Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 123 (2005) 93–110
www.elsevier.com/locate/entcs

Towards a Programming Language in Cellular Computing

Miguel A. Gutiérrez–Naranjo[1] Mario J. Pérez–Jiménez[1]
Agustín Riscos–Núñez[1]

Department of Computer Science and Artificial Intelligence
University of Sevilla
Sevilla, Spain

Abstract

Several solutions to hard numerical problems using P systems have been presented recently, and strong similarities in their designs have been noticed. In this paper we present a new solution, to the Partition problem, via a family of deterministic P systems with active membranes using 2-division. Then, we intend to show that the idea of a *cellular programming language* is possible (at least for some relevant family of **NP**-complete problems), indicating some "subroutines" that can be used in a variety of situations and therefore could be useful for designing solutions for new problems in the future.

Keywords: Membrane Computing, Complexity Class, Cellular Subroutine, **NP**-complete problem

1 Introduction

In [3], a new model of computation within the framework of *Natural Computing* was introduced, called *Membrane Computing.* It starts from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The computational devices of this model are called *P systems.*

Many results have been presented in the field during the last years by computer scientists, biologists, formal linguists and complexity theoreticians,

[1] Email: {magutier,marper,ariscosn}@us.es

doi:10.1016/j.entcs.2004.04.044

both confirming the relevance of the area and enriching it with many open problems and research lines.

In this paper we present a family of P systems that solves a numerical **NP**-complete problem, namely, the Partition problem. The design of this solution is inspired in several previous works on other problems, mainly the Subset-Sum and the Knapsack problems, but also the VALIDITY and SAT. The similarities between the design introduced here and the solutions presented in [5], [6], [8] and [10] will be highlighted and some conclusions will be extracted from them.

The paper is organized as follows: first some preliminary ideas about *recognizer P systems* and *complexity classes* are introduced in the next section; then, in section 3 a cellular solution for the Partition problem is presented, and some comments about the possibility of generalizing the design are given in section 4; finally, an example of application is shown in section 5 and some final remarks are given in section 6.

2 Preliminaries

Recall that a decision problem, X, is a pair (I_X, θ_X) such that I_X is a language over a finite alphabet (whose elements are called *instances*), and θ_X is a total boolean function over I_X. That is, the answer to each instance of the problem will be either TRUE (Yes) or FALSE (No). This is why we are interested in using a computing device that is able to receive an input, process it, and deliver a boolean answer.

In our case, we have chosen the class of P systems with input and with external output (special objects Yes and No will be used to implement the boolean answer). In order to obtain a significant speed-up, we will work in a cellular model using active membranes, and so we are allowed to use membrane division to obtain in polynomial time an exponential workspace. We also impose some restrictions, for instance we want the systems to be *confluent* (all computations with the *same* input lead to the *same* output), also every computation must be finite and, furthermore, we want that the answer is delivered in the last step of the computation, by sending to the environment a special object Yes or No.

2.1 Recognizer P systems with Active Membranes

Roughly speaking, a P system consists of a cell-like membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous, non-deterministic, maximally parallel manner.

A layman-oriented introduction can be found in [4] and further bibliography at [11].

Definition 2.1 A *P system with input* is a tuple (Π, Σ, i_Π), where: (a) Π is a P system, with working alphabet Γ, with p membranes labelled by $1, \ldots, p$, and initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_p$ associated with them; (b) Σ is an (input) alphabet strictly contained in Γ and the initial multisets are over $\Gamma - \Sigma$; and (c) i_Π is the label of a distinguished (input) membrane.

The computations of a P system with input $m \in M(\Sigma)$, a multiset over Σ, are defined in a natural way. The only novelty is that the initial configuration of (Π, Σ, i_Π) must be the initial configuration of the system associated with the input multiset $m \in M(\Sigma)$.

Definition 2.2 Let (Π, Σ, i_Π) be a P system with input. Let Γ be the working alphabet of Π, μ the membrane structure and $\mathcal{M}_1, \ldots, \mathcal{M}_p$ the initial multisets of Π. Let m be a multiset over Σ. The *initial configuration of* (Π, Σ, i_Π) *with input* m is $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots \mathcal{M}_p)$.

In the case of P systems with input and with external output, the concept of computation is introduced in a similar way but with a slight variant. We consider that it is not possible to observe the internal processes inside the P system, and we can only know if the computation has halted via some distinguished objects sent out to the environment. We can formalize these ideas in the following way.

Definition 2.3 A *recognizer P system* is a P system with input, (Π, Σ, i_Π), and with external output such that:

(i) The working alphabet contains two distinguished elements YES, NO.

(ii) All its computations halt.

(iii) If $\mathcal{C}$ is a computation of Π, then either some object YES or some object NO (but not both) must have been released into the environment, and only in the last step of the computation. We say that $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object YES (respectively, NO) appears in the external environment associated to the corresponding halting configuration of $\mathcal{C}$.

