HW IMPLEMENTATION OF A OPTIMIZED ALGORITHM FOR THE APPLICATION OF ACTIVE RULES IN A TRANSITION P-SYSTEM

Victor Martinez, Luis Fernandez, Fernando Arroyo, Abraham Gutierrez

Abstract: P systems or Membrane Computing are a type of a distributed, massively parallel and non deterministic system based on biological membranes. They are inspired in the way cells process chemical compounds, energy and information. These systems perform a computation through transition between two consecutive configurations. As it is well known in membrane computing, a configuration consists in a m*-tuple of multisets present at any moment in the existing* m *regions of the system at that moment time. Transitions between two configurations are performed by using evolution rules which are in each region of the system in a non-deterministic maximally parallel manner.*

This work is part of an exhaustive investigation line. The final objective is to implement a HW system that evolves as it makes a transition P-system. To achieve this objective, it has been carried out a division of this generic system in several stages, each of them with concrete matters.

In this paper the stage is developed by obtaining the part of the system that is in charge of the application of the active rules. To count the number of times that the active rules is applied exist different algorithms. Here, it is presents an algorithm with improved aspects: the number of necessary iterations to reach the final values is smaller than the case of applying step to step each rule. Hence, the whole process requires a minor number of steps and, therefore, the end of the process will be reached in a shorter length of time.

Keywords: Membrane Computing, Evolution Rules, Circuit design, Digital systems, Transition P System.

ACM Classification Keywords: D.1.m Miscellaneous – Natural Computing

Introduction

The new proposed models of bio-molecular (with DNA, RNA, proteins, or with membranes) and quantum computing maybe, in the future, will be probably very powerful technologies for advanced parallel super-computation. These new unconventional models are not simple improvements of the previous ones. Moreover, their potential advantage come from the inherent parallelism of the biological or physical processes they are inspired.

The Membrane Computing or P Systems (created by [Paun, 1998]) are computation systems based on the biomolecular processes of living cells. According to this, the investigations are based on the idea that the imitation of the procedures that take place in Nature and their application to machines, can lead to discover and to develop new computation models that will give place to a new generation of intelligent computers.

These systems perform a computation through transition between two consecutive configurations. Transitions between two configurations are performed by using evolution rules present in each region. If the system reaches a configuration in which there are no applicable rules at any membrane, it is said that the system reaches a halting configuration and, hence, the computation is successful.

There are many papers about software tools implementing different P-system variants [Arroyo 2003], [Arroyo 2004b] and [Fernandez, 2005a]. However, they are very interesting in order to define hardware implementation of these kinds of systems. Moreover, evolution of transition P- systems is very complicate to translate into hardware devices due mainly to the membrane dissolving or membrane division capabilities of rules. Besides that, the non-deterministic maximally parallel manner in which rules are applied inside membranes is much appropriated to be implemented in digital hardware devices. In the case of P-systems hardware implementations only can find a few of papers: a Hardware Circuit for Selecting Active Rules [Fernandez 2005b], connectivity arrays for membrane processors [Arroyo 2004a], a multisets and evolution rules representation in membrane processors [Arroyo 2004b] or even a hardware membrane system description using VHDL [Petreska 2003] .

This work is a part of a very ambitious project: to find and carry out a Hardware System able to simulate P systems evolution. This generic system has been divided into several stages. The first stage one develops a circuit able to determine active rules in a determined configuration for the membrane [Fernandez 2005b]. In this document, the second stage is developed: a circuit for the application of the active rules obtained in the first stage.

In order to proceed in an appropriate way, it is needed to define a data structure containing information about the initial membrane state, that is, the initial multiset of objects, the set of evolution rules and the corresponding priority relation among them. The circuit takes these data inputs and produce, as output, a set of evolution rules, active rules, which are able to produce the needed changes into the system in order to make evolve it to into the next configuration. Obviously, there are some needed constrains for the hardware design, and they are the following:

a. The cardinality *O = {a,b,c,d,e,f,g,h,i,j}* of the alphabet is limited to 10.
b. The multiset of objects associated to the membrane i, ω_i . It is represented in a hardware register of 4-bits words of length 10.
c. The finite set of evolution rules *Ri* associated to the membrane *I* (10 per membrane).
d. The priority relation ρ_i among the rules *Ri*.
e. Extra information to determine applicability of rules: existence register, for determining whether or not there exist children membranes for the next evolution step.

