P SYSTEMS WITH INPUT IN BINARY FORM

ALBERTO LEPORATI*
CLAUDIO ZANDRON†
Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy

and

MIGUEL A. GUTIÉRREZ-NARANJO‡
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Sevilla University
Avda Reina Mercedes s/n, 41012 Sevilla, Spain

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Current P systems which solve **NP**–complete numerical problems represent instances in unary notation. In classical complexity theory, based upon Turing machines, switching from binary to unary encoded instances generally corresponds to simplify the problem. In this paper we show that this does not occur when working with P systems. Namely, we propose a simple method to encode binary numbers using multisets, and a family of P systems which transforms such multisets into the usual unary notation.[a]

Keywords: Membrane Computing; P systems; Binary data; Partition

1. Introduction

P systems (also called *membrane systems*) were introduced in [**?**] as a new class of distributed and parallel computing devices, inspired by the structure and functioning of living cells. The basic model consists of a hierarchical structure composed by several membranes, embedded into a main membrane called the *skin*. Membranes divide the Euclidean space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and *evolution rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one. The rules are applied in a nondeterministic and maximally parallel way: all the objects that may evolve are

*leporati@disco.unimib.it.
†zandron@disco.unimib.it.
‡magutier@us.es

forced to evolve. A *computation* starts from an initial configuration of the system and terminates when no evolution rule can be applied. The result of a computation is the multiset of objects contained into an *output membrane* or emitted to the *environment* from the skin of the system.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems[b].

Many P systems which solve **NP**–complete decision problems have appeared in the literature during the last few years. Both in the field of *numerical* problems, that is, problems whose instances consist of sets or sequences of integer numbers (see for example Subset Sum [**?**], Knapsack [**?**], Bin Packing [**?**] or Partition [**?**] problems) or non-numerical problems as SAT [**?**, **?**] or QSAT [**?**].

It is well known [**?**, **?**] that the difficulty of such numerical problems is tied to the magnitude of the numbers which appear into the instance. For example, let us consider the PARTITION problem, which can be stated as follows:

Problem 1.1 NAME*:* PARTITION*.*

- INSTANCE*: a set* $A = \{a_1, a_2, \ldots, a_n\}$ *of positive integer numbers*
- QUESTION*: is there a subset* $A' \subseteq A$ *such that* $\sum_{a' \in A'} a' = \sum_{a \in A \setminus A'} a$*?*

The following algorithm solves the problem using the well known Dynamic Programming technique [**?**]. In particular, the algorithm returns 1 on positive instances, and 0 on negative instances.

```
PARTITION({a_1, a_2, ..., a_n})
s ← Σ_{i=1}^{n} a_i
if s mod 2 = 1 then return 0
for j ← 1 to s/2
    do M[1, j] ← 0
M[1, 0] ← M[1, a_1] ← 1
for i ← 2 to n
     do for j ← 0 to s/2
              do M[i, j] ← M[i − 1, j]
                  if j ≥ a_i and M[i − 1, j − a_i] > M[i, j]
                     then M[i, j] ← M[i − 1, j − a_i]
return M[n, s/2]
```

First of all, the algorithm computes the sum s of all elements in the instance. If s is odd then the instance is certainly negative, and thus the algorithm returns 0. If s is even then the algorithm checks for the existence of a subset $A' \subseteq A$ such that $\sum_{a' \in A'} a' = \frac{s}{2}$. In order to look for A', the algorithm uses a $n \times (\frac{s}{2} + 1)$ matrix M whose entries are from $\{0, 1\}$. It fills the matrix by rows, starting from the first row. Each row is filled from left to right. The entry $M[i, j]$ is filled with 1 if and

[b] A layman-oriented introduction can be found in [**?**]; a formal description in [**?**] and the latest information about P systems can be found on [**?**].

only if there exists a subset of $\{a_1, a_2, \ldots, a_i\}$ whose elements sum up to j. The given instance of PARTITION is thus a *positive* instance if and only if $M[n, \frac{s}{2}] = 1$ at the end of the execution.