This recognizer systems are specially suitable when trying to solve decision problems.

Definition 2.4 A P system with active membranes is a tuple $\Pi = (\Sigma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_m, R)$ where:

(i) $m \geq 1$, is the initial degree of the system;

(ii) Σ is an alphabet of symbol-objects;

(iii) H is a finite set of labels for membranes;

(iv) μ is a membrane structure, of m membranes, labelled (not necessarily in a one-to-one manner) with elements of H;

(v) $\mathcal{M}_1, \ldots, \mathcal{M}_m$ are strings over Σ, describing the initial multisets of objects placed in the m regions of μ;

(vi) R is a finite set of evolution rules, of the following forms:

(a) $[\, a \rightarrow \omega\,]_h^{\alpha}$ for $h \in H$,$\alpha \in \{+, -, 0\}$, $a \in \Sigma$, $\omega \in \Sigma^*$: This is an object evolution rule, associated with a membrane labelled with h and depending on the polarity of that membrane, but not directly involving the membrane.

(b) $a\,[\ \,]_h^{\alpha_1} \rightarrow [\, b\,]_h^{\alpha_2}$ for $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in \Sigma$: An object from the region immediately outside a membrane labelled with h is introduced in this membrane, possibly transformed into another object, and simultaneously, the polarity of the membrane can be changed.

(c) $[\, a\,]_h^{\alpha_1} \rightarrow b\ [\ \,]_h^{\alpha_2}$ for $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in \Sigma$: An object is sent out from membrane labelled with h to the region immediately outside, possibly transformed into another object, and simultaneously, the polarity of the membrane can be changed.

(d) $[\, a\,]_h^{\alpha} \rightarrow b$ for $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in \Sigma$: A membrane labelled with h is dissolved in reaction with an object. The skin is never dissolved.

(e) $[\, a\,]_h^{\alpha_1} \rightarrow [\, b\,]_h^{\alpha_2}\ [\, c\,]_h^{\alpha_3}$ for $h \in H$, $\alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$, $a, b, c \in \Sigma$: An elementary membrane can be divided into two membranes with the same label, possibly transforming some objects and their polarities.

These rules are applied according to the following principles:

- All the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non deterministic way), but any object which can evolve by one rule of any form, must evolve.
- If a membrane is dissolved, its content (multiset and internal membranes) is left free in the surrounding region.
- If at the same time a membrane labelled by h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, and then the division is produced. Of course, this process takes only one step.
- The rules associated with membranes labelled by h are used for all copies

of this membrane. At one step, a membrane can be the subject of *only one* rule of types (b)-(e).

Let us denote by $\mathcal{AM}$ the class of recognizer P systems with active membranes using 2-division.

2.2 The complexity class $\mathbf{PMC}_{\mathcal{F}}$

The first results about "solvability" of **NP**–complete problems in polynomial time (even linear) by cellular computing systems with membranes were obtained using variants of P systems that lack an input membrane. Thus, the constructive proofs of such results need to design one system for each instance of the problem.

If we wanted to perform such a solution of some decision problem in a laboratory, we will find a drawback on this approach: a system constructed to solve a concrete instance is useless when trying to solve another instance. This handicap can be easily overtaken if we consider a P system with input. Then, the same system could solve different instances of the problem, provided that the corresponding input multisets are introduced in the input membrane.

Instead of looking for a single system that solves a problem, we prefer designing a family of P systems such that each element decides all the instances of *equivalent size*, in some sense.

Definition 2.5 Let $\mathcal{F}$ be a class of recognizer P systems. We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\Pi = (\Pi(n))_{n\in\mathbb{N}^+}$, of $\mathcal{F}$, and we denote this by $X \in \mathbf{PMC}_{\mathcal{F}}$, if the following is true:

- The family Π is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine constructing $\Pi(n)$ from $n \in \mathbb{N}^+$ in polynomial time.
- There exists a pair (g, h) of polynomial-time computable functions $g : L \to \bigcup_{n\in\mathbb{N}^+} I_{\Pi(n)}$ and $h : L \to \mathbb{N}^+$ such that for every $u \in L$ we have $g(u) \in I_{\Pi(h(u))}$, and
  - · The family Π is polynomially bounded with regard to (g, h); that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(h(u))$ with input $g(u)$ is halting and, moreover, it performs at most, $p(|u|)$ steps.
  - · The family Π is sound, with regard to (X, g, h); that is, for each $u \in I_X$ it is verified that if there exists an accepting computation of $\Pi(h(u))$ with input $g(u)$, then $\theta_X(u) = 1$.
  - · The family Π is complete, with regard to (X, g, h); that is, for each $u \in I_X$

it is verified that if $\theta_X(u) = 1$, then every computation of $\Pi(h(u))$ with input $g(u)$ is an accepting one.

In the above definition we have imposed every P system $\Pi(n)$ to be *confluent*, in the following sense: every computation with the *same* input produces the *same* output.

