A P System Simulator for Logic Gates

Amged Fathey, Amr Badr, and Ibrahim Farag

***Abstract*—The objectives of this paper are representing a simulator for the logic gates using P systems with priorities rules, and making use of the P system parallel computing in order to reduce the time used to test or evaluate a logic circuit (set of logic gates), which may change the vision of the current logic gates systems. Also, introducing the basic logic gates and how they work together, the development of the appropriate P system models for these gates are represented, and putting all together in order to get logic circuits which are P system based, finally a simulation and a test for them using a P Lingua language simulator, and an example is introduced to illustrate and making test of the model.**

***Index Terms*—Logic gates, membrane computing, p-lingua, p system, simulator.**

I. Introduction

The current systems are mostly digital, that may be software or hardware systems (based on the binary numbers 0, 1), which are implemented using a set of logic gates or logic circuits. These logic gates operate with each other to generate the desired output of the circuits, which is the output of the system. This is the traditional digital system. Membrane computing, as introduced by G. Păun [1], appeared with a different way of computation which is imported from the living cells. A model for simulating Boolean circuits with DNA algorithms is proposed [2], and also this issue is discussed using a P systems [3]. We propose a simpler and different model for simulating logic circuits with P systems. So, we propose an idea that can be performed using the same logic gate operations, but with P system computation. This is discussed before [3] using P system with catalysts, but, we use P system with rule priorities that make it different and reduce the number of used membranes. In this case, the system will be different, as well as the way of computation using the parallel computation feature of the P-System which reduces the operations time which will be shown within computation. Also, we introduce a definition of membrane computing, and some notes on Logic gates and circuits. The main topic, the simulation of Logic circuits with P systems, the simulated gates are AND, OR, NAND, and XOR. We simulate and test a set of connected logic gates. Then the same set of gates is implemented and tested using a software simulator called P-Lingua. Finally, we conclude the work and introduce the proposed future work.

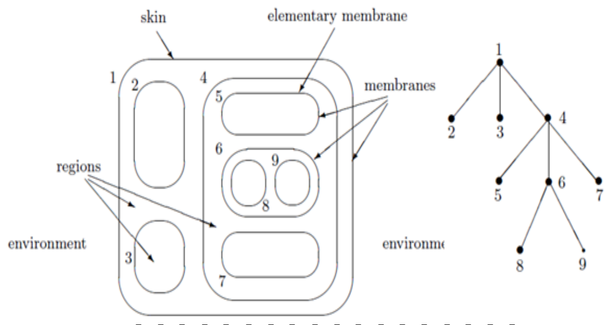
Manuscript received December 26, 2011; revised January 10, 2012.

The authors are with the Computer Science Department, Faculty of Computers Science, MTI University, Cairo, Egypt (e-mail: amged_fathey@hotmail.com).

II. Membrane Computing

Membrane computing is a branch of natural computing with an initial goal of abstracting computing models from the structure and functioning of living cells [1]. The initial goal was learning something possibly useful for computer science from cell biology, and giving the area developed in this direction [1], [4].

Membrane structure (which is the basic idea of building a membrane system) is a structure composed of several cell-membranes, hierarchically embedded in a main membrane called the skin membrane [5]. A plane representation of a membrane structure (Fig. 1), can be given by means of a Venn diagram, without intersected sets and with a unique superset. The membranes delimit regions and associate with each region a set of objects, described by some symbols over an alphabet, and a set of evolution rules [1, 6].



$[_1\,[_2\,]_2\,[_3\,]_3\,[_4\,[_5\,]_5\,[_6\,[_8\,]_8\,[_9\,]_9\,]_6\,[_7\,]_7\,]_4\,]_1$

Fig. 1. Membrane structures

A. Membrane Computing Computation

The computation mainly starts with an initial configuration of the system, where the input data of a problem is encoded [7].

A computation for a P system can be described as follows:

C0 =>C1=>…=>Cn, n>=0,

where *C0* is the initial configuration.

The transition from one configuration to another is performed by applying rules to the objects placed inside the regions. A computation of the system is a tree of configurations, the rules is applied in a non-deterministic maximal parallel manner (all objects which can evolve by such rules have to evolve), in each step, in each region; each object that can be evolved according to some rules must do it by transitions until reaching a halting configuration, where no more rules may be applied. The result of a halting computation is usually defined through the multi-set associated with a specific output membrane, or the environment, which is the space out of the skin.

