On the Reachability Problem for P Systems with Symport/Antiport

Gheorghe Păun[1] **Mario J. Pérez-Jiménez**[2]
Fernando Sancho-Caparrini[2]

[1]Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania, and
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
gpaun@imar.ro, gp@astor.urv.es

[2]Department of Computer Science and Artificial Intelligence
University of Seville. {Mario.Perez,Fernando.Sancho}@cs.us.es

Abstract. We address the problem of deciding whether or not a given configuration of a P system can be reached by correct transitions starting from a given initial configuration. Specifically, we consider P systems with symport/antiport rules, an attractive class which was recently introduced. As expected, the problem is undecidable in general, due to the large generative power of P systems, but, somewhat surprisingly, the reachability is decidable for configurations which take into account also the objects which are sent out of the system during the computation (the language describing such configurations is proved to be context-sensitive, hence recursive). These assertions are true both for halting configurations and for arbitrary configurations.

1 Introduction

P systems are distributed parallel computing models which start from the observation that the processes which take place in the complex structure of a living cell can be considered computations. Abstracting from the structure and the functioning of living cells, the basic ingredients of a P system are the *membrane structure*, consisting of several membranes embedded in a main membrane (called the *skin*) and delimiting *regions* where multisets of *objects* are placed; the objects evolve according to given *evolution rules*, which are applied non-deterministically (the rules to be used and the objects to evolve are randomly chosen) in a maximally parallel manner (in each step, all objects which can evolve must do it). The objects can also be communicated from a region to another one. In this way, we get *transitions* from a *configuration* of the system to another configuration. A sequence of transitions constitutes a *computation*; with each *halting computation* we associate a *result*, the number of objects in a specified *output membrane* (this is an *elementary* membrane, that is, a membrane without any other membrane inside). The objects can also be described by strings over a given alphabet, and in this case the evolution rules are based on string processing operations, and the result of a computation can be a number or a set of strings (a language), but we do not consider this case here.

Since these computing devices were introduced ([11]) several variants have been considered. Many of them were proved to be computationally complete, able to compute all Turing computable sets of natural numbers (all recursively enumerable languages, in the case of string-objects). Moreover, it was shown that when membrane division, membrane creation, or string replication is allowed, NP-complete problems can be solved in linear time. Details (including the current bibliography of the domain and several papers which can be downloaded) can be found at the web address `http://psystems.disco.unimib.it`.

The communication of objects through membranes is one of the most important ingredients of a P system, and, roughly speaking, communication gives power to these computing devices. This led to the question (see [12]) to closely investigate the power of communication, to consider "purely communicative" systems, where the objects are not changed during a computation, but they just change their place with respect to the compartments of a P system. A first attempt to solve this problem was done in [8], where the idea of plasmids, or of vectors from gene cloning is captured by considering certain "vehicle-objects" which carry other objects through membranes. Another biochemical idea, that of membrane transport in pairs of chemicals, was recently followed in [10]. When two chemicals can pass through a membrane only together, in the same direction, the process is called *symport*. When the two chemical pass simultaneously, but in opposite directions, the process is called *antiport*. For biochemical details about symport/antiport processes at the level of alive cell membranes we refer to [1] and to [2].

The idea of coupled transport through membranes was captured in [10] by considering rules of the form $(ab, in), (ab, out)$ (for symport), and $(a, out; b, in)$ (for antiport). Also, more complex rules can be allowed, for instance, of the form $(ab, out; cd, in)$. (In all cases, a, b, c, d are symbol-objects.)

Both P systems with carriers and with symport/antiport were shown to be computationally universal. In the latter case, when rules of the forms $(ab, in), (ab, out), (a, out; b, in)$ are used, five membranes were necessary, while systems with only two membranes are universal in the case of rules $(u, out; v, in)$, with u, v strings of length at most two.

P systems with symport/antiport are attractive from several points of view: (i) they are directly inspired from biochemistry and do not contain any "artificial" ingredient, inspired from theoretical computer science, (2) they observe the conservation law, as no object is created or destroyed during a computation, (3) they essentially take into account the environment, (4) they are mathematically elegant and computationally powerful. Note that usually the first three features are not present in the case of other types of P systems.

Here we address a question which was not considered yet in the membrane computing area, that of reachability of a given configuration with respect to a given system.

The question can be raised for any type of P systems, and we expect similar results in all cases: (i) undecidability of the reachability for systems which are computationally universal (intuitively, because there are P systems which generate non-recursive sets of numbers/languages, just consider a configuration which is precisely associated with the result of a computation; whether or not it is reachable is equivalent with the membership of the associated result to the generated non-recursive set/language), (ii) decidability of the reachability of *extended configurations* (this result is obtained both for halting configurations and for configurations which are not necessarily halting). In the case of *conservative* systems, those which observe the conservation law, we call extended configuration a description of both the system (membranes and objects present in their regions) and of objects sent out of the system during the computation. When also erasing rules are allowed, we have to also take into account the objects which are "erased". Because here we deal only with systems with symport/antiport, and they are conservative, we do not enter into details for the latter case, its study remains as a research topic (the first problem is to give a precise meaning to the idea of "taking into account the objects which are erased").

An "explanation" for the decidability of the reachability of extended configurations is the fact that they preserve the whole "working space", hence, intuitively, they can be computed by context-sensitive Chomsky rules. Because the context-sensitive languages are recursive, we find that the reachability of extended configurations is decidable. Actually, we will prove a stronger result: the language of extended configurations (halting or not) which are reachable in a given P system with symport/antiport rules can be generated by a so-called matrix grammar with appearance checking without using λ-rules; such grammars generate a strict subfamily of the family of context-sensitive languages.

2 P Systems with Symport/Antiport

In this section, we introduce the notion of P systems with symport/antiport in a semi-formal manner, following [10]; a completely formal definition will be given in Section 3.

The language theory notions we use here are standard, and can be found in any of the many monographs available, in particular, in [20]; also, one can use [19]. In particular, we denote by V^* the free monoid generated by an alphabet V under the operation of concatenation; the empty string is denoted by λ and the length of $x \in V^*$ is denoted by $|x|$. The families of context-free, context-sensitive, and recursively enumerable languages are denoted by CF, CS, RE, respectively.

The reader may also want to consult [15] for an introduction to the membrane computing area.

A membrane structure is pictorially represented by a Venn diagram, and it can be mathematically represented by a tree or by a string of matching parentheses. Let X be a finite and non-empty set (that is, X is an alphabet). A multiset over a set X is a mapping $M : X \longrightarrow \mathbf{N} \cup \{\infty\}$ (we allow infinite multiplicity); the *support* of M is the set $supp(M) = \{a \in X \mid M(a) > 0\}$. The multisets whose elements have only finite multiplicities can be represented in a natural way by strings over X, such that the number of occurrences of symbols $a \in X$ in a string $w \in X^*$ gives the multiplicity of a in the multiset associated with w; clearly, all permutations of w represent the same multiset.

We consider here only P systems with symport/antiport with rules of the general form $(u, out; v, in)$, where u, v are strings of a length which is not prescribed in advance; the strings u, v are interpreted as representing multisets.

