# Rete Algorithm for P System Simulators

Carmen Graciani, Miguel A. Gutiérrez-Naranjo, Agustín Riscos-Núñez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
{cgdiaz,magutier,ariscosn}@us.es

**Summary.** The Rete algorithm is a well-known algorithm in rule-based production systems which builds directed acyclic graphs that represent higher-level rule sets. This allows the rule-based systems to avoid complete re-evaluation of all conditions of the rules each step in order to check the applicability of the rules and, therefore, the computational efficiency of the production systems is improved. In this paper we study how these ideas can be applied in the improvement of the design of computational simulators in the framework of Membrane Computing.

## 1 Introduction

Rules is one of the most used paradigms in Computer Science for dealing with information. Given two pieces of knowledge $V$ and $W$, expressed in some language, the rule $V \to W$ is usually considered as a causal relation between $V$ and $W$. The interpretation of the rule can change according to the context, but roughly speaking, the rule $V \to W$ claims that the statement $W$ can be *derived* from the statement $V$. The problem of knowing if a piece of information $G$ can be obtained via *derivation* from a set of current statements $A$ and a set of rules $R$ arises in a natural way. This is usually called a *reasoning problem* and it will be denoted by $\langle A, R, G \rangle$.

In Computer Science, there are two basic methods for seeking a solution of a reasoning problem, both of them based on the inference rule known as Modus Ponens:

$$\frac{V \quad V \to W}{W}$$

which allows to obtain $W$ from the rule $V \to W$ and the piece of information $W$. The first method is data-driven and it is known as *forward chaining*, the latter is query-driven and it is called *backward chaining* [1]. A study of these methods in the framework of Membrane Computing can be found in [12, 13].

The piece of information $V$ (the left-hand side of the rule or LHS) is usually split into unit pieces $v_1, v_2, \ldots, v_n$. The forward chaining derivation of $W$ (the right

```
Forward chaining
INPUT: A reasoning problem ⟨A, R, G⟩
INITIALISE: Memory = A, Deduced = ∅
  if G ∈ Memory then
    return true
  end if
  while Memory ≠ Deduced do
    Deduced ← Memory
    for all (v₁v₂...vₙ → W) ∈ R such that
           {v₁, v₂, ..., vₙ} ⊆ Deduced do
      if W = G then
        return true
      else
        Deduced ← Deduced ∪ {W}
      end if
    end for
  end while
  return false
```

**Fig. 1.** From a computational point of view, the reasoning problem $\langle A, R, G \rangle$, can be solved with the forward chaining algorithm

hand side of the rule or RHS) according to the Modus Ponens via the rule $V \rightarrow W$ needs to check if the statements $v_1, v_2, \ldots, v_n$ belong to the set of statements currently accepted. Figure 1 shows a detailed description of the forward chaining method.

The key point of this algorithm is to check for all rules $v_1 v_2 \ldots v_n \rightarrow W$ whether $\{v_1, v_2, \ldots, v_n\} \subseteq Deduced$ or not. A naive algorithm for this checking consists on

1. Enumerating the rules and the *Deduced* set.
2. Performing a sequential pattern matching between them.

In the framework of Expert Systems [10], a solution to this problem was proposed by Charles L. Forgy in his Ph.D. dissertation at the Carnegie-Mellon University in 1979 [8, 9]. The solution was called the *Rete*[1] *algorithm*. The Rete algorithm places pieces of information in the nodes of a graph and gets faster response time than the naive algorithm of checking one by one the information units in the LHS of the rules.

