Forward and Backward Chaining with P Systems

Sergiu Ivanov[1,2], Artiom Alhazov[1,3], Vladimir Rogojin[1,4],
Miguel A. Gutiérrez-Naranjo[5]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E-mail: {sivanov,artiom}@math.md
[2] Technical University of Moldova, Faculty of Computers,
Informatics and Microelectronics,
Ştefan cel Mare 168, Chişinău MD-2004 Moldova
[3] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
[4] Research Programs Unit, Genome-Scale Biology,
Faculty of Medicine, Helsinki University,
Biomedicum, Haartmaninkatu 8, Helsinki 00014, Finland
E-mail: vladimir.rogojin@helsinki.fi
[5] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mail: magutier@us.es

Summary. On the one hand, one of the concepts which lies at the basis of membrane computing is the multiset rewriting rule. On the other hand, the paradigm of rules is profusely used in computer science for representing and dealing with knowledge. Therefore, it makes much scene to establish a "bridge" between these domains, for instance, by designing P systems reproducing forward and backward chaining which can be used as tools for reasoning in propositional logic. Our work shows again, how powerful and intuitive the formalism of membrane computing is and how it can be used to represent concepts and notions from totally unrelated areas.

1 Introduction

The use of rules is one of the most common paradigms in computer science for dealing with knowledge. 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. This representation is universal in science. For example, in chemistry, V and W can be metabolites and $V \to W$ a chemical reaction. In this case,

V represents the reactants which are consumed in the reaction and W is the obtained product. In ecology, W may represent the population obtained from the set of individuals V after a time unit. In computer science, V and W are pieces of information (usually split into unit pieces $v_1, v_2, \ldots, v_n$ and $w_1, w_2, \ldots, w_m$) and the rule $V \to W$ is the representation of the precedence relation between V and W. In propositional logic, $V \to W$ is a representation of the clause $\neg v_1 \neg v_2 \vee \cdots \vee \neg v_n \vee w_1 \vee w_2 \vee \cdots \vee w_m$.

Besides representing knowledge using this paradigm, scientists are interested in the derivation of new knowledge from a known piece of information: given a knowledge base $KB = (A, R)$, where A is a set of known atoms and a set R of rules of type $V \to W$, the problem is to know if a new atom g can be obtained from the known atoms and rules. We will call this problem a *reasoning problem* and it will be denoted by $\langle A, R, g \rangle$.

In computer science, there are two basic method for seeking a solution of a reasoning problem, both of them based on the inference rule know as Generalized Modus Ponens: the former is data-driven and it is known as *forward chaining*, the latter is query-driven and it is called *backward chaining*.

In this paper we will consider knowledge bases on propositional logic and prove that both types of chaining can be simulated by P systems. In this way, given a reasoning problem we present several methods for building P systems Π_b and Π_f which produce the objects YES or NO if and only if the corresponding chaining method gives a positive or negative answer.

As one should observe, even though logic inference rules and multiset rewriting rules originate from totaly different areas of mathematics and computer science and represent unrelated notions, their concepts have some similarities. In particular, no information about the ordering of elements in both left and right sides of the rules of both types is used. On the other hand, the inference rules could be thought of as set rewriting rules, while multiset rewriting rules operate at multisets. However, multiset rewriting rules could be interpreted as set rewriting rules if one ignores the multiplicity of elements of the multiset. Therefore we could represent sets of facts in P systems as multisets of objects and inference rules as multiset rewriting rules. When one considers the set of facts represented in a region of a P systems, one only considers the underlying set of the region's multiset.

The paper is organized as follows. First we recall some basic definitions related to the reasoning problem. Then we present our constructions and prove that the obtained P systems produce the same answer as the corresponding chainings. Finally, some conclusions and open research directions are proposed.

2 Definitions

2.1 Transitional P Systems

A *transitional* membrane system is defined by a tuple

$\Pi = (O, \mu, w_1, w_2, \cdots, w_m, R_1, R_2, \ldots, R_m, i_0)$, where

O is a finite set of objects,

μ is a hierarchical structure of m membranes, bijectively labeled by $1, \ldots, m$; the interior of each membrane defines a region; the environment is referred to as region 0,

w_i is the initial multiset in region $i, 1 \leq i \leq m$,

R_i is the set of rules of region $i, 1 \leq i \leq m$,

i_0 is the output region; in this paper i_0 is the skin and could be omitted.

The rules of a membrane systems have the form $u \to v$, where $u \in O^+$, $v \in (O \times Tar)^*$. The target indications from $Tar = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$ are written as a subscript, and target *here* is typically omitted. In case of non-cooperative rules, $u \in O$. In this paper we will not consider target indications.

The rules are applied in a maximally parallel way: no further rule should be applicable to the idle objects. In case of non-cooperative systems, the concept of maximal parallelism is the same as evolution in L systems: all objects evolve by the associated rules in the corresponding regions (except objects a in regions i such that R_i does not contain any rule $a \to u$, but these objects do not contribute to the result). The choice of rules is non-deterministic.