Since each entry is considered exactly once to determine its value, the time complexity of the algorithm is proportional to $n(\frac{s}{2} + 1) = \Theta(ns)$. This means that the difficulty of the problem depends on the value of s, that is, on the magnitude of the values in A. In fact, let us denote by K the maximum element of A. If K is polynomially bounded w.r.t. n then also $s = \sum_{i=1}^{n} a_i \leq Kn$ is polynomially bounded w.r.t. n, and thus the above algorithm works in polynomial time. On the other hand, if K is exponential w.r.t. n, say $K = 2^n$, then also s is exponential and the above algorithm works in exponential time and space. This behavior is usually referred to in the literature by telling that the PARTITION problem is a *pseudo–polynomial* **NP**–complete problem.

The fact that in general the above algorithm is not a polynomial time algorithm for PARTITION can be immediately understood by comparing its time complexity with the instance size. The usual size for the instances of PARTITION is $\Theta(n \log K)$ (also $O(n \log s)$ in [**?**, page 91]), since for conciseness every "reasonable" encoding is assumed to represent each element of A using a string whose length is $O(\log K)$. Here all logarithms are taken with base 2. Stated differently, the size of the instance is usually considered to be the number of bits which must be used to represent in binary all the integer numbers which occur in A. If we would represent such numbers using the unary notation, then the size of the instance would be $\Theta(nK)$. But in this case we could write a program which first converts the instance in binary form and then uses the above algorithm to solve the problem in polynomial time with respect to the new instance size. We can thus conclude that the difficulty of a numerical **NP**–complete problem depends also on the measure of the instance size we adopt.

The fact that the difficulty of a problem generally depends upon how we measure the instance size is even more apparent if we consider the FACTORIZATION problem:

Problem 1.2 NAME*:* FACTORIZATION*.*

- INSTANCE*: a positive integer number n which is the product of two prime numbers p and q*
- OUTPUT*: p*

This problem is generally considered *intractable*, which means that no polynomial time algorithm is known that solves it on every instance. The conjectured intractability of this problem is often exploited in Cryptography: a notable example is the RSA cryptosystem [**?**]. Here the natural instance size for the problem is $\Theta(\log n)$, the number of bits which are needed to represent n in binary form. Also for this problem, if we let the instance size be $\Theta(n)$ then the trivial algorithm which tries to divide n by every number comprised between 1 and $\sqrt{n}$ is a polynomial time algorithm which solves the FACTORIZATION problem.

For these reasons we believe that it is important to show that P systems which solve **NP**–complete numerical problems do not take their power from the fact that the instances are represented in unary notation. Hence in this paper we first propose

a simple method to represent positive integer numbers in binary notation using multisets of objects. Then, we propose a family of P systems which transforms this binary encoding into the unary notation used in [**?**, **?**, **?**, **?**].

The paper is organized as follows. In section 2 we introduce our encoding of binary numbers using multisets. In section 3 we propose a family of simple P systems which can be used to transform a given positive integer number from such encoding to unary notation. Section 4 concludes the paper and gives some directions for future research.

2. Encoding binary numbers using multisets

First of all let us show how a given positive integer number x can be represented in binary notation using a multiset. Let $x_n, x_{n-1}, \ldots, x_1$ be the binary representation of x, so that $x = \sum_{i=1}^{n} x_i 2^{i-1}$. We use the objects from the following alphabet:

$$\mathcal{A}_n = \{\langle b, j\rangle \mid b \in \{0,1\}, j \in \{1,2,\ldots,n\}\} \tag{1}$$

Object $\langle b, j\rangle$ is used to represent bit b into position j in the binary encoding of an integer number. Hence, to represent the above number x we will use the following multiset (actually, a set) of objects:

$$\langle x_n, n\rangle, \langle x_{n-1}, n-1\rangle, \ldots, \langle x_1, 1\rangle$$

Let us remark that the alphabet $\mathcal{A}$ depends on the length of the binary representation of the number x, i.e., with the alphabet $\mathcal{A}_n$ we can represent from 1 to $2^n - 1$.

On the other hand, the unary representation of x is obtained by choosing a symbol from an alphabet, say the symbol a from alphabet $\mathcal{A}'$, and putting into the multiset x copies of such symbol: a^x. Hence, unary notation is exponentially longer than binary notation. Our transformation thus solves another problem raised by the solutions exposed in [**?**, **?**, **?**, **?**]: in order to provide the input values to the P systems, we should insert into such systems an exponential (with respect to the instance size) number of objects. This means that an exponential amount of work to prepare the system is required.

