# 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 much the same way as the promise of quantum and DNA computing) once future bio-technology gives way to practical bio-realization. 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 [11]. 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. As a branch of Natural Computing to explore new models, ideas, paradigms from the way nature computes,

⋆ 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`).

membrane computing has been quite successful: many models have been introduced, most of them Turing complete. (See http://psystems.disco.unimb/it for a large collection of papers in the area, and in particular the monograph [12].) Due to the maximal parallelism inherent in the model, P systems have a great potential for implementing massively concurrent systems in an efficient way that would allow us to solve currently intractable problems (in much the same way as the promise of quantum and DNA computing) once future bio-technology gives way to a practical bio-realization. Given this potential, the Institute for Scientific Information (ISI) has selected membrane computing as a fast "Emerging Research Front" in Computer Science (http://esi-topics.com/erf/october2003.html).

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 for P systems has 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 (in)finite 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, multimembranes 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 $u \to v$, 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 polyhedra encoding and SPIN, for 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. One of the purposes of the experiments is to let us know if the maximally parallelism and "lack of control-flow structure" in P systems would cause existing symbolic encodings 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.

**Most of the proofs can be found in the Appendix, which may be read by the PC members at their discretion.**

## 2 P Systems and Their CTL Model-checking Problems

We use $\mathbf{N}$ to denote the set of nonnegative integers. Let $\Sigma = \{a_1, \ldots, a_k\}$ 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 $\Sigma$); $u$ 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 $\Sigma$ 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 $\Sigma$ 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. (That is, linear formulas are closed under quantification.)

A P system $G$ 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 membranes may contain other membranes. The dynamics of $G$ is governed by a set of rules associated with each membrane. Each rule specifies how objects evolve and move into neighboring membranes. 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 $u \to v$ where $u$ and $v$ are multisets over alphabet $\Sigma$. A configuration in $M$ is a multiset. As with the standard semantics of P systems [11–13], 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 $u_i \to v_i$, $1 \leq i \leq m$, 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 $u_i \to v_i$, for each $1 \leq i \leq m$. Let $U$ and $V$ be two configurations (multisets) over $\Sigma$. 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 $U \to_M V$, if there is a maximally enabled rule multiset $R$ such that $V$ is the result of applying $R$ over $U$. Formally, $U \to_M V$ iff

$$\exists r_1, \ldots, r_m \in \mathbf{N}.\ \mathrm{MaxEnable}(r_1, \ldots, r_m, U) \wedge\ \mathrm{Apply}(r_1, \ldots, r_m, U, V), \qquad (1)$$

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, the formula in (1) is a Presburger formula. Hence, 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: $f ::= A \mid f \wedge f \mid f \vee f \mid \neg f \mid \exists \circ f \mid \forall \circ f \mid \exists\diamond f \mid \forall\Box f$, 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 $M$)
- $[\exists \circ f_1]^M$ is the set of configurations $U_1$ such that, for some execution $U_1 \rightarrow_M U_2 \rightarrow_M \ldots$, we have $U_2 \in [f_1]^M$;
- $[\forall \circ f_1]^M$ is the set of configurations $U_1$ such that, for any execution $U_1 \rightarrow_M U_2 \rightarrow_M \ldots$, we have $U_2 \in [f_1]^M$;
- $[f_1 \;\exists\mathcal{U}\; f_2]^M$ is the set of configurations $U_1$ such that, for some 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 $n$;
- $[f_1 \;\forall\mathcal{U}\; f_2]^M$ is the set of configurations $U_1$ such that, for 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 $n$.

(Notice that $U \rightarrow_M U$ is always true by definition when $U$ is halting; i.e., none of the rules is enabled under $U$.) The $\mathrm{CTL}^{\mathcal{A}}$ model-checking problem is to decide the answer of the following question:

**Given**: a P system $M$, a $\mathrm{CTL}^{\mathcal{A}}$ formula $f$,

**Question**: is $[f]^M$ 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 $U_{\mathrm{init}}$ satisfies $f$ can also be formulated in our definition as follows: is $[A_{\mathrm{init}} \wedge f]^M$ empty? where $A_{\mathrm{init}}$ is an atomic regular predicate where $U_{\mathrm{init}}$ 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 $u \to v$ 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 proofs 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* $\mathrm{INIT}$, $A$ *and* $H$ *are regular formulas in* $\mathrm{REG}$.

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* $\mathrm{LIN}$.

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

We do not currently know whether Theorem 2 still holds when $\mathrm{CTL}^{\mathrm{LIN}}$ model-checking problem is considered (i.e., Theorem 1 can be further strengthened to special non-cooperative BPSs). We say that $M$ is a *$\Lambda$-free-special* non-cooperative BPS if it is special and it does not have erasing rules (i.e., rules like $a \to \Lambda$). For such a $M$, Theorem 2 generalizes to $\mathrm{CTL}^{\mathrm{LIN}}$.

**Theorem 5.** *The* $\mathrm{CTL}^{\mathrm{LIN}}$ *model-checking problem for* $\Lambda$*-free-special and noncooperative 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 $u \to v$, 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 $u \to v$, 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 [15]. A CPS has multiple membranes labeled $1, 2, ...$, where 1 is the skin membrane. The rules in any membrane are of the forms: (1). $a \to a_x$, (2). $ab \to a_x b_y$, (3). $ab \to a_x b_y c_{come}$, where $a, b, c$ are objects, $x, y$ (which indicate the directions of movements of $a$ and $b$) can be $here$, $out$, or $in_j$. The designation $here$ means that the object remains in the membrane containing it, $out$ means that the object is transported to the membrane directly enclosing the membrane that contains the object (or to the environment if the object is in the skin membrane). The designation $in_j$ means that the object is moved into the membrane, labeled $j$, that is directly enclosed by the membrane that contains the object. A rule of the form (3) can only appear in the skin membrane. When such a rule is applied, $c$ is imported through the skin membrane from the environment (i.e., outer space) and will become an element in the skin membrane. In one step, all rules are applied in a maximally parallel manner. For notational convenience, when the target designation is not specified, we assume that the symbol remains in the membrane containing the rule.

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

It is known that a language $L \subseteq o^*$ is accepted by a deterministic (nondeterministic) CPS if and only if it is accepted by a deterministic (nondeterministic) multicounter machine. (Again, define the language accepted by a multicounter machine $Z$ to be $L = \{o^n \mid Z$ when given $n$ has a halting computation $\})$. The "if" part was shown in [15]. 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 [9], 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 [9] 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 (using Theorem 7 below) 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 6.** *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 following theorem,

**Theorem 7.** *Let $L \subseteq o^*$. Then the following statements are equivalent: (1). $L$ is accepted by a linear-bounded multicounter machine, (2). $L$ is accepted by a $log\ n$ space-bounded TM (with a read-only input with left and right end markers), (3). $L$ is accepted by a two-way multihead FA (with a read-only input with left and right end markers). These equivalences hold for both the deterministic and nondeterministic versions.*

Theorem 7 can be generalized to non-unary inputs, i.e., inputs of the form $a_1^{i_1}...a_k^{i_k}$, where $a_1, ..., a_k$ are distinct symbols.