We have the class $\mathbf{PMC}_{\mathcal{F}}$ is closed under polynomial–time reduction and complement.

3 Solving the Partition problem in linear time

In this section we present a linear time solution to the *Partition* problem, in terms of P systems with active membranes, constantly comparing it with the solutions to the Subset–Sum and Knapsack problems given in [5] and [6].

The *Partition* problem can be stated as follows: *Given a set A of n elements, where each element has a "weight" $w_i \in \mathbb{N}$, decide whether or not there exists a partition of A into two subsets with the same total weight.*

We will represent the instances of the problem using tuples of the kind $(n, (w_1, \dots, w_n))$, where n is the size of the set A and $(w_1, \dots, w_n)$ is the list of weights of the elements from A. We can define in a natural way an additive function w that corresponds to the data in the instance.

We address the resolution of the problem via a brute force algorithm. The strategy can be roughly split into the following subgoals:

- *Generation stage*: use membrane division to get a single membrane for each subset.
- *Calculation stage*: compute in each membrane the weight of its associated subset and the weight of its complementary.
- *Checking stage*: in each membrane, compare $w(B)$ with $w(B^c)$, where B is the associated subset.
- *Output stage*: the answer is delivered according to the results of the checking.

The family presented here is $\Pi = \{(\Pi(n), \Sigma(n), i(n)) : n \in \mathbb{N}\}$. For each element of the family, the input alphabet is $\Sigma(n) = \{x_1, \dots, x_n\}$, the input membrane is always the same, $i(n) = e$, and the P system $\Pi(n) = (\Gamma(n), \{e, r, s\}, \mu, \mathcal{M}_e, \mathcal{M}_r, \mathcal{M}_s, R)$ is defined as follows:

- Working alphabet:
  $\Gamma(n) = \{a_0, a, b_0, b, c, d_0, d_1, d_2, e_0, \dots, e_n, g, \bar{g}, \hat{g}, h_0, h_1, i_1, i_2, i_4, i_5,$

$p, \bar{p}, q, x_0, \ldots, x_n, Yes, No, No_0, z_1, \ldots, z_{2n+1}, \#\}$.

- Membrane structure: $\mu = [\ [\ ]_e\ [\ ]_r\ ]_s$.
- Initial multisets: $\mathcal{M}_e = e_0 g$; $\mathcal{M}_r = b_0 h_0$ and $\mathcal{M}_s = z_1$.

- The set of evolution rules, R, consists of the following rules:

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

The goal of these rules is to generate one membrane for each subset of A. Indeed, exactly the same two schemes of rules are used for the generation stages in the Subset-Sum and the Knapsack case. Here is how these rules work: in each step (according to the index of e_i), we consider an element of A and either we add it to the subset associated with the membrane, B, or we put it in the complementary subset, B^c. Note that a membrane can proceed to the checking stage only after it gets negative charge; a positively charged membrane where the object e_n appears will get blocked (it will be dissolved, see rules in (i)).

(b) $[x_0 \rightarrow a_0]_e^0$; $[x_0 \rightarrow \bar{p}]_e^+$.
$[x_i \rightarrow x_{i-1}]_e^+$, $[x_i \rightarrow \bar{p}]_e^-$, for $i = 1, \ldots, n$.

At the beginning, the multiplicities of the objects x_j (with $1 \leq j \leq n$) encode the weights of the corresponding elements of A. They are not present in the definition of the system, but they are inserted as input in the membrane labelled by e before starting the computation: for each $a_j \in A$, w_j copies of x_j have to be added to the input membrane. During the computation, at the same time as elements are added to the subset associated with a membrane, objects a_0 and $\bar{p}$ are generated to store the weight of this subset and of its complementary. Again, these schemes of rules are almost identical to the ones used for the calculation stages of the Subset-Sum and Knapsack (see [5], [6]), the only difference is that here the weight of the complementary is kept, and in the mentioned papers it was just removed.

Indeed, it is worth noticing that the index-rotation technique was already used in [8] and [10] to deal with the set of variables in an ordered manner, even though there was no weight calculation there.

(c) $[q \rightarrow i_1]_e^-$; $[\bar{p} \rightarrow p]_e^-$; $[a_0 \rightarrow a]_e^-$.

When a membrane gets negatively charged, the two first stages (i.e., generation and calculation stages) end, and then some transition rules are applied. Objects a_0 and $\bar{p}$, whose multiplicities encode the weights of the associated subset, $w(B)$, and of its complementary, $w(B^c)$, are renamed for the next stage, when their multiplicities are compared (similar renaming rules can be found in the designs of the Subset-Sum and Knapsack solutions). The renaming is useful to avoid conflicts between rules from the checking stage and rules from the generation and calculation stages.

(d) $[a]_e^- \rightarrow [\ ]_e^0 \#; \quad [p]_e^0 \rightarrow [\ ]_e^- \#$.

