A New Strategy to Improve the Performance of PDP-Systems Simulators

Carmen Graciani, Miguel A. Martínez-del-Amor, and Agustín Riscos-Núñez

Department of Computer Science and Artificial Intelligence,
Research Group on Natural Computing, Universidad de Sevilla, Seville, Spain
{cgdiaz,mdelamor,ariscosn}@us.es

Abstract. One of the major challenges that current P systems simulators have to deal with is to be as efficient as possible. A P system is syntactically described as a membrane structure delimiting regions where multisets of objects evolve by means of evolution rules. According to that, on each computation step, the applicability of the rules for the current P system configuration must be calculated. In this paper we extend previous works that use Rete-based simulation algorithm in order to improve the time consumed during the checking phase in the selection of rules. A new approach is presented, oriented to the acceleration of Population Dynamics P Systems simulations.

Keywords: Rete algorithm · P systems · Membrane computing · Rule applicability · Simulator performance

1 Introduction

In Membrane Computing it is relatively common to find in the literature designs of P systems where a collection of rules is described by means of a single template (usually using indexed objects). P-Lingua standard allows the definition of rule patterns with parameters, thus getting closer to the usual syntax used in the papers. For example, the following rule pattern represents one thousand evolution rules acting over different objects:

$$[o_i \rightarrow x_i], 1 \leq i \leq 1000.$$

Nevertheless, when designing simulator software, we commonly assume that the rules will be handled individually. For instance, all built-in simulators in the *pLinguaCore* library (even those from *PMCGPU project* [27]) always unwrap any rule pattern within the `.pli` file, and then they load in memory every single rule obtained. Therefore, in the previous example, our simulators would handle those 1000 rules separately. It seems clear that, if we were able to process rule patterns without unwrapping them, then the performance would improve dramatically.

This paper provides a step forward in this direction, proposing an improvement of the first phase of the simulation loop based on a Rete network: checking which rules are applicable (an how many times).

The paper is structured as follows. First, some preliminary notions about production systems and the classical Rete algorithm are recalled. Then, Sect. 3 discusses how to bring a Rete-like approach into the pseudocode of simulation engines used in membrane computing software tools. Some further details are given for the case of PDP systems in Sect. 4. Finally, the paper concludes with some final remarks and future work.

2 Production Systems

A rule-based production system is a model of computation that has been widely used in the field of Artificial Intelligence for a wide range of tasks in many domains. It is classically defined by a set of *rules* (rulebase), a set of *facts* (working memory), and a *rule engine* that controls the execution.

Each rule consists of a conjunction of *condition elements* and a set of *actions*. The general form is

`if` [condition]* `then` [action]*

Usually the condition part is known as the *left-hand side (LHS)* of the rule and each condition is called a *pattern*. The action part is known as the *right-hand side (RHS)* and describe the *effects* of applying the rule.

The rule engine repeatedly performs the operations described in Algorithm 1 until no more rule is applicable (or an action element stops it).

applicable ← ∅;
foreach *rule* **do**
  Test LHS of the rule against the working memory;
  if *it matches* **then**
    add rule to *applicable*
  end
end
Choose one rule from *applicable* (if any);
Perform the actions of the RHS of the selected rule;

Algorithm 1. Match-act cycle

The actions produce, in most cases, the inclusion and/or deletion of facts within the working memory. Because of those changes, some rules may become applicable while, conversely, some other rules may stop being applicable.

It is well known that a large part of the time and memory consumed by a ruled-based production system is due to the matching phase; that is, determining which rules are applicable at any given instant, according to the current facts in the working memory. Thus, the main challenge of match algorithms is to update this information in an efficient way.

2.1 The Rete Algorithm

The Rete algorithm [8] is a classic and widely used algorithm for checking rule satisfaction. It takes advantage of two empirical observations:

- *Temporal redundancy:* The application of the rules does not change all the working memory. Only some facts are affected and the remaining ones (probably, most of them) stay unchanged. Rete maintains state information across cycles and performs incremental matching.
- *Structural similarity:* Several rules can (partially) share the same (or similar) conditions in the LHS. Rete recognises those identical features in order to avoid making the same tests multiple times.