As a corollary to Theorem 6, 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 6 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 6, 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 $u \to v$, 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 8.** *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 $u_i \to v_i$ and $u_j \to v_j$ 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 $ab \to ae$ and $cd \to d$ is separated. But the system with rules $ab \to ae$ and $cd \to e$ 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 $u_i \to v_i$ and $u_j \to v_j$ in $M$, multisets $u_i$ and $u_j$ are disjoint.

**Theorem 9.** *For separated bounded P systems, the model-checking problem for formulas in the form of* $\mathrm{INIT} \wedge \exists \diamond H$*, where* $\mathrm{INIT}$ *and* $H$ *are regular formulas in* $\mathrm{REG}$*, 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., $U \rightsquigarrow_M V$) 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 9 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 [15, 16, 5], and following the same ideas of the proof of Theorem 6 but using constructions in [15, 16, 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 10.** *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* $\mathrm{INIT}$ *and* $H$ *are linear formulas in* $\mathrm{LIN}$*, 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 10 to the full $\mathrm{CTL}^{\mathrm{LIN}}$.

**Theorem 11.** *The* $\mathrm{CTL}^{\mathrm{LIN}}$ *model-checking problem for bounded maximally parallel and 1-unbounded BPSs is decidable.*

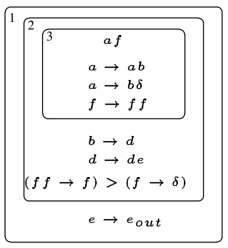
## 5 Experiments

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

[14]. 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 [13]. 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 $\delta$ 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 [13] 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. In particular, a Promela process is defined for each membrane, where the process exits when its corresponding membrane dissolves. Object-transfers across a membrane are simulated as 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 Appendix, which is not part of the paper.

## References

1. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model-checking. In *Concurrency (CONCUR 1997)*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer-Verlag, 1997.
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. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.
9. O. H. Ibarra. The number of membranes matters. In *Proc. 4th Workshop on Membrane Computing*, pages 218–231, 2003.
10. 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.
11. Gh. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
12. Gh. Paun. *Membrane Computing: An Introduction*. Springer-Verlag, 2002.
13. Gh. Paun and G. Rozenberg. A guide to membrane computing. *TCS*, 287(1):73–100, 2002.
14. W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992.
15. 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.
16. P. Sosik and R. Freund. P systems without priorities are computationally universal. In *WMC-CdeA2002*, volume 2597 of *LNCS*, pages 400–409. Springer, 2003.

## Appendix: Proofs and code not presented in the paper

**Proof of Theorem 1**:

*Proof.* The proof reduces the halting problem of two-counter machines to the $\mathrm{CTL}^{\mathrm{REG}}$ model-checking problem.

Consider a two-counter machine $\mathcal{M}$ that is a nondeterministic program with counters $x_1$ and $x_2$. Each counter stores a nonnegative integer value and can be incremented and decremented by 1 and tested against 0. (When $\mathcal{M}$ tries to decrement a 0-value counter, it crashes.) More precisely, the transition table of $\mathcal{M}$ contains finitely many instructions, each of which is in one of the following forms:

$s_i : x := x + 1;\ \texttt{goto}\ s_j;$
$s_i : x := x - 1;\ \texttt{goto}\ s_j;$
$s_i : (x == 0?);\ \texttt{goto}\ s_j;$ (fires only when $x == 0$)
$s_i : (x > 0?);\ \texttt{goto}\ s_j;$ (fires only when $x > 0$)

where $x$ is a counter, each $s$ is a state in $\mathcal{M}$ (there are only a finite number of them). In the above, we say that the instruction *leads from* $s_i$ *to* $s_j$. In particular, two states, $s_{\mathrm{start}}$ and $s_{\mathrm{halt}}$, are designated as the starting state and the halting state, respectively. HALT is the problem that decides whether there is an execution from $s_{\mathrm{start}}$, with both counters initially being 0, to $s_{\mathrm{halt}}$. It is well-known that HALT is undecidable [10].

Given a two-counter machine $\mathcal{M}$, we now construct a non-cooperative BPS $M$ and a $\mathrm{CTL}^{\mathrm{REG}}$ formula $f$ as follows. Each state $s$ in $\mathcal{M}$ is a symbol $s$ in $M$. Suppose that $\mathcal{M}$ has $t$ instructions, $I_1, \ldots, I_t$, for some $t$. For each instruction $I_h$, we create a new symbol, also denoted by $I_h$, in $M$. The rules in $M$ are created for each instruction $I_h$. Suppose that instruction $I_h$ is in one of the forms shown in above and leads from state $s_i$ to state $s_j$. The instruction is simulated by two (maximally parallel) moves in $M$; i.e., we add the following (totally $2t$) rules to $M$, where $1 \le h \le t$:

$s_i \to I_h$
$I_h \to s_j$.

Initially in $M$, $\#(s_{\mathrm{start}}) = 1$ (one copy of the symbol $s_{\mathrm{start}}$), and $\#(s_i) = 0$ for each other $s_i$ and $\#(I_h) = 0$ for each $h$. Clearly, the above rules only keep the state transition (instead of counter behavior) for each instruction in $\mathcal{M}$.

We now take care of the counter behavior for each instruction in $\mathcal{M}$, by adding the following (totally 16 ) rules to $M$:

(the increment rule-set)
$c_x \to d_x$
$c_x \to +_x$
$+_x \to a_x$
$d_x \to c_x$
(the decrement rule-set)
$a_x \to b_x$
$a_x \to -_x$
$-_x \to e_x$
$b_x \to a_x$

where $x \in \{x_1, x_2\}$. Hence, in $M$, we have totally $2t + 16$ rules. Notice that $\#(a_x)$ in $M$ corresponds to the counter value $x$ in $\mathcal{M}$. For each counter $x$, the symbols $c_x, d_x, +_x, b_x, -_x, e_x$ are all auxiliary. Initially, the multiplicities for $a_x, d_x, +_x, b_x, -_x, e_x$ are all 0. Hence, only the multiplicities for $c_x$ are possibly not 0. The two rule sets, for each $x$, run in maximally parallel, as follows. The increment rule-set evolves every $c_x$ (the $c_x$'s serve as the supplies for all the increments over $x$) into an intermediate $d_x$ or into $+_x$. Then, each evolved $+_x$ is changed into $a_x$ (i.e., an "increment" is made to $x$) and each intermediate $d_x$ is changed back to $c_x$. In parallel to these, the decrement rule-set evolves every $a_x$ into an intermediate $b_x$ or into $-_x$. Then, each evolved $-_x$ is changed into $e_x$ (since $e_x$ is not enabled at any time, the rule set essentially makes a "decrement" to $x$). The counter behavior in each instruction of $\mathcal{M}$ is simulated by two maximally parallel moves of the 16 rules.