Working with binary encoded numbers, instead, allows one to prepare the system by inserting a polynomially bounded number of objects.

3. Converting from binary to unary notation

In this section we propose a family of simple P systems which allows to convert a given positive integer number x, expressed in binary notation as exposed in the previous section, to the usual unary notation.

The objects used by the P systems form a subset of alphabet $\mathcal{A}$ of equation (**??**). Namely, in order to represent x in binary notation we will use only the objects which correspond to the bits of x which are equal to 1. For example, if $x = 25$ then its binary representation is 11001, and we will use the objects $\langle 1, 5\rangle$, $\langle 1, 4\rangle$, and $\langle 1, 1\rangle$ to represent it. Since the first element in the pairs of $\mathcal{A}$ used is always equal to 1,

we can be more concise by omitting it. Once omitted the first element of the pair, also angular parenthesis are superflous.

The family of P systems which performs the transformation is formally defined as follows:

$$\Pi(n) = (A(n), \mu, w, R(n), i_{\text{in}}, i_{\text{out}})$$

where:

- $A(n) = \{1, 2, \ldots, n\} \cup \{a\}$ is the alphabet;
- $\mu = [\,]_{skin}$ is the membrane structure consisting of the skin only;
- $w = \emptyset$ is the multiset of objects initially present in region 1;
- $R(n)$ is the following set of evolution rules associated with region 1:

$$[j \to (j-1)^2]_{skin} \qquad \text{for all } j \in \{2, 3, \ldots, n\}$$
$$[1 \to a]_{skin}$$

- $i_{\text{in}} = skin$ specifies the input membrane of Π;
- $i_{\text{out}} = skin$ specifies the output membrane of Π.

The semantics of the rules is the usual for evolution rules. All they are applied in a maximal parallel mode. The number of cellular steps of the P system is bounded by n and the computation halts when no more rules can be applied. When this happens, the multiset placed in the output membrane (the only one membrane) is the *output* of the computation.

Computations proceed as follows. The objects which denote the positions of 1's in the binary representation of x are initially put into the region enclosed by the skin. Then the computation starts, and the rules from R are applied. It is easily seen that the presence of object j, with $j \in \{1, 2, \ldots, n\}$, will produce 2^{j-1} copies of object a. Hence at the end of the computation, when no more rules from R can be applied, the skin will contain x copies of object a, that is, the unary representation of x.

We conclude this section with an example of computation of the above P systems.

Let us consider again the value $x = 25$; as previously said, it will be represented by means of objects 5, 4, and 1 (each in a unique copy). At the first step of computation, we apply in parallel the rules $1 \to a$, $4 \to 3, 3$ and $5 \to 4, 4$, obtaining the multiset $a, 3, 3, 4, 4$.

Then, we apply in parallel the rule $3 \to 2, 2$ on each copy of the symbol 3, thus obtaining four copies of the symbol 2, and the rule $4 \to 3, 3$ on each copy of the symbol 4, thus obtaining four copies of the symbol 3. The multiset we obtain after the second step of computation will be $a, 2, 2, 2, 2, 3, 3, 3, 3$.

Hence, we apply the rules $2 \to 1, 1$ and $3 \to 2, 2$ obtaining $a, 1^8, 2^8$. By means of the rules $1 \to a$ and $2 \to 1, 1$ we then obtain the multiset $a^9, 1^{16}$ and finally, applying again $1 \to a$ we obtain the multiset a^{25} which is exactly the unary codification of the initially binary coded number.

From the previous definition and example, it is easy to see that the cardinality of the alphabet and the number of computation steps are linear with respect to the input size.

4. Composition of P systems

In the previous section, a method for converting natural number from binary into unary notation has been described. Intuitively, such a P system could be *composed* with a P system which solves an instance of a problem with input in unary form and to get a new P system which solves the same problem with input in binary form.

The formalization of such intuition has several technical details and, to the best of our knowledge, the composition of P systems has not been defined.