In spite of the notable differences between the semantics of the rules in Expert Systems and in Membrane Computing, the problem of checking if the restrictions of the LHS of the rule hold is common in both paradigms. In Membrane Computing, a rule of the form

$$u_1^{n_1} \ldots u_k^{n_k} \, [\, v_1^{m_1} \ldots v_l^{m_l} \,]_i^\alpha \rightarrow u' \, [\, v' \,]_i^{\alpha'}$$

---

[1] *Rete* means net in Latin.

is applicable if, in the current configuration, there exists a membrane labelled by $i$, with polarisation $\alpha$, containing enough objects $v_1 \ldots v_l$ and such that its surrounding membrane contains enough objects $u_1 \ldots u_k$. Although the *application* of the rule is different in both paradigms (in Membrane Computing, the objects in the LHS are consumed and the objects in the RHS are created; in Expert Systems the *information* in the LHS does not change, and the one in the RHS is considered *true*), in both cases it is necessary a *checking* of the conditions in order to decide the applicability of the rule.

In this paper we explore if the successful ideas underlying the Rete algorithm can be adapted to the current P systems simulators and contribute to improve their efficiency so that they can face medium-size instances of real life problems.

The paper is organised as follows: Next, we recall some preliminaries on the derivation process in logic and rule-based expert systems. In Section 3 a short description of the Rete algorithm is provided. Section 4 shows how this algorithm can be adapted to P system simulators. Some final remarks and lines for future research are provided in the last section.

## 2 Production Systems

Next, we recall some preliminaries on production systems and the derivation of pieces of knowledge by using rules.

### 2.1 Formal Logic Preliminaries

An *atomic formula* (also called an *atom*) is a formula with no deeper structure. An atomic formula is used to express some fact in the context of a given problem. The *universal set* of atoms is denoted with $U$. A *knowledge base* is a construct $KB = (A, R)$ where $A = \{a_1, a_2, \ldots, a_n\} \subseteq U$ is the set of known atoms and $R$, a set of rules of the form $V \to W$ with $V, W \subseteq U$, is the set of *production rules*.

In propositional logic, the *derivation* of a proposition is done via the inference rule known as Generalised Modus Ponens

$$\frac{P_1, P_2, \ldots, P_n \quad P_1 \wedge P_2 \wedge \cdots \wedge P_n \to Q}{Q}$$

The meaning of this rule is as follows: if $P_1 \wedge P_2 \wedge \cdots \wedge P_n \to Q$ is a production rule and $P_1, P_2, \ldots, P_n \subseteq A$ then $Q$ can be derived from this knowledge. Given a knowledge base $KB = (A, R)$ and an atomic formula $g \in U$, we say that $g$ can be derived from $KB$, denoted by $KB \vdash g$, if there exists a finite sequence of atomic formulas $F_1, \ldots, F_k$ such that $F_k = g$ and for each $i \in \{1, \ldots, k\}$ one of the following claims holds:

- $F_i \in A$.
- $F_i$ can be derived via Generalised Modus Ponens from $R$ and the set of atoms $\{F_1, F_2, \ldots, F_{i-1}\}$

## 3 Rule-based Expert Systems

Instead of viewing computation as a specified sequence of operations, production systems view computation as the process of applying transformation rules in a sequence determined by the data.

A classical production system has three major components: (1) a global database (or working memory) that contains facts or assertions about the particular problem being solved, (2) a rulebase that contains the general knowledge about the problem domain, and (3) a rule interpreter that carries out the problem solving process.

The facts in the global database can be represented in any convenient formalism. The rules have the form `IF <condition> THEN <action>`

In general, the LHS or condition part of a rule can be any pattern that can be matched against the database. It is usually allowed to contain variables that might be bound in different ways, depending upon how the match is made. Once a match is made, the right-hand-side (RHS) or action part of the rule can be executed. In general, the action can be any arbitrary procedure employing the bound variables. In particular, it can result in addition/elimination of facts to the database, or modification of old facts in the database.

What follows is the basic operation for the rule interpreter (this operation is repeated until no more rules are applicable):

1. The condition part of each rule (LHS) is tested against the current state.
2. If it matches, then the rule is said to be *applicable.*
3. From the applicable rules, one of them is chosen to be applied.
4. The actions of the selected rule are performed.

Production systems may vary on the expressive power of conditions in production rules. Accordingly, the pattern matching algorithm which collects production rules with matched conditions may vary.