A sequence of transitions is called a computation. The computation halts when such a configuration is reached that no rules are applicable. Since in this paper we will focus on deciding P systems, we are only interested in the presence of one of the special symbols $\{\text{YES}, \text{NO}\}$ in the halting configuration of a computation.

In transitional P systems with promoters/inhibitors we consider rules of the following forms:

- $u \to v|_a$, $a \in O$ – this rule is only allowed to be applied when the membrane it is associated with contains at least an instance of a; a is called the *promoter* of this rule;
- $u \to v|_{\neg a}$, $a \in O$ – this rule is only allowed to be applied when the membrane it is associated with contains no instances of a; a is called the *inhibitor* of this rule.

Rules do not consume the corresponding promoters/inhibitors. A rule may have both a promoter and an inhibitor at the same time, in which case it can only be applied when there is at least one instance of the promoter *and* no instances of the inhibitor in the region. Note also, that a single instance of an object may act as a promoter for more than one instance of rewriting rules during the same transition.

2.2 P Systems with Active Membranes

A P system with *active membranes* is defined by a tuple

$\Pi = (O, H, \mu, w_1, w_2, \ldots, w_m, R, i_0)$, where

O is a finite set of objects,

H is the alphabet of names of membranes,

μ is the initial hierarchical structure of m membranes, bijectively labeled by $1, \ldots, m$;

w_i is the initial multiset in region $i, 1 \leq i \leq m$,

R is the set of rules,

i_0 is the output region; in this paper i_0 is the skin and could be omitted.

The rules in P systems with active membranes can be of the following five basic types:

(a) $[a \to v]_h^e$, $h \in H$, $e \in \{+, -, 0\}$, $a \in O$, $v \in O^*$; in this paper we consider the extension $a \in O^*$;
(b) $a[\,]_h^{e_1} \to [b]_h^{e_2}$, $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, $a, b \in O$;
(c) $[a]_h^{e_1} \to [\,]_h^{e_2} b$, $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, $a, b \in O$;
(d) $[a]_h^e \to b$, $h \in H \backslash \{s\}$, $e \in \{+, -, 0\}$, $a, b \in O$;
(e) $[a]_h^{e_1} \to [b]_h^{e_2} [c]_h^{e_3}$, $h \in H \backslash \{s\}$, $e_1, e_2, e_3 \in \{+, -, 0\}$, $a, b, c \in O$.

The rules apply to elementary membranes, i.e. membranes which do not contain other membranes inside.

The rules are applied in the usual non-deterministic maximally parallel manner, with the following details: any object can be subject of only one rule of any type and any membrane can be subject of only one rule of types (b)–(e). Rules of type (a) are not counted as applied to membranes, but only to objects. This means that when a rule of type (a) is applied, the membrane can also evolve by means of a rule of another type. If a rule of type (e) is applied to a membrane, and its inner objects evolve at the same step, it is assumed that first the inner objects evolve and then the division takes place, so that the result of applying rules inside the original membrane is replicated in the two new membranes.

2.3 Formal Logic Preliminaries

Definition 1. *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. For a set A, $|A|$ is the number of elements in this set (cardinality).

Definition 2. *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 *is the set of 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 Generalized Modus Ponens:

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

The meaning of this rule is as follows: if $P_1 \wedge P_2 \wedge \cdots \wedge P_n \rightarrow Q$ is a known 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 Generalized Modus Ponens from R and the set of atoms $\{F_1, F_2, \ldots, F_{i-1}\}$

It is important to remark that for rules $V \rightarrow W$ we can require $|W| = 1$ without losing generality. Indeed, $V \rightarrow W = \neg V \vee W$. If $W = w_1 \wedge w_2 \wedge ... \wedge w_n$,

$$V \rightarrow W = \neg V \vee \bigwedge_{i=1}^{n} w_i = \bigwedge_{i=1}^{n} \neg V \vee w_i = \bigwedge_{i=1}^{n} V \rightarrow w_i$$

This conclusion also makes it clear how to transform set of rules R to R' with the property that all right-hand sides contain no more than one symbol.

This definition of derivation provides two algorithms to answer the question of knowing if an atom g can be derived from a knowledge base KB. The first one is known as *forward chaining* and it is an example of data-driven reasoning, i.e., the starting point is the known data. The dual situation is the *backward chaining*, where the reasoning is query-driven.

Definition 3. Forward chaining *decides $KB \vdash g$ by constructing the closure K of the set of known facts A under the operation of adding new facts to the set by applying Generalized Modus Ponens and checking $g \in K$.*

Definition 4. Backward chaining *decides $KB \vdash g$ by attempting to find some resolution of the goal fact g to the set of known facts A by substituting the right-hand sides of the rules with their corresponding left-hand sides.*

We will call *reduction* the substitution of the right-hand side of a rule with the corresponding left-hand side in the process of backward chaining.

A deep study of both algorithms is out of the scope of this paper. We briefly recall their basic forms.