Unfortunately, the counter behavior is not faithfully simulated by the 16 rules. There are two reasons. First, the state transitions that are kept in the $2t$ rules shown earlier are not effectively associated with the 16 increment/decrement rules. Second, the amount of "increment" and "decrement", due to the nature of maximally parallelism, is not necessarily 1 that is indicated in an increment/decrement instruction. To overcome the problems, we construct a $\mathrm{CTL}^{\mathrm{REG}}$ formula $f$ to "regulate" $M$'s executions.

For each instruction $I_h$, we construct a regular $A_h$ (i.e., a Boolean combination of atomic regular predicates) as follows (we only show the construction for instructions concerning the counter $x_1$; the case for $x_2$ can be handled symmetrically):

- The instruction $I_h$ is in the form of
$$s_i : x_1 := x_1 + 1;\ \texttt{goto}\ s_j.$$
In this case, we define the formula $A_h$ to be
$$\#(I_h) = 1 \Rightarrow (\#(+_{x_1}) = 1 \wedge \#(-_{x_1}) = 0 \wedge \#(+_{x_2}) = 0 \wedge \#(-_{x_2}) = 0).$$
That is, when the $2t$ rules evolve the initial object $s_{\mathrm{start}}$ into object $I_h$ (i.e., the instruction $I_h$ is being simulated), we require that the amount of "increment" to $x_1$, when the increment rule-set for $x_1$ runs, is 1 (i.e., $\#(+_{x_1}) = 1$ shown in the formula) and the amount of "decrement" to $x_1$, when the decrement rule-set for $x_1$ runs, is 0 (i.e., $\#(-_{x_1}) = 1$ shown in the formula). Additionally, the counter $x_2$ does not change (i.e., $\#(+_{x_2}) = 0 \wedge \#(-_{x_2}) = 0$ shown in the formula).
- The instruction $I_h$ is in the form of
$$s_i : x_1 := x_1 - 1;\ \texttt{goto}\ s_j.$$
In this case, we define the formula $A_h$, similar to the above case, to be
$$\#(I_h) = 1 \Rightarrow (\#(-_{x_1}) = 1 \wedge \#(+_{x_1}) = 0 \wedge \#(+_{x_2}) = 0 \wedge \#(-_{x_2}) = 0).$$
- The instruction $I_h$ is in the form of
$$s_i : (x_1 == 0?);\ \texttt{goto}\ s_j.$$
In this case, we define the formula $A_h$ to be
$$\#(I_h) = 1 \Rightarrow (\#(b_{x_1}) = 0 \wedge \#(-_{x_1}) = 0 \wedge \#(+_{x_1}) = 0 \wedge \#(+_{x_2}) = 0 \wedge \#(-_{x_2}) = 0).$$
Notice that when $\#(I_h) = 1$, the number $\#(b_{x_1})$ of intermediate symbols $b_{x_1}$ corresponds to the number of $a_{x_1}$ (the current value of $x_1$ when $I_h$ is executed), as long as both $x_1$ and $x_2$ do not change (i.e., $\#(-_{x_1}) = 0 \wedge \#(+_{x_1}) = 0 \wedge \#(+_{x_2}) = 0 \wedge \#(-_{x_2}) = 0$).

– The instruction $I_h$ is in the form of

$$s_i : (x_1 > 0?);\ \texttt{goto}\ s_j.$$

In this case, we define the formula $A_h$ to be

$$\#(I_h) = 1 \Rightarrow (\#(b_{x_1}) > 0 \wedge \#(-_{x_1}) = 0 \wedge \#(+_{x_1}) = 0 \wedge \#(+_{x_2}) = 0 \wedge \#(-_{x_2}) = 0).$$

We use $A$ to denote the conjunction of all the $A_h$'s; i.e.,

$$A = \bigwedge_{1 \le h \le t} A_h.$$

Notice that the halting condition of $\mathcal{M}$ can be expressed as an atomic regular predicate $H$ that is $\#(s_{\text{halt}}) = 1$. As we have mentioned earlier, initially, $M$ starts with one copy of $s_{\text{start}}$; i.e.,

$$\#(s_{\text{start}}) = 1 \wedge \bigwedge_{s_{\text{start}} \neq q \in \{s_1, \ldots, s_k, I_1, \ldots, I_t\}} \#(q) = 0. \tag{2}$$

In the above, $s_1, \ldots, s_k$ are all the states in $\mathcal{M}$. Additionally, the initial multiplicities for symbols $a_x, d_x, +_x, b_x, -_x, e_x$, where $x \in \{x_1, x_2\}$, are all 0; i.e.,

$$\bigwedge_{q \in \{a_x, d_x, +_x, b_x, -_x, e_x\}, x \in \{x_1, x_2\}} \#(q) = 0. \tag{3}$$

Notice that, initially, we do not restrict $\#(c_x)$. We use INIT to denote the conjunction of (2) and (3). To faithfully simulate $\mathcal{M}$, we would like $M$ to start with some multiset (e.g., some multiplicities for symbols $c_{x_1}$ and $c_{x_2}$) satisfying INIT, and then the "regulator" $A$ is always been satisfied, and finally, the halting condition $H$ is satisfied; i.e., $M$ satisfies the CTL$^{\text{REG}}$ formula $f$ defined as INIT $\wedge$ $(A\ \exists\mathcal{U}\ H)$. From above, it is clear that HALT holds for counter machine $\mathcal{M}$ iff $[f]^M$ is not empty for the non-cooperative BPS $M$. Since INIT, $A$ and $H$ are all regular formulas in REG, the result follows. ∎

**Proof of Theorem 2**:

*Proof.* Let $M$ be a special and non-cooperative BPS whose alphabet is $\Sigma$ and $f$ be a CTL$^{\text{REG}}$ formula. The CTL$^{\text{REG}}$ formula $f$ is built up from atomic regular predicates in the form of $\#(a) \sim n$, where $a \in \Sigma$, $n \in \mathbf{N}$ and $\sim \in \{>, <, =, \ge, \le\}$, using Boolean connectors and temporal operators. We use constant $B$ to denote the maximal constant $n$ that appears in the atomic regular predicates in $f$.

