A Rete-based Algorithm for Rule Selection in P Systems

CARMEN GRACIANI*, MIGUEL Á. GUTIÉRREZ-NARANJO,
IGNACIO PÉREZ-HURTADO, AGUSTÍN RISCOS-NÚÑEZ,
ÁLVARO ROMERO-JIMÉNEZ

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Escuela Técnica Superior de Ingeniería Informática
University of Sevilla
Av/ Reina Mercedes, s/n
41012 Sevilla (Spain)

Received May 27th, 2013; In final form ...

The Rete algorithm is a well-known pattern matching algorithm conceived to make rule-based production system implementations more efficient. It builds a directed acyclic graph, representing higher-level rule sets, that allows the implementation to avoid checking each step the applicability of all the rules. Instead, only those affected by a change in the collection of facts are checked. In this paper we study how the underlying ideas of this algorithm can be adapted to improve the design of computational simulators within the framework of Membrane Computing.

Key words: Rete algorithm, P systems, Membrane Computing

1 INTRODUCTION

When dealing with information, knowledge and reasoning in Computer Science, rule-based representation is one of the most popular choices. Given two

* email: cgdiaz@us.es

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 order to solve such a reasoning problem, there are two basic methods, 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 V. The first method is data-driven and it is known as *forward chaining*, the second one is query-driven and it is called *backward chaining* [1]. A study of these methods within the framework of Membrane Computing can be found in [20, 21].

The piece of information V (the left-hand side of the rule or LHS) is usually split into elementary pieces $v_1, v_2, \ldots, v_n$. The forward chaining derivation of W (the right hand side of the rule or RHS) according to the Modus Ponens via the rule $V \to W$ 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 pseudocode briefly describing the forward chaining method.

What this algorithm essentially does is to check for all rules if they produce new knowledge from the information already present in the set $Deduced$. A naive implementation of the loop from Figure 1 can be achieved performing a sequential pattern matching between rules and elements in the set $Deduced$. This latter set is dynamically increased with new knowledge, so the previously checked rules must be reconsidered by restarting the sequential pattern matching. Such implementation provides an exhaustive checking, but it is extremely inefficient.

Within the framework of Expert Systems [16], a better solution to this problem was proposed by Charles L. Forgy in his Ph.D. dissertation at the Carnegie-Mellon University in 1979 [12, 13]. The solution was called the *Rete** *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.

* *Rete* means net in Latin.

Forward chaining
INPUT: A reasoning problem $\langle A, R, G\rangle$
INITIALISE: $Deduced = A$

if $G \in Deduced$ **then**
  return `true`
end if
for all $(v_1 v_2 \dots v_n \to W) \in R$ such that
    $\{v_1, v_2, \dots, v_n\} \subseteq Deduced \wedge W \notin Deduced$ **do**
  $Deduced \leftarrow Deduced \cup \{W\}$
  if $W = G$ **then**
    return `true`
  end if
end for
return `false`

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

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

In what follows we will use the following template for rules in Membrane Computing

$$u\,[\,v\,]_i^{\alpha} \to u'\,[\,v'\,]_i^{\alpha'}$$

which can be considered as a generalisation of many kinds of rules usually present in different P systems models.

A rule following the mentioned scheme is *applicable* if, in the current configuration, there exists a membrane labelled by i, with polarisation α, such that it includes the multiset v, and its surrounding membrane includes the multiset u. Although the *application* of the rule is different in both paradigms (for the Membrane Computing case, the objects in the LHS are consumed and the objects in the RHS are created; while for 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: In Section 2 we provide a brief overview of production systems and the Rete algorithm. Section 3 shows how this algorithm can be adapted to P system simulators. In Section 4 some experimental results are presented. 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, informally speaking, a formula with no inner structure. Atomic formulas are used to express facts 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 $A = \{a_1, a_2, \ldots, a_m\} \subseteq U$ is the current set of atoms known (or believed) to be true, and R is the set of *production rules*, of the form $V \to W$ with $V, W \subseteq U$.

