P Systems with Membrane Creation and Rule Input

Miguel Angel Gutiérrez-Naranjo, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mail: {magutier, marper}@us.es

Summary. When a uniform family of recognizer P systems is designed to solve a problem, the data of a concrete instance of the problem is usually provided via a multiset which is placed in the so-called *input membrane*. In this paper we present a new definition for recognizer P systems, called with *rule input*, where the data of the instance is provided via a set of rules which are introduced in the system at the beginning of the computation. We also discuss a new semantic for P systems with membrane creation and, as an example, a uniform family of recognizer P systems with *rule input* which solves the Subset Sum problem is provided.

1 Introduction

Solving **NP**-complete problems is done in the membrane computing framework by generating an exponential amount of workspace in polynomial time and using the parallelism to check simultaneously all the candidate solutions.

The way in which this exponential number of membranes is created in polynomial time is based on biological processes inspired in living cells. Basically, two processes are used in order to produce new membranes: *mitosis* (membrane division) and *autopoiesis* (membrane creation), see [**?**]. Both ways of generating new membranes have given rise to different variants of P systems: *P systems with active membranes*, where the workspace is generated by membrane division, and *P systems with membrane creation*, where the new membranes are created from objects.

Both models are universal from a computational point of view, but technically, they are pretty different. In fact, nowadays there does not exist any

theoretical result which proves that these models can simulate each other in polynomial time.

P systems with active membranes have been successfully used to design solutions to well-known **NP**-complete problems, as SAT [**?**], Subset Sum [**?**], Knapsack [**?**], Bin Packing [**?**] and Partition [**?**], but as Gh. Păun pointed in [**?**] *"membrane division was much more carefully investigated than membrane creation as a way to obtain tractable solutions to hard problems"*. Recently, the first results related to the power and design of algorithms to solve NP problems by means of P systems using membrane creation have arisen (see [**?**, **?**, **?**]).

In these solutions, both in the model of P systems with active membranes and P systems with membrane creation, a uniform family of recognizer P systems is designed to solve the problem and the data of a concrete instance of the problem is usually provided via a multiset which is placed in the so-called *input membrane*.

In this paper we present a new definition for recognizer P systems with *rule input*, where the data of the instance is provided via a new set of rules. We also discuss a new semantics for P systems with membrane creation and, as an example, a uniform family of recognizer P systems with *rule input* which solves the Subset Sum problem is provided.

The paper is organized as follows. P systems with membrane creation are remembered in the next section, together with a short disscusion about their semantics. In section 3 recognizer P systems with input rules are presented. As an example, a uniform family of recognizer P systems with rule input which solves the Subset Sum problem is presented in Section 4. Finally, some formal details and conclusions are given in the last sections.

2 A new semantics for P systems with membrane creation

Since Gh. Păun presented the cellular computation with membranes, many different variants have been proposed. If the membrane structure is considered to set a classification among these different variants, two big groups are obtained: P systems where the initial structure does not change along computations and P systems where the tree structure of the membranes vary (or can do it) along computation. The decrease of the number of membranes is made by applying the so-called *dissolution rules*, of the form $[a]_e \rightarrow b$, where the object a inside a membrane with label e produces the dissolution of the membrane, the object a disappears and a new element b and the rest of the multiset in the membrane go to its father (more precisely, they go to the

closest non-dissolved ancestor in the membrane hierarchy, since several membranes can dissolve in the same step). Increasing the number of membranes are usually made via division of existing ones or creating new membranes from objects[1].

We recall that a *P system with membrane creation* is a construct of the form

$$\Pi = (O, H, \mu, w_1, \ldots, w_m, R),$$

where:

1. $m \geq 1$ is the initial degree of the system;
2. O is the alphabet of *objects*;
3. H is a finite set of *labels* for membranes;
4. μ is a *membrane structure* consisting of m membranes labelled (not necessarily in a one-to-one manner) with elements of H;
5. $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
6. R is a finite set of *rules*, of the following forms:
   a) $[a \to v]_h$, where $h \in H$, $a \in O$, and v is a string over O describing a multiset of objects. These are *object evolution rules* associated with membranes and depending only on the label of the membrane.
   b) $a[\,]_h \to [b]_h$, where $h \in H$, $a, b \in O$. These are *send-in communication rules*. An object is introduced in the membrane, possibly modified during the process.
   c) $[a]_h \to [\,]_h\, b$, where $h \in H$, $a, b \in O$. These are *send-out communication rules*. An object is sent out of the membrane, possibly modified during the process.
   d) $[a]_h \to b$, where $h \in H$, $a, b \in O$. These are *dissolution rules*. In reaction with an object, a membrane is dissolved, while the object specified in the rule can be modified.
   e) $[a \to [v]_{h_2}]_{h_1}$, where $h_1, h_2 \in H$, $a \in O$, and v is a string over O describing a multiset of objects. These are *creation rules*. In reaction with an object, a new membrane is created. This new membrane is placed inside of the membrane of the object which triggers the rule and has associated an initial multiset and a label.