### 3.1 The Rete Algorithm

The Rete algorithm is a well-known algorithm for efficiently checking the many pattern/many object pattern match problem [8], and it has been widely used mainly in production systems. In rule-based systems the checking process takes place repeatedly. This algorithm takes advantage of two empirical observations:

- *Temporal redundancy:* The application of the rules does not change all the current knowledge. Only some pieces of information are changed and the remaining ones (probably, most of them) keep unchanged.
- *Structural similarity:* Several rules can (partially) share the same conditions in the LHS.

This algorithm provides a generalised logical description of an implementation of functionality responsible for matching data from the current state of the

system against productions rules in a pattern-matching. It reduces or eliminates certain types of redundancy through the use of node sharing. It stores partial matches when performing joins between different fact types. This allows the rule-based systems to avoid complete re-evaluation of all facts each step. Instead, the production system needs only to evaluate the changes to working memory.

The Rete algorithm builds directed acyclic graphs that represent higher-level rule sets. They are generally represented at run-time using a network of in-memory objects. These networks match rule conditions (patterns) to facts (relational data tuples) acting as a type of relational query processor, performing projections, selections and joins conditionally on arbitrary numbers of data tuples. In other words the set of rules is preprocessed yielding a network in which each node comes from a condition of a rule. If two or more rules share a condition then they usually share that node in the constructed network. The path from the root node to a leaf node defines a complete rule LHS.

Facts flow through the network and are filtered out when they fail a condition. At any given point, the contents of the network captures all the checked conditions for all the present facts.

This network (a directed graph) has four kind of nodes:

- **Root:** acts as input gate to the network. Receives the changes in the knowledge base and then those tokens pass to the root successors.
- **Alpha nodes:** perform conditions which depend on just one pattern. If the test succeeds, then received token passes to the node successors. There are different alpha nodes depending on the considered pattern.
- **Beta nodes:** perform inter-patterns conditions, for example, if two patterns have a common variable. It receives tokens from two nodes and stores the tokens that arrive from each parent in two different memories. If a token arrives from one of its input, then the condition will be checked against all the tokens in the another input's local memory. For each successfully checked pair, a new token, combining both of them, passes to the node successors.
- **Terminal nodes:** receive tokens which match all the patterns of the LHS of a rule and produce the output of the network.

For example, if the following set of production rules and facts are considered, then the network displayed in Figure 2 will be created. The figure also shows how tokens corresponding to different facts pass through the network.

```
    Rule: R1                          Fact: f1  H1(2, 1).
        Exists H2(Y, Z, Z).           Fact: f2  H1(2, 4).
        Exists H1(X, Y > 3).          Fact: f3  H2(4, 3, 3).
            => ...                     Fact: f4  H2(5, 9, 9).

    Rule: R2                          Fact: f5  H3(3).
        Exists H2(Y, Z, Z).           Fact: f6  H3(9).
        Exists H3(Z).
            => ...
```
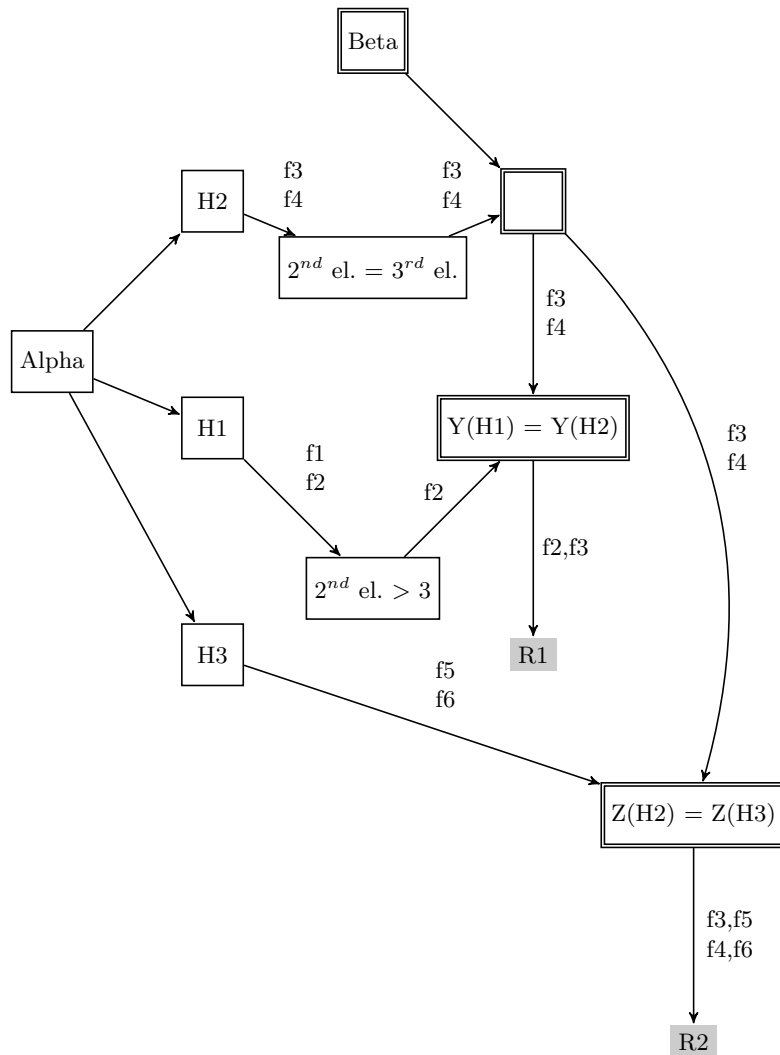
**Fig. 2.** Example of a Rete network and tokens flow