Therefore, the previous *Circuit for Selecting Active Rules* determine which of the evolution rules present inside a membrane are active (in binary positive logic) accordingly to the multiset of objects present in the membrane.

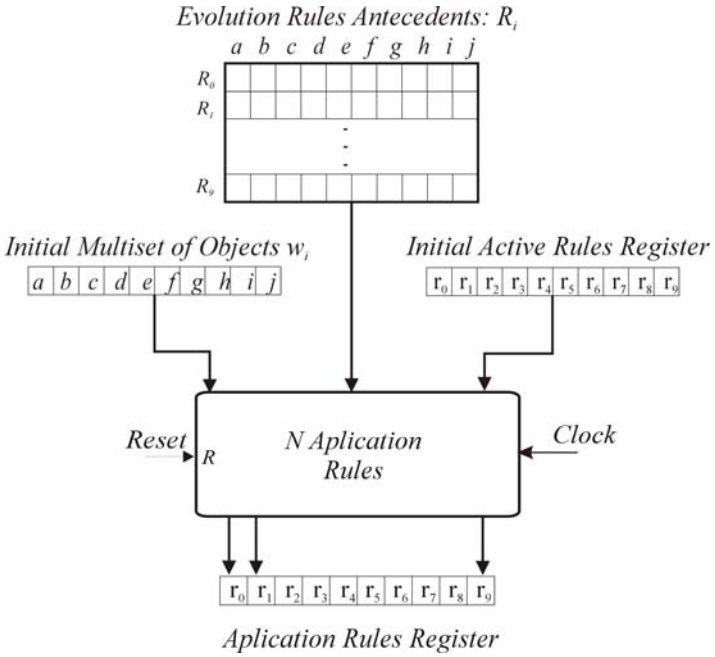

Fig. 1: General structure of the circuit for Application of Active Rules.

The objective of the work that now shows up will consist in obtaining a HW circuit that indicates the rules to be applied to a Transition P System. The inputs are the following registers: multiset of objects associated to the membrane *i*; Evolution Rules antecedents and Initial Active Rules. The output will be a register that contains the number of times that each rule (See Fig.1) should be applied. These values associated to each rule will serve to carry out, in a later process, the communication of elements among regions.

Application of the Active Rules

The application of the active evolution rules is a repetitive process that can be carried out in different ways. The paper [Fernandez, 2006] allows us to obtain circuits or systems optimized as for level of complexity and resources utilization.

A first option to apply the rules could consist on the step-by-step application of the group of active rules. In each iteration step, one of the active rules is randomly chosen. If the elected rule is applicable, it is applied. Every time a rule is applied, a particular accountant is increased. These accountants, associated to each rule, will represent, at the end of the process, the number of times that rule has been applied. If the elected rule is not applicable, this rule is eliminated of the active rules group. The process concludes when there is not any more applicable rule, that is to say, the group is empty.

To improve the previous algorithm, we can apply the rules certain number of times bigger than the unit in a single evolution step. The maximum number of times a rule can be applied into a multiset ω_i of state of the cell in an evolution step will be called "MAX value." This value is obtained considering that you apply only this rule, that is to say, without keeping in mind the other rules. In other words, the *MAX* value represents the biggest number of times that a rule can be applied supposing that only this rule is applied

The following example illustrates the concept of *MAX* value: If we have a multiset of objects $\omega_i=a^{10}b^5c^7$ and the following active rules $R_1=a^1$, $R_2=b^2c$ and $R_3=bc^2$;el value MAX of R_1 is 10, the value MAX of R_2 is 2 and the value MAX of R_3 is 3.