The rules are applied according to the following principles:

- The rules are used as customary in the framework of membrane computing, that is, in a maximal parallel way. In one step, each object in a membrane

[1] Recently, new operations to change the membrane structure, such as *merging membranes*, or the operations of *endocytosis*, *exocytosis* or *gemmation* have been explored.

can only be used by one rule (non-deterministically chosen when there are several possibilities), but any object which can evolve by a rule of any form must do it (with the restrictions indicated below).

- If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external one. The skin membrane is never dissolved.
- All the elements which are not involved in any of the operations to be applied remain unchanged.
- The rules associated with the label l are used for all membranes with this label, irrespective of whether or not the membrane is an initial one or it was obtained by creation.
- At one step, different rules can be applied to different membranes with the same label, but one membrane can just be the subject of only one rule of types (b), (c) or (d).
- Several rules of the type (a) or (e) can be applied to different objects in the same membrane simultaneously.

Hence, the rules of type (a) and (e) are applied in parallel, that is, all objects which can evolve by such rules must do it, while the rules of type (b), (c) and (d) are used sequentially, in the sense that one membrane can be used by at most one rule of these types in each step (time unit).

The main difference with respect to the usual semantics of P systems with membrane creation is related to the communication rules (*send-in* and *send-out*). In [**?**, **?**, **?**], several objects can cross out one membrane simultaneously. On the contrary, in the semantics used in this paper, only one element can cross a membrane in each time unit. This is closer to the semantics of P systems with active membranes, where only one object can cross a membrane in one time unit.

From a theoretical point of view, both semantics are acceptable and, as shown in this paper, **NP**-complete problems can be solved in both models. Another matter of discussion is the parallelism of the creation of new membranes. In this paper, in a similar way as in [**?**, **?**, **?**], several membranes can be created simultaneously inside one membrane, and there are no restrictions with respect to the label of the objects contained in the created membrane.

Each of these semantics makes sense from a theoretical point of view. Another question is to know which of them is closer to the biological inspiration. Deciding the best semantics of these models is an open discussion in the P systems community.

3 Recognizer P systems with membrane creation

Recognizer P systems were introduced in [**?**] and are the natural framework to study decision problems, since deciding if an instance has an affirmative or negative solution is equivalent to deciding if a word (*yes* or *no*) belongs or not to the language.

In the literature, recognizer P systems are associated in a natural way with P systems with *input*. The data related to an instance of the decision problem has to be provided to the P system in order to compute the appropriate answer. The formal definition of such P systems consider a specific label and calls *input membrane* the membrane associated to this label. A multiset encoding the instance of the problem is then placed in the input membrane in the initial configuration and the computation starts.

In this paper we propose a different way to supply the data related to the instance of the problem. In a similar way to the classical definition, a label is marked as *input label*, but instead of associating a multiset to the membrane with this label, we associate a set of rules which encodes the data of the problem. The main differences between this model and the classic model are:

- The data of the instance is involved in the computation when the rules are triggered.
- Since the membrane structure can change along the computation, the number of membranes with input label can also change. All these membranes are *input membranes*.

This can be formalized as follows.

Definition 1. *A* P system with rule input *is a tuple* (Π, i_Π, R_{i_Π})*, where* Π *is a P system, with working alphabet* Γ *and* n *membranes labelled by* $1, \ldots, n$*, and initial multisets* $w_1, \ldots, w_n$ *associated with them, and a finite set of rules* R*,* i_Π *is a distinguished label and* R_{i_Π} *is a set of rules disjoint of* R*. The set of rules* R_{i_Π} *is a new set of rules where the data of the problem is codified. It is associated to the input label* i_{Pi} *and it is added to the set of rules* R*. None of the rules from* R *is modified.*

In a certain sense, providing the instance of the problem via a set of rules is the *dual* process of providing it as a multiset. In both cases, the information is provided before the computation starts and the P system acts as a black box, but, on the one hand, if we provide an *initial multiset*, these elements interact with a set of rules which do not depend on the instance of the problem. On the other hand, if the data is supplied as a set of rules, these rules have to interact with the object of an initial configuration which does not depend on the instance.

The way in which the user obtains the computed answer is similar to usual recognizer P systems. We use an *external output* in the classical way (see, for example, [**?**]).

Definition 2. *A* recognizer P system with rule input and with external output*, is a P system with rule input* (Π, i_Π, R_{i_Π})*, such that*

- *The working alphabet contains two distinguished elements:* yes*,* no*.*
- *All of its computations halt.*
- *If* C *is a computation of* Π*, then either an object* yes *or an object* no *(but not both) has to be sent out to the external environment, and only in the last step of the computation.*

Let us note that a recognizer P system with rule input is also a confluent system in the following sense: every computation with the same initial configuration has the same output.