The most important issue regarding performance is the order of the conditions in the LHS of the rule. This lead us to consider the following strategies in order to improve the efficiency.

- Most specific to most general. If the rule activation can be controlled by a single data, place it first.
- Data with the lowest number of occurrences in the working memory should go near the top.

- Volatile data (ones that are added and eliminated continuously) should go last, particularly if the rest of the conditions are mostly independent.

With those strategies we are trying to minimise (in general) the number of *beta* nodes that will exist in the network and, therefore, the number of checks performed until a token arrived in a terminal node.

## 4 Membrane Computing

In this section we explore how the Rete algorithm can be adapted to Membrane Computing simulators. For a first approximation we have chosen to focus on rules handling polarisations, which can be written in the following form

$$\mathtt{u}_1^{\mathtt{n}_1} \cdots \mathtt{u}_{\mathtt{k}}^{\mathtt{n}_{\mathtt{k}}} [\mathtt{v}_1^{\mathtt{m}_1} \cdots \mathtt{v}_1^{\mathtt{m}_1}]_{\mathtt{i}}^{\alpha} \rightarrow \mathtt{u}'[\mathtt{v}']_{\mathtt{i}}^{\alpha'}$$

($\mathtt{k}$ and/or $\mathtt{l}$ can be 0) with $\mathtt{u}_1$, ..., $\mathtt{u}_{\mathtt{k}}$, $\mathtt{v}_1$, ..., $\mathtt{v}_1 \in \Gamma$, $\mathtt{u}'$, $\mathtt{v}' \in \Gamma^*$, and either $\mathtt{v}_1^{\mathtt{m}_1} \cdots \mathtt{v}_1^{\mathtt{m}_1} \neq \lambda$ or $\mathtt{v}' \neq \lambda$.

