We can convert M to the system M' as follows:

1. If $a \to a_x$ is a rule in membrane i of M, then $a_i \to a_j$ is a rule in M, where j is the index of the membrane into which a is transported to, as specified by x.
2. If $ab \to a_x a_y$ is a rule in membrane i of M, then $a_i b_i \to a_j b_k$ is a rule in M, where i and j are the index numbers of the membranes into which a and b are transported to, as specified by x and y.

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

Clearly, M' accepts $o_{i_0}^n$ if and only if M accepts o^n, and M' is a deterministic (nondeterministic) bounded P system. Now it is easy to show that the emptiness problem for deterministic linear-bounded multicounter machines (i.e., given Z, is there an input n such that Z halts?) is undecidable. Hence, we have:

Theorem 5. *It is undecidable to determine, given a deterministic (nondeterministic) BPS M and a fixed multiset w, whether there is an n such that M starting with multiset wo^n has a halting computation.*

For the next result, we need the fact that linear-bounded multicounter machines, $log\ n$ space-bounded TMs, and two-way multihead FAs are all equivalent (for both the deterministic and nondeterministic versions). As a corollary to Theorem 5, we can show that Theorem 4 does not hold for deterministic (nondeterministic) bounded P systems, even in very simple cases. Recall that $Halt$ is a regular formula in REG that defines all the halting configurations. For a fixed multiset w, the set of all wo^n is clearly definable by a regular formula I_w in REG. Theorem 5 essentially says that the emptiness of $[I_w \wedge \exists \diamond Halt]^M$ is undecidable. Hence, in contrast to Theorem 4, we have,

Corollary 1. *The* $\mathrm{CTL}_{-}^{\mathrm{REG}}$ *model-checking problem for (nondeterministic) bounded P systems is undecidable. The undecidability remains even for* $\mathrm{CTL}_{-}^{\mathrm{REG}}$ *formulas in the form of* $\mathrm{INIT} \wedge \exists \diamond H$ *where* INIT *and* H *are regular formulas in* REG.

We have seen that the emptiness problem for deterministic bounded P systems is undecidable. We now look at a special case when the cardinality of the maximally parallel multiset of rules applicable at each step is at most 1. Thus the computation of the system would be sequential. More generally, consider a (nondeterministic) bounded P system whose computation is restricted in that at every step, only one nondeterministically selected rule is applied. Call such a system a sequential bounded P system. In contrast to Theorem 5, We show that the emptiness problem for sequential bounded P system is decidable. In fact, this result is true even if the system is not bounded, i.e., in the rules of the form $u \rightarrow v$, we no longer require that $|v| \leq |u|$. We can show that such a sequential P system is equivalent to a partially blind multicounter machine (PBCM). Note that a PBCM [6] can increment/decrement any counter by 1 or leave it unchanged; however, it can not test a counter for zero. When there is an attempt to decrement a zero counter, the machine gets stuck and the computation is aborted. The machine starts with the input counter set to a value n with all other counters set to zero. We say that the machine accepts if it eventually halts in an accepting state with *all* the counters zero.

It can be shown that a language $L \subseteq o^*$ is accepted by a sequential P system if and only if it is accepted by a PBCM. Since the emptiness problem for PBCMs is decidable (as this problem is reducible to the reachability problem for vector addition systems (i.e., Petri nets)) [6], we have:

Theorem 6. *The emptiness problem for sequential P systems (and, hence, also for sequential bounded P systems) is decidable.*

A BPS M is *separated* if for any two distinct rules $u_i \rightarrow v_i$ and $u_j \rightarrow v_j$ in M, the multiset union of u_i and v_i is disjoint with the multiset union of u_j and v_j. For instance, the system with rules $ab \rightarrow ae$ and $cd \rightarrow d$ is separated. But the system with rules $ab \rightarrow ae$ and $cd \rightarrow e$ is not. In contrast to Corollary 1, we have the following result. Currently, we do not know whether the result still holds when we modify the above "separated" definition into the following: for any two distinct rules $u_i \rightarrow v_i$ and $u_j \rightarrow v_j$ in M, multisets u_i and u_j are disjoint.