These rules implement the comparison mentioned above (that is, they check whether $w(B) = w(B^c)$ holds or not). They work as a loop that erases objects a and p one by one alternatively, changing the charge of the membrane in each step. Exactly the same method can be used to compare the multiplicities of whatever two objects of the working alphabet, so again we find rules that might be re-used when attacking other numerical problems.

(e) $[i_1 \rightarrow i_2]_e^-; \quad [i_2 \rightarrow i_1]_e^0$.

A marker that controls the previous loop is described here. The index of i_j and the electric charge of the membrane give enough information to point out if the number of objects a is greater than (less than or equal to) the number of objects p.

Here we find the first important difference in the design with respect to the ones for Subset-Sum and Knapsack. There counters were used, and the schemes of rules depended on the number of steps that the checking was going to last. But now this number of steps depends on the total weight of the set A, and we cannot use this information if we want an uniform design. However, there are good news: the rules used here can be used also in general, so new versions of the solutions to Subset-Sum and Knapsack using this subroutine can be given.

(f) $[i_1]_e^0 \rightarrow [\ ]_e^+ No$.

This rule, together with the ones in the next item, take care of the result of the checking. If a subset $B \subseteq A$ verifies that $w(B) > w(B^c)$, then at the end of the calculation stage there will be less objects p than a inside the membrane associated with it. This forces the loop described in (e) to halt: the moment will come when there are no objects p left, and then the rule $[i_2 \rightarrow i_1]_e^-$ will be applied but it will not be possible to apply the rule $[p]_e^0 \rightarrow [\ ]_e^- \#$ at the same time. Thus, an object i_1 will be present in the membrane and the latter will be neutrally charged, so the rule (f) will be applied, ending the checking stage with a negative result.

(g) $[i_2 \rightarrow i_4 c]_e^-$.
$[c]_e^- \rightarrow [\ ]_e^0 \#; \quad [i_4 \rightarrow i_5]_e^0$.
$[i_5]_e^0 \rightarrow [\ ]_e^+ Yes; \quad [i_5]_e^- \rightarrow [\ ]_e^+ No$.

If, on the contrary, $w(B) \leq w(B^c)$ holds, then the objects a will be exhausted before the objects p. It is important to distinguish between the cases where the multiplicity of p is strictly greater than the multiplicity of a and the cases where these multiplicities coincide. This is why object c gives again neutral charge to the membrane and then object i_5 checks if a rule $[p]_e^0 \rightarrow [\;]_e^- \#$ is applied or not.

(h) $[p \rightarrow \#]_e^+; \quad [a \rightarrow \#]_e^+$.

If after the checking loop of rules in (d) has finished there still are some objects p or a in the membrane, they can be erased (just for "cleaning" purposes).

(i) $[e_n]_e^+ \rightarrow \#$.
$[a_0 \rightarrow \#]_s^0; \quad [\bar{p} \rightarrow \#]_s^0; \quad [g \rightarrow \#]_s^0$.

These rules also perform a "cleaning" task, dissolving the membranes that are not meaningful and erasing the objects that these membranes leave in the skin membrane. This is not essential in the design, but it is helpful.

(j) $[z_i \rightarrow z_{i+1}]_s^0$, for $i = 1, \ldots, 2n$.
$[z_{2n+1} \rightarrow d_0 d_1]_s^0$.
$[d_1]_s^0 \rightarrow [\;]_s^+ d_1$.

Before the answer is sent out, the system has to make sure that all the relevant membranes have finished their checking stages. To do this, first we wait for $2n+1$ steps and then we activate the process. This needs to be done in order to make sure that the division process is over, and thus we know that from this moment on, the membranes that finish their checking stage, and only them, will have positive charge (see the rules in (f) and (g) for the end of the checking and note that we get rid of the spare membranes via the rules in (i)).

(k) $[g]_e^- \rightarrow [\;]_e^- \bar{g}$.
$[\bar{g} \rightarrow \hat{g}]_s^+$.
$\hat{g}[\;]_e^+ \rightarrow [\hat{g}]_e^0$.

As we said before, we need to check if all the relevant membranes have finished their checking stages. This is done using the objects g that are present in the skin and the auxiliary membrane labelled by r (see the next set of rules). There must be 2^n copies of g, because each relevant membrane sends one, and there is one relevant membrane for each subset of A, that is 2^n in all.

(l) $d_0[\;]_r^0 \rightarrow [d_0]_r^-$.
$[h_0 \rightarrow h_1]_r^-; \quad [h_1 \rightarrow h_0]_r^+$.
$[b_0]_r^- \rightarrow [\;]_r^+ b; \quad \hat{g}[\;]_r^+ \rightarrow [\hat{g}]_r^-$.