(Fig. 2) give an example of a P system that computes the squares of natural non-null numbers (the output is read in membrane 1, which have to be appeared elementary at the end of a computation) [1].

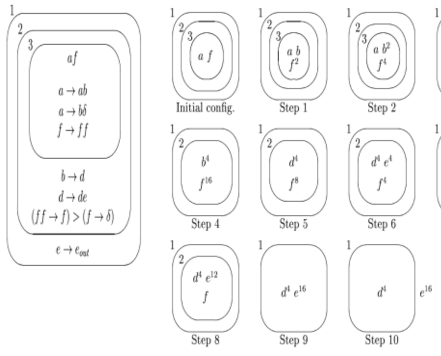


Fig. 2. A P system computes squares of natural non-null numbers

Based on [1], we made use of the formal definition of a P system and also the priorities rules.

A P system with priorities rules is a construct

$\Pi = (O, \mu, w_1, \dots, w_m, (R_1,P_1), \dots, (R_m,P_m), i_o)$

Where:

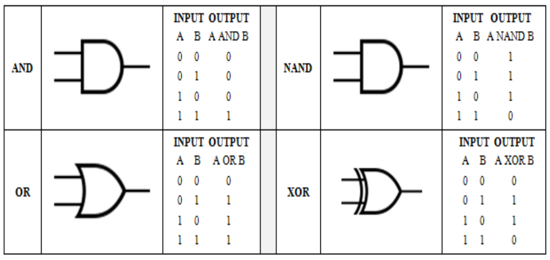
- O is the finite and non empty alphabet of objects,
- μ is a membrane structure, consisting of m membranes, labeled 1; 2,…m
- $w_1, \dots, w_m$ are strings over O representing the multi-sets of objects Presented in regions 1,2,…,m of the membrane structure,
- $R_1, \dots, R_m$ are finite sets of evolution rules associated with regions 1,2,..,m of the membrane structure,
- $p_1,\dots, p_m$ are finite sets of priorities of the rules
- i_o is either one of the labels 1,2,..,m , and the respective region is the output region of the system, or it is 0, and the result of a computation is collected in the environment of the system.

For more details about rule priorities relation we can describe it as follows: A rule $r_1{:}u_1 \to v_1$ from a set R_i is used in a transition step with respect to II only if there is no rule $r_2{:}u_2 \to v_2$ in R_i which can be applied at the same step and $r_2 > r_1$.that is, at each step, only rules which are maximal among the rules applicable in that region are allowed to be applied [1, 6].

III. Logic Gates

Logic gates are electronic circuits that operate on one or more input signals to produce an output signal [8] .Digital systems are constructed by using logic gates as software or hardware machines. These gates are the AND, OR, NOT, NAND, NOR, XOR and XNOR gates. As an example, (Fig. 3) shows the AND, OR, NAND and XOR gates. The logic gates behavior can be described by truth tables.

Most of the gates have at least 2 inputs (may be more); except the inverter (known as NOT) have one input; but it has also one output.



INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

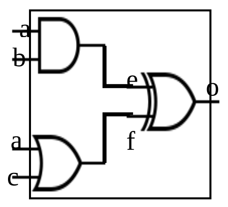
INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3. AND /OR/NAND/XOR gates symbols and truth tables

For example, if we have a set of gates like in Fig. 4, which contains 3 gates AND with inputs (a, b), OR gate with inputs (a, c), and XOR gate with inputs (e, f), the output of these gates is appeared as in the truth table (Fig. 4).

A.



INPUT			OUTPUT
A	B	C	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

B.

Fig. 4. Logic Gates example with output

IV. Logic Gates Simulation

This section introduces the proposed P system model for the logic gates separately. The following paragraph describes the initial configuration of the membranes that implement the AND, OR, NAND, and XOR gates, as shown in (Figs. 5, 6, 7, and 8).

As mentioned before this simulation was introduced in [2,3] but this reduces the number of membranes and number of rules and use a P system with priorities among the evolution rules in which its model is discussed in section 2 in the membrane computation model part [6].

When using this model, the inner rules can work in a parallel way according to the computation way of the P system. The Example will show the computation which work parallel and the other works sequentially.

In this simulation we will consider the input of the gates exist in the lowest level (inner membrane). The result of the computation will be sent out (in most cases skin or the container gate if exist).