Before we proceed further with the proof, we need some more definitions. Let $\Sigma = \{a_1, \ldots, a_k\}$. Consider a formula in the form of

$$\bigwedge_{1 \le i \le k} \#(a_i) \sim_i n_i \tag{4}$$

where each $\sim_i \in \{=, \ge, >\}$ and each $0 \le n_i \le B$. A *B-tube* is the set of tuples $(\#(a_1), \ldots, \#(a_k))$ in $\mathbf{N}$ that satisfy a formula in (4). Clearly, a configuration of $M$ is a multiset over $\Sigma$, which can be represented as a vector in $\mathbf{N}^k$. We use the notation

$$[n_1^{\sim_1}, \ldots, n_k^{\sim_k}] \tag{5}$$

to represent the tube. For instance, $[2^{>}, 4^{\geq}, 1^{=}]$ represents all the multisets with more than 2 $a_1$'s, at least 4 $a_2$'s, and 1 $a_3$. The empty set $\emptyset$ or a finite union of $B$-tubes is called a *$B$-regular set*, which serves as the symbolic representation of the interpretation set $[f]^M$ (which is also a set of configurations) in the following model-checking procedure for $\mathrm{CTL}^{\mathrm{REG}}$ formula $f$, which is analogous to the procedure proposed in [2].

Observe that, for each fixed $B$, $B$-regular sets are closed under Boolean combinations (i.e., $\neg$ (complement), $\cap$, $\cup$). To prove that $[f]^M$ is a $B$-regular set for every $\mathrm{CTL}^{\mathrm{REG}}$ formula $f$, we use an induction over the definition of $f$:

- $f$ is an atomic regular predicate. Clearly, $[f]^M$ is a $B$-regular set.
- $f$ is $\neg f_1, f_1 \wedge f_2$ or $f_1 \vee f_2$, and both $[f_1]^M$ and $[f_2]^M$ are $B$-regular sets. Using the observation, we know that $[f]^M$ is also a $B$-regular set, according to the definition of $[f]^M$.
- $f$ is $\exists \circ f_1$ and $[f_1]^M$ is a $B$-regular set. Then, by definition, $[\exists \circ f_1]^M = \{u : \exists v, u \rightarrow_M v \ \wedge \ v \in [f_1]^M\}$. We use $Pre([f_1]^M)$ to denote the set. It suffices for us to show that $Pre([f_1]^M)$ is a $B$-regular set when $[f_1]^M$ is a $B$-tube; e.g., in the form of $[n_1^{\sim_1}, \ldots, n_k^{\sim_k}]$ with each $0 \leq n_i \leq B$. We will construct each constituent $B$-tube $[m_1^{\sim'_1}, \ldots, m_k^{\sim'_k}]$ in $Pre([f_1]^M)$. For each symbol $b$, we use $H^{-1}(b)$ to denote all the symbols $a$ such that $a \rightarrow b$ is a rule in $M$. $H^{-1}(b)$ could be an empty set. Because $M$ is special, all of the sets $H^{-1}(b)$ are disjoint for distinct $b$'s. Now, we fix any $i$ with $H^{-1}(a_i) \neq \emptyset$. Suppose that $a_{i_1}, \ldots, a_{i_t}$ (for some $t$) are all the symbols in $H^{-1}(a_i)$. Then the $B$-tube $[m_1^{\sim'_1}, \ldots, m_k^{\sim'_k}]$ must satisfy the following conditions:
  - if $\sim_i$ is $=$, then $\sum_{1 \leq j \leq t} m_{i_j} = n_i$ and, for each $j$,
    - $\sim'_{i_j}$ is $=$ when $a_{i_j} \rightarrow \Lambda$ is not a rule in $M$.
    - $\sim'_{i_j}$ is $\geq$ when otherwise.

    This is because each of the $n_i$ copies of symbol $a_i$ is evolved from one of the $a_{i_j}$'s.
  - if $\sim_i$ is $\geq$, then $\sum_{1 \leq j \leq t} m_{i_j} = n_i$ and each $\sim'_{i_j}$ is $\geq$.
  - if $\sim_i$ is $>$, then $\sum_{1 \leq j \leq t} m_{i_j} = n_i$ and, among $\sim'_{i_1}, \ldots, \sim'_{i_t}$, there is exactly one $>$ and the rest are all $\geq$.

  The above three conditions are called CONDITION($i$) for $i$ satisfying $H^{-1}(a_i) \neq \emptyset$. Now, we define CONDITION($i$) for $i$ satisfying $H^{-1}(a_i) = \emptyset$. That is, there is no rule like $a \rightarrow a_i$ in $M$, for any $a$. There are two subcases:
  - $a_i \rightarrow \Lambda$ is a rule in $M$. In this case, due to maximal parallelism, there shouldn't be any $a_i$-objects in a configuration that is reached from a configuration in $[m_1^{\sim'_1}, \ldots, m_k^{\sim'_k}]$. Hence, if $\sim_i$ is $>$ or $n_i > 0$, then we have already calculated $Pre([f_1]^M)$, which is simply $\emptyset$. That is, the CONDITION($i$) is simply false. Otherwise, the CONDITION($i$) is as follows: $m_i = 0$ and $\sim'_i$ is $\geq$.
  - $a_i \rightarrow \Lambda$ is not a rule in $M$; i.e., $a_i$ is not enabled by any rule in $M$. In this case, every copy of $a_i$ is kept after a maximal parallel move; i.e., the CONDITION($i$) is $m_i = n_i$ and $\sim'_i = \sim_i$.

  We use CONDITION to denote the conjunction of all CONDITION($i$) for all $i$. When CONDITION is false, $Pre([f_1]^M)$ is empty. Otherwise, $Pre([f_1]^M)$ is exactly the union of all the $[m_1^{\sim'_1}, \ldots, m_k^{\sim'_k}]$ that satisfy CONDITION. Notice that such $[m_1^{\sim'_1}, \ldots, m_k^{\sim'_k}]$ is always a $B$-tube. Therefore, $B$-regular sets are closed under $Pre$; i.e., $[f]^M = Pre([f_1]^M)$ is also a $B$-regular set.
- $f$ is $\forall \circ f_1$ and $[f_1]^M$ is a $B$-regular set. Noticing that $[f]^M = \neg Pre(\neg [f_1]^M)$, the result that $[f]^M$ is also a $B$-regular set can be followed easily.

- $f$ is $f_1\ \exists\mathcal{U}\ f_2$ and both $[f_1]^M$ and $[f_2]^M$ are $B$-regular sets. Using [2], the set $[f]^M$ can be calculated as the limit of the monotonic sequence $Q_0, Q_1, \ldots$, where $Q_0 = [f_2]^M$, and each $Q_{i+1} = Q_i \cup ([f]^M \cap Pre(Q_i))$. Since, as shown above, $B$-regular sets are closed under Boolean operations and $Pre$. Therefore, each $Q_i$ is a $B$-regular set. Also, since there are only a finite number of distinct $B$-regular sets (for the given $B$), the monotonic sequence $Q_0, \ldots$ of $B$-regular sets must converge; i.e., $[f]^M = Q_{i_0}$ for some $i_0$. Hence, $[f]^M$ also a $B$-regular set.
- $f$ is $f_1\ \forall\mathcal{U}\ f_2$ and both $[f_1]^M$ and $[f_2]^M$ are $B$-regular sets. This case can be argued similarly, since, using [2], the set $[f]^M$ is the limit of the monotonic sequence $Q_0, Q_1, \ldots$, where $Q_0 = [f_2]^M$, and each $Q_{i+1} = Q_i \cup ([f_1]^M \cap Pre(Q_i) \cap (\neg Pre(\neg Q_i)))$.