4 Linear solution to the Subset Sum problem

In this section we present a family $\prod$ of recognizer P systems with rule input that solves the Subset Sum problem in linear time.

The Subset-Sum problem can be stated as follows: *Given a finite set,* A*, a weight function,* $w : A \to \mathbb{N}$*, and a constant* $k \in \mathbb{N}$*, determine whether or not there exists a subset* $B \subseteq A$ *such that* $w(B) = k$*. If* A *has* n *elements with weights* $w_1, \ldots, w_n$*, one instance of the problem can be encoded as* $(n, (w_1, \ldots, w_n), k)$*.*

As usual in the framework of P systems, the solution of the problem is based on an algorithm of brute force where an exponential amount of workspace is built in linear time. In a similar way to the solution of **NP**-problems with P systems with active membranes, the algorithm is split in four stages:

- *Generation stage*: for every subset of A, a membrane is created.
- *Weight calculation stage*: in each working membrane the weight of the associated subset is calculated.
- *Checking stage*: in each membrane it is checked whether or not the weight of its associated subset is exactly k.
- *Output stage*: when the previous stage has been completed in all membranes, the system sends out the answer (*yes* or *no*) to the environment.

Next we describe the family $\prod$ of recognizer P systems with rule input that solves the Subset Sum problem. Given an instance $(n, (w_1, \ldots, w_n), k)$ of the

problem, the *recognizer P system with rule input* (Π, i_Π, R_{i_Π}) is built, where Π is a P system which only depends on n and the *input label* i_Π also depends only on n. The data corresponding to this specific instance are provided in the set of rules R_{i_Π} which, obviously, depends on k, on the weights $(w_1, \ldots, w_n)$ and on the label i_Π.

• Alphabet:

$$O = \left\{ \begin{array}{l} z_0, \ldots, z_{3n+11}, a_1, \ldots a_n, na_1, \ldots, na_n, w_1, \ldots, w_n, s_1, \ldots, s_{n-1}, \\ u_1, \ldots, u_{n-1}, i, k, q, i_0, \ldots, i_5, q_0, \ldots, q_8, ba, ca, da, br, cr, dr, \\ m_1, \ldots, m_6, dr_1, da_1, c, c_1, c_2, \#, no_0, no_1, h_0, \ldots, h_n, yes_0, \ldots, yes_3, \\ fn_1, fn_2, fn_3, yes, no \end{array} \right\}$$

• Set of labels: $H = \{e_0, e_1, \ldots, e_n, r, a, dr, da, d, f, h\}$
• Initial membrane structure: $\mu = [\,]_{e_0}$.
• Initial multiset $\mathcal{M}_s = z_0 a_1 n a_1$
• The set R of evolution rules consists of the following rules:

(a) $2n$ rules

$[a_i \to [a_{i+1} n a_{i+1} w_i]_{e_i}]_{e_{i-1}}$, for $i = 1, \ldots, n-1$.
$[na_i \to [a_{i+1} n a_{i+1}]_{e_i}]_{e_{i-1}}$, for $i = 1, \ldots, n-1$.
$[a_n \to [w_n s_1]_{e_n}]_{e_{n-1}}$,
$[na_n \to [s_1]_{e_n}]_{e_{n-1}}$,

These $2n$ rules create 2^n membranes at depth n (if the skin is at depth zero) in n steps. Note how the objects $w_1, \ldots, w_n$ are distributed in the created membranes. If each object w_i in a membrane e_j was sent to *all* children of e_j and the process goes on until every w_i were in a elementary membrane, we would obtain each one of the 2^n subsets of $\{w_1, \ldots, w_n\}$ in one of the 2^n elementary membranes. This is done by the next set of rules.

(b) $n^2 - n$ rules

$[w_i \to u_i^2]_{e_j}$, for $i = 1, \ldots, n-1$ and $j = i, \ldots, n-1$.
$u_i[\,]_{e_j} \to [w_i]_{e_j}$, for $i = 1, \ldots, n-1$ and $j = i+1, \ldots, n$.

Each w_i have to be sent to *two* membranes. This is done in two steps. In the first one, w_i evolves to two copies of u_i and then, in the second step, each copy of u_i send w_i into a child.

(c) $n - 1$ rules

$[s_i \to s_{i+1}]_{e_n}$, for $i = 1, \ldots, n-2$.
$[s_{n-1} \to ikq]_{e_n}$

The membranes with label e_n are created with the element s_1. This element is the first one of a sequence $s_1, \ldots, s_{n-1}$ which is used as a counter. When the element s_{n-1} is reached, it evolves to the set $\{i, k, q\}$. The element k

will evolve in the next step to represent the value of the constant k in an unary representation. In the meanwhile, i and q evolve to i_0 and q_0 in a waiting step by using rules of the following set.

(d) $n-1$ rules

$[a_i \to a_{i-1}]_{e_n}$, for $i = 2, \ldots, n$.