A. Membrane AND Gate

Boolean AND gate can be simulated by P systems with rewriting priorities rules using only one membrane including its rules in exactly two steps. The formal model definition of the membrane AND gate of degree 1 is a tuple II, and also we assume that the final result will be at the skin.

$$\Pi = (O, \mu, w_1, (R_1, P_1), 0)$$

where

$O = \{0, 1\}$,

$\mu = \{[_1]_1\}$,

$w_1 = \lambda$,

$R_1 = \{r_1\text{: } 11 \to (\text{out}, 1),\ r_2\text{: } 1 \to (\text{out}, 0),\ r_3\text{: } 0 \to (\text{out}, 0)\}$,

$P_1 = \{r_1 > r_2\}$.

The initial configuration of the membrane AND gate appeared in Fig. 5.

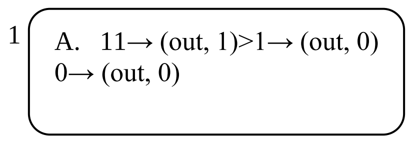


Fig. 5. AND gate membrane

According to the model and tracing the inputs, the available scenarios that might happen are as follows when the pairs enter the membrane 1 are showed in TABLE I.

TABLE I: AND GATE OUTPUT

Pair	Output
(0,0)	0
(0,1)	0
(1,1)	1

B. *Membrane OR Gate*

Boolean OR gate as AND gate also can be simulated by P systems with rewriting priorities rules using only one membrane including its rules in exactly two steps. The formal model definition of the membrane OR gate of degree 1 is a tuple II, and also we assume that the final result will be at the skin.

$\Pi = (O, \mu, w_1, (R_1, P_1), 0)$

where

$O= \{0, 1\}$,

$\mu = \{[_1\,]_1\}$,

$w_1 = \lambda$,

$R_1 = \{r_1\text{: } 1 \rightarrow (\text{out}, 1), r_2\text{: } 00 \rightarrow (\text{out}, 0)\}$,

$P_1 = \{r_1 > r_2\}$.

The initial configuration of the membrane OR gate appeared in (Fig. 6).

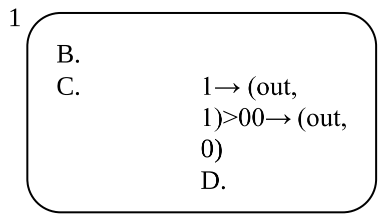


Fig. 6. OR gate membrane

TABLE II shows the tracing of the inputs to the membrane, and the all available scenarios that might happen when the pairs enter the membrane.

TABLE II: OR GATE OUTPUT

Pair	Output
(0,0)	0
(0,1)	1
(1,1)	1

C. *Membrane NAND Gate*

Boolean NAND gate as in the case of previous gates can be simulated by P systems with rewriting priorities rules using only one membrane including its rules in only two steps. The formal model definition of the membrane NAND gate consisting of degree 1 is a tuple II, and also we assume that the final result will be at the skin.

The formal model definition of the membrane NAND gate of degree 1 is a tuple II

$\Pi = (O, \mu, w_1, (R_1, P_1), 1)$

where

$O= \{0,1\}$,

$\mu = \{ [_1]_1\}$,

$w_1 = \lambda$,

$R_1 = \{r_1\text{: } 11 \rightarrow (\text{out}, 0), r_2\text{: } 1 \rightarrow (\text{out}, 1), r_3\text{: } 00 \rightarrow (\text{out}, 1)\}$,

$P_1 = \{r_1 > r_2\}$.

The initial configuration of the membrane NAND gate appeared in (Fig. 7). Like the previous membrane gates, the inputs came to the membrane from the outsider membrane.

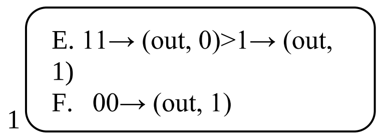


Fig. 7. NAND gate membrane

According to the model and tracing to the inputs, the all available scenarios that might happen are showed, when the pairs enter the membrane 1 are expressed in (TABLE III).

TABLE III: NAND GATE OUTPUT

Pair	Output
(0,0)	1
(0,1)	1
(1,1)	0

D. *Membrane XOR Gate*

At last, Boolean XOR as in the other gate also can be simulated by P systems with rewriting priorities rules using only one membrane including its rules in only two steps. The formal model definition of the membrane XOR gate that has degree 1 is a tuple II, and also we assume that the final result will be at the skin.