This rule is associated with any membrane with label $i$. In such a rule we can distinguish three kinds of conditions:

- Membrane label is $i$ and charge must be $\alpha$: $[]_{\mathtt{i}}^{\alpha}$
- Outside the membrane there must be at least $\mathtt{n}_{\mathtt{j}}$ copies of element $\mathtt{u}_{\mathtt{j}}$: $\mathtt{u}_{\mathtt{j}}^{\mathtt{n}_{\mathtt{j}}}$
- Inside the membrane there must be at least $\mathtt{m}_{\mathtt{i}}$ copies of element $\mathtt{v}_{\mathtt{i}}$: $[\mathtt{v}_{\mathtt{i}}^{\mathtt{m}_{\mathtt{i}}}]$

Now conditions can be reordered in order to follow the proposed strategies for production systems. For example, consider the following rules:

- (R1) $\mathtt{b}^3[\mathtt{ef}]_2^+ \rightarrow \mathtt{u}[\mathtt{v}]_2^{\alpha}$
- (R2) $\mathtt{b}^3[\mathtt{fe}^2]_2^+ \rightarrow \mathtt{u}'[\mathtt{v}']_2^{\alpha'}$.

We can describe them as follows in order to put at the beginning common conditions:

- (R1) $[]_2^+ [\mathtt{f}]\mathtt{b}^3[\mathtt{e}] \rightarrow \mathtt{u}[\mathtt{v}]_2^{\alpha}$
- (R2) $[]_2^+ [\mathtt{f}]\mathtt{b}^3[\mathtt{e}^2] \rightarrow \mathtt{u}'[\mathtt{v}']_2^{\alpha'}$.

To complete the example let us now consider a configuration where a membrane labelled by $2$ has positive charge, objects $\{\mathtt{f}^3, \mathtt{e}^7, \mathtt{o}^4, \mathtt{x}\}$ are inside it and the objects $\{\mathtt{c}, \mathtt{b}^8, \mathtt{g}^3\}$ are in the surrounding membrane. This configuration leads to the applicability of both rules.

Figure 3 shows the network associated to the rules of this example and how objects of the considered configuration go through different nodes. When there is not enough objects to pass trough a node they remain in it. Notice that from the output of the network we deduce that each rule can be used at most two times.

Figure 4 shows a generic algorithm to simulate a P system with active membranes using such networks. The halting conditions can be: A prefixed number of repetitions, there is no rule that can be applicable, occurrence of specific objects...
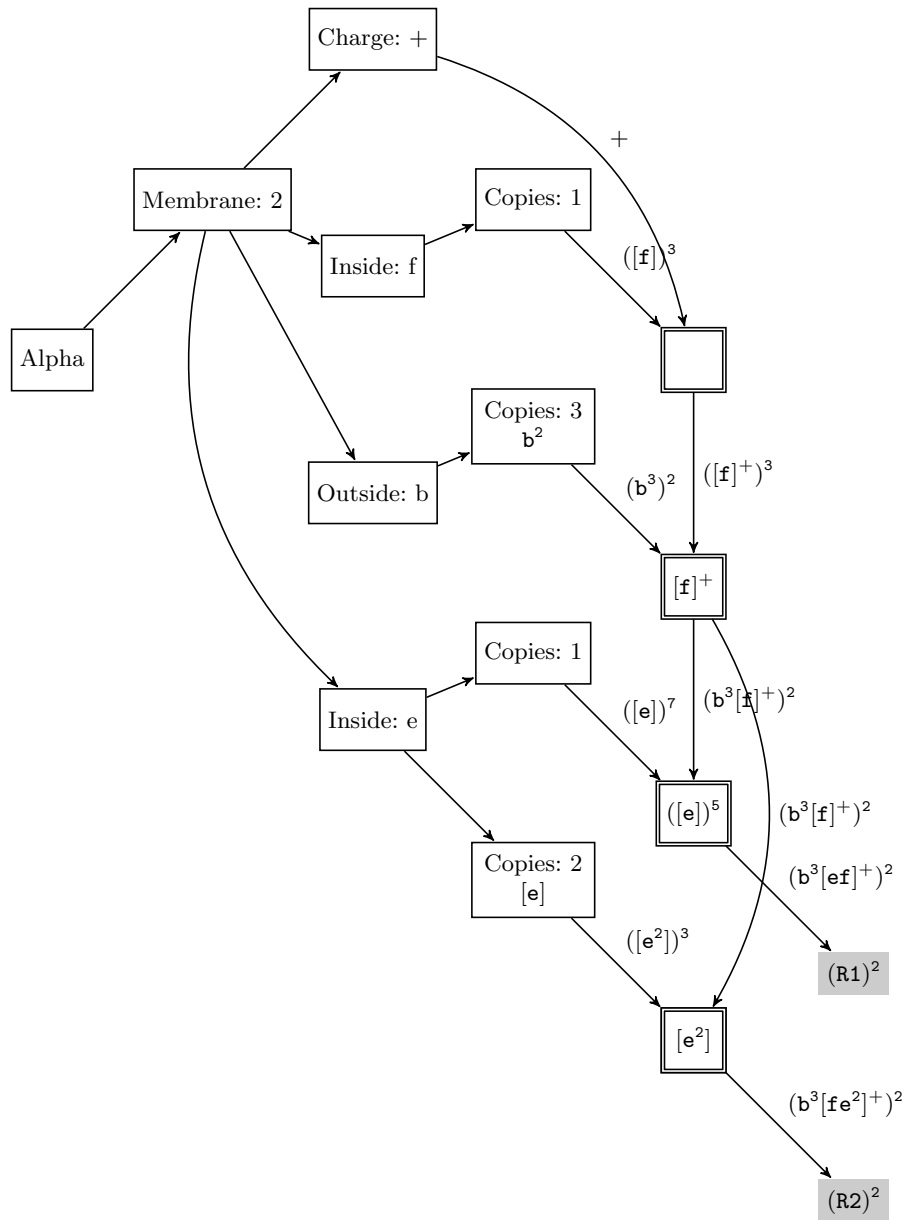
Charge: +