In this section we present a definition for composing P systems which fit into our purposes. The general definition and the study of its properties lies out of the scope of this paper. First we define a *joint* of two P systems

A *joint* $P_1 \circ P_2$ of P systems P_1 and P_2 is a new P system where the skin membrane of P_2 is identified to an elementary membrane of $P1$. In this way we obtain a new labelled membrane structure. Each labelled membrane keeps its initial multiset and set of rules. In the initial configuration, the new membrane obtained by identification, the initial multiset and set of rules are the union of the multisets and sets of rules of the identified membranes. Next we give a formal definition.

Definition 1 *Let* $P_1 = (O_1, H_1, EC_1, \mu_1, w_1^1, \ldots, w_{m_1}^1, R_1)$ *and* $P_2 = (O_2, H_2, EC_2, \mu_2, w_1^2, \ldots, w_{m_2}^2, R_2)$ *be two P systems where:* $m_1, m_2 \geq 1$ *are the initial degrees of the systems;* O_1 *and* O_2 *are the alphabets of objects;* H_1 *and* H_2 *are two disjoint finite set of labels for membranes;* $EC_1 = EC_2$ *are the finite sets of electrical charges for membranes;* μ_1 *and* μ_2 *are the membrane structures consisting (resp.) of* m_1 *and* m_2 *membranes labelled (not necessarily in a one-to-one manner) with elements of* H_1 *and* H_2*;* $w_1^i, \ldots, w_m^i$ *are strings over* O_i*, describing the multisets of objects placed in the* m_i *regions of* μ_i *(for i=1,2);* R_1 *and* R_2 *are the finite sets of rules associated to* P_1 *and* P_2*.*

Let i_1 *be the label of an elementary membrane of* P_1 *and* s_2 *the label of the skin membrane of* P_2*. And let* μ *be the membrane structure obtained by identifying* i_1 *with* s_2*. Since the none* i_1 *is a leave in* μ_1 *and the node* s_2 *is the root of* μ_2 *the new graph is also a membrane structure. We keep the same label for all membranes and label the joint membranes by* α*.*

We define a joint $P_1 \circ P_2$ *as a P system*

$$P_1 \circ P_2 = (O, H, EC, \mu, w_1, \ldots, w_m, R)$$

where $m = m_1 + m_2 - 1$ *is the initial degree of the systems;* $O = O_1 \cup O_2$ *is the alphabet of objects;* $H = (H_1 - \{i_1\}) \cup (H_2 - \{s_2\}) \cup \{\alpha\}$ *is the finite set of labels for membranes;* $EC = EC_1 = EC_2$ *is the finite set of electrical charges for membranes;* μ *and is the membrane structure labelled with elements of* H*;* $w_1, \ldots, w_k$ *are strings over* O*, describing the multisets of objects placed in the* m *regions of;* $R = R_1 \cup R_2$ *is the finite set of rules.*

Note that given two P systems with the same set of electrical charges (which can be empty) there exists several posibilities of getting a *joint*: One for each elementary membrane of P_1. Next we define the *composition* of two P systems. In order to define such composition we need two P systems with *input* and *output*.

Definition 2 *Let P_1 and P_2 be two P systems with input and output such that:*

- *The input membrane of P_1 is an elementary membrane. We will denote by i_1 the label of such membrane.*
- *The output membrane of P_2 is the skin membrane. We will denote by s_2 the label of such membrane.*

The composition $P_1 \circ P_2$ is the joint obtained by identifying i_1 with s_2.

Note that if P_2 sends the output to the environment, we can consider a new external membrane surrounding the whole P system which becomes the new skin. With this new skin we can consider the composition with another P system.

5. A Case Study

In this section we describe two familes of P systems Π_B ans P_{part}:

- The family $\Pi_B = \{P^B(n,d) \,:\, n,d \in N\}$ converts multisets of natural numbers from binary into unary notation. The P system $P^B(n,d)$ depends on the number of elements that we want to convert and on d, where d is defined by

$$d = Ent[log_2(max\, A)] + 1 \tag{2}$$

  where A is the set of numbers to convert.

- The family $\Pi_{part} = \{P^{part}(n) \,:\, n, \in N\}$ is a uniform family which solves the **NP**-problem PARTITION. It is based on the solution presented in ... but with small changes. Each P system $P^{part}(n)$ solves all instances of the problem with n elements. The solution is obtained in polynomial time on n and the input has to be provided in unary form.