$[i \to i_0]_{e_n}$

$[q \to q_0]_{e_n}$

Besides the waiting rules for i and k, we also have rules for decreasing the index of elements a_i. By using this sequence $\{a_n, \ldots, a_1\}$ we get the synchronous apparition of a multiset of elements a_1 which encodes the whole weight of the corresponding subset.

(e) 16 rules

$$[a_1 \to ba, ca, da]_{e_n} \qquad [r \to br, cr, dr]_{e_n}$$

$$\begin{array}{lll} [ba \to [m_1]_a]_{e_n} & ca\,[\,]_r \to [c_1]_r & [c_1 \to c_2]_a \\ [br \to [m_1]_r]_{e_n} & cr\,[\,]_a \to [c_1]_a & [c_1 \to c_2]_r \\ [da \to da_1]_{e_n} & [da_1 \to [m_2]_{da}]_{e_n} & ca[\,]_{da} \to [c]_{da} \\ [dr \to dr_1]_{e_n} & [dr_1 \to [m_2]_{dr}]_{e_n} & cr[\,]_{dr} \to [c]_{dr} \end{array}$$

$$[c_2]_a \to \# \qquad [c_2]_r \to \#$$

The task of this set or rules is to compare the number of elements a_1 and r in each elementary membrane (labelled by e_n). The idea is the following: For each element a_1 a new element ca and a new membrane (labelled by a) will be created. Analogously, for each element r a new element cr and a new membrane (labelled by r) will be created. In the next steps, elements ca will be introduced (after renaming) into membranes labelled with r. These elements will produce the dissolution of such membranes. Analogously, elements cr will be introduced (after renaming) into membranes labelled with a. These elements will also produce the dissolution of such membranes.

If the initial number of elements a_1 and r are the same, after an appropriate number of steps all the membranes labelled with a or r disappear. Otherwise, a membrane labelled with a or r remains and it will be used as a flag for the computation in this membrane.

This comparison has other technical details. For example, we do not desire that *two* objects ca go into the same membrane labelled with r, so we need extra rules, but the process is essentially as described above.

(f) 10 rules

$[i \to i_0]_{e_n}$ $\quad$ $[i_0 \to i_1]_{e_n}$
$i_5[\,]_a \to [i_5]_a$ $\quad$ $[i_1 \to i_2]_{e_n}$
$i_5[\,]_r \to [i_5]_r$ $\quad$ $[i_2 \to i_3]_{e_n}$
$[i_5]_a \to no_0$ $\quad$ $[i_3 \to i_4]_{e_n}$
$[i_5]_r \to no_0$ $\quad$ $[i_4 \to i_5]_{e_n}$

As we saw above, by using the rules from set (e), if the number of elements a_1 and r are the same, then all membranes labelled with a or r dissappear. But if this does not happen, after an appropriate number of steps, at least one of this type of membranes remains. In this set (f) we have 10 rules which are the complement to this process. We have a counter with an initial object i (generated via a rule of the set (c)) and the sequence $i_0, i_1, \ldots, i_5$. When i_5 appears, if there exists a membrane with label a or r, this element goes into the membrane and dissolves it sending the element no_0. Otherwise, the element i_5 does not trigger any rule and the object no_0 does not appear.

(g) 16 rules

$[m_1 \to m_2]_a$ $\quad$ $[m_1 \to m_2]_r$
$[m_2 \to m_3]_a$ $\quad$ $[m_2 \to m_3]_r$ $\quad$ $[m_2 \to m_3]_{da}$ $\quad$ $[m_2 \to m_3]_{dr}$
$[m_3 \to m_4]_a$ $\quad$ $[m_3 \to m_4]_r$ $\quad$ $[m_3]_{da} \to \#$ $\quad$ $[m_3]_{dr} \to \#$
$[m_4 \to m_5]_a$ $\quad$ $[m_4 \to m_5]_r$
$[m_5 \to m_6]_a$ $\quad$ $[m_5 \to m_6]_r$
$[m_6]_a \to \#$ $\quad$ $[m_6]_r \to \#$

Membranes with label a or r are auxiliary membranes created for the comparison between the number of objects a_1 and r. When the comparison ends, these membranes are dissolved. This process is carried out by a counter. The membranes are created with the object m_1. When the counter $m_1, m_2, \ldots$ reaches m_6 the membrane is dissolved.
Membranes with labels da and dr are also created for the comparison. They catch the excess of objects ca and cr. When this work is done, they are also dissolved.

(h) $n + 8$ rules

$[q_i \to q_{i+1}]_{e_n}$, for $i = 0, \ldots, 7$.
$[q_8]_{e_j} \to q_8$, for $j = 2, \ldots, n$.
$[q_8]_{e_1} \to \#$,
When the element q_8 is generated, it produces sequentially the dissolution of all membranes except the skin.