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

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

The meaning of this rule is as follows: if $P_1 \wedge P_2 \wedge \ldots \wedge P_n \to Q$ is a production rule and $P_1, P_2, \ldots, P_n \in A$, 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 $\{F_1, F_2, \ldots, F_{i-1}\}$.

2.2 Rule-based Expert Systems

Instead of viewing computation as a sequence of operations specified by a program, in production systems computation is seen 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 rule base 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):

> The condition part of each rule (LHS) is tested against the current state.
> If it matches, then the rule is said to be *applicable*.
> From the applicable rules, one of them is chosen to be applied.
> 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. Nevertheless, a common feature for rule-based systems is that the process of finding applicable rules is called repeatedly at each step.

2.3 The Rete Algorithm

The Rete algorithm is a well-known algorithm for efficiently checking the many pattern/many object pattern match problem [12], and it has been widely used mainly in production systems. 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, while the rest (probably, most of them) remain unaltered.

- *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 LHS of 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 LHS of a rule.

Facts flow through the network, and they 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 has four kinds of nodes (see Figure 2):

- **Alpha Root:** (marked with 'Alpha') 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:** (rectangle nodes) perform conditions which depend on just one pattern. If the test succeeds, then the received token passes to the node successors. There are different alpha nodes depending on the considered pattern.

- **Beta Root:** (marked with 'Beta') For each rule present in the network there is a path from beta root through beta nodes ending in a terminal

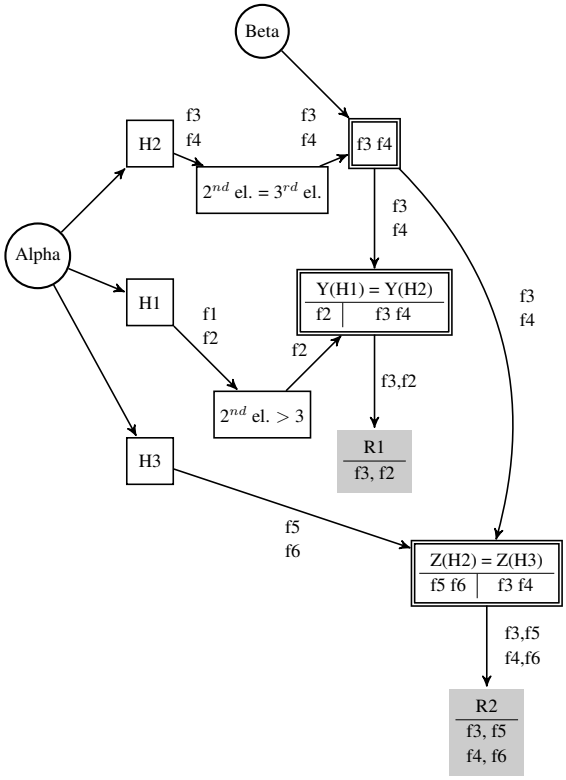

FIGURE 2
Example of a Rete network and tokens flow

node. This path follows rule conditions and interrelate each condition with the next one (as they appear in the rule). Beta nodes can be reused for different rules if their first conditions are the same.

- **Beta nodes:** (double rectangle nodes) perform inter-patterns conditions, for example, if two patterns have a common variable. Each one receives tokens from two nodes and stores the tokens that arrive from each parent in two different slots. When a token arrives through one of the inputs, then the condition is checked against all the tokens in the slot for the other input. For each successfully checked pair, a new token, combining both of them, passes to the node successors. Beta

nodes directly connected to the beta root are special ones; they only have one slot and tokens always pass through them.

- **Terminal nodes:** (grey 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. Notice that the first beta node is common for both rules. The figure also shows how tokens corresponding to different facts flow through the network. The output is that rule `R1` can be fired be the pair of facts `f3, f2` and rule `R2` by any of the pairs of facts `f3, f5` and `f4, f6`.

```
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).
                                Fact: f5  H3(3).
Rule: R2                        Fact: f6  H3(9).
    Exists H2(Y, Z, Z).
    Exists H3(Z).
         => ...
```

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, then 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 these strategies we are trying to minimise (in general) the number of *beta* nodes in the network and, therefore, the number of checks required until a token arrives into a terminal node.