Both families are designed with *input* and *output* and it has sense to consider the composition of P systems of both families. We obtain the following family:

$$\Pi = \{P^{part}(n) \circ P^B(n,d) \,:\, n,d \in N\}$$

where each P system $P(n,d) = P^{part}(n) \circ P^B(n,d)$ is a cellular device which solves all the instances of the PARTITION problem with the same parameters n and d.

5.1. The family Π_B

The P systems of this family are adapted from the model presented in the section **??**. The differences are mainly two: Two membranes are considered, one as *input* membrane and the second one (the *skin*) is the *output* membrane. In this way we prepare the composition with P systems of the second family.

The second difference is due to technical reasons. We add new elements which has no meaning in the encoding of the information, but they make sense after the composition (objects e_0, z and f) and a counter v_1.

The problem can be stated as follows: *Given a multiset A of natural numbers expressed in binary form, to get a multiset A' with such numbers expressed in binary form.*

We adapt the description from section refsec:enc. Instead of codifying a single natural number, we look for a P system which convert a *multiset* of numbers. So, for each element in the multiset, we consider a mark $\{x_1, x_2, \ldots\}$, so in this way, following section **??** the multiset $\{3, 4, 3, 11\}$ can be expressed in binary form as the set of pairs

$$\{(x_1, 1), (x_1, 1), (x_2, 3), (x_3, 1), (x_3, 2), (x_4, 1)(x_4, 2), (x_4, 4)\}$$

where (x_i, j) represents that the $i - th$ element in the enumeration of the multiset has *one* in the $j - th$ position of the binary representation.

We define the family $\Pi_B = \{P^B(n, d) : n, d \in N\}$ where each $P^B(n, d)$ solves all the instances of the problem with the same number of elements n and the same bound d, defined in the equation **??**. (In fact, these n and d are upper bounds).

The P system $P^B(n, d) = (O(n, d), H, EC, \mu, w_t, w_s, \ldots, R(n, d))$ is defined as follows:

- $O(n, d) = \{e_0, z, f\} \cup \{y_1, \ldots, y_n\} \cup \{v_1, \ldots, v_{d+1}\}$
  $\cup \{(x_i, j) : 1 \leq i \leq n, 1 \leq j \leq d\}$
- $H = \{t, s\}$ with t the label of the input membrane and s, the *skin* the label of the output membrane.
- $EC = \emptyset$ (We can also consider the membranes with neutral charge along all the computation)
- $\mu = [\,[\,]_t\,]_s$
- $w_t = \{e_0, f, v_1\}$; $w_s = \emptyset$
- The following set of rules $R(n, d)$:

  $[(x_i, j) \to (x_i, j - 1)]_t$ for all $i \in \{1, \ldots n\}$ and $j \in \{1, \ldots, d\}$.
  $[(x_i, 1) \to y_i]_t$ for all $i \in \{1, \ldots n\}$.
  $[v_j \to v_{j+1}]_t$ for all $j \in \{1, \ldots, d\}$.
  $[v_{d+1}]_t \to z$.

All the rules are associated to the label t and are *object evolution rule*. The only exception is the last one, which is a *dissolution* rule.

At the beginning of the computation, the *input* codifying the multiset of natural numbers in binary form (as described above) is placed in the *input membrane*. The P system evolves as described in section **??**. After d steps the membrane t contains the elements e_0, f, v_{d+1} and a multiset of elements y_i, $1 \leq i \leq n$ codifying the *input*. In the next step v_{d+1} dissolves the membrane t and is transformed into z in

the skin. The remaining objects also go to the skin. No more rules can be applied and the computation halts.

The computation is deterministic and halts after $d + 1$ steps.