Theorem 7. *For separated bounded P systems, the model-checking problem for formulas in the form of* $\mathrm{INIT} \wedge \exists \diamond H$*, where* INIT *and* H *are regular formulas in* REG*, is decidable.*

Notice that separated systems can demonstrate nonlinear reachability relations. For instance, consider such a system M with rules $ea \to a$ and $ccb \to cbd$. Define INIT to be $\#(b) = 1 \wedge \#(a) = 1 \wedge \#(d) = 0$ and H to be $\#(e) > 0 \wedge \#(c) \geq 2$. Then, the set of all $V \in [H]^M$ that is reachable from some $U \in [\mathrm{INIT}]^M$ (i.e., $U \leadsto_M V$) is exactly the set of V satisfying the following nonlinear relation: $\#(e) > 0 \wedge \#(c) \geq 2 \wedge \#(a) = 2^{\#(d)}$. We believe that Theorem 7 can be generalized to the entire $\mathrm{CTL}^{\mathrm{REG}}$, further investigation of which will be left for the full version of the paper.

We now investigate the case when a BPS M is *bounded maximally parallel*; i.e., there is a constant K such that on every execution of M, every maximally parallel move only fires at most K instances of rules. Examples of such M include purely catalytic systems [13, 14, 5], and following the same ideas of the proof of Theorem 5 but using constructions in [13, 14, 5], one can show that simple reachability queries like formulas $\mathrm{INIT} \wedge \exists \diamond H$ in $\mathrm{CTL}^{\mathrm{REG}}$ are undecidable for these M's. To make the query decidable, we add one more restriction. A maximally parallel move from $u = (t_1, \ldots, t_k)$ (the vector representation of the multiset u) to $v = (s_1, \ldots, s_k)$ is 1-non-monotonic if $t_2 \leq s_2, \ldots, t_k \leq s_k$. M is 1-non-monotonic if its executions consist of 1-non-monotonic maximally parallel moves only. With this restriction, we can show that linear reachability queries are decidable:

Theorem 8. *For bounded maximally parallel and 1-non-monotonic BPSs, the model-checking problem for formulas in the form of* $\mathrm{INIT} \wedge \exists \diamond H$*, where* INIT *and* H *are linear formulas in* LIN*, is decidable.*

Let N be a constant. A configuration $u = (t_1, \ldots, t_k)$ is 1-unbounded if each of $t_2, \ldots, t_k$ is bounded by N (i.e., only the first t_1 is possibly larger than N). M is 1-unbounded if its executions consist of 1-unbounded configurations only. In this case, we can generalize Theorem 8 to the full $\mathrm{CTL}^{\mathrm{LIN}}$.

Theorem 9. *The* $\mathrm{CTL}^{\mathrm{LIN}}$ *model-checking problem for bounded maximally parallel and 1-unbounded BPSs is decidable.*

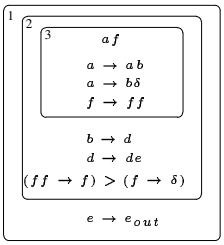
5 Experiments

From the results presented so far, even simple reachability queries like formulas $\mathrm{INIT} \wedge \exists \diamond H$ in $\mathrm{CTL}^{\mathrm{REG}}$ are undecidable for a bounded P system M in general. In this section, we investigate *conservative* behavior approximations that can be applied over M such that every execution of the approximated system is also an execution of the original M. Hence, such a conservative behavior approximation at least provides a way to help us analyze the original system, partially. This resembles similar approximation techniques in traditional model-checking of (in)finite state transition systems.

One such approximation is to let M to execute for at most B maximally parallel steps for a given constant B. Clearly, since B is a fixed, the reachability relation of M now is expressible as a Presburger formula, which can be calculated with a Presburger

manipulator like Omega [12]. Another approximation is to force M to crash whenever M reaches a multiset with more than S objects for a given constant B. Under this approximation, M can be simulated by a finite-state transition system and, accordingly, tools like the LTL model-checker SPIN [7] can be used to analyze it. In fact, these two approximations are applicable to a general P system (which is not necessarily a BPS) with multi-membranes, priority rules and dissolving membranes. Below, we briefly report our experiences in using Omega and SPIN to conservatively analyze a general P system, which is taken from literature [11]. Since Omega (resp. SPIN) has been proved effective in handling even fairly large infinite (resp. finite) real-world applications [2, 7], the primary purpose of our experiments is to identify whether these tools are also effective for a general P system with a reasonable size, where the inherent maximal parallelism makes the model highly nondeterministic, concurrent, and, more importantly, lack of control-flow structure.