3 MEMBRANE COMPUTING

In this section we explore how the Rete algorithm can be adapted to Membrane Computing simulators. We assume that the reader is familiar with basic concepts related to this area, for an extensive bibliography and documentation please refer to the handbook [27] and the P systems webpage [32].

Since there is no implementation *in vivo* nor *in vitro* of P systems, the development of *in silico* simulators has been one of the most active research lines in the area [10, 19]. In [15], a specification language to define membrane computing systems called P-Lingua has been presented. This language aims to be a standard to define P systems. The P-Lingua framework also includes a Java library called pLinguaCore, which is able to handle input files in P–Lingua format defining P systems from a number of different models [8, 9, 24]. Moreover, the library includes several built–in simulators for each supported model. It is an Open Source software tool available at [33].

In this software tool, the checking of the applicability of the rules is carried out by a sequential process. When the P system is read and analysed from the P-Lingua file, the set of rules is classified according to the label of the associated membrane (if the model uses charges for membranes, then they are also taken into account).

Figure 3 shows part of the "selecting rules" phase if the P system model uses active membranes (the complete simulation algorithm can be found in [30]). Such method only simulates one possible computation, so it is used for confluent P systems (that is, systems for which all the computations with the same input lead to the same result).

It is worth stressing the fact that the Rete-based algorithm that we introduce in this paper is completely independent from the computation mode of the considered model (sequential, maximal/minimal parallelism, etc). Indeed, the Rete network contains information about which rules are "individually" applicable. When calculating applicable multisets of rules, the computation mode comes into play.

For a first approximation to the study of how to use Rete algorithm ideas within Membrane Computing we have chosen to focus on rules handling polarisation. Remember the scheme introduced in Section 1 for such rules

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

A rule of this kind is associated with any membrane with label i. Note that this general template covers, among others, rules used by P systems with active membranes. For example, an evolution rule of the form $[v \rightarrow v']_i^{\alpha}$ can

Selecting rules
INPUT: $C_t =$ `Current configuration`,
$R =$ `P system classified set of rules`
INITIALISE: $R_{sel} = \emptyset$
for all $m \in C_t$ **do**
$h \leftarrow$ label of m; $\alpha \leftarrow$ charge of m; $EvolutionOnly \leftarrow$ FALSE
for all $r \in R$ with label h and charge α **do**
if r has the form $[a \rightarrow b]_h^\alpha$ and a appears in m **then**
$N \leftarrow$ multiplicity of a in m; $R_{sel} \leftarrow R_{sel} \cup \{(r, m, N)\}$
Remove all objects a from m
end if
if $\neg EvolutionOnly$ **then**
if r has the form $a[\,]_h^\alpha \rightarrow [b]_h^\beta$ and a appears in m'
(m' parent of m) in C_t **then**
$R_{sel} \leftarrow R_{sel} \cup \{(r, m, 1)\}$; $EvolutionOnly \leftarrow$ TRUE
Remove one object a from m'
end if
end if
... (rest of considered rule types)
end for
end for

FIGURE 3
Part of the "selecting rules" phase for the simulation algorithm of P systems with active membranes

be expressed as $[v]_i^\alpha \rightarrow [v']_i^\alpha$.

From the previous rule scheme three kinds of conditions can be considered:

- Charge must be α: $[\,]^\alpha$
- Outside there must be at least n copies of element u: u^n
- Inside there must be at least m copies of element v: $[v^m]$

P systems found in the literature often contain rule collections, defined through objects with subscripts (for example $[a_j]_i^\alpha \rightarrow \ldots$ for j in a given set

J). Instead of considering each rule of such collections separately, subscripts conditions have been added to deal directly with them.

Figure 4 shows different alpha nodes derived from the described rule conditions.