5.2. The family Π_{part}

This family is a uniform family[c] of P systems in the framework of active membranes (see ...) which solves the **NP**-problem PARTITION. It is based on the solution presented in ... but with small changes. Each P system $P^{part}(n)$ solves all instances of the problem with n elements. The solution is obtained in polynomial time on n and the input has to be provided in unary form. Each P sysstem of the family, $P^{part}(n)$, $n, \in N$ consists on the following elements:

- $O(n) = \{a_0, a, b_0, b, c, d_0, d_1, d_2, f, g, g_0, g_1, h_0, h_1, p_0, p, q, z, \#, yes, no, no_0\} \cup \{e_0, \ldots, e_n\} \cup \{i_1, i_2, i_3, i_4\} \cup \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\} \cup \{z_1, \ldots, z_{2n+1}\}$
- $H = \{e, r, s\}$; the skin s, the label e for the working membranes and a label r for the membrane of control.
- $EC = \{+, -, 0\}$
- $\mu = [\,[\,]r\,[\,]_e\,]_s$
- $w_e = \emptyset$; $w_s = \emptyset$; $w_r = \{b_0, h_0\}$
- The set of rules $R(n)$ described below. We follow the design for solving PARTITION with active membranes presented in ..., with small changes due to technical reasons. A detailed description and motivation for the rules can be found there.

[c] In the sense of ...

Set (a)
$[e_i]_e^- \to [q]_e^0[e_i]_e^+$, for $i = 1, \ldots, n$,
$[e_i]_e^+ \to [e_{i+1}]_e^-[e_{i+1}]_e^+$, for $i = 1, \ldots, n-1$.

Set (b)
$[x_1 \to a_0]_e^-, \quad [x_1 \to p_0]_e^+$,
$[x_i \to x_{i-1}]_e^+$, for $i = 2, \ldots, n$,
$[x_i \to p_0]_e^0$, for $i = 2, \ldots, n$.

Set (c)
$[e_n]_e^+ \to \#$, $[a_0 \to \#]_s^0$,
$[p_0 \to \#]_s^0, \quad [g \to \#]_s^0$.

Set (d)
$[q \to i_1]_e^0, \quad [p_0 \to p]_e^0$,
$[a_0 \to a]_e^0, \quad [g]_e^0 \to [\ ]_e^0 g_0$.

Set (e)
$[a]_e^0 \to [\ ]_e^- \#, \quad [p]_e^- \to [\ ]_e^0 \#$.

Set (f)
$[i_1 \to i_2]_e^0, \quad [i_2 \to i_1]_e^-$.

Set (g)
$[i_1]_e^- \to [\ ]_e^+ no$.

Set (h)
$[i_2 \to i_3 c]_e^0$,
$[c]_e^0 \to [\ ]_e^- \#, \quad [i_3 \to i_4]_e^-$,
$[i_4]_e^- \to [\ ]_e^+ Yes, \quad [i_4]_e^0 \to [\ ]_e^+ no$.

Set (i)
$[p \to \#]_e^+, \quad [a \to \#]_e^+$.

Set (j)
$[z_i \to z_{i+1}]_s^0$, for $i = 1, \ldots, 2n$,
$[z_{2n+1} \to d_0 d_1]_s^0$,
$d_0[\ ]_r^0 \to [d_0]_r^-, \quad [d_1]_s^0 \to [\ ]_s^+ d_1$,
$[g_0 \to g_1]_s^+$,
$g_1[\ ]_e^+ \to [g_1]_e^-$.

Set (k)
$[h_0 \to h_1]_r^-, \quad [h_1 \to h_0]_r^+$,
$[b_0]_r^- \to []_r^+ b, \quad g_1[\ ]_r^+ \to [g_1]_r^-$,
$b[\ ]_r^- \to [b_0]_r^+, \quad [g_1]_r^+ \to [\ ]_r^- g_1$,
$[h_0]_r^+ \to [\ ]_r^+ d_2, \quad [d_2]_s^+ \to [\ ]_s^- d_2$.

Set (l)
$[no \to no_0]_s^-$,
$[yes]_s^- \to [\ ]_s^0 yes$,
$[no_0]_s^- \to [\ ]_s^0 no$.

The following set of rules does not appear in the design presented in ...

Set (m)
$[z]_e^0 \to [\ ]_e^- z_1, \qquad [e_0 \to e_1]_e^0, \qquad [f \to g]_e^0$
$[y_i \to x_i]_e^0$ for $i = 1, \ldots, n$

The first rule of the set, $[z]_e^0 \to [\ ]_e^- z_1$ is the starting point of the clock. The remaining rules are simply renaming.