Before the match-act cycles take place, the set of rules is preprocessed yielding a network (a directed acyclic graph). During the match-act cycles, tokens associated with facts flow through this network each time that they are added/deleted to the working memory. At any given point, the contents of the network correspond to the conditions that have already been checked against the current facts.

We will use the set of rules in Fig. 1(a), and the network associated to it (displayed below the set of rules) in order to illustrate the description of the different components of such a network and the process followed to construct it. Figure 1(b) shows how tokens, corresponding to different facts added to the working memory, pass through the network during a match-act cycle.

The network constructed for a given set of rules has two roots and three kinds of nodes:

- **Root** α is the entry to the α-nodes subnetwork. During the match-act cycles this root receives the changes in the working memory (added or removed facts) and pass those tokens to its successors (α-nodes).

  In the figure, root α is represented as a squared node with a symbol α inside.
- **α-nodes**, children of the root α. They are included in the network for each different pattern appearing in any of the LHS of the rules. α-nodes perform the checking for the associated condition to all the tokens they receive. Only when the test is successful, the token passes to the successors (β-nodes).

  In the figure, α-nodes are represented as rectangles, showing their associated condition inside them. For example, in order to match the first condition of rule `R1`, an object must verify `p1` relation, and its first argument must be equal to number `3`, as described in the corresponding α-node. Since in this small example there are only three different patterns in the LHS of `R1` and `R2`, the network contains only three different α-nodes.
- **Root** β is the entry to the β-nodes subnetwork.

  In the figure, root β is the double squared node with a symbol β inside.
- **β-nodes** perform inter–patterns conditions. β-nodes receive tokens from two nodes (an α-node and a β-node) and have two different memories to store the tokens that arrive from each parent. Every time a new token arrives, the condition will be checked for all possible combinations of this token with

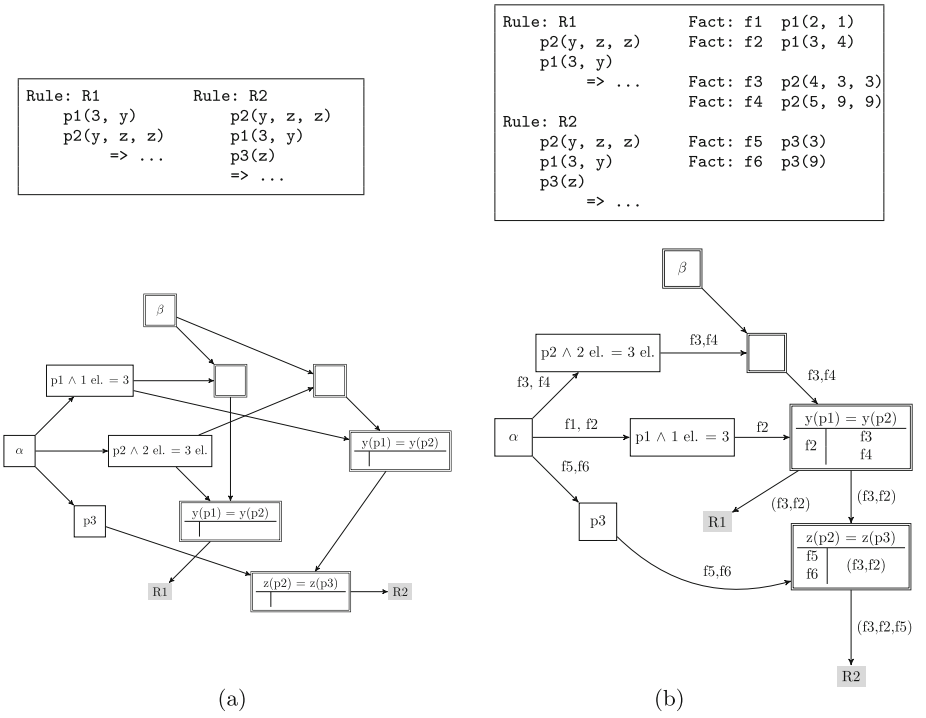


Fig. 1. Two different Rete networks for `R1` and `R2`. The one on the right also illustrates the flow of tokens for a given working memory `f1`, ..., `f6`.

tokens from the local memory of the other parent. Successful combinations (if any) are passed on to successor nodes (either terminal or β-nodes).