The fact of applying a rule a value bigger than one implies that it consumes, in an evolution step, a bigger number of elements from the multiset ω. Hence, the whole process requires a minor number of steps and, therefore, the end of the process will be reached in a shorter length of time. The following pseudo-code sequence illustrates the necessary process to obtain the number of times that each active rule should be applied in a region:

```
(1)  R ← InitialActiveRules
(2)  BEGIN
(3)    DO
(4)       r_i ← Aleatory(R)
(5)       MAX ← Applicability(r_i, ω)
(6)       N ← Aleatory(1, MAX)
(7)       ω ← (ω − (N * Antecedent(r_i)))
(8)       counts(N, r_i)
(9)       R ← ActiveRules
(10)   WHILE |R| ≥ 1
(11)  END
```

Explanation of the algorithm:

- (1) The process uses the group of active available rules initially R.
- (4) At each iteration, one of the rules r_i of this group will be applied. Such rule will be randomly obtained.
- (5) On the selected rule, the value of applicability *MAX* is obtained and it is applied with an aleatory multiplicity *N* between 1 and the value *MAX* (6)
- (7) The application of selected rule r_i will consist in subtracting from the starting multiset ω, the values of the antecedent elements multiplied by the value *N* of rule application. In turn, we will increase *N* times the particular accountant that counts the number of times that rule has been applied (8).
- (9) On the new obtained multiset ω it has been proved again the applicability of the available rules.
- (10) Every time the group of applicable rules is upgraded, the end of the process has been controlled. The stop condition is obtained when the number of applicable rules is zero. While R is bigger or the same as the unit, it executes, once more, a new iteration of the process.

Basic Functional Units (F.U.)

This section defines the previous step to design the complete circuit of active rules application to the evolution of a transition P system. It will consist on obtaining certain basic operative functional units. These functional units will solve each one of the simple tasks needed to obtain the complete application. The design of the final circuit will be based on the assembling of these modules together with the corresponding combinational and sequential logic which allows their integrated operation.

Applicability MAX F.U.:

In this case, we will obtain the design of the circuit that obtains the *MAX* applicability of a rule. This value supposes the largest number of times that a rule can be applied independently of the other ones. To do so, we will only keep in mind the multiplicities of their elements and the multiplicities of the elements of the multiset ω.

Therefore, the inputs into this circuit will be two registers: one with the content of the values of the multiset ω of state of the membrane and another with the antecedent of the rule we want to get their value *MAX*. The output will be the value *MAX* of this rule. To obtain this value we should carry out the division among each couple of elements similar of the multiset ω_i with r_i. From each division, we will participated the maximum that will represent the largest number of times that mentioned element could be consumed in an evolution step without bearing in mind the other elements. We will take the smallest value out of all the partial results of these divisions.

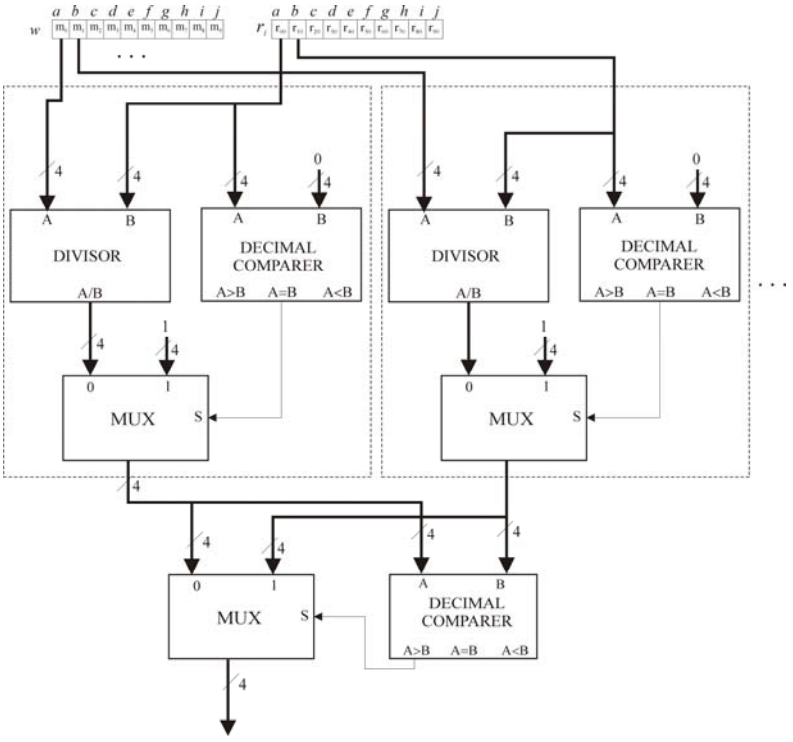

Fig. 2: Part of applicability MAX of a rule circuit..