A *P system with symport/antiport* (of degree $m \geq 1$) is a construct

$$\Pi = (V, \mu, M_1, \ldots, M_n, M_e, R_1, \ldots, R_n, i_o),$$

where:

1. V is the alphabet of *objects*;

2. μ is a membrane structure with n membranes (injectively labelled by positive integers $1, 2, \ldots, n$; the skin membrane is labelled with 1);

3. $M_1, \ldots, M_n$ are the multisets of objects initially present in the regions of the system, and M_e is the multiset of objects present outside the system, in the *environment*; the "internal multisets" have finite multiplicities of objects, while for each $a \in V$ we have either $M_e(a) = 0$ or $M_e(a) = \infty$ (outside the system, an object is either absent, or present in arbitrarily many copies); due to these restrictions, in what follows we will represent the internal multisets by strings, as mentioned before, and the multiset M_e by its support;

4. $R_1, \ldots, R_n$ are finite sets of *rules* of the form $(u, out; v, in)$, for $u, v \in V^*$ with $uv \neq \lambda$;

5. $i_o \in \{1, \ldots, n\}$ is an elementary membrane of μ (the output membrane).

For a rule $(u, out; v, in)$, we say that $(|u|, |v|)$ is the *radius* of the rule; if all rules from Π have the radius componentwise smaller than or equal to (r, s), for given $r, s \geq 0$, then we say that Π is of radius (r, s). If $u = \lambda$ or $v = \lambda$, then instead of $(u, out; v, in)$ we write (v, in), (u, out), respectively (and such a rule is a symport rule).

The meaning of the rules $(u, out; v, in)$ from $R_i, 1 \leq i \leq n$, is obvious: when applying such a rule, the objects indicated by u will exit the region of membrane i while the objects indicated by v will enter this region. Of course, in order to use such a rule, all objects specified by u must be present in the region of membrane i and all objects specified by v must be present in the region immediately outside (in the environment, if the rule is applied in the skin membrane), with the multiplicity at least as large as specified by u, v, respectively.

The multisets of objects present in the m regions of Π constitute the *configuration* of the system; $(M_1, \ldots, M_n)$ is the initial configuration. We pass from a configuration to another configuration by using the rules from $R_1, \ldots, R_n$, as customary in P systems: the rules are applied in the non-deterministic maximally parallel manner, in the sense that we apply the rules in parallel, to all objects which can be processed, non-deterministically choosing the rules and the objects. Thus, a transition means a redistribution of objects among regions (and environment), which is maximal for the chosen set of rules. A sequence of transitions between configurations of the system constitutes a *computation*; a computation is successful if it *halts*, i.e., it reaches a configuration where no rule can be applied to any of the objects.

The *result* of a successful computation is the number of objects present within the membrane with the label i_o in the halting configuration. A computation which never halts yields no result. The set of all the numbers computed by Π is denoted by $N(\Pi)$.

The family of all sets $N(\Pi)$, computed as above by systems Π of degree at most $n \geq 1$ and of radius at most $(r, s), r, s \geq 0$, is denoted by $NPP_n(r, s)$. When one of the parameters n, r, s is not bounded, we replace it with $*$.

Also, we use NRE to denote the family of recursively enumerable sets of natural numbers; this is the family of the length sets of recursively enumerable languages (the family of sets of natural numbers which can be recognized by a Turing machine).

The following result is proved in [10].

Theorem 2.1 *$NRE = NPP_n(r, s)$, for all $n, r, s \geq 2$.*

It is important to note that if in a P system Π with symport/antiport rules we have rules of the form (v, in) in R_1, and v specifies objects which are available in the environment in an arbitrarily large number of copies, then no computation with respect to Π can halt, hence $N(\Pi) = \emptyset$. Indeed, by using such a rule we bring arbitrarily many copies of the objects from v into the system, but, because we assume that the environment is inexhaustible, the rule can be used again and again. Because the existence of such a rule can easily be decided by simply inspecting the system, in what follows we always assume that such rules are not allowed (this does not means that always our systems generate non-empty sets of numbers, as the computations can run forever also because of other rules). This restriction is crucial for the positive decidability result from Section 6.

3 A Formalization

In this section we give a mathematical formalization of the definition of a P system with symport/antiport rules, as well as of its functioning (hence also of the set of numbers generated by it). This formalization follows the style of [18], where P systems with multiset rewriting rules, regulated by a priority relation, also providing the membrane dissolving action, were considered; as the reader can see, the case of P systems with symport/antiport rules is much easier, because simpler ingredients are used. We also mention that the problem of completely formalizing the definition of a P system of a given type and, mainly, of its computations and results of computations, is a non-trivial one, it was explicitly formulated in [13], and addressed with completely different techniques in [3] and [9].

3.1 A syntax for P systems with symport/antiport

Definition 3.1 *A* multiset *over a set, V, is an application $m : V \longrightarrow \mathbf{N} \cup \{\infty\}$. The* support *of m is the set $supp(m) = \{a \in V \mid m(a) > 0\}$. We denote by $\mathbf{M}(V)$ the set of multisets over V (usually we denote it briefly by $\mathbf{M}$). The following subsets of $\mathbf{M}$ will be considered:*

$$\mathbf{M}^{\infty} = \{m \in \mathbf{M} \mid \forall\, a \in V\ (m(a) = 0 \vee m(a) = \infty)\}$$
$$\mathbf{M}^{*} = \{m \in \mathbf{M} \mid \forall\, a \in V\ (m(a) \in \mathbf{N})\}$$

Definition 3.2 *Given $m_1, m_2 \in \mathbf{M}$, we define:*

- Inclusion*: $m_1 \leq m_2 \Leftrightarrow \forall\, a \in V(m_1(a) \leq m_2(a))$.*
- Strict inclusion*: $m_1 < m_2 \Leftrightarrow m_1 \leq m_2 \wedge m_1 \neq m_2$.*
- Union*: $+ : \mathbf{M}^2 \rightarrow \mathbf{M},\ (m_1 + m_2; a) = m_1(a) + m_2(a), \forall a \in V$.*
- Difference*: $- : \mathbf{M}^2 \rightarrow \mathbf{M},\ (m_1 - m_2; a) = \max\{m_1(a) - m_2(a), 0\}, \forall a \in V$.*
- Amplification*: $\otimes : \mathbf{N} \times \mathbf{M} \rightarrow \mathbf{M},\ (n \otimes m_1; a) = n \cdot m_1(a), \forall a \in V$.*

In above inequalities the following conventions are used: $n + \infty = \infty + n = \infty - n = \infty,\ n \leq \infty,\ \infty \leq \infty,\ n \cdot \infty = \infty$.

Definition 3.3 *A* membrane structure *is a rooted tree $\mu = (N(\mu), E(\mu))$, where the nodes from $N(\mu)$ are called* membranes*, the root (denoted below by x_0) is called* skin*, and the leaves are called* elementary membranes*.*

As in [18], for a given node x of μ, we will denote by $f(x)$ the father of x in μ, that is, the membrane immediately external to x, and by $Ch(x)$, the set of children of x in μ, that is, the collection of membranes delimiting "from bellow" the region x.