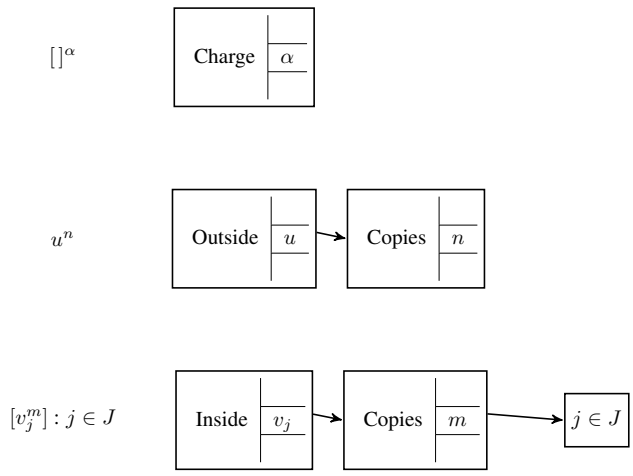

FIGURE 4
Examples of alpha nodes derived from different rule conditions

Following the proposed strategies for production systems, conditions should be sorted in an appropriate order. For example, let us consider the following rules associated with a certain label i:

- (R1) $b^3[ef]^+ \rightarrow \ldots$
- (R2) $b^3[fe^4]^+ \rightarrow \ldots.$

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

- (R1) $[]^+[f]b^3[e] \rightarrow \ldots$
- (R2) $[]^+[f]b^3[e^4] \rightarrow \ldots.$

To complete the example, let us now consider a configuration where there exists a membrane with the same label as the rules, with positive charge, objects $\{f^3, e^7, o^4, x\}$ inside it and objects $\{c, b^8, g^3\}$ in the surrounding membrane. Both rules are applicable to this membrane in this configuration.

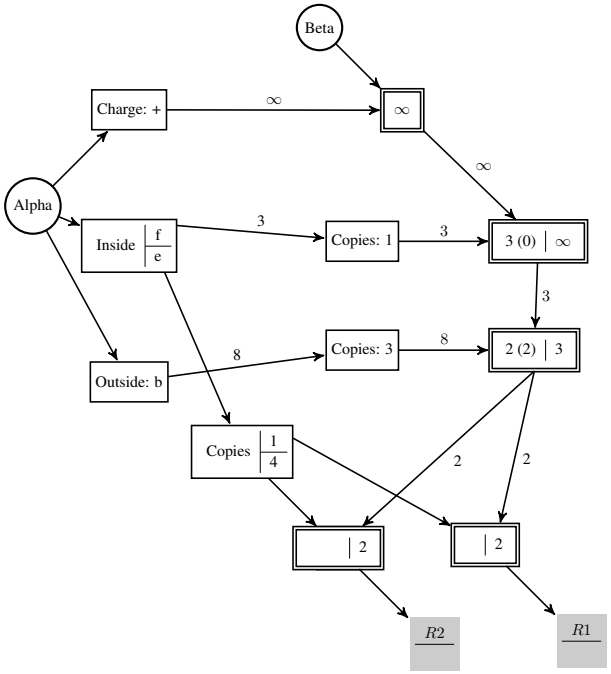

FIGURE 5
A Rete network for (R1) and (R2): Before including e^7

Figures 5 and 6 show the network associated with the rules of this example and how objects of the considered configuration go through different nodes. In the former (Figure 5), part of the configuration is included: membrane has positive charge, objects $\{f^3\}$ are inside and objects $\{b^8\}$ are outside. Objects o and x inside the membrane and objects c and g outside the membrane, have no effects for this example. In the latter (Figure 6), objects e^7 are included inside the membrane. In the network built for the P system, beta nodes contain the following: in their left slot a pair $n(m)$, indicating that, from its parent condition `Copies: k`, there is at most n possible applications (namely, k copies are needed for an application of that condition and $m + (n \cdot k)$ objects are available); in their right slot the maximum number r of possible applications derived from previous conditions. Each time the contents of a beta node is updated, if the minimum between n and r changes, then that amount is transmitted to the right slot of the successor beta node. The symbol ∞ is

used to denote that there is no limit of possible applications from previous conditions. The output of the network in Figure 6 shows that rule (R1) is applicable at most two times and rule (R2) at most one.