When r_i is equal to zero, the result is forced to 1´s all. In the following comparison process, the minor value is obtained. The all 1´s result is not chosen and, therefore, does not have any incidence in the final result. The figure 2 shows the part to are carried obtain the dividing minimum between the first two elements of the circuit. To obtain the total circuit it will be necessary to add other dividers until the total elements are covered and the opportune comparisons to obtain the smallest value carry out.

1 Aleatory Active Rule F.U.

This circuit will randomly select a rule from the ActiveRules register to be applied. The ActiveRules register contains binary values indicating, with positive logic, the active rules from existing rules of the system. The output

of this circuit will be a register in which only one rule will be randomly selected. One input Enable *"E"* will activate the starting of the clock that attacks the random generator.

This generator will select decimal numbers aleatory in a serial way until getting some one that corresponds with the number of the active rule. When this value is obtained, the aleatory number generation should stop.

The main part of the circuit will be a 1 bit Decimal Multiplexer. The selection inputs of the Multiplexer correspond with the outputs from the Decimal aleatory generator. This generator will stop when some of the outputs of the Multiplexer are set to 1. The position that indicates this output corresponds with the randomly selected active rule.

When only one position of the ActiveRules register is present, it means that only one active rule exists. In this case it is not necessary to carry out the process of aleatory selection. To avoid a loss of efficiency that could happen in this case, the circuit has been endowed with a special operation condition. The solution consists in detecting this condition previously and to provide the Multiplexer with a specific input *"ALL"* that allows to get the content of all the multiplex inputs (See Fig.3).

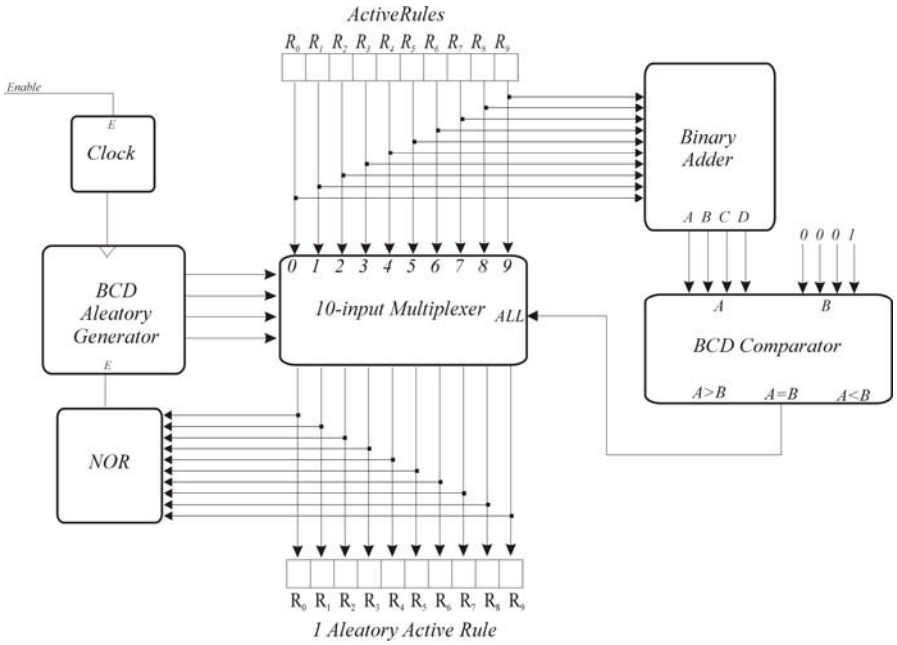

Fig. 3: Internal structure of the circuit for 1 Aleatory Active Rule.

Application r_i Aleatory 1-*MAX* F.U.

This circuit obtains an output register associated to the application of the rule r_i. This register contains the multiplicity of those elements that they should be subtracted from the multiset associated to the membrane region.