Definition 3.4 *A* cell with environment *over an alphabet V is a 3-tuple (μ, M, M_e), where $\mu = (N(\mu), E(\mu))$ is a membrane structure, M is an application $M : N(\mu) \longrightarrow \mathbf{M}(V)$, such that $\forall\, x \in N(\mu)\ (M(x) \in \mathbf{M}^*)$, and $M_e \in \mathbf{M}^{\infty}$.*

In the sequel, if x_0 is the root of the tree of a cell with environment (μ, M, M_e), then we will assume $M(f(x_0)) = M_e$.

Definition 3.5 *Let (μ, M, M_e) be a cell with environment over an alphabet V, and let $x \in N(\mu)$. A* transport rule *associated with x is a 2-tuple $r = (u_r, v_r) \in \mathbf{M}^* \times \mathbf{M}^*$, with $u_r + v_r \neq \mathbf{0}$. If x is the skin membrane, then the restriction $u_r \neq \mathbf{0}\ \vee\ \neg(v_r \subseteq M_e)$ is considered.*

Informally, an evolution rule, r, for a membrane $x \in N(\mu)$, has the form:

$$(u, out; v, in), \text{ with } uv \neq \lambda.$$

In this formalization $u_r = u$ and $v_r = v$.

Definition 3.6 *Let $C = (\mu, M, M_e)$ be a cell with environment over an alphabet V, and $x \in N(\mu)$. A* collection $\mathcal{R}$ *of* transporter rules *associated with C is a function with the domain $N(\mu)$ such that for every membrane $x \in N(\mu)$, $R(x) = \{r_1^x, \ldots, r_{s_x}^x\}$ (denoted R_x) is a finite (possibly empty) set of transport rules associated with x.*

Definition 3.7 *A* P system with symport/antiport *is a 4-tuple $\Pi = (V, C_0, \mathcal{R}, i_0)$, where:*

- *V is a non-empty finite set (usually called base alphabet).*
- *$C_0 = (\mu_0, M_0, M_e^0)$ is a cell with environment over V, such that $M_e^0 \in \mathbf{M}^\infty$.*
- *$\mathcal{R}$ is a collection of transporter rules associated with C_0.*
- *i_0 is a distinguished node of μ_0, which specifies the output membrane of* Π.

3.2 A semantic for P systems with symport/antiport

Definition 3.8 *A* configuration, *C, of a* P *system with symport/antiport,* $\Pi = (V, C_0, \mathcal{R}, i_0)$, *with $C_0 = (\mu_0, M_0, M_e^0)$, is a cell with environment, $C = (\mu_0, M, M_e)$, over V. The configuration C_0 will be called the* initial configuration of Π.

Definition 3.9 *Let $C = (\mu_0, M, M_e)$ be a configuration of a* P *system with symport/antiport* $\Pi = (V, C_0, \mathcal{R}, i_0)$, *and $x \in N(\mu_0)$. We say that* the rule $r \in \mathcal{R}_x$ is applicable to C *if the membrane associated with x has all necessary objects to apply the rule, that is, $u_r \leq M(x)$, and the membrane immediately outside to x has all necessary objects to apply the rule, that is, $v_r \leq M(f(x))$.*

Definition 3.10 *We define the* index of applicability of $r \in \mathcal{R}_x$ in C over the node x, *denoted $N_{Ap}(r, C, x)$, as the maximum number of times the rule r can be applied to C in the node x. That is, $N_{Ap}(r, C, x) = 0$ if r is not applicable to C, otherwise*

$$N_{Ap}(r, C, x) = \min\{\max\{n \mid n \otimes u_r \leq M(x)\}, \max\{n \mid n \otimes v_r \leq M(f(x))\}\}$$

In transition P systems, the application of a rule in a membrane is not affected by the content of the other membranes [18], but in P systems with symport/antiport a rule of a membrane can be applied only when this membrane and the membrane immediately external to it verifies the applicability conditions imposed the rule says.

Definition 3.11 *Let $C = (\mu, M, M_e)$ a configuration of a* P *system with symport/antiport,* Π. *We will say that $H : N(\mu) \longrightarrow \mathbf{N}^{\mathbf{N}}$ is an* applicability matrix over C, *denoted $H \in \mathbf{M_{Ap}}(C)$, if the following conditions are satisfied (usually, we write H_x instead of $H(x)$):*

- *H_x has a correct size, i.e., $\forall\, x \in N(\mu)\ \forall\, j\ (j > s_x \rightarrow H_x(j) = 0)$ (by s_x we have denoted the number of rules associated with x).*

- *Every rule can be applied as many times as H indicates, i.e.,*
$$\forall\, x \in N(\mu)\ \forall\, j\ (1 \leq j \leq s_x \to H_x(j) \leq N_{Ap}(r_j^x, C, x)).$$
- *All the rules can be applied simultaneously, i.e.,*
$$\forall\, x \in N(\mu)\ (\sum_j H_x(j) \otimes u_{r_j^x} + \sum_{y \in Ch(x)} (\sum_i H_y(i) \otimes v_{r_i^y}) \leq M(x)).$$
- *There are enough elements in the environment, i.e.,* $\sum_j H_{x_0}(j) \otimes v_{r_j^{x_0}} \leq M_e$.
- *It is maximal, i.e.,* $\neg\exists\, H'\ (H < H' \wedge H' \in \mathbf{M_{Ap}}(C))$.

Definition 3.12 *The* execution *of an applicability matrix, H, over a given configuration, $C = (\mu, M, M_e)$, produces the configuration $PH(C) = (\mu, M', M'_e)$, where:*

$$\begin{aligned} M'(x) = \quad & M(x) \quad - \sum_j H_x(j) \otimes u_{r_j^x} + \sum_j H_x(j) \otimes v_{r_j^x} + \\ & \quad + \sum_{y \in Ch(x)} (\sum_j H_y(j) \otimes u_{r_j^y} - \sum_j H_y(j) \otimes v_{r_j^y}), \\ M'_e = \quad & M_e \quad + \sum_j H_{x_0}(j) \otimes u_{r_j^{x_0}} - \sum_j H_{x_0}(j) \otimes v_{r_j^{x_0}}. \end{aligned}$$

Definition 3.13 *We will say that a configuration C_1 of a* P *system with symport/antiport* Π *yields a configuration, C_2, by a transition in one step of Π, denoted $C_1 \Rightarrow_\Pi C_2$, if there exists a non–zero applicability matrix H over C_1 such that $H(C_1) = C_2$.*

Once we have defined the relationship of transition in one step among configurations, we can define the relation of transition as its transitive closure.

Definition 3.14 *We say that configuration C yields configuration C' in k transition steps ($k \geq 0$), if there are configurations $C_1, \ldots, C_{k+1}$ such that*

$$C_1 = C \wedge C_{k+1} = C' \wedge \forall i\ (1 \leq i \leq k \Rightarrow C_i \Rightarrow_\Pi C_{i+1}).$$

We say that configuration C yields configuration C', denoted $C \overset{\star}{\Rightarrow}_\Pi C'$, if there is $k \geq 0$ such that C yields C' in k transition steps.