$b[\ ]_r^- \to [_r b_0]_r^+$; $\quad [\hat{g}]_r^+ \to [\ ]_r^- \hat{g}$.
$[h_0]_r^+ \to [\ ]_r^+ d_2$; $\quad [d_2]_s^+ \to [\ ]_s^0 d_2$.

The membrane labelled by r is present in the initial configuration, but remains inactive until an object d_0 "wakes it up". The purpose of this membrane is to perform a loop where the objects $\hat{g}$ are involved, in such a way that if there are no objects $\hat{g}$ available in the skin, the loop will halt. Thus, we can detect if there are no objects $\hat{g}$ present in the skin region. This fact will mean that all the relevant membranes have finished their checking stage, and that the system is ready to send out the answer (Yes or No).

(m) $[No \to No_0]_s^-$.
$[Yes]_s^- \to [\ ]_s^0 Yes$.
$[No_0]_s^- \to [\ ]_s^0 No$.

Finally, the output process is activated. The skin membrane needs to be negatively charged before the answer is sent out. Object d_2 takes care of this (see the previous set of rules) and then, if the answer is affirmative, an object Yes will be sent out recovering the neutral charge for the skin. Note that the answer Yes has some priority over the negative answer, in the sense that we first check if there is any object Yes and then, if it is not the case, the answer No will be sent out. This little trick of changing the electrical charge of the skin membrane and using the auxiliary object No_0 is also used in the other two designs, so hopefully this feature can be also saved for future designs.

3.1 Some comments on computational complexity

The main goal of the design presented above is not just solving the Partition problem, but to do it *efficiently*, according to some prefixed standards (see the *complexity classes* presented in Subsection 2.2). Recall that the active membranes model allows us to create an exponential workspace in polynomial time during the computation. We exploit this fact, and thus we are only concerned with the time complexity (number of cellular steps of any computation) and about the resources *initially* needed to build the P systems.

In this line, note that the family of P systems introduced above has a recursive description and it requires only a polynomial amount of resources:

- size of the alphabet: $4n + 27 \in \Theta(n)$.
- number of initial membranes: $3 \in \Theta(1)$.
- size of initial multisets: $5 \in \Theta(1)$.
- number of evolution rules: $6n + 39 \in \Theta(n)$.

Furthermore, it can be proved that the system is confluent and that the number of steps of any computation is of linear order with respect to the size[2] of the input multiset. That is, $Partition \in \mathbf{PMC}_{\mathcal{AM}}$.

Analogous complexity results were obtained and proved for the SAT [8], VALIDITY [10], Subset-Sum [5] and Knapsack [6] problems.

4 Generalizing the rules: towards a programming language

In this section we will present an overview of the computation, commenting how the rules work and sketching the first instructions that could be added to the library of subroutines of the *programming language* that we intend to create.

In the first step of the computation, the rule $[e_i]_e^0 \rightarrow [q]_e^- [e_i]_e^+$ is applied, for $i = 0$. From this moment on, the rest of division rules will be applied in turns, in such a way that whenever a negatively charged membrane is created, it will not divide anymore. The concept of subset *associated* with an internal membrane is an abstraction, because there are no witness-objects in the membrane to encode it, but we can agree to "associate" subsets with membranes following this definition:

- The subset associated with the initial membrane is the empty one.
- When an object e_j appears in a neutrally charged membrane (with $j < n$), then the j-th element of A is selected and added up to the previous associated subset. Once the stage is over, the associated subset will not be modified anymore.
- When a division rule is applied, the two newborn membranes inherit the associated subset from the original membrane.

As we already said, the rules in (a) are exactly repeated in the designs for Subset-Sum and Knapsack (see [5] and [6], respectively). Thus, we could create a new instruction, valid to use it in the designs of P systems, called for example

$$gen\_subsets(n)$$

This is just a notation; whenever we find this in a design we should replace it by the set of rules described in (a). We can also make use, if needed, of the semantic notion of associated subset in further stages of the computation.

For instance, this is done in the calculation stage. The weights of the elements are added only if the element is selected for the associated subset.

[2] The instance is encoded in a 1-ary fashion in the system, through the input multiset

$$\mathit{calc\_weight}(n) \equiv \left(\begin{array}{ll} [x_0 \rightarrow \mathit{subs}]_e^0, & \\ [x_0 \rightarrow \mathit{compl}]_e^+, & \\ [x_i \rightarrow x_{i-1}]_e^+, & \text{for } \ i = 1, \ldots, n \\ [x_i \rightarrow \mathit{compl}]_e^-, & \text{for } \ i = 1, \ldots, n \end{array} \right)$$

The object *compl* can be substituted by λ if we do not want information about the complementary, or by other objects, depending on the concrete problem that we are addressing (maybe we could add a second variable that says if the weight of the complementary should be computed or not). The object *subs* encodes with its multiplicity the weight of the associated subset; we are free to use any object instead, it depends on our specific notation. For instance, in the Knapsack problem, two functions have to be computed: the weight and the value of the subset. Thus, we have to include twice the rules, using two different sets of indexed objects, one for each function (for instance, in [6], x_j and y_j were used, for $j = 0, \ldots, n$).