This result is reached in the case the rule r_i is applied between a value 1 and its *MAX* value, elected in an aleatory way. To count the number of times each rule is being applied, we should also extract this aleatory value of application *N*.

The inputs are, therefore, the register that contains the multiset of objects associated to the region ω and the antecedent r_i. Also, we will endow the circuit with an input Enable *"E"* that allows to select witch rules will act in each evolution step. The output will be stored in another register ω. Later on, these values will be subtracted from the state multiset in order to obtain the resulting new values.

Internally, the circuit is formed by an "F.U. Applicability MAX of a rule" (to obtain the value MAX) and an "F.U. Product multiset by N". Also, a generating circuit of aleatory numbers will select one random value between 1 and *MAX*. This number will be the value for which we will multiply by the antecedent of the rule. Finally, a series of AND gates allows to enable or to disable the output in function of the sign Enable (*E*) :

General Structure of the circuit

The general circuit is the result of the assembling of the different Functional Units, together with the sequential logic of control. The sequential logic determines the evolution of the internal steps the circuit should travel though until reaching the stop condition. This condition will be given when the register *R* of active rules is empty.

The sequence of events that the *sequential controller* should activate is based on the use of a 2 bits counter that allows reaching 4 states. Each state determines an evolution event. The counting continues in a recurrent way until the stop condition is reached, and then the accountant will be stopped. Fig. 4 shows the circuit to determine the times the active rules should be applied.