(i) 9 rules

$[c_2 \to \#]_{e_n}$ $[c \to \#]_{e_n}$ $[m_4 \to \#]_{e_n}$
$[m_6 \to \#]_{e_n}$ $[i_5 \to \#]_{e_0}$ $[q_8 \to \#]_{e_0}$
$[no_2 \to \#]_{e_0}$ $[yes_2 \to \#]_{e_0}$ $[fn_3 \to \#]_{e_0}$

These are *cleaning rules*. When an object becomes useless, it is sent to #.

(j) 4n+12 rules

$[z_i \to z_{i+1}]_{e_0}$, for $i = 0, \ldots, 2n+5$.

$[z_{2n+6} \to z_{2n+7} h_0]_{e_0}$,

$[z_i \to z_{i+1}]_{e_0}$, for $i = 2n+7, \ldots, 3n+10$.

$[h_i \to h_{i+1}^2]_{e_0}$, for $i = 0, \ldots, n-1$.

$[h_n \to [yes_0]_h]_{e_0}$,

The counter z_i starts at z_0 and finishes at z_{3n+11}. When the object z_{2n+6} is reached it generates z_{2n+7} but also a new element h_0. This is the beginning of a new counter h_i. When h_n is reached, it creates a new membrane with label h containing yes_0.

(k) 5 rules

$[yes_0 \to yes_1]_h$ $no_0[\,] \to [no_0]_h$
$[yes_1 \to yes_2]_h$ $[no_0]_h \to no_1$
$[yes_2]_h \to yes_3$

Membranes with label h are generated with the object yes_0 inside. This object evolves t0 yes_2, and yes_2 dissolves the membrane and sends out an object yes_3. Simultaneously, if there exists an object no_0, it goes into the membrane and dissolves it. The evolution of the yes needs one step more, so at the end of the process there will exist objects yes_3 only if there are more membranes with label h than objects no_0.

(l) 20 rules

$[z_{3n+11} \to b_0, f_0]_{e_0}$,

$[yes_3 \to yes_4, d_0]_{e_0}$

$[yes_i \to yes_{i+1}]_{e_0}$ for $i = 4, 5, 6$.

$[yes_7[\,]_d \to [yes_7]_d$

$[yes_7[\,]_f \to [yes_7]_f$

$[yes_7]_d \to \#$

$[yes_7]_f \to yes$

$[f_i \to f_{i+1}]_{e_0}$ for $i = 0, 1$.

$[f_2 \to [fn_1]_f]_{e_0}$

$[fn_i \to fn_{i+1}]_f$ for $i = 1, 2$.

$[fn_3]_f \to no$

$[d_0 \to [\,]_d]_{e_0}$
$b_0[\,]_d \to [b_0]_d$
$b_0]_d \to \#$
$[yes]_{e_0} \to yes[\,]_{e_0}$
$[no]_{e_0} \to no[\,]_{e_0}$

Each workspace membrane which finds a positive response in the checking stage sends an object yes_3 to the skin. This set of rules controls the output stage:

- If there does not exist any object yes_3 in the skin, one object no is sent to the environment and the P system halts.
- If there exists some objects yes_3 in the skin, only one object yes is sent to the environment and the P system halts.

This finishes the description of the regular set of rules. This set of rules is the same for all instances of the problem Subset Sum where the set A has n elements. It does not depend on k or the weight of the objects. This information is supplied to the system through a specific set of rules which is described in the next section.

4.1 Rules for unary representation

The solution to the Subset Sum problem presented in this work is made via a family of P systems in a similar way to the solution of other numeric **NP** problems reached via P systems with active membranes.

In these families, the set of rules only depends on the *size* of the problem and the concrete data which describes a particular instance of the problem is provided as elements added in the initial configuration in a particular membrane called *input membrane*. This membrane is identified by its label, which can be called the *input label*.

In the solution presented above to solve Subset Sum, we also have a family of P systems which have a set of rules which depends of the size of the problem. In this model we also have *input membranes*, identified by the *input label*. The main difference is that the input which codifies the concrete instance of the problem is not a multiset introduced into the initial membranes, but a set of rules associated to the *input label*.

These rules are a set of $n + 1$ rules which express the weights $w_1, \ldots, w_n$ and k into the unary representation used by P systems. These rules are

$[w_i \to a_i^{w(a_i)}]_{e_n}$, for $i = 1, \ldots, n$.
$[k \to r^k]_{e_n}$

That is, when the element w_i reaches a membrane with label e_n, in the next

step this element evolves to a multiset which consists of as many objects a_i as the weight of the element a_i in the description of the subset sum problem. Analogously, when the object k reaches a membrane with label e_n, it evolves to a mutiset with K copies of r. For example, if we consider the Subset Sum problem with the set $A = \{a_1, a_2, a_3\}$ (i.e., $N = 3$), the weights $w_1 = w(a_1) = 1$, $w_2 = w(a_2) = 2$, $w_3 = w(a_3) = 1$, and $K = 3$, then the input of the P system will be codified as the following for rules:

$[w_1 \to a_1]_{e_3}$

$[w_2 \to a_2 a_2]_{e_3}$

$[w_3 \to a_3]_{e_3}$

$[k \to rrr]_{e_3}$

5 How it works

In this section we informally describe how the P system works. The comments are general, but we illustrate them with an example. As above, we consider the Subset Sum problem with the set $A = \{a_1, a_2, a_3\}$ (i.e., $N = 3$), the weights $w_1 = w(a_1) = 1$, $w_2 = w(a_2) = 2$, $w_3 = w(a_3) = 1$, and $K = 3$. The initial configuration does not depend on the instance of the problem:

$$[z_0, a_1, na_1]_{e_0}$$

The rules for the unary representation of the data are provided in the previous section.