The generation and calculation stages end in a membrane when it gets negative charge for the first time, and we have at our disposal a witness-object q that appears in the membrane exactly in that moment. If we want to perform now the comparison between the multiplicities of two objects, we need to rename all the objects in the membrane, to make sure that there does not exist overlaping, i.e. we want to avoid nondeterminism. The renaming step depends strongly on the problem, because the new objects that are needed depend on how many stages we want to perform later on.

The next set of rules is (d). When these two rules are applied iteratively, a loop is created. The charge of the membrane changes from negative to neutral and back to negative in every loop, until one of the two objects that are being used is exhausted and the loop halts.

$$\mathit{check\_weight} \equiv \left(\begin{array}{l} [\mathit{obj}1]_e^- \rightarrow [\ ]_e^0 \# \\ [\mathit{obj}2]_e^0 \rightarrow [\ ]_e^- \# \end{array} \right)$$

Observe that this time the scheme of rules does not depend on n, we just compare the number of occurrences of two objects, $\mathit{obj}1$ and $\mathit{obj}2$. The names of this objects can be customized, as well as the two charges that are used. We can again recall the design of the Knapsack in [6] as an example, because two checking stages were carried out there, with different objects and different charges, but the same changing-charge-loop design.

Next, let us pay attention to the rules that take care of the result of the checking. As we said before, instead of using a counter that increases its index in each step we use two objects as markers, and this suffices to detect when the loop has halted. The evolution of these markers is as rules from item (e) show.

$$[i_{same} \rightarrow i_{diff}]_e^-; \quad [i_{diff} \rightarrow i_{same}]_e^0$$

Let us concentrate on the termination of the checking stage. First of all, if the multiplicity of $obj1$ is greater than the multiplicity of $obj2$ then the moment will come when the rule $[obj1]_e^- \rightarrow [\ ]_e^0\#$ will be applied but its counterpart in the loop ($[obj2]_e^0 \rightarrow [\ ]_e^-\#$) will not be applied in the next step, and so the membrane will keep a neutral charge for two consecutive evolution steps. This fact is detected by the marker and in the following step the rule

$$[i_{same}]_e^0 \rightarrow [\ ]_e^+ z_{more}$$

will be applied, bringing the checking stage of this membrane to its end. In the case of the Partition problem, the fact that there are more copies of object $obj1$ than of $obj2$ means a negative answer, so we replace z_{more} by No, but in other problems it could be replaced by Yes or by other special objects in order to activate further stages.

If, on the contrary, the multiplicity of $obj1$ is less than or equal to the multiplicity of $obj2$, then the membrane will have a negative charge for two consecutive steps, but we need to check if any object $obj2$ is still present in the membrane. This can be done using the rules

$$[i_{diff} \rightarrow aux_1 c]_e^-; \quad [c]_e^- \rightarrow [\ ]_e^0\#$$

then, in the next step the rule $[aux_1 \rightarrow aux_2]_e^0$ will be applied, and the charge will change if and only if there are some objects $obj2$ left (via the rule $[obj2]_e^0 \rightarrow [\ ]_e^-\#$). Finally, we differentiate the results depending on the electrical charge:

$$[aux_2]_e^0 \rightarrow [\ ]_e^+ z_{equal}; \quad [aux_2]_e^- \rightarrow [\ ]_e^+ z_{less}$$

Again, in our problem we have customized z_{equal} to be Yes and z_{less} to be No, but this depends on the condition that we are checking. Maybe in some problems instead of using objects No we are interested in blocking the membrane, and this can be done simply removing the corresponding rule.

$$marker\_eq \equiv \left(\begin{array}{l} [i_{same} \rightarrow i_{diff}]_e^-; \quad [i_{diff} \rightarrow i_{same}]_e^0 \\ [i_{same}]_e^0 \rightarrow [\ ]_e^+ No \\ [i_{diff} \rightarrow aux_1 c]_e^-; \quad [c]_e^- \rightarrow [\ ]_e^0\# \\ [aux_1 \rightarrow aux_2]_e^0 \\ [aux_2]_e^0 \rightarrow [\ ]_e^+ Yes; \quad [aux_2]_e^- \rightarrow [\ ]_e^+ No \end{array} \right)$$

The corresponding variants $marker\_leq$ and $marker\_geq$ can be defined if we consider that the successful result of the checking is to detect that the number of objects $obj1$ is less than (or greater than, respectively) the number of $obj2$.

It is time now to comment how the answer process is managed. It was already hinted in the previous section that there is a membrane, labelled by r, that plays a central role. Let us explain the process step by step (see Figure 1 for a graphical description of the detection loop).