β-nodes directly connected to the root β are a particular case; they do not use any local memory, they just let tokens pass trough them.

In the figure, β-nodes are the double squared ones where the associated inter–pattern condition is displayed inside them. Below the condition there are two cells, where the tokens stored in each local memory are shown.

For the given example, the first non-elementary β-node is associated to the following condition: the second argument of the token verifying the `p1` relation must be the same as the first argument of the token that verifies the `p2` relation (we denote this as $\mathtt{y(p1)} = \mathtt{y(p2)}$).

– **Terminal-nodes** receive tokens which match all the conditions of the LHS of a rule (including inter–patterns conditions), and produce the output of the network. The set of applicable rules is composed by the rules whose terminal-node are not empty.

In the figure, terminal-nodes are the grey ones.

The path from the root β, through different β-nodes, down to a terminal-node defines the complete LHS of a rule. Unless otherwise indicated, inter–pattern conditions are checked in the same order as they occur in the rule.

Note that a very simple change in the order of `R1` conditions yields a very different network, as shown in Fig. 1(b). Now, the set of rules not only share conditions `p2(y, z, z)` and `p1(3,y)`, but also that they occur at the beginning of the LHS, and moreover in the same order. Therefore, they continue sharing the α-nodes for those conditions (like in Fig. 1(a)), but now they also share the first β-nodes.

Algorithm 2 describes the general process that constructs the network for a given set of rules.

NET $\leftarrow$ graph ($\{\alpha, \beta\}, \emptyset$);
foreach *rule, R* **do**
  $C \leftarrow$ first condition of R;
  $A \leftarrow$ α-node in NET associated to C;
  `// if it does not exist, then add it to NET as an` α `child`
  $B \leftarrow$ β-node in NET child of A and β;
  `// if it does not exist, then add it to NET`
  foreach *condition D in R (in order of occurrence after C)* **do**
    $A \leftarrow$ α-node in NET associated to D;
    `// if it does not exist, then add it to NET as an` α `child`
    $B \leftarrow$ β-node in NET, child of A and B, associated to inter-pattern condition between D and previous conditions in R;
    `// if it does not exist, then add it to NET`
  end
  $T \leftarrow$ terminal-node in NET, child of B;
  `// if it does not exist, then add it to NET as child of` B
  Add R to T memory;
end

Algorithm 2. Network construction

The most important issue regarding performance is the order of the conditions in the LHS of the rules. This leads to consider the following strategies 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.

Those strategies try to minimise (in general), not only the number of β-nodes that will exist in the network (and, therefore, the number of checks performed until a token arrives into a terminal node), but also the number of β-nodes that must be updated each time that a fact flows through their memories.

In resume, the key advantage of Rete is that rule conditions are only re-evaluated when a fact is asserted or deleted. In this way, asserting a new fact is

simply a case of passing a token through the network, and a smaller number of matching operations are performed. In a naive implementation, each new fact would be compared against every single pattern of every rule, which means a greater time complexity. Retracting a fact is identical to assertion, but items are removed from node memories.

3 Rete and P System Simulation

In this section we explore how the Rete algorithm and the strategies described in the previous section 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 [23] and the P systems webpage [25].

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 [7,12]. In [9], a specification language for membrane 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 parse (plain-text) files in P-Lingua format defining P systems from a number of different models [6,14,18], checking whether they contain any syntax or semantics errors. P-Lingua files can also be exported into xml or binary formats, so that the converted files can then be used as the input for simulation tools. Moreover, the library includes several built-in simulators for each supported model. It is an Open Source software tool available at [26].

We will now discuss about the functioning of such simulator engines provided by *pLinguaCore*. After parsing the P system defined in the input P-Lingua (`.pli`) file, the simulation process of each computation step is carried out in two phases: selection and execution of rules. In the first phase, the checking of the applicability of the rules is made sequentially. 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).