The formal model definition of the membrane XOR gate has the degree 1 is a tuple II

$\Pi = (O, \mu, w_1, (R_1, p_1), 1)$

where

$O = \{0, 1\}$,

$\mu = \{[_1]_1\}$,

$w_1 = \lambda$,

$R_1 = \{r_1\text{: } 11 \rightarrow (\text{out}, 0), r_2\text{: } 10 \rightarrow (\text{out}, 1), r_3\text{:}00 \rightarrow (\text{out}, 0)\}$,

$P_1 = \{r_1 > r_2 >, r_3 > r_2\}$.

The initial configuration of the membrane XOR gate appeared in (Fig. 8).

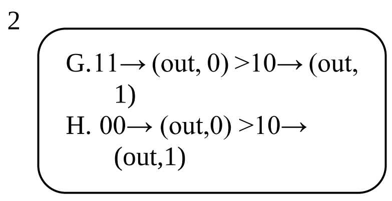


Fig. 8. XOR gate membrane

TABLE VI shows the tracing of the inputs to the membrane, and the all available scenarios that might happen when the pairs enter the membrane.

TABLE VI: XOR GATE OUTPUT

Pair	Output
(0,0)	0
(0,1)	1
(1,1)	0

V. LOGIC CIRCUITS SIMULATION

After approaching all the logic models of membrane computing, the computational processes in the nested levels are highly independent, and as a general mapping:

Membrane System : Gates

Skin Membrane : Gate Tester

Membrane : Gate

Rules : Instructions (Gate Operation)

Objects : I/O

Structure of Membranes: Structure of Gates (more than one gate)

The required design of any constructed circuit using the simulated gates is a nested set of gates;

In the next example; the last operation is the main container, and the first operation is subset of the upper level so that the result of each circuit step out to the higher level or gate to get the final result.

Logic Circuit Example

Using this mapping, we give an example of how constructing a P system which simulates a logic circuit, using the basic P systems constructed in the section 4 (AND, OR, NAND, and XOR). The following example contains three gates connected together to check the output of the final P system, which is the output of the logic gates, for example, if we have gates connected together as in Fig. 4 will produce a P system with tree representation of the membrane structure as shown in Fig. 9 and membrane structure as shown in Fig. 10.

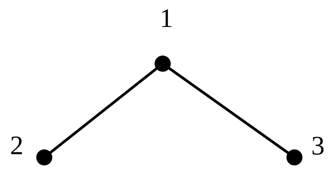


Fig. 9. The tree representation of the membrane structure of the example

The corresponding P system Definition will be as follows

$\Pi = (O, \mu, w_1, \ldots, w_3, (R_1, P_1), \ldots, (R_1, P_3), 0)$

O= {0, 1},

$\mu = \{[_1\ [_2]_2]\ [_3]_3]_1\}$,

$w_1 = \lambda$,

$w_2 = \lambda$,

$w_3 = \lambda$,

R_1= {r_1: 11→ (out, 0), r_2: 10→ (out, 1), r_3:00 → (out, 0)},

P_1= {$r1>r2>$, $r3>r2$}.

R_2= {$r1$: 11→ (out, 1), $r2$: 1→ (out, 0), r_3: 0→ (out, 0)},

P_2= {$r_1>r_2$}.

$R3$= {r_1: 1→ (out, 1), r_2: 00→ (out, 0)},

P_3= {$r_1>r_2$}.

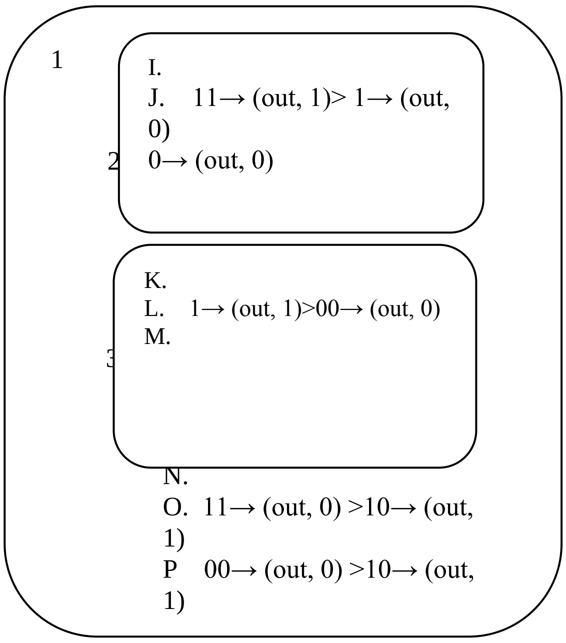

1
I.
J. 11→ (out, 1)> 1→ (out, 0)
2
0→ (out, 0)
K.
L. 1→ (out, 1)>00→ (out, 0)
M.
3
N.
O. 11→ (out, 0) >10→ (out, 1)
P 00→ (out, 0) >10→ (out, 1)