Next we describe the *input* of the P system. In this input we provide the data of the instance of the PARTITION in unary mode. For that we consider the objects y_i of the alphabet. Given a multiset of natural numbers $\{k_1, \ldots, k_m\}$ representing an instance of the problem, we place in the input membrane as many objects y_i as indicated by k_i. For example, the multiset associated to $\{2, 3, 2, 1, 3\}$ is $\{y_1^2, y_2^3, y_3^3, y_4, y_5^3\}$. We also place in the input set three objects: e_0, f, z. These objects do not depend on the instance of the problem and can be considered as the starting point of the clock of the P system.

Notice that the amount of resources needed to build a P system $P^{part}(n)$ (size of the alphabet, number of rules, maximal length of the rules, number of membranes and of objects in the initial configuration) is linearly bounded with respect to n. The number of steps to get an answer is also lineal in n.

5.3. Composing Π_B and Π_{part}

As described above, $\Pi_B = \{P^B(n,d) : n,d \in N\}$ is a family of P systems where each element $P^B(n,d)$ converts multisets of natural numbers from binary into unary notation and the family $\Pi_{part} = \{P^{part}(n) : n, \in N\}$ is a family of P systems where each P system $P^{part}(n)$ solves all instances of the PARTITION problem with n elements with the input provided in unary form. In each P system $P^B(n,d)$, the *output* membrane is the skin and in $P^{part}(n)$ the input membrane is an elementary membrane, so we are on the conditions of the definition **??**. By composing $P^B(n,d)$ and $P^{part}(n)$ for each n we obtain the following family:

$$\Pi = \{P^{part}(n) \circ P^B(n,d) : n,d \in N\}$$

where each P system $P(n,d) = P^{part}(n) \circ P^B(n,d)$ is a cellular device which solves all the instances of the PARTITION problem with the same parameters n and d.

In the P systems $P^{part}(n) \circ P^B(n,d)$ the input membrane is the membrane with label t from $P^B(n,d)$. We place in this membrane the instance of the problem in binary form. After $d+1$ steps, we obtain the output of $P^B(n,d)$ in the skin, which is identified with the membrane e of $P^{part(n)}$. Note that in $P^{part(n)}$ no rule can be applied till the input is placed in the corresponding membrane, so at this moment the computation of P^{part} starts and provides the answer to the problem.

6. Conclusions and directions for future work

When solving numerical **NP**–complete problems using P systems, integer numbers are usually represented in unary notation. However, in classical complexity theory such numbers are assumed to be represented in binary notation, which is an exponentially more compact encoding with respect to unary notation.

Switching from binary to unary notation simplifies **NP**–complete numerical problems, because it modifies the way me measure the size of instances, as well as the relation between instance size and the running time of algorithms which solve the problem.

The eventual composition between our systems and the ones exposed in literature allows to solve **NP**–complete numerical problems working on instances whose numbers are encoded in binary form. Moreover, since the instances must be injected into the systems before starting computations, working with binary notation allows to prepare such systems with a polynomially bounded effort. The preparation of these systems requires instead an exponential amount of work when dealing with instances whose numbers are encoded in unary form.

In this paper we present a family of P systems which solves PARTITION when the data are provided in binary form. This family is obtained by the composition of two families of P system. The first one is a semi-uniform family which converts multiset of natural numbers in binary form into a multiset which codifies these numbers in unary form. It is a semi-uniform family because the P system not only depends on the cardinal on the multiset, but on the maximal number of the multiset.

On the other hand, we have a uniform family of P systems which solves PARTITION, where each P system only depends on the cardinal of the instance of the problem.

The composition of both families is a semi-uniform family which solves PARTITION with the data in unary form.

This paper extend the work from ... and the main contributions are the presentation of P systems for the conversion of natural numbers from binary into unary notation; a first attempt to define the *composition* of P systems and the solution of PARTITION with the data in binary form. It remains the open problem of studying if is is possible to find a *uniform* family which solves PARTITION (or whatever **NP**-complete) problem with the data in binary form.

Acknowledgements

The present paper has been inspired by joint work with Seville research group during the Third Brainstorming Week held in Seville from January 31^{st} to February 4^{th}, 2005. Miguel A. Gutiérrez-Naranjo acknowledges 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.