Membrane: 2

Alpha

Inside: f

Copies: 1

+

$([\mathbf{f}])^3$

Copies: 3
$\mathbf{b}^2$

Outside: b

$(\mathbf{b}^3)^2$

$([\mathbf{f}]^+)^3$

$[\mathbf{f}]^+$

Inside: e

Copies: 1

$([\mathbf{e}])^7$

$(\mathbf{b}^3[\mathbf{f}]^+)^2$

Copies: 2
$[\mathbf{e}]$

$(([\mathbf{e}])^5$

$(\mathbf{b}^3[\mathbf{f}]^+)^2$

$(\mathbf{b}^3[\mathbf{ef}]^+)^2$

$([\mathbf{e}^2])^3$

$(\mathbf{R1})^2$

$[\mathbf{e}^2]$

$(\mathbf{b}^3[\mathbf{fe}^2]^+)^2$

$(\mathbf{R2})^2$

**Fig. 3.** Rete Network for P system

```
Create the network associated to the rules of the system
Include in the network objects from the initial configuration
while some halting condition do
    while there is no unmarked rule in the output of the network do
        Select one rule from the output of the network (or a set)
        Eliminate in the network all elements from the LHS of that rule
        Mark the charge if its going to change (and marked any rule of
        the output that change the charge for the same membrane)
        Accumulate the objects that must be included in the system
    end while
    Eliminate marks to rules of the output
    Change in the network the marked charges
    Add all the accumulated objects
end while
```

**Fig. 4.** Generic pseudocode of a simulation algorithm

## 5 Final Remarks

Let us notice some final considerations:

- As there exists a big number of different P systems models (both syntactically and semantically different), it is not possible to melt together all of them to have a single way to construct the network. So, the basic lines shown in this paper should be adapted to each specific model in order to improve the efficiency of the designed simulator.
- One of the key points of the efficiency of the algorithm is the proper order in the conditions of the LHS of the rule and this is a final choice of the designer of the P system. For example, electrical charge is usually used as a controlling condition, but it is the user who decided its role.
- Little syntactic or semantic changes on the model can have drastic influence on the efficiency of the algorithm. As an illustrative example, we can consider two similar models such that in the first one the membranes are injectively labelled and in the second, two different membranes can share the same label. This apparently slight difference requires a major change in the algorithm.