Fig. 10. P system initial configuration for the logic gates example

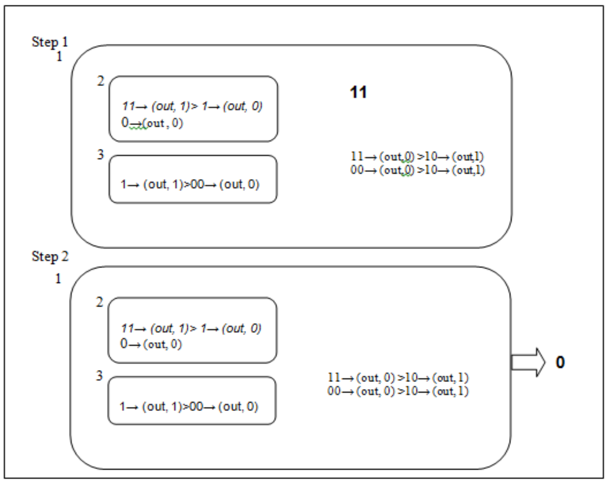


Fig. 11. Steps of P system computation for the logic gates example

When the objects "110" used as inputs a = 1,b = 1and c = 0 (according to Fig. 4) region 2 will receive "11" and region 3 will receive "10" according to the circuit design; the output is "1" from region 2 and "1" from region 3 which will be the input objects to the region 1 which works and generates the final output "0" as shown in the previous computation steps (step 1, and step 2), the right answer according to the truth table appeared in Fig. 4.

The computation of membranes will start with parallel and sequence arrangement in the same time which means that: when regions 2 and 3 receive their inputs they will start to work and generate their outputs to region 1 which wait to work because some rules may start working with the generated objects unless it waits for another one, this reduces the time needed to finish the computation. So, the computation calculated in a non-deterministic parallel way [6], this means that it take less time compared with sequential way because the two gates will be worked in the same time to get their output to push it to the last one as in the example. On the other hand, this indicates that the system got the desired output.

VI. Simulation Implementation Using P-Lingua

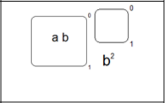
P-Lingua is a programming language for membrane computing developed by the Research Group on Natural Computing members, University of Seville [9, 10].

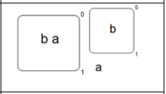
The designs of the gates mentioned in Fig. 4 are implemented and simulated using P-lingua. TABLE V list all simulated gates, for each gate there is the P-Lingua code to implement it and its output when it works like the one in section 3 and Fig. 3.

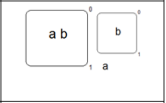
Note that we replaced the "1" with "a" and "0" with "b" for implementation.

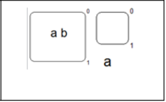
TABLE V: Gates Code and Run Steps Using P-Lingua

Gate	Code	Time	Output

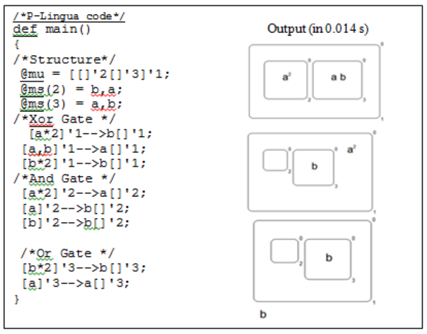
AND	def And(a,b)
	{
	@mu= []'1;
	@ms(1)= a,b;
	[a*2]'1-->a[]'1;
	[a]'1-->b[]'1;
	[b]'1-->b[]'1;
	}
		0.012 s	

OR	def Or(a,b)
	{
	@mu= []'1;
	@ms(1)= a,a;
	[b*2]'1-->b[]'1;
	[a]'1-->a[]'1;
	}
		0.0050 s	

NAND	def Nand(a,b)
	{
	@mu= []'1;
	@ms(1)= a,b;
	[a*2]'1-->b[]'1;
	[a]'1-->a[]'1;
	[b*2]'1-->a[]'1;
	}
		0.0050 s	

XOR	def Xor(a,b)
	{
	@mu= []'1;
	@ms(1)= b,b;
	[a*2]'1-->b[]'1;
	[a,b]'1-->a[]'1;
	[b*2]'1-->b[]'1;
	}
		0.0050 s	