5.1 The generation and weight calculation stages

In the solution presented here the *generation stage* and the *weight calculation stage* are developed simultaneously in the first $2n$ evolution steps. The algorithm is deterministic at this stage and it is the same for all the instances with the same number of elements n. Only at the end of the stage the rules for the unary representation are applied. In the first step two rules of the set (a) are applied and one of the set (j). Two new membranes are created and the counter z increases one unit. The new configuration has two membranes at level 1 (with label e_1) and the skin (with label e_0):

$$[z_1,\, [a_2, na_2, w_1]_{e_1},\, [a_2, na_2]_{e_1}]_{e_0}$$

Note that the elements a_1 and na_1 have created new membranes with label e_1. The index of the label e denotes its level. In general, elements a_i and na_i

create membranes with index e_i. The membrane created by a_i includes the element w_i. The evolution of these w_i will determine the differences among the workspace membranes.

The configuration at time two is

$$\begin{array}{l}[z_2,\ [\ u_1,u_1,\ [a_3,na_3,w_2]_{e_2}\ \ [a_3,na_3]_{e_2}\ ]_{e_1}\\ \qquad [\qquad\ \ [a_3,na_3,w_2]_{e_2}\ \ [a_3,na_3]_{e_2}\ ]_{e_1}]_{e_0}\end{array}$$

The counter z goes on increasing and four new membranes are created from the objects a_2 and na_2. The rule $[w_1 \to u_1, u_1]_{e_1}$ has been applied. In the first membrane with label e_1,

$$[u_1, u_1, [a_3, na_3, w_2]_{e_2}[a_3, na_3]_{e_2}]_{e_1}$$

there are two elements u_1 and two membranes with label e_2. In the next step the objects u_1 will send objects w_1 into these membranes. The configuration at time 3 is

$$\begin{array}{l}[z_3,\ [\ [u_2,u_2,w_1,\ [w_3,s_1]_{e_3}\ [s_1]_{e_3}]_{e_2}\ [w_1,\ [w_3,s_1]_{e_3}[s_1]_{e_3}]_{e_2}\ ]_{e_1}\\ \qquad [\ [u_2,u_2,\qquad [w_3,s_1]_{e_3}\ [s_1]_{e_3}]_{e_2}\ [\quad\ [w_3,s_1]_{e_3}[s_1]_{e_3}]_{e_2}\ ]_{e_1}]_{e_0}\end{array}$$

In this configuration we have already reached 2^3 workspace membranes (with label e_3). A new counter s appears. It is synchronized with the arrival of the objects w_i to the working space. When an object w_i arrives to the working space, it is split into the unary representation with the appropriate rule. When the counter s reaches s_{n-1}, this element generates the objects i, k, q. In the next step the generation stage finishes. In our example, this happens at time 6:

$$\begin{array}{l}[z_6,\ [[[q_0i_0r^3a_1^4]_{e_3}\ [q_0i_0r^3a_1^3]_{e_3}\ ]_{e_2}\ [[q_0i_0r^3a_1^2]_{e_3}\ [q_0i_0r^3a_1]_{e_3}\ ]_{e_2}]_{e_1}\\ \qquad [[[q_0i_0r^3a_1^3]_{e_3}\ [q_0i_0r^3a_1^2]_{e_3}\ ]_{e_2}\ [[q_0i_0r^3a_1]_{e_3}\ [q_0i_0r^3]_{e_3}\quad\ ]_{e_2}]_{e_1}]_{e_0}\end{array}$$

In every workspace membrane there are n copies of the object r, the objects q_0 and i_0 and different copies of a_1. These copies of a_i represent the weighs of the different subsets which can be built from the set A.

5.2 The checking stage

In this stage the 2^n workspace membranes work in parallel. In each membrane we have to compare if the number of objects r and a_1 are equal. This is done in a constant number of steps, exactly 8, regardless of the parameters N and K.

The cost we have to pay in order to develop the checking stage in a constant number of steps is the creation of a large amount of auxiliary membranes.

In order to follow this stage we focus our attention only to two workspace membranes from our example:

$$M_1 \equiv [q_0 i_0 r^3 a_1^2]_{e_3} \qquad M_2 \equiv [q_0 i_0 r^3 a_1^3]_{e_3}]_{e_2}$$

The answer in M_1 has to be negative, and positive (more precisely, *no answer*) in $M2$.