Starting from the initial configuration, we can build the computation tree associated with the P system: the nodes of this tree are the configurations of the computation, and the edges are the applicability matrices used to get one configuration from another.

Definition 3.15 *The* computation tree of a P system with symport/antiport Π, *denoted* $\mathbf{Comp}(\Pi)$, *is a rooted labelled maximal tree defined as follows: The root of the tree is the initial configuration, C_0, of Π. The children of a node are the configurations that follow in one step of transition. Nodes and edges are labelled by configurations and applicability matrices, respectively, in such a way that two labelled nodes C, C' are adjacent in* $\mathbf{Comp}(\Pi)$, *by means of an edge labelled with H, if and only if $H \in \mathbf{M_{Ap}}(C) - \{\mathbf{0}\} \wedge C' = H(C)$.*

The maximal branches of $\mathbf{Comp}(\Pi)$ *will be called* computations *of* Π. *We will say that a computation of* Π halts *if it is a finite branch. The configurations verifying* $\mathbf{M_{Ap}}(C) = \{\mathbf{0}\}$ *will be called* halting configurations*, and we will denote the set of halting configurations by* $Halt(\Pi) = \{C \mid \mathbf{M_{Ap}}(C) = \{\mathbf{0}\}\}$.

In the case of P systems with symport/antiport, every halting configuration is a *successful configuration.*

Definition 3.16 *Given* $\mathcal{C} \equiv (C_0 \Rightarrow_\Pi \dots C_k \Rightarrow_\Pi C_{k+1} \dots)$ *a computation of a* P *system* $\Pi = (V, C_0, \mathcal{R}, i_0)$ *with symport/antiport, the* extended configuration *associated with the configuration* $C_i = (\mu, M, M_e)$ *of* $\mathcal{C}$ *is the tuple* (M, e_i) *where* e_i *is the multiset of elements sent into the environment during the computation* $\mathcal{C}$ *until reaching the configuration* C_i.

That is, if $H^{(k)}$ *is the applicability matrix over* C_{k-1} *such that* $C_k = H^{(k)}(C_{k-1})$, *then we define by recursion the multiset of elements sent into the environment during the computation* $\mathcal{C}$ *until reaching the configuration* C_i *as follows:*

$$\begin{cases} e_0 = \emptyset, \\ e_i = e_{i-1} + \sum_j H^{(i)}_{x_0}(j) \otimes u_{r_j^{x_0}}, \quad \textit{for } i \geq 1. \end{cases}$$

Definition 3.17 *Let* $\Pi = (A, C_0, \mathcal{R}, i_0)$ *a P system. The* set of natural numbers generated by Π, *denoted* $\mathbf{N}(\Pi)$, *is defined as follows:*

$$\mathbf{N}(\Pi) = \{|M_C(i_0)| \mid C \in Halt(\Pi) \wedge C = (\mu_C, M_C, M_e^C)\}.$$

4 Matrix Grammars with Appearance Checking

In the proofs from the next two sections we need the notion of a *matrix grammar with appearance checking*, hence we briefly introduce it here; details can be found in [4] and in the chapter of [19] devoted to regulated rewriting.

Such a grammar is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of ordered sequences of the form $(A_1 \to x_1, \dots, A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \dots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied – therefore we say that these rules are applied in the *appearance checking* mode.)

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form generated by grammars with arbitrary rules (that is, including erasing rules) is denoted by MAT_{ac}^{λ}. If the set F is empty, then the grammar is said to be without appearance checking; if no erasing rule is used, then the grammar is said to be λ-free. If we use only grammars without appearance checking or without erasing rules, then we remove the symbols ac and λ, respectively, from the notation MAT_{ac}^{λ}.

The relations $CF \subset MAT \subset MAT_{ac} \subset CS \subset MAT_{ac}^{\lambda} = RE$, $MAT \subseteq MAT^{\lambda} \subset RE$ are known. The languages $L \in MAT^{\lambda}$, $L \subseteq a^*$, are regular [5]; hence, $CS - MAT^{\lambda} \neq \emptyset$, but it is not known whether or not MAT^{λ} contains languages which are not in CS.

5 A Non-Decidable Reachability Case

The fact that P systems with symport/antiport are computationally universal makes expected the fact that most decidability questions about them are not solvable algorithmically. This is obviously the case with questions about the generated sets of numbers, such as membership, emptiness, finiteness, etc. Although not directly following from the universality, the same assertion holds true for questions about the configurations a P system can reach during a computation, in particular, for the reachability question:

Given a P system Π *and a configuration* C *of* Π*, determine whether or not the initial configuration* C_0 *of* Π *yields configuration* C*.*

In the proof of this result we will make an essential use of the proof of the inclusion $NRE \subseteq NPP_2(2,2)$ on which Theorem 2.1 is based, hence we recall its proof from [10], also slightly simplifying it.

To this aim, we use the equality $RE = MAT_{ac}^{\lambda}$, with the obvious consequence that any set from NRE is the length set of a language from MAT_{ac}. In particular, we can consider a language over the one-letter alphabet.

Thus, let $G = (N, \{a\}, S, M, F)$ be a matrix grammar with appearance checking in the binary normal form, with $N = N_1 \cup N_2 \cup \{S, \#\}$, and with M containing matrices $m_i : (X \to \alpha, A \to x)$, $X \in N_1, \alpha \in N_1 \cup \{\lambda\}, A \in N_2, x \in (N_2 \cup \{a\})^*$, for $i = 1, \dots, k$ (matrices without rules used in the appearance checking manner), and $m_i : (X \to Y, A \to \#)$, $X, Y \in N_1, A \in N_2$, for $i = k+1, \dots, n$ (matrices with rules to be used in the appearance checking manner), for some $k \geq 1, n \geq k$. We denote the initial matrix $(S \to XA)$ of G by $(S \to X_0A_0)$, where $X_0 \in N_1$ and $A_0 \in N_2$.

We construct the P system with symport/antiport rules (the internal multisets are represented by strings)

$$\Pi = (V, (\mu, M, M_e), \mathcal{R}, 2),$$

with

$$\begin{aligned}
V &= N_1 \cup N_2 \cup \{a, c, f, h, Z\} \cup \{c_i, c_i', d_i \mid 1 \leq i \leq n\},\\
\mu &= (\{1,2\}, \{(1,2)\}), \text{ (that is, } \mu = [_1[_2\ ]_2]_1)\\
M(1) &= cX_0A_0,\\
M(2) &= \lambda,\\
supp(M_e) &= N_1 \cup N_2 \cup \{a, f, h, Z\} \cup \{c_i, c_i', d_i \mid 1 \leq i \leq n\},
\end{aligned}$$

and with the following sets of rules:

$$\begin{array}{rcl}
\mathcal{R}_1 & = & \{(cX, out; c_iY, in),\ (c_iA, out; cc_i', in),\ (c_i', out; u, in) \mid \\
& & \text{for } m_i : (X \rightarrow Y, A \rightarrow u), 1 \leq i \leq k,\ \text{with } X, Y \in N_1, A \in N_2, u \in (N_2 \cup \{a\})^*\} \\
& \cup & \{(cX, out; c_if, in),\ (c_iA, out; u, in) \mid \\
& & \text{for } m_i : (X \rightarrow \lambda, A \rightarrow u), 1 \leq i \leq k,\ \text{with } X \in N_1, A \in N_2, u \in \{a\}^*\} \\
& \cup & \{(cX, out; c_id_i, in),\ (d_i, out; Yh, in),\ (c_iA, out; Z, in),\ (c_ih, out; c, in) \mid \\
& & \text{for } m_i : (X \rightarrow Y, A \rightarrow \#), k+1 \leq i \leq n,\ \text{with } X, Y \in N_1, A \in N_2\}, \\
\mathcal{R}_2 & = & \{(a, in)\} \\
& \cup & \{(\alpha, in),\ (\alpha, out) \mid \alpha \in \{Z\} \cup \{c_i \mid 1 \leq i \leq k\}\} \\
& \cup & \{(fD, in),\ (fD, out) \mid D \in N_2\}.
\end{array}$$