Hence, for the CTL$^{\mathrm{REG}}$ formula $f$, its interpretation $[f]^M$ is always a $B$-regular set. The decidability follows, since it takes linear time to check whether the calculated $B$-regular set $[f]^M$ is empty or not. However, the above induction process is hard for complexity analysis when treated as a symbolic model-checking procedure. In fact, there is a much easier algorithm.

For a multiset $u$, we use $|u|$ to denote the total number of objects in $u$. Observe that, for any $B$-regular set $S$, $S$ is not empty iff there is a multiset $u \in S$ with $|u| \leq k \cdot (B+1)$, where $k = |\Sigma|$. Since the calculated $[f]^M$ is a $B$-regular set, to check the emptiness of $[f]^M$, it suffices to restrict the P system $M$ such that it only starts with multisets $u \in S$ with $|u| \leq k \cdot (B+1)$. Since $M$ is a BPS, every reachable multiset $v$ also satisfies $|v| \leq k \cdot (B+1)$. Therefore, the restricted $M$ is a finite state transition system, whose state space is bounded by $O((k \cdot (B+1))^k)$. Model-checking $f$ (i.e., the traditional CTL model-checking [3]) over the finite state transition system takes time $O(t \cdot (k \cdot (B+1))^k)$, where $t$ is the size of formula $f$. ∎

**Proof of Theorem 3**:

*Proof.* Without loss of generality, we may assume that the rules in $M$ are in the form of $a \to b$ (i.e., $M$ does not have erasing rules like $a \to \Lambda$). Let $\Gamma$ be an alphabet in which each symbol is a pair of two symbols in $\Sigma$; i.e., $\Gamma = \Sigma \times \Sigma$. We now build a finite automaton $A$ that works on an input word $w$ over alphabet $\Gamma$, as follows. First, $A$ guesses a nonempty subset $T$ of $\Gamma$, called the reachability table. Then, for every pair $(a, b) \in T$, $A$ nondeterministically simulates a copy of $M$ over the object $a$; all the copies run in parallel. Nondeterministically at some moment, $A$ shuts down the simulations simultaneously for all the copies. Then, $A$ makes sure that, for each pair $(a, b)$, the copy of $M$ for the pair ends up with the object $b$. Now, $A$ starts to read the input $w$. For each symbol in $\Gamma$ (i.e., a pair $(a, b) \in \Sigma \times \Sigma$) that $A$ reads, $A$ makes sure that the pair is in the table $T$. At the end of the input, $A$ accepts. We use $u$ (resp. $v$) to denote the multiset formed by collecting all the objects in the first (resp. second) coordinate for all pairs in $w$. Clearly, $w$ is accepted by $A$ iff $u \rightsquigarrow_M v$. Notice that the language $L(A)$ accepted by $A$ is a regular and hence semilinear language. From here, one can show that $\rightsquigarrow_M \subseteq \mathbf{N}^k \times \mathbf{N}^k$ defines a semilinear set; i.e., $\rightsquigarrow_M$ is definable by a linear formula. ∎

**Proof of Theorem 4**:

*Proof.* Let $M$ be a non-cooperative BPS, whose alphabet is $\Sigma = \{a_1, \ldots, a_k\}$. We will show that, for any CTL$_{-}^{\mathrm{LIN}}$ formula $f$, the set $[f]^M$ is effectively semilinear (i.e., definable by a linear formula). Therefore, testing the emptiness of $[f]^M$ is immediately decidable. Hence, the result follows. We use an induction on the definition of CTL$_{-}^{\mathrm{LIN}}$ formulas $f$. For the base case when $f$ is an atomic linear formula, $[f]^M$ is clearly semilinear (i.e., definable by the atomic linear formula). The cases when $f$ is $\neg f_1, f_1 \wedge f_2$ or $f_1 \vee f_2$ are all straightforward, since semilinear sets are closed under Boolean operations. We now focus on the remaining cases of $f$:

- $f$ is $\exists \circ f_1$ and $[f_1]^M$ is semilinear. Notice that $[f]^M = \{u : \exists v. u \rightarrow_M v \wedge v \in [f_1]^M\}$. Observe that the one-step maximally parallel move $\rightarrow_M$ is definable by a linear formula and hence semilinear. Using the fact that semilinear sets are closed under Boolean operations as well as quantification, we conclude that $[f]^M$ is also semilinear. The case when $f$ is $\forall \circ f_1$ can be handled similarly.
- $f$ is $\exists \diamond f_1$ and $[f_1]^M$ is semilinear. Notice that $[f]^M = \{u : \exists v. u \rightsquigarrow_M v \wedge v \in [f_1]^M\}$. From Theorem 3, $\rightsquigarrow_M$ is definable by a linear formula. From the arguments made in the above item (by replacing $\rightarrow_M$ with $\rightsquigarrow_M$), we conclude that $[f]^M$ is also semilinear. The case when $f$ is $\forall \Box f_1$ can be handled similarly.

∎

**Proof of Theorem 5**:

*Proof.* Let $M$ be a $\Lambda$-free-special and non-cooperative BPS, whose alphabet is $\Sigma = \{a_1, \ldots, a_k\}$. We will show that, for any $\text{CTL}^{\text{LIN}}$ formula $f$, the set $[f]^M$ is effectively semilinear (i.e., definable by a linear formula). Therefore, the result follows. Similar to the proof of Theorem 4, we use an induction on the definition of $f$. We only focus on the case when $f$ is $f_1 \exists \mathcal{U} f_2$ and both $[f_1]^M$ and $[f_2]^M$ are semilinear; the other cases are either easier or similar.

We first define a function $H : \Sigma \rightarrow \Sigma$ as follows. For each $a$, if $a \rightarrow b$ (for some $b$) is a rule in $M$, then $H(a) = b$. If there no such $b$, then $H(a) = a$. Since $M$ is $\Lambda$-free-special, the $H$ is unique. For a multiset $u$, we use $H(u)$ to denote the resulting multiset after replacing simultaneously every object $a$ with object $H(a)$. In particular, $H^i(u)$ is the result of applying $H$ over $u$ for $i$ times; by default, $H^0(u) = u$.