In the next step (time 7) the counters i and q increase untill i_1 and q_1. Each object a_1 and r generates three new objects ba, ca, da, and br, cr, dr, respectively.

$$M_1 \equiv [q_1, i_1, ba^2, ca^2, da^2, br^3, cr^3, dr^3]_{e_3}$$
$$M_2 \equiv [q_1, i_1, ba^3, ca^3, da^3, br^3, cr^3, dr^3]_{e_3}$$

The basic idea is the following. Each object ba creates one membrane with label a and, analogously, each object br creates one membrane with label r. In the step, the elements ca are sent into membranes with label r and elements cr are sent into membranes with label a. This is done in a maximal parallel manner and, if a rule can be applied, it is applied. So, if after this process there exists an object ca or cr in the membrane with label e_n, the reason is because the number of elements a_1 and r was different. If the number of $_1$ and r are equal, none of these objects remains in the membrane e_3. In the next step, we have:

$$M_1 \equiv [ca^2, cr^3, \ldots, [m_1]_r, [m_1]_r, [m_1]_r, [m_1]_a, [m_1]_a]_{e_3}$$
$$M_2 \equiv [ca^3, cr^3, \ldots, [m_1]_r, [m_1]_r, [m_1]_r, [m_1]_a, [m_1]_a, [m_1]_a]_{e_3}$$

and in the following step

$$M_1 \equiv [cr, \ldots, [m_2, c_1]_r, [c_1, m_2]_r, [m_2]_r, [m_2, c_1]_a, [m_2, c_1]_a]_{e_3}$$
$$M_2 \equiv [\ldots, [m_2, c_1]_r, [m_2, c_1]_r, [m_2, c_1]_r, [m_2, c_1]_a, [m_2, c_1]_a, [m_1, c_1]_a]_{e_3}$$

In the next step, we do not want that the object cr that has not evolved in M_1 go into a membrane with label a. In P systems with electric charges, this can avoided by changing the polarization, but we do not have that tool in this model.

This problem is solved by using the objects da and dr which create new membranes whose task is to collect these remaining objects. The whole process can be seen in the next example (where $[\ldots]_p^n$ denotes that there exist n membranes $[\ldots]_p$.

At time t: $[i_0, q_0, a_1, r^2]$
At time t+1: $[i_1, q_1, ba, ca, da, br^2, cr^2, dr^2]$
At time t+2: $[i_2, q_2, [m_1]_a, ca, da_1, [m_1]_r^2, cr^2, dr_1^2]$
At time t+3: $[i_3, q_3, [m_2, c_1]_a, [m_2]_{da}, [m_2]_r, [m_2, c_1]_r, cr, [m_2]_{dr}^2]$
At time t+4: $[i_4, q_4, [m_3, c_2]_a, [m_3]_{da}, [m_3]_r, [m_3, c_2]_r, [m_3, c]_{dr}, [m_3]_{dr}]$
At time t+5: $[i_5, q_5, m_4^2, c, \#^5, [m_4]_r]$
At time t+6: $[q_6, \#^8, [m_5, i_5]_r]$
At time t+7: $[q_7, \#^8, m_6, no_0]$
At time t+8: $[q_8, \#^9, no_0]$

Some comments to this example:

- This stage has the same number of steps regardless of N, K, and the weights of the elements.
- The stage is not deterministic. The element cr that appears at time 3 in the next step can go to a membrane with label dr (as it happens) or with label a, but the system is confluent, since both membranes dissolve and the useless objects are sent to #.
- An element no_0 appears only if there exists an alive membrane with label a or r when the counter i reaches i_5. As said before, if the number of elements a_1 are equal at the beginning of the checking stage, no membrane with label a or r survives untill i_5. In this case, the absence of no_0 is considered as a positive answer.

Following our example ($A = \{a_1, a_2, a_3\}$ (i.e., $N = 3$), the weights $w_1 = w(a_1) = 1$, $w_2 = w(a_2) = 2$, $w_3 = w(a_3) = 1$, and $K = 3$), at time 14 the next configuration is obtained:

$$[z_{14}, h_1^2\ [[[q_8\#^{21}no_0]_{e_3}\ [q_8\#^{18}i_5]_{e_3}\ ]_{e_2}\ [[q_8\#^{15}no_0]_{e_3}\ [q_8\#^{12}no_0]_{e_3}\ ]_{e_2}]_{e_1}$$
$$[[[q_8\#^{18}i_5]_{e_3}\ [q_8\#^{15}no_0]_{e_3}\ ]_{e_2}\ [[q_8\#^{12}no_0]_{e_3}\ [q_8\#^{9}no_0]_{e_3}\ ]_{e_2}]_{e_1}]_{e_0}$$

In six workspace membranes the object no_0 is obtained. In the remaining two, this object does not appears. These two membranes have obtained a positive answer. Note that in the skin the counter h has already started to grow.