The symbols c, c_i control the simulation of matrices $m_i, 1 \leq i \leq k$, in the following way. After using the rule $(cX, out; c_iY, in)$, we have c_i in the system. If the rule $(c_iA, out; c_i'c, in)$ cannot be used, then the symbol c_i will go forever back and forth through membrane 2, hence the computation will never finish. If the rule $(c_iA, out; c_i'c, in)$ is used, then at the next step c_i' will exit, bringing into the system the string u, which completes the simulation of the matrix $m_i : (X \rightarrow Y, A \rightarrow u)$.

In the case of terminal matrices, that is, with the first rule of the form $X \rightarrow \lambda$, we bring the symbols c_i, f in the system, c_i simulates the second rule of the matrix, and f checks whether or not the derivation was a terminal one. In the negative case, the rules $(fD, in), (fD, out)$ of $\mathcal{R}_2$, for $D \in N_2$, are used forever (the symbol c is no longer present, hence no other rule can be used).

If we start with a rule of the form $(cX, out; c_id_i, in)$, for some $k + 1 \leq i \leq n$, for $m_i : (X \rightarrow Y, A \rightarrow \#)$, then at the next step we have to use the rule $(d_i, out; Yh, in)$. If in the skin membrane there is any copy of A, then at the same time the rule $(c_iA, out; Z, in)$ has to be used, and the computation will never finish, because of the rules $(Z, in), (Z, out)$ from $\mathcal{R}_2$. If no copy of A is present, then c_i waits in the skin membrane. At the next step, together with H, the symbol c_i can leave the system, and at the same time c is brought again inside. In this way, the application of matrix m_i is simulated.

Note that in all cases the symbol c is again available, hence the process can be iterated. Consequently, $N(\Pi) = \{n \mid a^n \in L(G)\}$, which completes the proof of the inclusion $NRE \subseteq NPP_2(2, 2)$.

Now, the undecidability of the reachability problem immediately follows: a configuration $C = (\mu, M, M_e)$ where $M(1) = f$ and $M(2) = a^n$ can be reached if and only if $n \in N(\Pi)$; for a set $N(\Pi)$ which is not recursive, this question cannot be algorithmically answered; because $NPP_2(2, 2) = NRE$, there are such non-recursive sets $N(\Pi)$.

It is important to note that the previous result directly follows from the proof of the equality $NRE = NPP_2(2, 2)$ because of the fact that in any halting configuration we precisely know which are the objects present in the system: only f in the skin membrane, and a certain number of copies of a in the inner membrane. However, if some other objects would be present in the system in a halting configuration, when also a distinguished symbol f is introduced, then we can arrange that f will help them to leave the system, by using rules of the form $(f\alpha, out)$, for all objects α which we want not to have them inside, and (f, in). A similar trick can probably be used also for other types of P systems, hence a negative result concerning the reachability question is expected to be valid for all classes of P systems which are computationally universal.

Clearly, the previous negative decidability result holds both for halting configurations and, by default, for arbitrary configurations.

6 A Decidable Reachability Case

We consider now an apparently inoffensive extension of the notion of a configuration, also taking into account the objects which were sent out of the system during the computation. Because we do not allow rules of the form (v, in) for multisets v whose elements appear in the environment in arbitrarily many copies, the number of objects sent out of the system is always finite. Thus, we can represent their multiset by a string, as usual for internal multisets.

If the membrane structure has m membranes, $\{1, \ldots, m\}$, following Definition 3.16, an extended configuration (M, e) can be represented as the $(m+1)$-tuple $(z_1, \ldots, z_m, z_{m+1})$, of strings over V, where z_i, $1 \leq i \leq m$ represents $M(i)$, the multiset of objects present in membrane i, and z_{m+1} represents the multiset of objects sent into the environment during a computation in Π which has led to $(z_1, \ldots, z_m)$. For an easier representation, we write such a configuration as a string over $V \cup \{d\}$, where d is a new symbol (a delimiter), representing the comma, namely, by $z_1dz_2d \ldots dz_mdz_{m+1}$. Furthermore, we assume the alphabet V ordered, $V = \{a_1, \ldots, a_p\}$, and we write the strings z_j in the "standard form", that is, in the form $a_1^{i_1} \ldots a_p^{i_p}$, where $i_1, \ldots, i_p$ are the multiplicities of objects $a_1, \ldots, a_p$, respectively, in the multiset represented by z_j (of course, we can have $i_k = 0$, which means that the corresponding symbol a_k does not appear in z_j). We will denote by w_i the string that represents the objects present in the membrane i in the initial configuration

We denote by $econ(\Pi)$ the language of all strings of this form, describing (in the standard form) extended configurations of Π which can be reached from the initial configuration $w_1dw_2d \ldots dw_md$ (with λ describing the multiset of objects sent into the environment before starting any computation), and by $ehalt(\Pi)$ the language of strings describing (in the standard form) halting extended configurations which can be reached from $w_1dw_2d \ldots dw_md$.

The following two lemmas are the main results of this section.

Lemma 6.1 *For every P system Π with symport/antiport, the language $econ(\Pi)$ belongs to the family MAT_{ac}.*

Proof. Let us consider a P system with symport/antiport $\Pi = (V, (\mu, M, M_e), \mathcal{R}, i_o)$. As usual, we assume that the skin membrane has label 1 and that different membranes have different labels; specifically, $N(\mu) = \{1, \ldots, m\}$. Remember that for a membrane i from μ we denote by $f(i)$ the membrane immediately outside i; for $i = 1$ we put $f(1) = 0$.

Assume that V contains g elements. Moreover, we assume that the rules of $\mathcal{R}$ are labelled in a one-to-one manner, by $r_1, \dots, r_s$, with r_1 being a rule from $\mathcal{R}_1$ (if $\mathcal{R}_1$ is empty, then the changes in the construction below are straightforward and it is left to the reader).

We construct the matrix grammar

$$G = (N, T, S, M, F),$$

with

$$\begin{aligned} N &= \{d, S, X, Y, Z, \#\} \cup \{X_i \mid 1 \leq i \leq s\} \\ &\cup \{(a,i)', (a,i)'' \mid a \in V, 0 \leq i \leq m\}, \\ T &= \{(a,i) \mid a \in V, 0 \leq i \leq m\} \cup \{d\}, \end{aligned}$$

and the set of matrices constructed in the following way.