Checking the applicability of rules normally consumes plenty of time in *pLinguaCore* simulators, and in fact, it is mainly in this checking subroutine where the complexity of the simulation algorithm resides. For P systems where the rules have an associated probability, there is an additional difficulty: deciding how to implement the semantics, which informally indicate that rules should be applied in a "maximally parallel way, according to their probabilities". In particular, *pLinguaCore* includes a variety of simulation algorithms for PDP systems: *Binomial Block Based* (BBB) algorithm [1] does a random loop over blocks of rules (i.e. rules having the same LHS), and assigns a maximal number of applications to each one; *Direct Non Deterministic distribution with Probabilities* (DNDP) algorithm [6] does also a random loop, but over the rules, and assigning a probabilistic number of applications; and *Direct distribution based on Consistent Blocks Algorithm* (DCBA) [16] performs a proportional distribution of objects among blocks of rules before assigning a maximal, but probabilistic number of

applications to each rule. The main difference among these three approaches is not their performance, but the fact that they produce significantly different behaviours. DCBA is the one that tries to perform the selection of rules to be applied in a more realistic or accurate way, from an ecological point of view. Since it is the most common choice for PDP systems simulation, in what follows we will focus particularly on it.

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 P-system model (sequential, maximal/minimal parallelism, distributed, etc.). Indeed, the Rete network contains information about which rules are "individually" applicable. When calculating applicable multisets of rules, the computation model 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, which can be written in the following form

$$\mathtt{u}_1^{\mathtt{n}_1} \cdots \mathtt{u}_\mathtt{k}^{\mathtt{n}_\mathtt{k}} [\mathtt{v}_1^{\mathtt{m}_1} \cdots \mathtt{v}_\mathtt{l}^{\mathtt{m}_1}]_\mathtt{s}^\mathtt{c} \to \dots$$

($\mathtt{k}$ and/or $\mathtt{l}$ can be 0) with $\mathtt{u}_1, \dots, \mathtt{u}_\mathtt{k}, \mathtt{v}_1, \dots, \mathtt{v}_\mathtt{l} \in \Gamma$.

Also, on many occasions the symbols of the alphabet have subscripts (generally numerical) used to describe rulesets. In general, the following possibilities occur:

- Subscripts belong to a fixed set of possibilities:

$$\mathtt{u}_\mathtt{i} [\mathtt{v}_\mathtt{j}]_\mathtt{s}^\mathtt{c} \to \dots \quad \text{such that } \mathtt{1} \leq \mathtt{i}, \mathtt{j} \leq \mathtt{10}$$

- The value of a subscript of an object is determined by other subscripts values:

$$\mathtt{u}_{\mathtt{i},\mathtt{i}+\mathtt{1}} []_\mathtt{s}^\mathtt{c} \to \dots \quad \text{such that } \mathtt{1} \leq \mathtt{i} \leq \mathtt{10}$$

- Subscripts from different symbols may also be related:

$$\mathtt{u}_{\mathtt{i},\mathtt{j}} [\mathtt{v}_{\mathtt{i}+\mathtt{1},\mathtt{j}-\mathtt{1}}]_\mathtt{s}^\mathtt{c} \to \dots \quad \text{such that } \mathtt{1} \leq \mathtt{i}, \mathtt{j} \leq \mathtt{10}$$

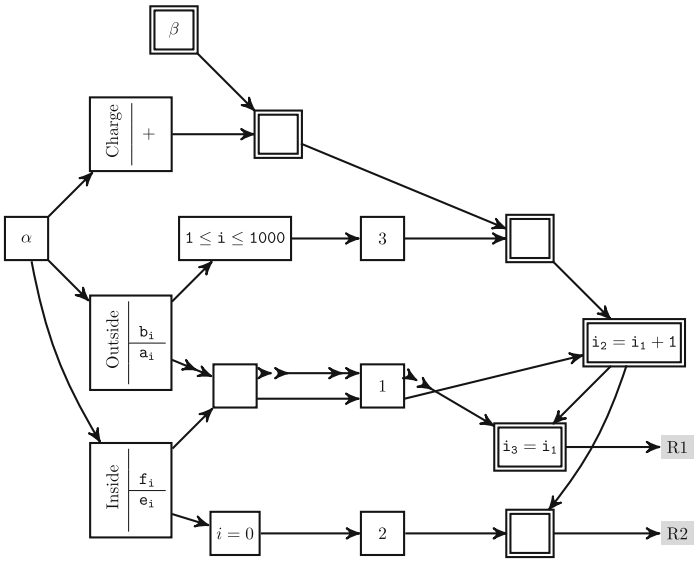
Generalising the above, rulesets schemes would be something of the form:

$$\mathtt{u}_{\mathtt{j}_1:\Gamma_1}^{\mathtt{n}_1} \dots \mathtt{u}_{\mathtt{j}_\mathtt{k}:\Gamma_\mathtt{k}}^{\mathtt{n}_\mathtt{k}} : \gamma_\mathtt{k} [\mathtt{v}_{\mathtt{i}_1:\theta_1}^{\mathtt{m}_1} : \theta_1 \dots \mathtt{v}_{\mathtt{i}_{\mathtt{k}'}:\theta_{\mathtt{k}'}}^{\mathtt{m}_{\mathtt{k}'}} : \theta_{\mathtt{k}'}]_\mathtt{s}^\mathtt{c} \to \dots$$

where each Γ is a relation over the symbol subscripts and each γ is a relation over the symbol subscripts and all the subscripts of the previous symbols. In such a scheme we can distinguish three kinds of conditions:

- A membrane labelled with $\mathtt{s}$ must have charge $\mathtt{c}$: $[]_\mathtt{s}^\mathtt{c}$
- Outside the membrane there must be at least $\mathtt{n}_\mathtt{j}$ copies of element $\mathtt{u}_\mathtt{j}$ and its subscripts must verify $\Gamma_\mathtt{j}$: $\mathtt{u}_{\mathtt{j}:\Gamma_\mathtt{j}}^{\mathtt{n}_\mathtt{j}} : \gamma_\mathtt{j}$.

  This condition is interrelated with the previous one as the membrane has to be the same as in the previous conditions. Also, the object subscripts are related in terms of $\gamma_\mathtt{j}$ with the subscripts of the objects in the former conditions.



R1. $[]_2^+ \mathtt{b}^3_{\mathtt{i}_1:1\leq \mathtt{i}_1\leq 1000}[\mathtt{f}_{\mathtt{i}_2} : \mathtt{i}_2 = \mathtt{i}_1 + 1]\mathtt{a}_{\mathtt{i}_3} : \mathtt{i}_3 = \mathtt{i}_1 \rightarrow \ldots$
R2. $[]_2^+ \mathtt{b}^3_{\mathtt{i}_1:1\leq \mathtt{i}_1\leq 1000}[\mathtt{f}_{\mathtt{i}_2} : \mathtt{i}_2 = \mathtt{i}_1 + 1][\mathtt{e}^2_{\mathtt{i}_3:\mathtt{i}_3=0}] \rightarrow \ldots$

Fig. 2. Rete network for P systems

– Inside the membrane there must be at least $\mathtt{m}_\mathtt{i}$ copies of element $\mathtt{v}_\mathtt{i}$ and its subscripts must verify $\Theta_\mathtt{i}$: $[\mathtt{v}^{\mathtt{m}_\mathtt{i}}_{\mathtt{i}:\Theta_\mathtt{i}} : \theta_\mathtt{i}]$.

Equivalent interrelation to those listed above have to be taken into account.

So, for each symbol in a given P system configuration, the token passing trough the network must contain information about the number of copies and its subscripts. As the symbol is, at the same time, inside a membrane and, probably, outside of one or more membranes a different token is sent for each symbol situation.

Following the work introduced in [10], α-nodes can be divided into several nodes, one for each condition over the arguments of a pattern. Moreover, we consider here a strategy to reduce the amount of redundant information in the network. We will allow these new detailed α-nodes to be used by several patterns, in such a way that each pattern will not be associated to a single α-node, but to a path from root α to a β-node.

For example, let us consider the following rules

1. $\mathtt{a}_\mathtt{i}\mathtt{b}^3_\mathtt{i}[\mathtt{f}_{\mathtt{i}+1}]^+_2 \rightarrow \ldots$ for $1 \leq \mathtt{i} \leq 1000$
2. $\mathtt{b}^3_\mathtt{i}[\mathtt{f}_{\mathtt{i}+1}\mathtt{e}^2_0]^+_2 \rightarrow \ldots$ for $1 \leq \mathtt{i} \leq 1000$

It is important to highlight that this strategy allows us to handle only two templates of LHS, instead two thousand different (but similar) individual rules.