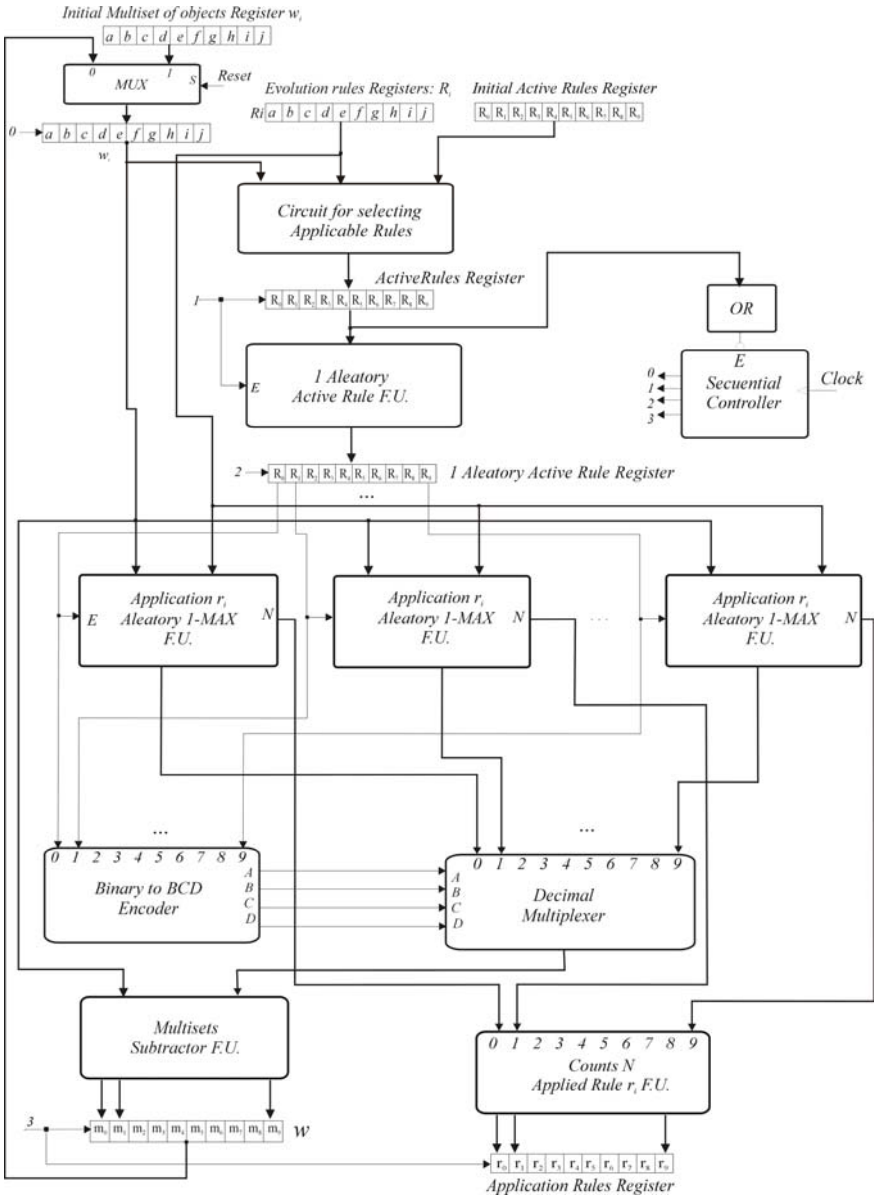

Fig. 4: Internal structure of the circuit to determine the times that the active rules should be applied.

The general structure of the circuit is based on the following sequence of states:

a. Initialization (Reset) phase: The sign *R = 1* load the register ω with the values of the initial multiset of objects. On the other hand, the Initial Active Rules Register has been obtained according to the paper [Fernandez, 2005b].

b. State 0: It proceeds to load the register ω and to calculate active rules. The Active Rules Register is obtained by checking in each evolution step the condition of applicability for the rules active initials and for the new multisets of objects. Applicable rules are those rules accomplishing that their antecedent is included in the multiset of objects found inside the membrane.
c. State 1: The Active Rules Register is loaded and begins the process to obtain 1 Aleatory Active Rule.
d. State 2: The 1 Aleatory Active Rule register is loaded. The application of the selected rule begins in order to obtain the new multiset of objects ω and the calculation of the number of times that such rule will be applied.
e. State 3: The Multiset of Objects Register ω and the Application Rules Register are loaded. The Application Rules Register will store the number of times each rule has been applied to. This register will be the output result we want to obtain.
f. Turn to the state 0: to load the register ω and starting a new calculation cycle with the new values.

Conclusions

This paper presents a direct way to design a circuit able to obtain the number of active rules application inside the membrane in a non deterministic and massively parallel application. The final objective is implementing a hardware circuit accomplishing the outlined initial requirement. That is, given an initial multiset of objects ω, a finite set of evolution rules and an initial Active Rules, the circuit provides the number set of application of rules to a membrane.

The more interesting aspects of this design are:

- The used algorithm allows a process with a minor number of steps and, therefore, we will reach the end of the computation in a shorter length of time.
- The hardware implementation is founded on basic components like registers, logical gates, multiplexer and sequential elements.
- The development of the digital system can be carried out using hardware-software architectures like Handel C or VHDL.
- The physical implementation can be accomplished on hardware programmable devices like FPGA's.

The next step in the process for the developing of a membrane processor is to design a circuit able to communicate elements from one region to another.

Bibliography

[Arroyo 2003] F. Arroyo, C. Luengo, A.V. Baranda, L.F. de Mingo, A software simulation of transition P systems in Haskell, Pre-Proceedings of Second Workshop on Membrane Computing, Curtea de Arges, Romania, August 2002 and Proc. of WMC02, Curtea de Arges, Romania, (Gheorghe Paun, Grzegorz Rozenberg, Arto Saloma, Claudio Zandron Eds.) Lecture Notes in Computer Science 2597, Springer-Verlag, 2003, 19-32.

[Arroyo 2004a] F. Arroyo, C. Luengo, J. Castellanos, L.F. de Mingo: A binary data structure for membrane processors: Connectivity arrays, Artiom Alhazov, Carlos Marín-Vide and Gheorghe Paun (eds.): Preproceedings of the Workshop on Membrane Computing, Tarragona, July 17-22 2003, 41-52 and in (Carlos Marín-Vide, Giancarlo Mauri, Gheorghe Paun, Grzegrorz Rozenberg, Arto Saloma Eds.) Lecture Notes in Computer Science, 2933, Springer Verlag, 2004, 19-30.

[Arroyo 2004b] F.Arroyo, C.Luengo, L.Fernandez, L.F.Mingo, J.Castellanos. Simulating membrane systems in digital computers. International Journal Information Theories and Applications, 11, 1 (2004), 29-34.