Notice that, starting from a multiset $u$, the execution of the $M$ is unique:

$$H^0(u), H^1(u), \ldots.$$

In particular, there is a constant $n_0$ (which is independent of the size of $u$) such that, for any $i > n_0$, $H^i(u)$ equals some multiset $H^j(u)$ with $j \leq n_0$. That is, the prefix with length $n_0 + 1$ carries "all" the information of the entire execution. Hence, to compute $[f_1 \exists \mathcal{U} f_2]^M$, one needs only to look at the prefix. More precisely, for any $u$, the following items are equivalent:

- $u \in [f_1 \exists \mathcal{U} f_2]^M$,
- $u$ satisfies the following:

$$\exists i_0 \leq n_0 . \forall 0 \leq i \leq i_0 . H^i(u) \in [f_1]^M \wedge H^{i_0+1}(u) \in [f_2]^M. \tag{6}$$

The result follows, since all the $u$'s satisfying (6) can be definable by a Presburger formula (noticing that, in (6), the $n_0$ is a constant independent of $u$). Hence, $[f]^M$ (i.e., $[f_1 \exists \mathcal{U} f_2]^M$) is semilinear. ∎

**Proof of Theorem 7**:

*Proof.* The equivalence of (2) and (3) is well known. That a linear-bounded multicounter machine can be simulated by a two-way multihead FA is obvious. We now show the converse.

Define a generalized linear-bounded multicounter machine as follows. As before, at the start of the computation, the input counter is set to a value $n$ (for some $n$), and all other counters are set to zero. Now we only require that there is a positive integer $c$ such that at any time during the computation, the value of any counter is at most $cn$. (Thus, we no longer require that the sum of the values of the counters is at most $n$.)

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

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

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

Now we show that a two-way multihead FA $W$ with $m$ heads $H_1, ..., H_m$ can be simulated by a generalized multicounter machine $Z$ with $2m$+1 counters, $D, C_1, ...C_m$, $E_1, ..., E_m$. $Z$ with input $n$ in counter $D$, and zero in the other counters first decrements $D$ and stores $n$ in counters $C_1, .., C_m$. Then $Z$ simulates the actions of head $H_i$ of $W$ using the counters $C_i$ and $E_i$. ∎

**Proof of Theorem 9**:

*Proof.* (Sketch) Let $M$ be a separated BPS, whose alphabet is $\Sigma = \{a_1, \ldots, a_k\}$. Without loss of generality, we assume that, in INIT $\wedge \exists \diamond H$, INIT is given as

$$\bigwedge_{1 \le i \le k} \#(a_i) \sim_i^{\text{INIT}} n_i^{\text{INIT}}$$

and $H$ is given as

$$\bigwedge_{1 \le i \le k} \#(a_i) \sim_i^{H} n_i^{H}$$

where $\sim_i^{\text{INIT}}, \sim_i^{H} \in \{=, >\}$ and $n_i^{\text{INIT}}, n_i^{H} \in \mathbf{N}$. We first assume that there is only one rule, $u \to v$, in $M$. Clearly, one may treat multisets $u$ and $v$ as vectors $(t_1, \ldots, t_k)$ and $(s_1, \ldots, s_k)$, respectively, in $\mathbf{N}^k$. Let $m$ be a number. Consider the following statement, denoted by $P(m)$:

"there are two vectors (multisets) $U$ and $V$ such that, $U \in [\text{INIT}]^M$, $V \in [H]^M$, and $U$ can reach $V$ in at most $m$ maximally parallel moves."

Clearly, to decide $[\text{INIT} \wedge \exists \diamond H]^M \neq \emptyset$, one need only to decide whether $\exists P$ holds (i.e., $P(m)$ holds for some $m$). Observe that for any fixed $m$, the truth of $P(m)$ can be easily decided (in this case, all the pairs of $U$ and $V$ satisfying the condition form a Presburger relation).

We first assume that there is some $1 \le i \le k$ such that both $\sim_i^{\text{INIT}}$ and $\sim_i^{H}$ are $=$. With this assumption, $\exists P$ can be decided easily:

- $n_i^{\text{INIT}} = n_i^H$. In this case, if $t_i \neq s_i$, then $\exists P$ can be decided using $P(1)$ (using one maximally parallel move). Now, we handle the scenario when $t_i = s_i$. If $n_i^{\text{INIT}} < t_i$ (i.e., the rule is not enabled on $U$), then clearly $\exists P$ can be decided using $P(1)$. Otherwise (i.e., $n_i^{\text{INIT}} \geq t_i$), the $a_i$-objects are always kept the same in all the $V$'s that are reached from the same $U$ in $[\text{INIT}]^M$. That is, the symbol $a$ can be dropped from $\Sigma$ and the rule. Hence, using an induction over the size of $\Sigma$ (the theorem clearly holds when $k = 1$), $\exists P$ can be decided.
- $n_i^{\text{INIT}} > n_i^H$. In this case, $\exists P$ can be decided using $P(1)$ when $t_i \leq s_i$, or, $t_i > s_i$ and $n_i^{\text{INIT}} < t_i$. When $n_i^{\text{INIT}} \geq t_i > s_i$, $\exists P$ can be decided using $P(n_i^{\text{INIT}} - n_i^H)$ ( each maximally parallel move decrements at least one $a_i$).
- $n_i^{\text{INIT}} < n_i^H$. This case is analogous to the above item.

We then assume the assumption does not hold. That is, for each $1 \leq i \leq k$, either $\sim_i^{\text{INIT}}$ or $\sim_i^H$ is $>$. Similar to the above, we can argue that when, for some $1 \leq i \leq k$, one of the following conditions is true,

- $s_i = 0$,
- $\sim_i^{\text{INIT}}$ is $=$, $\sim_i^H$ is $>$, and $t_i \geq s_i$,
- $\sim_i^{\text{INIT}}$ is $>$, $\sim_i^H$ is $=$, and $t_i \leq s_i$,

$\exists P$ can be decided easily (either use induction on a smaller $\Sigma$ or decide $\exists P$ with $P(m_0)$ for some constant $m_0$). The remaining case is when, for each $1 \leq i \leq k$, $s_i > 0$ and one of the following conditions is true,

- $\sim_i^{\text{INIT}}$ is $=$, $\sim_i^H$ is $>$, and $t_i < s_i$,
- $\sim_i^{\text{INIT}}$ is $>$, $\sim_i^H$ is $=$, and $t_i > s_i$,
- $\sim_i^{\text{INIT}}$ is $>$ and $\sim_i^H$ is $>$.

We now define a new system $M'$ with the only rule

$$u' = (t'_1, \ldots, t'_k) \to v' = (s'_1, \ldots, s'_k),$$

and with new INIT$'$ as

$$\bigwedge_{1 \leq i \leq k} \#(a_i) \sim_i^{\text{INIT}'} n_i^{\text{INIT}'}$$

and new $H'$ as

$$\bigwedge_{1 \leq i \leq k} \#(a_i) \sim_i^{H'} n_i^{H'}.$$

The new system $M$ is modified from $M$ as follows:

- when $\sim_i^{\text{INIT}}$ is $=$ and $\sim_i^H$ is $>$, or, $\sim_i^{\text{INIT}}$ is $>$ and $\sim_i^H$ is $>$, we have $t'_i = t_i$ and $s'_i = s_i$, In this case, we also have $\sim_i^{\text{INIT}'} = \sim_i^{\text{INIT}}$, $n_i^{\text{INIT}'} = n_i^{\text{INIT}}$, $\sim_i^{H'} = \sim_i^H$, $n_i^{H'} = n_i^H$.
- $\sim_i^{\text{INIT}}$ is $>$ and $\sim_i^H$ is $=$, we will switch the $s_i$ and the $t_i$; i.e., $t'_i = s_i$ and $s'_i = t_i$. Also, $\sim_i^{\text{INIT}'} = \sim_i^H$, $n_i^{\text{INIT}'} = n_i^H$, $\sim_i^{H'} = sim_i^{\text{INIT}}$, and $n_i^{H'} = n_i^{\text{INIT}}$.