5.3 Output stage

In this stage the answers from the workspace membranes are collected. It has two substages:

- *Simplification.* In this stage all the workspace and auxiliary membranes are dissolved. In this way, all the elements no_0 (if any exists) reach the skin.

- *Checking.* In this stage 2^n new auxiliary membranes with label h are generated and the objects no_0 are sent into them. If there exists 2^n objects no_0 in the skin, all these membranes will stop their inner process and the object no is sent to the environment. If the number of no_0 in the skin is less than 2^n, this means that at least one of the workspace membrane had a positive answer. In this case at least one of the membranes with label h is not stopped by no_0 and the object yes is sent to the environment.

The *simplification* is performed by the object q_8. It appears in all workspace membranes and dissolves them and the auxiliary membranes as well. In our example, at time 16 we have

$$[z_{16}, h_3^8 \, [q_8^4 \#^{54} i_5 no_0^3]_{e_1} \, [q_8^4 \#^{66} i_5 no_0^3]_{e_1}]_{e_0}$$

In the next step, both membranes at level 1 are dissolved under the action of object q_8, but the counter h has reached h_n and 2^n new membranes with label h are created. At time 17, we have

$$[z_{17}, \#^{122} q_8^6 i_5 no_0^3 \, [yes_0]_h^8]_{e_0}$$

The computation follows as described above, and at time 20 we have

$$[z_{20}, \#^{142} yes_3^2]_{e_0}$$

Here we find *two* objects yes_0, but only one element yes has to be sent to the environment. This stage is managed by rules of the set (l). For that, new membranes with label d and f are created. Finally, at time 26 we have again only one membrane

$$[\#^{144} f n_3 yes]_{e_0}$$

In the next step (time 27), the object yes is sent to the environment and the system reaches a halting configuration.

6 Some formal details

We have presented a uniform family of P systems which solves the Subset Sum problem. The construction of the P systems is linear in the parameter n, in the number of objects of the alphabet, and quadratic in the number of rules, so each P system can be built in polynomial time.

The number of steps of the P system is also linear in n:

- $3n + 18$ if the answer is *yes*,
- $3n + 19$ if the answer is *no*.

7 Conclusion and future work

The key for solving **NP**-complete problems in the membrane computing framework is the possibility to generate an exponential workspace in polynomial time.

The processes by which this exponential number of membranes is created in polynomial time are basically two: *mitosis* (membrane division) and *autopoiesis* (membrane creation). In this paper we explore a new semantics for P systems with membrane creation, closer to the semantics of P systems with active membranes than the semantics from [**?**, **?**, **?**]) and remark the necessity of set a standard semantics for this recently explored and fruitful model of P systems.

The main novelty of this paper is the concept of P systems with rule input. As pointed above, in a certain sense, providing the instance of the problem via a set of rules is the *dual* process of providing it as a multiset, since the P system which solves a problem only depends on the size of the problem and we only provide the data in a different way. In one case, we modify the multisets placed in the initial configuration adding the data and the set of rules remains unchanged. In the second case, the initial configuration remains unchanged and the data is supplied by *adding* new rules which codify the information.

One of the main drawbacks of this new approach is the fact that several membranes can be labelled with the input label (since new membranes with this label can be created) and in this way the input data can be supplied several times (even at different steps) along the computation and this feature goes against the simplicity principle. Therefore, a further and deeper study must be done in this line.

Acknowledgement

The support for this research through the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds, is gratefully acknowledged.

References

1. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: A fast P system for finding a balanced 2-partition. *Soft Computing*, in press.
2. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving SAT with membrane creation. Accepted paper for CiE 2005.
3. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A linear solution for QSAT with membrane creation. Submitted.

4. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A linear solution of Subset Sum by using membrane creation. Submitted.
5. P.L. Luisi: The chemical implementation of autopoiesis. In *Self-Production of Supramolecular Structures* (G.R. Fleishaker et al., eds.), Kluwer, Dordrecht, 1994.
6. Gh. Păun: Further open problems in membrane computing. In *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds.), Report RGNC 01/04, University of Seville, 2004, 354–365.
7. M.J. Pérez-Jiménez, A. Riscos-Núñez: Solving the Subset-Sum problem by active membranes. *New Generation Computing*, in press.
8. M.J. Pérez-Jiménez, A. Riscos-Núñez: A linear solution for the Knapsack problem using active membranes. In *Membrane Computing* (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), *Lecture Notes in Computer Science*, **2933**, 2004, 250–268.
9. M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving the BIN PACKING problem by recognizer P systems with active membranes. In *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds.), Report RGNC 01/04, University of Seville, 2004, 414–430.
10. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. In *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003* (E. Csuhaj-Varjú, C. Kintala, D. Wotschke, Gy. Vaszyl, eds.), 2003, 284–294.