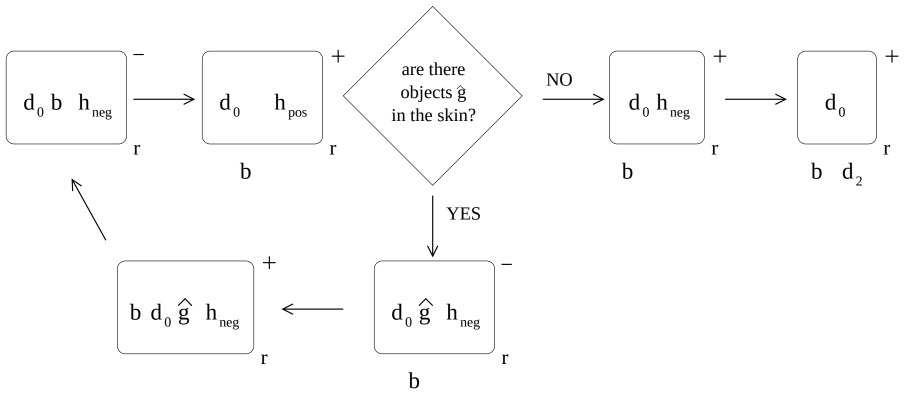

Fig. 1. Membrane r detection loop

We need to check if 2^n membranes have finished their checking stages, and to do this it seems a good idea to use 2^n objects, in order to make use of the parallelism of the model. To generate these objects we include an object g in the initial configuration and we use the rule $[g]_e^- \rightarrow [\ ]_e^- \bar{g}$. The object g is replicated every time the membrane divides, and this rule is applied when the membrane ends the generation and calculation stages and before starting the checking. We know that 2^n copies of $\bar{g}$ will be sent to the skin in some moment of the computation, but not simultaneously.

The idea is to make these objects return back to their membranes when the checkings are over. In order to avoid interferences with some checking stages that last longer than others, we add a counter and a renaming rule, and the process of counting how many membranes have finished their checking stage does not start until an object d_1 is sent out to the environment and the skin gets negative charge.

In the same step, an object d_0 enters the membrane r and activates the loop described by the rules in (l) and depicted in Figure 1. The idea is that membrane r tries to *fish* any object $\hat{g}$ present in the skin: the object b plays the role of *bait*, because it gives positive charge to the membrane allowing thus the rule $\hat{g}[\ ]_r^+ \rightarrow [\hat{g}]_r^-$ to apply (it is clear that if there are no objects $\hat{g}$ in the skin then the rule can not be applied). There is a marker inside the membrane that controls if any object actually entered the membrane, and in negative case, an object d_2 will be sent to the skin in order to finish the answer stage.

$$detector \equiv \left(\begin{array}{l} d_0[\ ]_r^0 \to [d_0]_r^- \\ [h_{neg} \to h_{pos}]_r^-; \quad [h_{pos} \to h_{neg}]_r^+ \\ [b]_r^- \to [_r]_r^+ b; \quad \hat{g}[\ ]_r^+ \to [\hat{g}]_r^-.\ b[\ ]_r^- \to [_r b_0]_r^+; \quad [\hat{g}]_r^+ \to [\ ]_r^- \hat{g} \\ [h_{neg}]_r^+ \to [\ ]_r^+ d_2 \\ [d_2]_s^+ \to [\ ]_s^0 d_2 \end{array} \right)$$

Then we just let the system output the answer, giving one step of advantage to object Yes, in such a way that when we obtain in the environment an object Yes or an object No we know that the system has halted (the computation has finished) and that the object answers correctly the instance of the problem that we were considering.

$$answer \equiv \left(\begin{array}{l} [No \to No_0]_s^- \\ [Yes]_s^- \to [\ ]_s^0 Yes \\ [No_0]_s^- \to [\ ]_s^0 No \end{array} \right)$$

5 Applications

As an example of the usefulness of the subroutines outlined in the previous section, and as a first step towards a programming language in cellular computing, let us see how the design of the solutions for the Subset-Sum [5], Knapsack [6], Bin Packing [7] and Partition problems would look like:

SUBSET-SUM	**KNAPSACK**	**BINPACKING**	**PARTITION**
gen_subsets (n)	*gen_subsets (n)*	for $i = 1, \ldots, b-1$ do	*gen_subsets (n)*
calc_weight (n)	*calc_weight1 (n)*	*gen_subsets* (n_i)	*calc_weight*$_{compl}$ *(n)*
rename	*calc_weight2 (n)*	*calc_weight* (n_i)	*rename*
check_weight	*rename*	*rename*	*check_weight*
marker_eq	*check_weight1*	*check_weight*	*marker_eq*
counter (n)	*marker_leq*	*marker_leq*	*counter (n)*
clean_dissolve	*rename*	*counter(n)*	*clean_dissolve*
detector	*check_weight2*	*clean_dissolve*	*detector*
answer	*marker_geq*	end for.	*answer*
	counter (n)	*calc_weight* (n_b)	
	clean_dissolve	*rename*	
	detector	*check_weight*	
	answer	*marker_leq*	
		counter (n)	
		clean_dissolve	
		detector	
		answer	