First of all, we can rewrite them as in Sect. 2.1, in order to put at the beginning common conditions. Figure 2 shows the new syntax, together with the constructed network. Note that there are specific detailed α-nodes for conditions about the membrane charge, about the region where the objects should be, about the index of the objects, and about their multiplicity.

Considering that, in a given configuration, several membranes may have the same label, all β-nodes (including those directly connected to root β) and terminal-nodes have different slots to distinguish between them.

In production systems, the changes in the working memory correspond mainly to adding or removing facts. In membrane computing, the modifications caused by the application of the rules mainly refer to polarization of the membranes or their associated multisets of objects. Each time that something is modified on a configuration, the corresponding tokens go trough the network, and in β-nodes the inter–relations between them are checked. Moreover, the checking does not yield a Boolean answer, but instead, the maximum number of times that the related rules could be used is updated.

4 Population Dynamics P Systems and DCBA Algorithm

Population Dynamics P systems are a variant of multienvironment P systems with extended active membranes [5]. As discussed before, the simulation of PDP systems has been a research topic for years. In total, up to 4 simulation algorithms were defined, each trying to improve both in accuracy and in performance for their predecessor. The latest defined algorithm is called DCBA [16], which implements a proportional distribution of objects among rules with overlapping LHS (i.e. competing for objects). Rules having the same LHS are arranged into blocks, and these are also restricted to the consistency condition: rules within a block must have the same LHS and the same charge in the RHS [16].

DCBA consists of 3 phases for the selection of rules: phase 1 (distribution), phase 2 (maximality) and phase 3 (probabilistic). The general scheme is the following:

1. Initialization of the algorithm: *static distribution table* (**columns:** blocks, **Rows:** (objects,membrane))
2. **Loop over Time**
3. **Selection** stage:
4. **Phase 1** (Distribution of objects along rule blocks)
5. **Phase 2** (Maximality selection of rule blocks)
6. **Phase 3** (Probabilistic distribution, blocks to rules)
7. **Execution** stage

As analysed in [17], Phase 1 is the bottleneck of the simulation in sequential mode, taking more than the half of the run time. Whereas Phase 2 performs a random loop over remaining blocks of rules to achieve maximality, and Phase 3 carries out a random multinomial distribution from blocks to rules, Phase 1 has to deal with all the defined blocks of rules, and distribute the objects among

foreach *environment* $e_j, 1 \leq j \leq m$ **do**
  Apply filters 1 and 2 to $\mathcal{T}_j$ using configuration C_t, obtaining the dynamic table $\mathcal{T}'_j$;
  Check *mutual consistency* for the blocks remaining in $\mathcal{T}'_j$. Launch an error if at least one inconsistency is found. Optionally, select a maximal subset of consistent blocks, and continue;
  Apply filter 3 to $\mathcal{T}'_j$ (delete empty rows);
  repeat
    Add all non-null values in the rows of $\mathcal{T}'_j$;
    Normalize the values of $\mathcal{T}'_j$ by using the total sum of rows;
    Multiply each row by the number of copies of the corresponding object in C_t;
    Calculate the minimum of the previous values per column;
    Select the block corresponding to the column with that minimum value;
    Delete the number of copies of the objects in the LHS according to that selection;
    Apply filters 2 and 3 to $\mathcal{T}'_j$;
  until *(Reached a maximum number of iterations)* $\vee$ *(All the column minimums are* 0*)*;
end

Algorithm 3. DCBA selection (Phase 1)

them. Algorithm 3 shows a brief overview of Phase 1 (more details can be seen in [16]).

Essentially, the proportional distribution of objects is carried out by using a table which implements the relationship between rules and their LHS as follows: each column corresponds to each rule block, each row to a pair (object, membrane), and the value in position (i, j) is $1/k$, if the object of row i appears k times in the LHS of block of j, or 0 otherwise. The algorithm always starts with a static table, that will be the same for each transition step. The checking of applicability of rules is carried by applying two filters to the static table, and generating a dynamic table in turn. Depending on the current configuration of the PDP system, the table is dynamically modified by deleting columns related to non-applicable blocks: due to the charge associated to the membrane in the LHS (filter 1), and due to the availability of objects in the LHS according to the configuration (filter 2).