Consider the morphisms $h_i, 1 \leq i \leq m$, defined by $h_i(a) = (a,i)'$, for each $a \in V$. Then,

$$\begin{aligned} M &= \{(S \to X\ h_1(w_1)d\ h_2(w_2)d \dots d\ h_m(w_m)d), \\ &\quad (S \to Z\ h_1(w_1)d\ h_2(w_2)d \dots d\ h_m(w_m)d)\} \\ &\cup \{(X \to X, (a_1,i)' \to (a_1, f(i))'', \dots, (a_u,i)' \to (a_u, f(i))'', \\ &\qquad\qquad (b_1, f(i))' \to (b_1, i)'', \dots, (b_v, f(i))' \to (b_v, i)'') \mid \\ &\quad (a_1 \dots a_u, out; b_1 \dots b_v, in) \in \mathcal{R}_i, 2 \leq i \leq m, u, v \geq 0, \\ &\quad \text{where } a_j, b_j \in V \text{ for all } j\} \\ &\cup \{(X \to X(b_{j_1}, 1)'' \dots (b_{j_q}, 1)'', (a_1, 1)' \to (a_1, 0)', \dots, (a_u, 1)' \to (a_u, 0)', \\ &\quad (b_{k_1}, 0)' \to (b_{k_1}, 1)''(b_{k_1}, 0), \dots, (b_{k_r}, 0)' \to (b_{k_r}, 1)''(b_{k_r}, 0)) \mid \\ &\quad (a_1 \dots a_u, out; b_1 \dots b_v, in) \in \mathcal{R}_1, u, v \geq 0, \text{ where} \\ &\quad a_j, b_j \in V \text{ for all } j, \{1, 2, \dots, v\} = \{j_1, \dots, j_q\} \cup \{k_1, \dots, k_r\}, \\ &\quad \{j_1, \dots, j_q\} \cap \{k_1, \dots, k_r\} = \emptyset, \text{ and } M_e(b_{j_l}) = \infty, \text{ for } 1 \leq l \leq r\} \\ &\cup \{(X \to X_1, (a_1, 1)' \to ck((a_1, 1)'), \dots, (a_u, 1)' \to ck((a_u, 1)'), \\ &\qquad\qquad (b_{k_1}, 0)' \to ck((b_{k_1}, 0)'), \dots, (b_{k_r}, 0)' \to ck((b_{k_r}, 0)')) \mid \\ &\quad r_1 = (a_1 \dots a_u, out; b_1 \dots b_v, in) \in \mathcal{R}_1, u, v \geq 0, \text{ where } a_j, b_j \in V \\ &\quad \text{for all } j, ck((\alpha, j)') \in \{(\alpha, j)', \#\}, \text{ for all } \alpha \text{ and } j \text{ as above, and there is} \\ &\quad \text{at least one } j \text{ such that } ck((a_j, i)') = \#, \text{ or} \\ &\quad ck((b_j, f(i))') = \#; \text{ moreover, } M_e(b_{k_l}) = 0, 1 \leq l \leq r, \text{ and, if} \\ &\quad l \in \{1, 2, \dots, v\} - \{k_1, \dots, k_r\}, \text{ then } M_e(b_l) = \infty\} \\ &\cup \{(X_{t-1} \to X_t, (a_1, i)' \to ck((a_1, i)'), \dots, (a_u, i)' \to ck((a_u, i)'), \\ &\qquad\qquad (b_1, f(i))' \to ck((b_1, f(i))'), \dots, (b_v, f(i))' \to ck((b_v, f(i))')) \mid \\ &\quad r_t = (a_1 \dots a_u, out; b_1 \dots b_v, in) \in \mathcal{R}_i, 2 \leq i \leq m, u, v \geq 0, \text{ where } a_j, b_j \in V \\ &\quad \text{for all } j,\ ck((\alpha, j)') \in \{(\alpha, j)', \#\}, \text{ for all } \alpha \text{ and } j \text{ as above, and there is} \\ &\quad \text{at least one } j \text{ such that } ck((a_j, i)') = \#, \text{ or} \end{aligned}$$

$$
\begin{aligned}
& ck((b_j, f(i))') = \#, 2 \leq t \leq s-1\} \\
\cup\ & \{(X_{t-1} \to X_t, (a_1,1)' \to ck((a_1,1)'), \dots, (a_u,1)' \to ck((a_u,1)'), \\
& \qquad (b_{k_1},0)' \to ck((b_{k_1},0)'), \dots, (b_{k_r},0)' \to ck((b_{k_r},0)')) \mid \\
& r_t = (a_1 \dots a_u, out; b_1 \dots b_v, in) \in \mathcal{R}_1, u, v \geq 0, \text{ where } a_j, b_j \in V \\
& \text{for all } j, ck((\alpha, j)') \in \{(\alpha, j)', \#\}, \text{ for all } \alpha \text{ and } j \text{ as above, and there is} \\
& \text{at least one } j \text{ such that } ck((a_j, i)') = \#, \text{ or} \\
& ck((b_j, f(i))') = \#; \text{ moreover, } M_e(b_{k_l}) = 0, 1 \leq l \leq r, \text{ and, if} \\
& l \in \{1, 2, \dots, v\} - \{k_1, \dots, k_r\}, \text{ then } M_e(b_l) = \infty, 2 \leq t \leq s-1\} \\
\cup\ & \{(X_{s-1} \to Y, (a_1,i)' \to ck((a_1,i)'), \dots, (a_u,i)' \to ck((a_u,i)'), \\
& \qquad (b_1, f(i))' \to ck((b_1, f(i))'), \dots, (b_v, f(i))' \to ck((b_v, f(i))') \mid \\
& r_s = (a_1 \dots a_u, out; b_1 \dots b_v, in) \in \mathcal{R}_i, 2 \leq i \leq m, u, v \geq 0, \text{ where } a_j, b_j \in V \\
& \text{for all } j, ck((\alpha, j)') \in \{(\alpha, j)', \#\}, \text{ for all } \alpha \text{ and } j \text{ as above,} \\
& \text{and there is at least one } j \text{ such that } ck((a_j, i)') = \#, \text{ or } ck((b_j, f(i)') = \#\} \\
\cup\ & \{(X_{s-1} \to Y, (a_1,1)' \to ck((a_1,1)'), \dots, (a_u,1)' \to ck((a_u,1)'), \\
& \qquad (b_{k_1},0)' \to ck((b_{k_1},0)'), \dots, (b_{k_r},0)' \to ck((b_{k_r},0)')) \mid \\
& r_s = (a_1 \dots a_u, out; b_1 \dots b_v, in) \in \mathcal{R}_1, u, v \geq 0, \text{ where } a_j, b_j \in V \\
& \text{for all } j, ck((\alpha, j)') \in \{(\alpha, j)', \#\}, \text{ for all } \alpha \text{ and } j \text{ as above,} \\
& \text{and there is at least one } j \text{ such that } ck(a_j, i) = \#, \text{ or} \\
& ck((b_j, f(i))') = \#; \text{ moreover, } M_e(b_{k_l}) = 0, 1 \leq l \leq r, \text{ and, if} \\
& l \in \{1, 2, \dots, v\} - \{k_1, \dots, k_r\}, \text{ then } M_e(b_l) = \infty\} \\
\cup\ & \{(Y \to Y, (a,i)'' \to (a,i)') \mid a \in V, 1 \leq i \leq m\} \\
\cup\ & \{(Y \to X, (a_1,i)'' \to \#, \dots, (a_g,i)'' \to \#), \\
& (Y \to Z, (a_1,i)'' \to \#, \dots, (a_g,i)'' \to \#) \mid 1 \leq i \leq m, V = \{a_1, \dots, a_g\}\} \\
\cup\ & \{(Z \to Z, (a,i)' \to (a,i)) \mid a \in V, 0 \leq i \leq m\} \\
\cup\ & \{(Z \to d)\}.
\end{aligned}
$$