For more addition to the correctness of the proposed gates, we will implement the same example appeared in Fig. 4 (simulated in last section) with P-lingua as a kind of test. As shown in Fig. 12 we can see the code and the output of the circuit. The simulation takes 0.014 s, which means less than the total times of the three used gates (this proves the parallel computations). Also we can observe that the output is as desired, actually all the combinations were tested; all of them gave the correct answer.



```
/*P-Lingua code*/
def main()
{
/*Structure*/
 @mu = [[]'2[]'3]'1;
 @ms(2) = b,a;
 @ms(3) = a,b;
/*Xor Gate */
  [a*2]'1-->b[]'1;
 [a,b]'1-->a[]'1;
 [b*2]'1-->b[]'1;
/*And Gate */
 [a*2]'2-->a[]'2;
 [a]'2-->b[]'2;
 [b]'2-->b[]'2;

 /*Or Gate */
 [b*2]'3-->b[]'3;
 [a]'3-->a[]'3;
}
```

Fig. 12. The code and the output of the circuit

VII. Conclusion and Future Work

We introduce in this paper the original design and the simulated version of AND, OR, NAND, and XOR logic gates using P system. An example is also given for logic gates and the simulated logic gates for the same example (logic circuit) which produce the same outputs like logic gates.

We notice that the logic gates used in most systems and its computation is sequential in most cases; in this paper; we consider reducing time and computations as much as we can through using the non-deterministic parallel computation of the P system by implementing the gates as membranes and rules with priorities, The models may need some extra developments to reach the full parallelism in both sides inside the gate and within gates when connecting a group together.

As shown in Fig. 12 we can see the code and the output of the circuit. The simulation takes 0.014 s which means less than the total times of the three used gates (this proves the parallel computations)

Comparing to the other simulated logic gates appeared in [2, 3] there is a fewer number of membranes and less number of rules that leads to fewer computations.

Other directions are in possible reduction of computation that we may use the rough P system [11, 12] based on the rough set theory [13] to minimize the gates computation and logic gate simplification as done in logic gates with P systems in the near future.

References

[1] Gh. Păun, "Computing with Membranes," *Journal of Computer and System Sciences*, vol. 61, Issue 1, pp. 108-143, 2000.

[2] O. Ogihara and M. Ray A.,"Simulating Boolean Circuits on a DNA Computer," *Technical Report* 631, Department of Computer Science, Rutgers University, 1996.

[3] R. Ceterchi and D. Sburlan," Simulating Boolean Circuits with P Systems," in *Membrane Computing, Proceedings of the International Workshop WMC* 2003, Tarragona, Spain, pp. 104–122, July 2003.

[4] G. Ciobanu, M. J. Perez-Jimenez, and G. Paun, Applications of Membrane Computing, Natural Computing Series, Springer, 2006.

[5] B. Nadia and Za. Claudio." Computing with Genetic Gates, Proteins and membranes," 7th International Workshop, WMC pp. 250–265, 2006.

[6] Gh. Paun, Membrane Computing, An Introduction, Springer-Verlag, Berlin, 2002.

[7] Gh. Păun, "Membrane Computing and Brane Calculi," *Electronic Notes in Theoretical Computer Science* vol. 171, Issue 2, pp. 3-10, 2007.

[8] M. Mano. *Logic and Computer Design Fundamentals*, vol. 1, Pearson/Prentice Hall, 2004.

[9] D. Diaz-Pernil, I.Perez-Hurtado, M. J Perez-Jimenez, and A. Riscos-Nunez, "A P-Lingua programming environment for membrane computing," In *Corne et al., Revised Selected and Invited Papers, Lecture Notes in Computer Science* vol.5391, pp. 187-203, 2009, Springer.

[10] M. Garcia-Quismondo and R. Gutierrez-Escudero, *et al*.," P-Lingua 2.0: New Features and First Applications," *Seventh Brainstorming Week on Membrane Computing*, Sevilla, pp. 141-167, 2009.

[11] A. D. Nolal, Gh. Păun, M. J. P ́erez-Jim ́enez, and F. Rossell, "Handling Imprecision in P Systems," *Brainstorming Workshop on Uncertainty in Membrane Computing*, Palma, p 1, 2004.

[12] G. Păun, "Rough P Systems," *Brainstorming Week on Membrane Computing*, Tarragona, p 9, 2003.

[13] J. Bazan, H. S. Nguyen, and M. Szczuka, "A View on Rough Set Concept Approximations," *Fundamenta Informaticae*, vol. 59, pp. 107–118, 2004.