Finally, there is a further restriction within phase 1: if two non-consistent blocks (having different associated right-hand charge) can be selected at the same time given a configuration, then the simulation algorithm will return an error, or optionally non-deterministically choose a subset of consistent blocks.

Evolution rules in PDP systems follow the scheme presented in previous section. Moreover, each environment contains a P system. Since they do not share objects directly, a separate Rete *evolution network* for each P system can be considered.

foreach *environment* $e_j, 1 \leq j \leq m$ **do**
  repeat
    Add all non-null values in the rows of $\mathcal{T}_j$;
    Normalize the values of $\mathcal{T}_j$ by using the total sum of rows;
    Multiply each row by the number of copies of the corresponding object in C_t;
    Calculate the minimum of the previous values per column;
    Select the block corresponding to the column with that minimum value;
    Delete the number of copies of the objects in the LHS according to that selection and send the corresponding tokens to the networks;
  until *(Reached a maximum number of iterations)* $\vee$ *(All the column minimums are* 0*)*;
end

Algorithm 4. Phase 1 reduction due to the use of Rete networks

Environments can send (receive) objects to (from) other environments, by means of a set of communication rules of the following form.

$$(x)_{e_j} \rightarrow (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$$

A new network for the set of communication rules has to be constructed, but this is quite simple as their LHS include just one single condition, the existence of an object in an specific environment.

These *communitation network* must be synchronised with the evolution networks for an accurate simulation. When the initial configuration is included, for each object in the environments, a token is sent to the evolution network associated to that environment and to the communication network. If during a computation step of a simulation, an evolution rule of any P system sends out to its environment an object, then a token removing it is sent to the corresponding evolution network and, also, a token adding it to the corresponding environment has to be sent to the communication network. Moreover, when a communication rule is used during a computation step, in addition to tokens sent trough the communication network, a token has to be sent to the evolution network associated to each receiving P system.

As mentioned before, each time that a token passes through the network the maximum number of times that any rule affected by this change in a configuration is updated. With this information, $\mathcal{T}_j$ is dynamically updated and there is no need to use an initial static distribution table (step 1 in DCBA is replaced by the construction of the networks). Indeed, it would not be necessary to apply any filter to $\mathcal{T}_j$. This updating includes checking mutual consistency launching an error if an inconsistency is found. Algorithm 4 briefly describes new Phase 1.

5 Conclusions and Future Work

In this paper we have presented how to use Rete-based checking for applicability to improve the time consuming by DCBA. For further work new simulators

have to be added to *pLinguaCore* (and also to *PMCGPU project*), not unwrapping rules and constructing Rete-based networks instead, and adapting selection phase. The basic lines shown should be adapted to each specific model in order to improve the efficiency of the designed simulator.

As is well known, one of the key points of the efficiency of the Rete algorithm is the proper order in the conditions of the LHS of the rule. On the other hand, one of its disadvantages is the memory consumed by β-nodes, what has led to modified algorithms for production systems as [19] and the more recent Rete* [24]. It will be interesting to study the impact of this drawback within Membrane Computing framework. In order to test the performance, it is desirable to work on a battery of examples as diverse and demanding as possible (e.g. in [17] a random generator of systems was used to stress the simulators).

On the other hand, the adaptation of the Rete 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 one-processor machine. Nonetheless, new hardware architectures are being used for simulating P systems [2–4,15,17,20–22], so the parallel versions of the Rete algorithm [11,13] and their relations with parallel simulators of P systems will be considered in the future.

Acknowledgements. The authors acknowledge the support of the project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain, cofinanced by FEDER funds.