The set F, of rules to be used in the appearance checking mode, consists of all rules of the form $\alpha \to \#$ from the matrices of M.

Let us briefly examine the work of the grammar G.

The symbols X, Y, Z, as well as $X_i, 1 \leq i \leq s$, control the work of the other rules, in such a way that a computation in Π is simulated by a derivation in G. Specifically, in the presence of X one simulates a nondeterministic maximally parallel use of rules from $\mathcal{R}$, while the symbols $X_1, \dots, X_s$ are used for checking that no further application of the rules $r_1, \dots, r_s$, respectively, is possible at that step. One starts from a description of the initial configuration where all symbols a present in a region i are replaced by the new symbol $(a,i)'$. When simulating the rules of Π, one passes from primed symbols $(a,i)'$ to double primed symbols $(a,j)''$, where j is the membrane outside membrane i in the case when the symbol a is sent *out*, and, conversely, i is the membrane outside j, in the case when a is sent *in*. The symbols sent out of the system are written in the form $(a,0)'$. All double primed symbols present inside the system are replaced by single primed symbols in the presence of Y. When this operation is complete, we either return to X, hence the process can be iterated, or we replace Y by Z. In the presence of Z we replace each primed or double primed symbol by the non-primed symbol, which is considered terminal with respect to G, and the derivation stops. All these operations are performed by using the rules of the form $\alpha \to \#$ in the appearance checking manner, with $\#$ being a trap symbol, which is never removed.

The first sentential form generated by G from its axiom S is $X\ h_1(w_1)d\ h_2(w_2)d \ldots d\ h_m(w_m)d$. During the work of G, the symbols corresponding to various membranes are mixed, while the symbols d are never rewritten. However, the second component of symbols of the form $(a,i)'$ present in any current sentential form identifies the membrane i where the symbol a is placed; the symbols sent out of the system appear in the form $(a,0)'$. After using any rule, the symbols from the system change their second component, according to the *in/out* indication of the rule, and it is double primed. This latter fact prevents the use of the same symbol by another rule at the same step.

In the case of membranes $2 \le i \le m$, the simulation of rules from $\mathcal{R}_i$ is rather easy. Some precautions should be taken in the case of rules from $\mathcal{R}_1$ with the symbols which are introduced from the environment into the system. If these symbols appear in the environment in arbitrarily many copies, then again there is no difficulty (such symbols are introduced by the rules of the form $X \to X(b_{j_1},1)'' \ldots (b_{j_q},1)''$). If the symbols to be introduced in the system are not from M_e, but previously sent out of the system, then they appear in a bounded number of copies in the environment, hence we have to make sure that enough copies of them are present; moreover, we still preserve copies of them in the environment, but without any prime (the corresponding rules are of the form $(b_{k_1},0)' \to (b_{k_1},1)''(b_{k_1},0), \ldots, (b_{k_r},0)' \to (b_{k_r},1)''(b_{k_r},0)$). The non-primed symbols which remain in the environment cannot be used by a further rule, they remain unchanged until the end of the derivation.

At any moment, we can use the matrix which starts with the rule $X \to X_1$. This matrix checks whether or not the rule r_1 can be applied once again to the symbols which are single primed. If this is the case, then the trap symbol $\#$ is introduced and the sentential form will never lead to a terminal string. Otherwise, we continue by introducing the symbol X_2, a step when we check whether or not the rule r_2 can be applied. One continues with $X_3, \ldots, X_{s-1}$; in each case, the applicability of the corresponding rules $r_2, \ldots, r_s$ is checked, and always the trap symbol is introduced if the rule can be applied. If no rule can be applied, that is, the computation was maximally parallel (hence correct with respect to Π), then the symbol Y is introduced.

In the presence of Y we simply replace each double primed symbol with its single primed variant. When this process is complete (this is checked by the rules of the form $(a_j,i)'' \to \#$, from the matrix which start with the rule $Y \to X$, then we return to the symbol X and the process can be iterated, we can simulate another transition with respect to Π.

At any moment when Y can be replaced by X, we can also introduce the control symbol Z. In the presence of this symbol we replace each primed symbol with its non-primed version. When all symbols are thus replaced by terminal symbols, we can also the symbol Z by d. Therefore, all configurations which can be reached after at least one transition in Π correspond to a sentential form generated by G.

In order to produce also the initial configuration, we use the matrix $(S \rightarrow Z\ h_1(w_1)d\ h_2(w_2)d \dots d\ h_m(w_m)d)$, then Z assists the removing of primes, and after that it is replaced by d.

Therefore, the strings from $L(G)$ are of the form dw, where w is a permutation of a string over $V \cup \{d\}$ describing a configuration which can be reached by a correct computation in Π. Let us now consider the regular language

$$\begin{aligned} R &= (a_1,1)^*(a_2,1)^* \dots (a_g,1)^* d (a_1,2)^*(a_2,2)^* \dots (a_g,2)^* d \dots \\ &\quad d(a_1,m)^*(a_2,m)^* \dots (a_g,m)^* d (a_1,0)^*(a_2,0)^* \dots (a_g,0)^* \end{aligned}$$

We denote by $Perm(L)$ the permutation closure of a language L. Consider also the morphism h which maps each symbol $(a,i), a \in V, 0 \leq i \leq m$, into a, and also maps d into d. Denote (as usually) by $\partial_x(L)$ the left derivative of a language L with respect to a string x, that is, $\partial_x(L) = \{y \mid xy \in L\}$. Then we obtain

$$econ(\Pi) = h(R \cap Perm(\partial_d(L(G))))$$

Indeed, ∂_d removes the leftmost occurrence of the symbol d (the symbol obtained from the symbol Z), $Perm$ rearranges arbitrarily the symbols, the intersection with R selects only those strings which have the symbols in the "correct" place, first the multiset from the first membrane, then the multiset from the second membrane, and so on, with the symbols in the standard order; finally, the morphism h returns the symbols (a,i) to a. (Note that this morphism is, in fact, a coding.)