[Arroyo 2004c] F. Arroyo, C. Luengo, J. Castellanos, L.F. de Mingo, Representing Multisets and Evolution Rules in Membrane Processors, Pre-proceedings of the Fifth Workshop on Membrane Computing (WMC5), Milano, Italy, June 2004, 126-137.

[Fernandez, 2005a] L.Fernandez, F.Arroyo, J.Castellanos, V.J.Martinez, Software Tools / P Systems Simulators Interoperability, Pre-proceedings of the 6th Workshop on Membrane Computing, Vienna - Austria, July 2005.

[Fernandez, 2005b] L.Fernandez, V.J.Martínez, F.Arroyo, L.F.Mingo, A Hardware Circuit for Selecting Active Rules in Transition P Systems, Workshop on Theory and Applications of P Systems. Timisoara (Rumanía), september, 2005.

[Fernandez, 2006] L.Fernandez, F.Arroyo, J.Castellanos, J.A.Tejedor, I.García, New Algorithms for Application of Evolution Rules based on Applicability Benchmarks, BIOCOMP06 International Conference on Bioinformatics and Computational Biology, Las Vegas (USA), july, 2006 (submitted).

[Paun 1998] Gh. Paun, Computing with membranes, Journal of Computer and System Sciences, 61 (2000), and Turku Center for Computer Science-TUCS Report No 208, 1998.

[Petreska 2003] B.Petreska, C.Teuscher, A hardware membrane system. Preproceedings of the Workshop on Membrane Computing (A.Alhazov, C.Martín-Vide and Gh.Paun, eds) Tarragona, July 17-22 2003, 343-355.

Authors' Information

Victor J. Martinez Hernando – Dpto. Arquitectura y Tecnología de Computadores de la Escuela Universitaria de Informatica de la Universidad Politécnica de Madrid, Ctra. Valencia, km. 7, 28031 Madrid (Spain); e-mail: victormh@eui.upm.es

Luis Fernandez Munoz – Dpto. Lenguajes, Proyectos y Sistemas Informaticos de la Escuela Universitaria de Informatica de la Universidad Politécnica de Madrid; Ctra. Valencia, km. 7, 28031 Madrid (Spain); e-mail: setillo@eui.upm.es

Fernando Arroyo Montoro – Dpto. Lenguajes, Proyectos y Sistemas Informaticos de la Escuela Universitaria de Informatica de la Universidad Politécnica de Madrid, Ctra. Valencia, km. 7, 28031 Madrid (Spain); e-mail: farroyo@eui.upm.es

Abraham Gutierrez – Dpto. Informatica Aplicada de la Escuela Universitaria de Informatica de la Universidad Politécnica de Madrid, Ctra. Valencia, km. 7, 28031 Madrid (Spain); e-mail: abraham@eui.upm.es

CONTRADICTION VERSUS SELFCONTRADICTION IN FUZZY LOGIC*

Carmen Torres, Susana Cubillo, Elena Castineira

Abstract: Trillas et al. introduced in [7] and [8] the concepts of both self-contradictory fuzzy set and contradiction between two fuzzy sets. Later, in [1] and [2] the necessity of determine not only the contradiction, but also the degree in that this property occurs, was considered. This paper takes up again these subjects, and firstly we study if there exists some connection between the two first notions. After that, taking into account that self-contradiction of a fuzzy set could be understood as the contradiction with itself, and starting from the degrees of contradiction between two fuzzy sets proposed in [5], we obtain degrees of self-contradiction. Finally, preservation of some intuitive properties both in the use of connectives and in the obtaining of new knowledge throughout compositional rule of inference, are tested.

Keywords: fuzzy sets, t-norm, t-conorm, strong fuzzy negations, contradiction, measures of contradiction, fuzzy relation, compositional rule of inference.

ACM Classification Keywords: F.4.1 Mathematical Logic and Formal Languages: Mathematical Logic (Model theory, Set theory); I.2.3 Artificial Intelligence: Deduction and Theorem Proving (Uncertainty, "fuzzy" and probabilistic reasoning); I.2.4 Artificial Intelligence: Knowledge Representation Formalisms and Methods (Predicate logic, Representation languages).

* This work is partially supported by CICYT (Spain) under project **TIN 2005-08943-C02-001** and by UPM-CAM (Spain) under project **R05/11240**.