References

1. Cardona, M., Colomer, M.A., Margalida, A., Palau, A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Sanuy, D.: A computational modeling for real ecosystems based on P systems. Nat. Comput. **10**(1), 39–53 (2011). Springer, Netherlands
2. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del Amor, M.Á., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulating a P system based efficient solution to SAT by using GPUs. J. Log. Algebr. Program. **79**(6), 317–325 (2010). Membrane computing and programming
3. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.Á., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulation of P systems with active membranes on CUDA. Briefings Bioinformatics **11**(3), 313–322 (2010)
4. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del Amor, M.Á., Pérez-Jiménez, M.J., Ujaldón, M.: The GPU on the simulation of cellular computing models. Soft Comput. **16**(2), 231–246 (2012)
5. Colomer, M.A., Martínez-del Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A uniform framework for modeling based on P systems. In: 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), pp. 616–621, September 2010
6. Colomer, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Comparing simulation algorithms for multienvironment probabilistic P systems over a standard virtual ecosystem. Nat. Comput. **11**(3), 369–379 (2012)

7. Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Software for P systems. In: [23], chap. 17, pp. 437–454. Oxford University Press Inc. (2010)
8. Forgy, C.L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem Expert Systems. IEEE Computer Society Press, Los Alamitos (1990)
9. García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: An overview of P-lingua 2.0. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 264–288. Springer, Heidelberg (2010)
10. Graciani, C., Gutiérrez-Naranjo, M.Á., Pérez-Hurtado, I., Riscos-Núñez, A., Romero-Jiménez, Á.: A Rete-based algorithm for rule selection in P systems. Int. J. Unconventional Comput. **9**(5–6), 367–384 (2013)
11. Gupta, A., Forgy, C., Newell, A., Wedig, R.: Parallel algorithms and architectures for rule-based systems. SIGARCH Comput. Archit. News **14**(2), 28–37 (1986)
12. Gutiérrez-Naranjo, M.Á., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Available membrane computing software. In: Ciobanu, G., et al. (eds.) Applications of Membrane Computing. Natural Computing Series, pp. 411–436. Springer, Heidelberg (2006)
13. Kuo, S., Moldovan, D.: The state of the art in parallel production systems. J. Parallel Distrib. Comput. **15**(1), 1–26 (1992)
14. Macías–Ramos, L.F., Pérez–Hurtado, I., García–Quismondo, M., Valencia–Cabrera, L., Pérez–Jiménez, M.J., Riscos–Núñez, A.: A P–Lingua based simulator for spiking neural P systems. In: Gheorghe, M., Păun, Gh., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) CMC 2011. LNCS, vol. 7184, pp. 257–281. Springer, Heidelberg (2012)
15. Martínez-del Amor, M.A., García-Quismondo, M., Macías-Ramos, L.F., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Simulating P systems on GPU devices: a survey. Fundamenta Informaticae **136**, 269–284 (2015)
16. Martínez-del-Amor, M.A., et al.: DCBA: simulating population dynamics P systems with proportional object distribution. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) CMC 2012. LNCS, vol. 7762, pp. 257–276. Springer, Heidelberg (2013)
17. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Gastalver-Rubio, A., Elster, A.C., Pérez-Jiménez, M.J.: Population dynamics P systems on CUDA. In: Gilbert, D., Heiner, M. (eds.) CMSB 2012. LNCS, vol. 7605, pp. 247–266. Springer, Heidelberg (2012)
18. Martínez-del Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-Lingua based simulator for tissue P systems. J. Log. Algebr. Program. **79**(6), 374–382 (2010)
19. Miranker, D.P.: TREAT: a better match algorithm for AI production systems. In: Proceedings of the National Conference on Artificial Intelligence, pp. 42–47. American Association for Artificial Intelligence, August 1987
20. Nguyen, V., Kearney, D., Gioiosa, G.: An extensible, maintainable and elegant approach to hardware source code generation in reconfig-P. J. Log. Algebr. Program. **79**(6), 383–396 (2010)
21. 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., Diaz-Pernil, D., Molina-Abril, H., Berciano, A., Kropatsch, W. (eds.) CAIP 2011, Part II. LNCS, vol. 6855, pp. 277–284. Springer, Heidelberg (2011)

22. 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. Int. J. Nat. Comput. Res. **2**(3), 25–34 (2011)
23. Păun, G., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press Inc., New York (2010)
24. Wright, I., Marshall, J.: The execution kernel of RC++: RETE*: a faster RETE with TREAT as a special case. Int. J. Intell. Games Simul. **2**(1), 36–48 (2003)
25. The P systems webpage. http://ppage.psystems.eu/
26. The P–Lingua web site. http://www.p-lingua.org/wiki
27. The PMCGPU project site. http://sourceforge.net/projects/pmcgpu/