Recent applications of P system techniques to real-world problems (e.g.,[2, 7]) require more and more efficient simulators. In the similar way to other areas in Computer Science, the availability of huge amount of data, together with the iteration of probabilistic process in an attempt of simulating natural processes needs of more and more efficient algorithms.

In this paper we recover a successful algorithm from the Expert System field and propose a first attempt to consider it in the Membrane Computing framework. The implementation is currently under development, and in principle it will be inserted into a simulator within the P-Lingua framework. However, upon completing the implementation, we are convinced that it will be possible to export this

technique into any other P system simulator. The adaptation of the algorithm has been made by considering that the computer where the software runs has only one processor and, in this way, the software simulation of the P systems is made sequentially in an one-processor machine. Nonetheless, new hardware architectures are being used for simulating P systems [3, 4, 5, 6, 15, 16, 17], so the parallel versions of the Rete algorithm [11, 14] and their relations with parallel simulators of P systems should be considered in the future.

## Acknowledgements

## References

1. Apt, K.R.: Logic Programming. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), pp. 493–574. The MIT Press (1990)
2. Cardona, M., Colomer, M.Á., Margalida, A., Palau, A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Sanuy, D.: A computational modeling for real ecosystems based on P systems. Natural Computing 10(1), 39–53 (2011)
3. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Implementing P systems parallelism by means of GPUs. In: Păun, G. *et al.* (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 5957, pp. 227–241. Springer, Berlin Heidelberg (2009)
4. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulating a P system based efficient solution to SAT by using GPUs. Journal of Logic and Algebraic Programming 79(6), 317–325 (2010)
5. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulation of P systems with active membranes on CUDA. Briefings in Bioinformatics 11(3), 313–322 (2010)
6. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Jiménez, M.J., Ujaldon, M.: The GPU on the simulation of cellular computing models. Soft Computing 16(2), 231–246 (2012)
7. Colomer, M.A., Margalida, A., Sanuy, D., Pérez-Jiménez, M.J.: A bio-inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study. Ecological Modelling 222(1), 33 – 47 (2011)
8. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19(1), 17–37 (1982)
9. Forgy, C.L.: On the efficient implementation of production systems. Ph.D. thesis, Department of Computer Science, Pittsburgh, PA, USA (1979)
10. Giarratano, J., Riley, G.: Expert systems: principles and programming. Thomson Course Technology (2005)

11. Gupta, A., Forgy, C., Newell, A., Wedig, R.: Parallel algorithms and architectures for rule-based systems. ACM SIGARCH Computer Architecture News 14(2), 28–37 (1986)
12. Gutiérrez-Naranjo, M.A., Rogojin, V.: Deductive databases and P systems. Computer Science Journal of Moldova 12(1), 80–88 (2004)
13. Ivanov, S., Alhazov, A., Rogojin, V., Gutiérrez-Naranjo, M.A.: Forward and backward chaining with P systems. International Journal on Natural Computing Research 2(2), 56–66 (2011)
14. Kuo, S., Moldovan, D.: The state of the art in parallel production systems. Journal of Parallel and Distributed Computing 15(1), 1 – 26 (1992)
15. Nguyen, V., Kearney, D., Gioiosa, G.: An extensible, maintainable and elegant approach to hardware source code generation in reconfig-P. Journal of Logic and Algebraic Programming 79(6), 383–396 (2010)
16. Peña-Cantillana, F., Díaz-Pernil, D., Berciano, A., Gutiérrez-Naranjo, M.A.: A parallel implementation of the thresholding problem by using tissue-like P systems. In: Real, P. *et al.* (eds.) CAIP (2). Lecture Notes in Computer Science, vol. 6855, pp. 277–284. Springer (2011)
17. Peña-Cantillana, F., Díaz-Pernil, D., Christinal, H.A., Gutiérrez-Naranjo, M.A.: Implementation on CUDA of the smoothing problem with tissue-like P systems. International Journal of Natural Computing Research 2(3), 25–34 (2011)