Similarly, we define $P'(m)$ to be

"there are two vectors (multisets) $U$ and $V$ such that, $U \in [\text{INIT}']^{M'}$, $V \in [H']^{M'}$, and $U$ can reach $V$ in at most $m$ maximally parallel moves."

It is left to the reader to verify that $\exists P$ holds for $M$ iff $\exists P'$ holds for the new $M'$. In fact, $\exists P'$ is always true for the new $M'$. Hence, $\exists P$ is decided simply to be true.

The proof can be generalized to the case when there are multiple rules in the separated system. ∎

**Proof of Theorem 10**:

*Proof.* (sketch) Let $M$ be a bounded maximally parallel and 1-non-monotonic BPS. We use counters $x_1, \ldots, x_k$ to denote $\#(a_1), \ldots, \#(a_k)$, respectively, in a configuration. One can show that $M$ can be faithfully simulated by a counter machine $A$ with counters $x_1, \ldots, x_k$, where the $x_1$ is the only unrestricted counter and the other counters $x_2, \ldots, x_k$ are nondecreasing. In particular, $A$ has instructions to test a counter against an integer constant. This capability is sufficient to simulate a bounded maximally parallel move (and check it is indeed maximally parallel) in $M$. Such an $A$ is studied in [8] and it is known that the reachability relation (over the counter values) of $A$ is definable by a linear formula (whose emptiness is decidable). The result follows. ∎

**Proof of Theorem 11**:

*Proof.* (sketch) Let $M$ be a bounded maximally parallel and 1-unbounded BPS. We use counters $x_1, \ldots, x_k$ to denote $\#(a_1), \ldots, \#(a_k)$, respectively, in a configuration. Similar to the above proof, one can show that $M$ can be faithfully simulated by a counter machine $A$ with counters $x_1, \ldots, x_k$, where the $x_1$ is the only unrestricted counter and the other counters $x_2, \ldots, x_k$ are all bounded by constant $N$. Therefore, $A$ is essentially a one-counter machine since one can build the other counters into $A$'s finite control. We now treat the unrestricted counter $x_1$ as a (unary) stack and hence $A$ is a stack machine. Let $f$ be a linear formula over variables $\#(a_1), \ldots, \#(a_k)$. Since $\#(a_1)$ (i.e., the $x_1$) is the only unbounded variable, essentially the formula can be rewritten into a one-variable linear formula over the $x_1$. When one treats $x_1$ as a unary word, the formula defines a regular set. That is, the $\text{CTL}^{\text{LIN}}$ model-checking problem for $M$ can be reduced to the CTL model-checking problem for the stack machine $A$ with respect to regular stack words. The result follows since the latter problem is known decidable [1]. ∎

**Omega code for the example P system in Section 5**

```
#Omega code for a maximally parallel move
#of the example P system


Max_Parallel_Move:={
[a1,a2,a3,b1,b2,b3,d1,d2,d3,e1,e2,e3,f1,f2,f3,delta2,delta3]->
[a1',a2',a3',b1',b2',b3',d1',d2',d3',e1',e2',
e3',f1',f2',f3',delta2',delta3']:
#we are talking about nonnegative integers here
```

```
a1>=0 && a2>=0 && a3>=0 && b1>=0 && b2>=0 && b3>=0 && d1>=0
&& d2>=0 && d3>=0 && e1>=0 && e2>=0 && e3>=0 && f1>=0
&& f2>=0 && f3>=0 &&  delta2>=0 && delta3>=0 &&
a1'>=0 && a2'>=0 &&  a3'>=0 && b1'>=0 && b2'>=0 && b3'>=0
&& d1'>=0 && d2'>=0 && d3'>=0 && e1'>=0 && e2'>=0 && e3'>=0
&& f1'>=0 && f2'>=0 && f3'>=0 &&  delta2'>=0 && delta3'>=0 &&

exists
(
#Auxiliary and quantified variables are needed to make the
#encoding easier
t1,t2,t3,t4,t5,t6,t7,t8,ya1,ya2,ya3,yb1,yb2,yb3,yd1,yd2,
yd3,ye1,ye2,ye3,yf1,yf2,yf3,ydelta2,ydelta3:

#The auxiliary variables are nonnegative
t1>=0 && t2>=0 &&t3>=0 && t4>=0 &&t5>=0 &&t6>=0 &&t7>=0
&&t8>=0 && ya1>=0 && ya2>=0 &&  ya3>=0 && yb1>=0 &&
yb2>=0 && yb3>=0 && yd1>=0 && yd2>=0 && yd3>=0 && ye1>=0
&& ye2>=0 && ye3>=0 && yf1>=0 && yf2>=0 && yf3>=0 &&
ydelta2>=0 && ydelta3>=0

&&

(
#start maxEnable
(
#start enable
(
(!(t1>0 || t3>0 || t4>0) or  delta3=0) &&
(!(t2>0 || t5>0 || t6>0 || t7>0) or delta2=0)
&&
#priority
(f2-2t7<2) &&
#enough objects to fire
(a3-t3-t1>=0) && (f3-t4>=0) && (b2-t5>=0) && (d2-t6>=0)
&& (f2-t7-t7-t2>=0) && (e1-t8>=0)
)
#end enable
&&
#startforall -- to make sure the t's are  maximal
(forall (t1',t2',t3',t4',t5',t6',t7',t8':
(
!(
(t1'>=t1 && t2'>=t2 && t3'>=t3 && t4'>=t4 && t5'>=t5 &&
t6'>=t6 && t7'>=t7 && t8'>=t8) &&
(
#start enable
(!(t1'>0 || t3'>0 || t4'>0) or  delta3=0) &&
 (!(t2'>0 || t5'>0 || t6'>0 || t7'>0)
 or delta2=0)
&&
#priority
(f2-2t7'<2) &&
#enough objects to fire
(a3-t3'-t1'>=0) && (f3-t4'>=0) && (b2-t5'>=0) &&
(d2-t6'>=0) && (f2-t7'-t7'-t2'>=0) && (e1-t8'>=0)
#end enable
)
)
or (t1'=t1 && t2'=t2 && t3'=t3 && t4'=t4 && t5'=t5 &&
t6'=t6 && t7'=t7 && t8'=t8)
)
)
)
#endforall
)
#above end maxEnable
)

&&

#each move is split into two: fire1 and fire2, where
#the y's are intermediate vars
#start fire1
(
(
(ya3=a3-t1-t3+t3) && (yb3=b3+t1+t3) && (yf2=f2-t2-t7-t7+t7)
&& (ydelta2 = delta2 + t2) && (ydelta3 =delta3+t1) && (yf3=f3-t4+t4+t4)
&& (yb2=b2-t5) && (yd2=d2+t5-t6+t6) && (ye2 =e2+t6) && (ye1=e1-t8)
&& (ye3=e3) && (yd1=d1) && (yd3=d3) && (yb1=b1) && (yf1=f1) &&
 (ya2=a2) && (ya1=a1)
)
)
#end fire1

&&

(
#start condition1
(
!(t1>0 && t2>0) or (
delta3'=ydelta3 && delta2'=ydelta2 &&
a1'=ya1+ya2+ya3 &&
b1'=yb1+yb2+yb3 &&
f1'=yf1+yf2+yf3 &&
d1'=yd1+yd2+yd3 &&
e1'=ye1+ye2+ye3 &&
a2'=0 && b2'=0 && f2'=0 && e2'=0 && d2'=0 &&
a3'=0 && b3'=0 && f3'=0 && e3'=0 && d3'=0
)
)
#end condition1
```

```
&&

#start condition2
(
!(t2>0 && t1=0) or (
delta3'=ydelta3 && delta2'=ydelta2 &&
a1'=ya1+ya2 &&
b1'=yb1+yb2 &&
f1'=yf1+yf2 &&
d1'=yd1+yd2 &&
e1'=ye1+ye2 &&
a2'=0 && b2'=0 && f2'=0 && e2'=0 && d2'=0 &&
a3'=ya3 && b3'=yb3 && f3'=yf3 && e3'=ye3 && d3'=yd3
)
)
#end condition2

&&

#start condition3
(
!(t2=0 && t1>0) or (
 delta3'=ydelta3 && delta2'=ydelta2 &&
 a2'=ya2+ya3 &&
b2'=yb2+yb3 &&
f2'=yf2+yf3 &&
d2'=yd2+yd3 &&
e2'=ye2+ye3 &&
a3'=0 && b3'=0 && f3'=0 && e3'=0 && d3'=0 &&
a1'=ya1 && b1'=yb1 && f1'=yf1 && e1'=ye1 && d1'=yd1
)
)
#end condition3

&&

#start condition4
(
!(t2=0 && t1=0) or (
 delta3'=ydelta3 && delta2'=ydelta2 &&
a1'=ya1 && b1'=yb1 && f1'=yf1 && e1'=ye1 && d1'=yd1
&& a3'=ya3 && b3'=yb3 && f3'=yf3 && e3'=ye3 && d3'=yd3
&& a2'=ya2 && b2'=yb2 && f2'=yf2 && e2'=ye2 && d2'=yd2
)
)
#end condition4
)
#end fire2
)
};
```