Because the family MAT_{ac} is closed under left derivatives, intersection with regular languages, permutation, and λ-free morphisms, it follows that $econ(\Pi) \in MAT_{ac}$, which concludes the proof. $\square$

In the previous proof we have considered all configurations which are accessible from the initial configuration, without taking care whether or not the corresponding computations will eventually halt. In order to obtain a matrix grammar which generates only the halting accessible configurations, we have to modify the previous construction in a way which is already suggested in the proof of Lemma 6.1. Specifically, after introducing the symbol Z, we do not directly replace all symbols $(a,i)'$ by (a,i), but we first check whether or not any further transition is possible. This amounts at checking whether or not any rule $r_1, \dots, r_s$ is applicable to the current configuration, and this can be done in the same way as when we have checked the maximality of the use of rules (in the phase starting with the use of the rule $X \rightarrow X_1$). Namely, by using new symbols $Z_1, \dots, Z_s$ in the same way as we have used the symbols $X_1, \dots, X_s$, we can check whether or not the rules $r_1, \dots, r_s$, respectively, can be applied. In the affirmative case, we introduce the trap symbol, in the negative case we (conclude that the computation halts, and we) remove the primes. We leave the technical details to the reader and we conclude by stating this observation as a lemma:

Lemma 6.2 *For every P system Π with symport/antiport, the language $ehalt(\Pi)$ belongs to the family MAT_{ac}.*

As we have mentioned in Section 4, $MAT_{ac} \subset CS$, while the context-sensitive languages are recursive. Therefore, the membership problem with respect to he languages $econ(\Pi), ehalt(\Pi)$ is decidable, and this proves the following result:

Theorem 6.1 *The reachability of extended configurations of P systems with symport/antiport is decidable, both in the general case, of considering arbitrary configurations, and when considering halting configurations.*

The proofs of Lemmas 6.1 and 6.2 also imply the inclusion $NPP_*(*, *) \subseteq NRE$: if also erasing rules are used, then we can get the language $\{a^n \mid n \in N(\Pi)\}$, hence this language is in the family MAT_{ac}^{λ}, which is equal to RE, and this implies that $N(\Pi) \in NRE$. Note that always in the P systems area, when proving equality of the form $NPP_*(*, *) = NRE$, one proves the inclusion $NRE \subseteq NPP_n(r, t)$, for some n, r, t, and one says that "the opposite inclusion is straightforward". Actually, the opposite inclusion is implied by the Turing-Church thesis, if we accept this thesis, and it is routine task – but a rather cumbersome one – to construct a Turing machine which simulates a P system of a given type. The constructions in the proofs of Lemmas 6.1 and 6.2 give for the first time a proof for the inclusion $NPP_*(*, *) \subseteq NRE$, moreover, not in terms of Turing machines (or Chomsky type-0 grammars), but in terms of the more restrictive formalism of matrix grammars with appearance checking.

7 Final Remarks

We have considered here a problem which so far was not investigated in the membrane computing area, namely, the reachability of a configuration in a given P system. We have dealt with systems with symport/antiport, but the same question can be formulated for any type of P systems. As expected, the problem is undecidable for the case of usual configurations, but it becomes decidable when taking into account also the symbols which are sent out of the system during a computation. Nothing was said here about the computation complexity of these positive decidability questions – this remains as a research topic.

These results refer to systems of arbitrary degree and form, hence known to be computationally universal. The case of systems of a particular type, with rules of a restricted form, and/or with a limited number of membranes, remains to be investigated, in the hope to find classes of systems with a decidable reachability problem even for restricted configurations. Techniques used in other areas, when dealing with the reachability question can be useful in this respect. For instance, we refer the reader to [6], [7], where also other types of decidability questions can be found, dealing with safety, blocking, etc.

We conclude with the remark that this paper can also be seen as a first answer to the (meta)question formulated in [13], to address new research problems about P systems, different from the "classic" ones, about the power, the normal forms, and the possibility of solving NP-complete problems in polynomial time. The decidability questions are such a class of new problems, and up to now only [16] has considered such questions (namely, the decidability of dissolving a membrane in a system where this operation is possible, with multiset rewriting rules, not with symport/antiport). In [17] a simulation of deterministic Turing machines by P systems with external output is given, and it is expected that this simulation can be also used to attack decidability questions about P systems by transferring corresponding results from Turing machines to P systems.

Acknowledgements

First author's research was partially supported under the Programa Cátedra of the Fundación Banco Bilbao Vizcaya Argentaria (BBVA), and by the Department of Computer Science and Artificial Intelligence of Sevilla University, and under the project TIC2002-04220-C03-03 of the Ministerio de Ciencia y Tecnología of Spain.

Last authors wish to thank the support of the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds.

References

[1] B. Alberts et al., *Essential Cell Biology. An Introduction to the Molecular Biology of the Cell*, Garland Publ. Inc., New York, London, 1998.

[2] I.I. Ardelean, The relevance of biomembranes for P systems, *Fundamenta Informaticae*, 49, 1-3 (2002), 35–43.

[3] A. V. Baranda, J. Castellanos, F. Arroyo, R. Gonzalo, Towards an electronic implementation of membrane computing: A formal description of nondeterministic evolution in transition P systems, *Proc. 7th Intern. Meeting on DNA Based Computers* (N. Jonoska, N.C. Seeman, eds.), Tampa, Florida, USA, 2001, 273–282.

[4] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[5] D. Hauschild, M. Jantzen, Petri nets algorithms in the theory of matrix grammars, *Acta Informatica*, 31 (1994), 719–728.

[6] O.H. Ibarra, Verification in Queue-Connected Multicounter Machines, *International Journal of Foundations of Computer Science*, 13 (2002), 115–127.

[7] O. H. Ibarra, T. Bultan, J. Su, Reachability Analysis for Some Models of Infinite-state Transition Systems, *Proceedings of 11th International Conference on Concurrency Theory*, Pennsylvania, 2000, 183–198.

[8] C. Martin-Vide, Gh. Păun, G. Rozenberg, Membrane systems with carriers, *Theoretical Computer Science*, 270 (2002), 779–796.

[9] A. Obtulowicz, Membrane computing and one-way functions, *Intern. J. Found. Computer Science*, 12, 4 (2001), 551–558.

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

[11] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* No 208, 1998 (www.tucs.fi).

[12] Gh. Păun, Computing with membranes (P Systems): Twenty six research topics, CDMTCS TR 119, Univ. of Auckland, 2000 (www.cs.auckland.ac.nz/ CDMTCS).

[13] Gh. Păun, Further research topics about P systems, *Pre-Proceedings of Workshop on Membrane Computing*, Curtea de Argeş, Romania, August 2001, Technical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 243–250.

[14] Gh. Păun, *Membrane Computing. An introduction*, Springer-Verlag, Berlin, 2002.

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

[16] Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, 41, 3 (2000), 259–266.

[17] M. J. Pérez-Jiménez, A. Romero-Jiménez, Simulating Turing Machines by P systems with external output, *Fundamenta Informaticae*, 49, 1-3 (2002), 273–287.

[18] M. J. Pérez-Jiménez, F. Sancho-Caparrini, A formalization of transition P systems, *Fundamenta Informaticae*, 49, 1-3 (2002), 261–272.

[19] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, 1997.

[20] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.