Also the solutions for SAT and VALIDITY problems could be rewritten in this form (we refer to [9] for the exhaustive description of the rules and of the similarities between the two designs).

```
gen_assignments (n)
calc_satisfied_clauses (n,m)
synchronization
check_truth_value (n,m)
counter (n,m)
answer
```

In this case, a division process is carried out at the beginning of the process to generate one membrane for each possible truth assignment of the n variables appearing in the formula. There, the electrical charges of the membranes in each step are meaningful, because they determine whether a variable will be assigned a TRUE value or a FALSE value. This strategy is very close to the one used in our generation stage.

In parallel with the division process, inside each membrane some objects keep track of which of the m clauses is (are) satisfied whenever we assign a truth value to a variable. This is somehow a weight calculation process. If the clause i gets a value of TRUE with the current assignment of variable j, then we add the "witness" of the clause, otherwise, we skip to the next variable. The technique of rotating the indexes is used here.

Concerning the checking process, the situation is different, because we need to check that all the clauses are satisfied, instead of comparing the multiplicities of two objects. However, the method that is used to control when the checking stage ends is a counter in the skin, and this is also used for the numerical problems. Also the answering process is very similar, the object Yes gets some priority over the object No by means of a counter and of the electric charge of the skin membrane.

6 Final Remarks

Up to now, the idea of a *programming language* has not been deeply discussed in the Membrane Computing area, but actually it is not hard to find some similarities between different designs conceived for different purposes. The use of the changes in the polarization (used in every design within the active membrane framework), the technique of working with indexed objects and making a rotation on the indexes (already used in [2], section 7.2, and later on by many other authors), the use of renaming rules in order to inhibit the evolution of an object until a specific instant in the computation (e.g., in [5] and [6]), and, of course, the use of counters (an indexed object that increases its index up to a certain value and then transforms into something different, see again [2]), among others. It is worth mentioning two examples of applying these strategies to the design of solutions for other numerical **NP**-complete problems: the multidimensional Knapsack problem in [1] and the Bin Packing problem in [7].

In this paper a first informal approach is made to find some "macro-rules" that may be used in a variety of situations. Of course, it is possible to define other subroutines for other variants of P systems. Anyway, much more work can be done in this field, increasing the list of instructions of this, so to say, programming language.

Acknowledgement

The support 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] Pan, L.; Martín-Vide, C.: Solving multidimensional 0-1 Knapsack problem by P systems with input and active membranes, *Proceedings of the Second Brainstorming Week on Membrane Computing*, Gh. Păun, A. Riscos, A. Romero and F. Sancho (eds.), Report RGNC 01/04, University of Seville, 2004, 342–353.

[2] Păun, Gh.: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.

[3] Păun, Gh.: Computing with membranes, *Journal of Computer and System Sciences*, **61**(1), 2000, 108–143.

[4] Păun, Gh. and Pérez-Jiménez, M.J.: Recent computing models inspired from biology: DNA and membrane computing, *Theoria*, **18**, 46 (2003), 72–84.

[5] Pérez-Jiménez, M.J.; Riscos-Núñez, A.: Solving the Subset-Sum problem by active membranes, *New Generation Computing*, in press.

[6] Pérez-Jiménez, M.J.; Riscos-Núñez, A.: A linear solution for the Knapsack problem using active membranes, *Membrane Computing*, C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg and A. Salomaa (eds.), Lecture Notes in Computer Science, vol. 2933, 2004, 250–268.

[7] Pérez-Jiménez, M.J.; Romero-Campero, F.J.: Solving the BIN PACKING problem by recognizer P systems with active membranes, *Proceedings of the Second Brainstorming Week on Membrane Computing*, Gh. Păun, A. Riscos, A. Romero and F. Sancho (eds.), Report RGNC 01/04, University of Seville, 2004, 414–430.

[8] Pérez-Jiménez, M.J.; Romero-Jiménez, A.; Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division, *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, E. Csuhaj-Varjú, C. Kintala, D. Wotschke and Gy. Vaszyl (eds.), 2003, 284-294.

[9] Pérez-Jiménez, M.J.; Romero-Jiménez, A.; Sancho-Caparrini, F.: Complexity classes in P systems, *Meeting of the European Molecular Computing Consortium. Volume of abstracts*, 2003, 18.

[10] Pérez-Jiménez, M.J.; Romero-Jiménez, A.; Sancho-Caparrini, F.: Solving VALIDITY problem by active membranes with input, *Proc. Brainstorming Week on Membrane Computing*, M. Cavalieri, C. Martín-Vide and Gh. Păun (eds.), Report GRLMC 26/03, University of Tarragona, 2003, 279–290.

[11] http://psystems.disco.unimib.it/