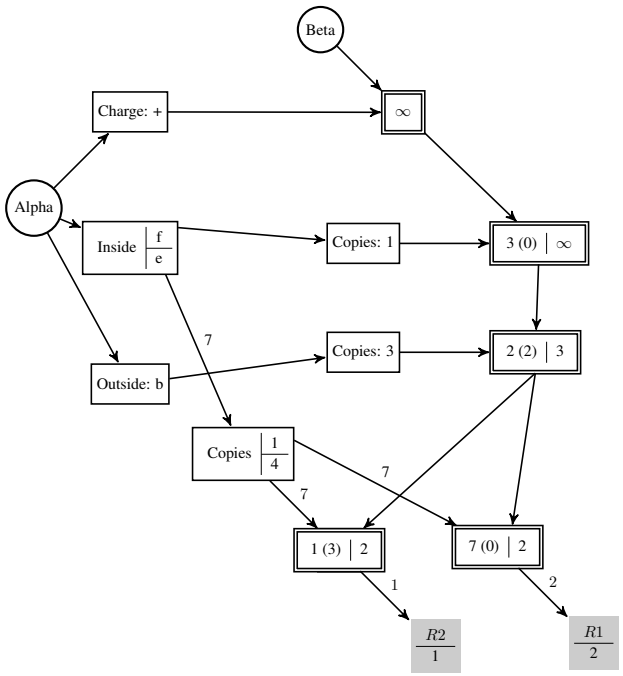

FIGURE 6
A Rete network for (R1) and (R2): After including e^7

Figure 7 shows a generic algorithm to simulate a P system using such networks. There exist several alternative halting conditions that can be used: a prefixed number of repetitions is reached; there is no applicable rule; occurrence of specific objects, etc. Notice that the network reflects how changes in a configuration affect to the usable rules. Once all the changes are considered in the network, its output contains a multiset of rules, all of them applicable for that configuration at most the number of times indicated by their multiplicity. According to the semantics of the model, rules from that multiset are applied to change the configuration. In general there will be more than one possible set of rules that can be selected to be applied.

```
Create the network associated with the rules of the system
Include in the network objects from the initial configuration
while not some halting condition do
  while the output of the network is not empty do
    Select a multiset of rules from the output of the network (according to the semantics of the model)
    Change the configuration applying the selected rules (include and eliminate objects also in the network)
  end while
end while
```

FIGURE 7
Generic pseudocode of a simulation algorithm

4 EXPERIMENTAL RESULTS

In order to check the rule selection based on the Rete algorithm, an experiment has been performed using the well-known *Subset Sum problem* [14]: Given a finite set A, a weight function, $w : A \to \mathbb{N}$ and a constant $k \in \mathbb{N}$, determine whether or not there exists a subset $B \subseteq A$ such that $w(B) = \sum_{b \in B} w(b) = k$.

This problem has already been used as a case study within Membrane Computing (see e.g., [11, 18, 23, 31]). As an illustrative experiment for this paper we have chosen the simulation of the first family of recogniser P systems included in [18] to solve Subset Sum. More precisely, the instance to be solved is a very simple one: $A = \{a_1, a_2\}$, $w(a_1) = 3$, $w(a_2) = 2$ and $k = 4$.

First, the Rete network for the rules of the selected instance has been created. After that, a simulation using the P-Lingua simulator has been performed for the same selected instance obtaining all the configurations and applied rules for each step. To finalise, each obtained configuration has been used as input for the Rete network. The set of applicable rules obtained each time coincides with the set of applied rules given by the P-Lingua simulator. The only difference is in the sending out rules. Such rules, because of the semantics of P systems with active membranes, can be applied only once. For example, for $rule9 : [a_0]^-_e \to []^0_e\#$, if in a membrane with label e and charge $-$ there are N copies of object a_0, then the Rete network will say that

$rule9$ is applicable, at least, N times. Nevertheless, during the simulation, it will be applied only once.

During this simulation process, in order to change from one configuration to the following one, the Rete network does not receive as input the new configuration. Instead the input solely consists on elements (charges and objects) consumed/added by applied rules given by the P-Lingua simulator.