The example P system M is shown in the figure below. It has three membranes where, in particular, membrane 2 (resp. membrane 3) are dissolved (i.e., objects in the membrane are immediately become objects in the outside membrane and the membrane along with the membrane's rules is all gone) whenever the rule in membrane 2 (resp. membrane 3) that contains δ fires. In membrane 2, the relation $ff \to f \ > \ f \to \delta$ says that, roughly, in a maximally parallel move, the former rule is given higher priority to fire than the latter rule. The P system is to compute a quadratic relation between certain objects; see [11] for details. Using Omega, we encode a maximally parallel move $\to_M$ in a Presburger relation which contains 34 variables (i.e., $\mathbf{N}^{17} \times \mathbf{N}^{17}$). Notice that a symbol may need up to three variables to represent, in order to specify its multiplicity in one of the three membranes. Additionally, a number of quantified variables are needed to encode the maximal parallelism, the priority rules and the dissolving membranes. Due to space limitation, we omit the detail of the Omega encoding. We used Omega to compute the reachability relation of M within B maximally parallel moves. Unfortunately, the tool crashed when computing with $B = 6$ (memory usage was 1.6GB including virtual memory), though it was successfully completed with $B = 5$ (in 489 CPU seconds).



To use SPIN, we encode M in Promela, the front-end specification language in SPIN. A Promela process is defined for each membrane, where the process exits when its corresponding membrane dissolves. Object-transfers across a membrane are simulated through rendezvous communications among processes, and the priority relation between evolution rules is implemented by carefully designed guards of the related selections. Again, we omit the detail of the Promela encoding. Using SPIN's default option, we checked the system for deadlock states. Unfortunately, SPIN could not finish any run within one hour as we varied the variable types from byte to short and long, respectively. Then, we checked a liveness property: eventually, the evolution of this P system will come to an end, i.e., only the skin membrane is left and no evolution rules in the skin can be applied; this is equivalent to checking that eventually all the three processes shall reach the ends of their bodies. Surprisingly, SPIN handled this property easily —

the total time consumed, as we varied the variable types from byte to short and long, increased merely from less than 0.1 second to several seconds and several minutes. The results of these checkings are all "false" since the inner membranes may not necessarily dissolve. Another property we checked about this P system is that: whenever the evolution of this P system comes to an end, the number of e objects outside the skin membrane is the square of the number of d objects inside the skin membrane. Again, SPIN gave the correct answer ("true") fairly fast (in less than 1 second) for each of the three cases (byte, short, long).

Through these preliminary experiments, we prefer SPIN over Omega to serve as the back-end solver in a future P system model-checker. On the other hand, Omega has its own strength in handling infinite state systems. Still, more research is needed for both approximation methods to create a more efficient encoding. All our experiments were run on a PC server with two 1GHz PIII processors running Linux with 1GB physical memory. The encodings can be found in the long version of the paper, which is available at `www.eecs.wsu.edu/~zdang`.

References

1. O. Andrei, G. Ciobanu, and D. Lucanu. Executable specifications of p systems. In *Proc. 5th Workshop on Membrane Computing*, pages 126–145, 2005.
2. T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: symbolic representations, approximations, and experimental results. *ACM Trans. Program. Lang. Syst.*, 21(4):747–789, 1999.
3. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.
4. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.
5. 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.
6. S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theor. Comput. Sci.*, 7:311–324, 1978.
7. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
8. O. H. Ibarra. The number of membranes matters. In *Proc. 4th Workshop on Membrane Computing*, pages 218–231, 2003.
9. Gh. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
10. Gh. Paun. *Membrane Computing: An Introduction*. Springer-Verlag, 2002.
11. Gh. Paun and G. Rozenberg. A guide to membrane computing. *TCS*, 287(1):73–100, 2002.
12. W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992.
13. P. Sosik. P systems versus register machines: two universality proofs. In *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Arges, Romania*, pages 371–382, 2002.
14. P. Sosik and R. Freund. P systems without priorities are computationally universal. In *WMC-CdeA2002*, volume 2597 of *LNCS*, pages 400–409. Springer, 2003.