**Promela code for the example P system in Section 5**

```
bool end_1=false,end_2=false,end_3=false;
byte out_e;
/*communication channel between membrane 2 and 1*/
chan chan_1 = [0] of {byte};
/*communication channel between membrane 3 and 2*/
chan chan_2 = [0] of {byte};

active proctype mem3() /*simulates membrane 3*/
{
    byte left_a=1,left_f=1,right_a,right_f,right_b;
    bool dissolve_3=false;
    do
    :: !dissolve_3 ->
        atomic{
            do /*one maximal parallel move*/
            :: (left_a>0) -> left_a--; right_a++; right_b++
            :: (left_a>0&&!dissolve_3) ->
                      left_a--; right_b++; dissolve_3=true
            :: (left_f>0) -> left_f--; right_f=right_f+2
            ::  else -> break
            od;
                 /*update objects after the move*/
            left_a=left_a+right_a;right_a=0;
            left_f=left_f+right_f;right_f=0
        }
    :: dissolve_3 ->
   /*membrane 3 dissolves and transfers objects to membrane 2*/
        chan_2!left_a;
        chan_2!left_f;
        chan_2!right_b;
        break
    od ;
    end_3=true
}

active proctype mem2() /*simulates membrane 2*/
{
    byte left_a, left_b, left_d, right_d, left_f,
         right_f, right_e, tmp;
    bool dissolve_2 = false;
    do
    :: !dissolve_2 -> atomic{
            do /*one maximal parallel move*/
            :: (left_b>0) -> left_b--; right_d++
            :: (left_d>0) -> left_d--; right_d++; right_e++
            :: (left_f>1) -> left_f=left_f-2; right_f++
            :: (left_f<=1&&left_f>0&&!dissolve_2) ->
                  left_f--; dissolve_2=true
            :: else -> break
```

```
             od;
             left_d=left_d+right_d;right_d=0;
             left_f=left_f+right_f;right_f=0
         }
        /*Take over objects from membrane 3*/
    ::  chan_2?tmp; left_a=left_a+tmp;
        chan_2?tmp; left_f=left_f+tmp;
        chan_2?tmp; left_b=left_b+tmp;
        tmp=0
       /*membrane 2 dissolves and transfer all objects to membrane 1*/
    :: dissolve_2 ->
        chan_1!left_a;
        chan_1!left_d;
        chan_1!right_e;
chan_1!1;
        break
    od;
    end_2=true
}

active proctype mem1()  /*simulates membrane 1*/
{
    bool no_progress=false,dissolve_2=false;
    byte left_a,left_d,left_e,tmp;
    do
    :: !no_progress -> atomic{
            do /*one maximal parallel move*/
            :: left_e>0 -> left_e--; out_e++
            :: else -> no_progress=true; break
            od
        }
       /*take over objects from membrane 2*/
    ::  chan_1?tmp; left_a=left_a+tmp;
        chan_1?tmp; left_d=left_d+tmp;
        chan_1?tmp; left_e=left_e+tmp;
        chan_1?tmp; dissolve_2=(tmp!=0);
tmp=0; no_progress=false
    :: (no_progress && dissolve_2) ->
break; /*exit after membrane 2 dissolved
              and no rules can be applied*/
    od;
    end_1=true
}

-------------------------------------------
Two properties (translated from LTL formulas) checked:
-------------------------------------------

never {    /* !(<> end_1&&end_2&&end_3) */
accept_init:
T0_init:
        if
        :: (! ((end_1))) -> goto accept_S3
        :: (((! ((end_2))) || (! ((end_3))))) ->
               goto accept_all
        fi;
accept_S3:
T0_S3:
        if
        :: (! ((end_1))) -> goto accept_S3
        fi;
accept_all:
        skip
}

never { /* end_1&&end_2&&end_3&&(out_e!=d_1*d_1) */
accept_init:
T0_init:
        if
        :: ((out_e!=d_1*d_1) && (end_1)
            && (end_2) && (end_3)) -> goto accept_all
        fi;
accept_all:
        skip
}
```