Figure 8 represents part of the Rete network built. Only nodes for some rules with label e are represented, namely the following ones:

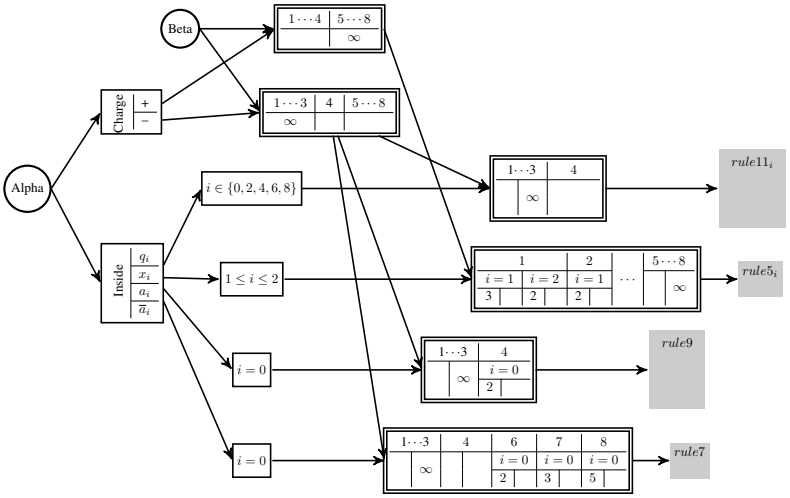

FIGURE 8
Part of Rete network for Subset Sum

$rule5_i : [x_i \rightarrow x_{i-1}]_e^+ : 1 \leq i \leq 2$
$rule7 : [\overline{a}_0 \rightarrow a_0]_e^-$
$rule9 : [a_0]_e^- \rightarrow [\,]_e^0 \#$
$rule11_i : [q_{2*i} \rightarrow q_{2*i+1}]_e^- : 0 \leq i \leq 4$

To perform this experiment beta nodes have been expanded in order to contain a memory for each existing membrane with the same label.

Elements (charges and objects) present in Figures 8 and 9 correspond to the simulation of the step 12. Before this step is performed, in the configuration there are 8 membranes with label e (distinguished by an index from 1 to 8 in beta nodes). One of them (membrane with index 4) has charge 0 and objects $a_0^2\ a^2\ q_5$. In such situation rules $[a]_e^0 \rightarrow [\,]_e^-\ \#$ and $[q_5 \rightarrow q_6]_e^0$ are

applied for that membrane. So objects a and q_5 are consumed, membrane 4 changes its charge, from 0 to $-$, and an object q_6 is added. The differences between Figures 8 and 9 capture the instant in which charge $-$ and object q_6 are included. The output of the network indicates that rules 9 and 11 are now applicable in membrane with index 4.

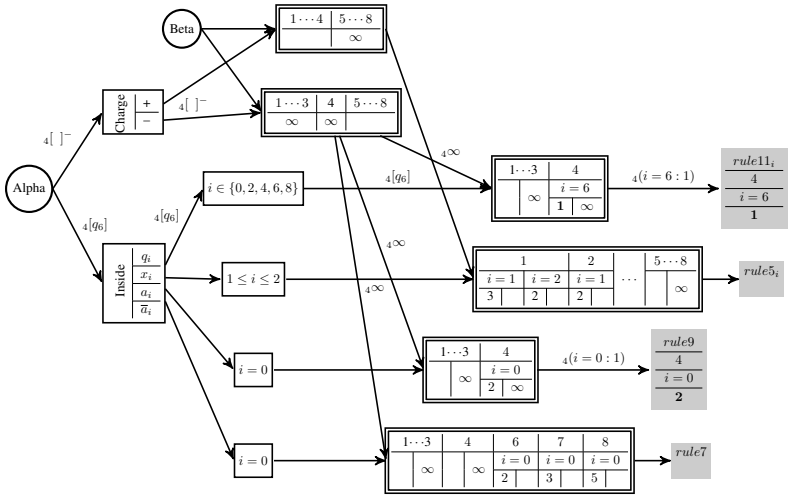

FIGURE 9
Effects of the inclusion of object q_6 and charge $-$ in membrane 4

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 decides its role.

- Little syntactic or semantic changes in 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. This requirement comes, in a similar way as in other areas within Computer Science, from the availability of huge amount of data, together with the iteration of probabilistic methods in an attempt of simulating natural processes.

In this paper, we consider a successful algorithm from the Expert Systems 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 a single-processor machine. Nonetheless, new hardware architectures are being used for simulating P systems [3, 4, 5, 6, 25, 28, 29], so the parallel versions of the Rete algorithm [17, 22] and their relations with parallel simulators of P systems should be considered in the future.

6 ACKNOWLEDGEMENTS

The authors acknowledge the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200 and the project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain, both cofinanced by FEDER funds.

REFERENCES

[1] Krzysztof R. Apt. (1990). Logic Programming. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 493–574. The MIT Press.

[2] Mónica Cardona, M. Ángels Colomer, Antoni Margalida, Antoni Palau, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, and Delfí Sanuy. (2011). A computational modeling for real ecosystems based on P systems. *Natural Computing*, 10(1):39–53.

[3] José M. Cecilia, José M. García, Ginés D. Guerrero, Miguel A. Martínez-del-Amor, Ignacio Pérez-Hurtado, and Mario J. Pérez-Jiménez. (2009). Implementing P systems parallelism by means of GPUs. In Păun *et al.* [26], pages 227–241.

[4] José M. Cecilia, José M. García, Ginés D. Guerrero, Miguel A. Martínez-del-Amor, Ignacio Pérez-Hurtado, and Mario J. Pérez-Jiménez. (2010). Simulating a P system based efficient solution to SAT by using GPUs. *Journal of Logic and Algebraic Programming*, 79(6):317–325.

[5] José M. Cecilia, José M. García, Ginés D. Guerrero, Miguel A. Martínez-del-Amor, Ignacio Pérez-Hurtado, and Mario J. Pérez-Jiménez. (2010). Simulation of P systems with active membranes on CUDA. *Briefings in Bioinformatics*, 11(3):313–322.

[6] José M. Cecilia, José M. García, Ginés D. Guerrero, Miguel A. Martínez-del-Amor, Mario J. Pérez-Jiménez, and Manuel Ujaldon. (2012). The GPU on the simulation of cellular computing models. *Soft Computing*, 16(2):231–246.

[7] M. Ángels Colomer, Antoni Margalida, Delfí Sanuy, and Mario J. Pérez-Jiménez. (2011). 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.

[8] M. Angels Colomer, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. (2012). Comparing simulation algorithms for multienvironment probabilistic P systems over a standard virtual ecosystem. *Natural Computing*, 11(3):369–379.

[9] Miguel A. Martínez del Amor, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. (2010). A P-lingua based simulator for tissue P systems. *Journal of Logic and Algebraic Programming*, 79(6):374–382.

[10] Daniel Díaz-Pernil, Carmen Graciani, Miguel A. Gutiérrez-Naranjo, Ignacio Pérez-Hurtado, and Mario J. Pérez-Jiménez. (2010). Software for P systems. In [27], pages 437–454.

[11] Daniel Díaz-Pernil, Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. (2007). Solving subset sum in linear time by using tissue P systems with cell division. In José Mira and José R. Álvarez, editors, *IWINAC (1)*, volume 4527 of *Lecture Notes in Computer Science*, pages 170–179, Berlin Heidelberg. Springer.

[12] Charles L. Forgy. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37.

[13] Charles L. Forgy. (1979). *On the efficient implementation of production systems.* PhD thesis, Department of Computer Science, Pittsburgh, PA, USA. AAI7919143.

[14] Michael Garey and David S. Johnson. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman & Co Ltd

[15] Manuel García-Quismondo, Rosa Gutiérrez-Escudero, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. (2009). An Overview of P-lingua 2.0. In Păun *et al.* [26], pages 264–288.

[16] Joseph C. Giarratano and Gary D. Riley. (2005). *Expert systems: principles and programming*. Thomson Course Technology.

[17] Anoop Gupta, Charles L. Forgy, Allen Newell, and Robert Wedig. (1986). Parallel algorithms and architectures for rule-based systems. *ACM SIGARCH Computer Architecture News*, 14(2):28–37.

[18] Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. (2004). On descriptive complexity of P systems. In Giancarlo Mauri, Gheorghe Păun, Mario J. Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Workshop on Membrane Computing*, volume 3365 of *Lecture Notes in Computer Science*, pages 320–330. Springer.

[19] Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. (2006). Available membrane computing software. In Gabriel Ciobanu, Mario J. Pérez-Jiménez, and Gheorghe Păun, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 411–436. Springer.

[20] Miguel A. Gutiérrez-Naranjo and Vladimir Rogojin. (2004). Deductive databases and P systems. *Computer Science Journal of Moldova*, 12(1):80–88.

[21] Sergiu Ivanov, Artiom Alhazov, Vladimir Rogojin, and Miguel A. Gutiérrez-Naranjo. (2011). Forward and backward chaining with P systems. *International Journal on Natural Computing Research*, 2(2):56–66.

[22] Steve Kuo and Dan Moldovan. (1992). The state of the art in parallel production systems. *Journal of Parallel and Distributed Computing*, 15(1):1 – 26.

[23] Alberto Leporati, Giancarlo Mauri, Claudio Zandron, Gheorghe Păun, and Mario J. Pérez-Jiménez. (2009). Uniform solutions to SAT and subset sum by spiking neural P systems. *Natural Computing*, 8(4):681–702.

[24] Luis F. Macías-Ramos, Ignacio Pérez-Hurtado, Manuel García-Quismondo, Luis Valencia-Cabrera, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. (2011). A P-lingua based simulator for spiking neural P systems. In Marian Gheorghe, Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Sergey Verlan, editors, *Int. Conf. on Membrane Computing*, volume 7184 of *Lecture Notes in Computer Science*, pages 257–281. Springer.

[25] Van Nguyen, David Kearney, and Gianpaolo Gioiosa. (2010). An extensible, maintainable and elegant approach to hardware source code generation in reconfig-P. *Journal of Logic and Algebraic Programming*, 79(6):383–396.

[26] Gheorghe Păun, Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, Grzegorz Rozenberg, and Arto Salomaa, editors. (2010). *Membrane Computing, 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009. Revised Selected and Invited Papers*, volume 5957 of *Lecture Notes in Computer Science*, Berlin Heidelberg. Springer.

[27] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors (2010). *The Oxford Handbook of Membrane Computing* Oxford University Press, Oxford, England.

[28] Francisco Peña-Cantillana, Daniel Díaz-Pernil, Ainhoa Berciano, and Miguel A. Gutiérrez-Naranjo. (2011). A parallel implementation of the thresholding problem by using tissue-like P systems. In Pedro Real, Daniel Díaz-Pernil, Helena Molina-Abril, Ainhoa Berciano, and Walter G. Kropatsch, editors, *CAIP (2)*, volume 6855 of *Lecture Notes in Computer Science*, pages 277–284. Springer.

[29] Francisco Peña-Cantillana, Daniel Díaz-Pernil, Hepzibah A. Christinal, and Miguel A. Gutiérrez-Naranjo. (2011). Implementation on CUDA of the smoothing problem with tissue-like P systems. *International Journal of Natural Computing Research*, 2(3):25–34.

[30] Ignacio Pérez-Hurtado. (2010). *Desarrollo y aplicaciones de un entorno de programación para Computación Celular: P-Lingua*. PhD thesis, Department of Computer Science and Artificial Intelligence. University of Seville. In Spanish.

[31] Mario J. Pérez-Jiménez and Agustín Riscos-Núñez. (2005). Solving the subset-sum problem by P systems with active membranes. *New Generation Computing*, 23(4):339–356.

[32] The P Systems Webpage: `http://ppage.psystems.eu/`

[33] The P–Lingua Web Site: `http://www.p-lingua.org/wiki`