Forward chaining
INPUT: A reasoning problem $\langle A, R, g \rangle$
INITIALIZE: $Deduced = A, Deduced' = \emptyset$
while $Deduced \neq Deduced'$ **do**

```
    Deduced' ← Deduced
    for all (P1P2...Pn → Q) ∈ R such that {P1, P2, ..., Pn} ⊆ Deduced' do
      if Q = g then
        return true
      else
        Deduced ← Deduced ∪ {Q}
      end if
    end for
  end while
  if g ∉ Deduced then
    return false
  end if
```

Backward chaining

INPUT: A reasoning problem $\langle A, R, Targets\rangle$

```
  if Targets = ∅ then
    return true
  else
    Found ← false
    Actual ← SelectOne(Targets)
    for all (P1P2...Pn → Actual) ∈ R do
      NewTargets ← ({P1, P2, ..., Pn}\A) ∪ (Targets\{Actual})
      Found ← Found ∨ (Backward Chaining⟨A ∪ {Actual}, R, NewTargets⟩)
    end for
    return Found
  end if
```

Instead of only having one goal in the input of the backward chaining algorithm, we consider a set of goal facts $Targets$. In the case of only one goal fact, this set is initially $\{g\}$. Note that this algorithm does not always produce a correct result in the cases when the inference rules form cycles like, for example, $\{a \to b, b \to a\}$.

In this paper we present several different transformations of a tuple $\langle A, R, g\rangle$ into P systems and prove than forward chaining and backward chaining can be represented and performed in the usual semantics of membrane computing. We will write multisets in string notation. We will use the symbol $(\cdot)$ to denote multiset union.

The problem of deriving a new piece of knowledge from given ones and how such derivation can be made automatically has been studied for centuries. In this paper we explore a small part of this problem. In other logic systems, as relational logic, clausal logic, first or higher order logic, many other problems as the unification of terms must be considered. We refer the interested reader to [1].

In the paper, $v_1v_2\ldots v_n$ may mean either a conjunction of atoms in an inference rule or a multiset of objects representing such a conjunction. Which of these is actually meant should be clear from the context.

3 Forward Chaining

Let us consider the reasoning problem $\langle A, R, g\rangle$. Forward chaining basically consists in finding all facts that can be derived from A according to R and checking whether g is among these facts.

We will now try to design a transitional P system which will implement forward chaining. We will focus on constructing a non-uniform solution, because in this way we will be able to map inference rules directly to multiset rewriting rules in P systems.

Intuitively, the forward chaining algorithm consists of successive application of rules. All rules can be applied in any order and produce the same result. This leads us to the conclusion that the standard maximally parallel strategy of applying rules in P systems is suitable for carrying out forward chaining.

In this first approach we will look at the propositional rule $V \to W$ from our knowledge base as an evolution rule of a P system where all the objects in both sides of the rule have multiplicity one. Before we start, we need to introduce some considerations.

- First of all, in P systems the objects in the LHS of the rule are consumed when the rule is applied. This is a serious drawback for a direct translation of propositional logic into P systems. This limitation can be avoided if we introduce a copy of the LHS into the RHS of the rule, thus considering a multiset rewriting rule $V \to VW$ for each propositional rule $V \to W$.
- Copying the LHS into the RHS introduces new undesirable effects. One of them is that a rule can be applied indefinitely many times, since the objects which trigger the rule will be in the membrane forever. This can be also avoided by introducing a new object γ_i for each propositional rule $r_i \equiv V \to W$ and adding it to the LHS of the membrane computing rule $\gamma_i V \to VW$. This object γ_i is consumed and allows the rule to be applied at most once.
- The answer YES can be easily produced by using a rule $g \to YES$. As soon as g is generated, the object YES is produced. The answer NO should be obtained if new atoms can be deduced. From a membrane computing point of view, it is not so easy to check if a membrane has a new object different from the previous configuration, but we can consider an upper bound on the number of steps in order to check if g has been produced or not. This upper bound is related to the number of rules, since each rule can be only applied once.

We construct a P system implementing chaining according to the remarks given above:

$$
\begin{aligned}
\Pi_0 &= (U_0, [\ ]_s, w_s^{(0)}, R_0, s), \text{ where}\\
U_0 &= U \cup \{\text{YES}\} \cup \{\gamma_i \mid 1 \le i \le n\},\\
w_s^{(0)} &= \gamma_1\gamma_2 \dots \gamma_n,\\
R_0 &= \{\gamma_i V \to VW \mid (r_i : V \to W) \in R, 1 \le i \le, n\} \cup \{g \to \text{YES}\},\\
n &= |R|.
\end{aligned}
$$

Note that labeling the inference rules in R is done injectively.

We placed the initial set of facts A into the skin membrane and let some multiset rewriting rules easily obtained from R to simulate the forward-chaining inference process according the set of inference rules R. The rule $g \to \text{YES}$ is waiting for the goal to appear in the region. As soon as the goal appears, the rule produces a YES-object.

Π_0 is very simple and illustrates vividly how easily very basic forward chaining can be done in P systems. Π_0 always stops, and there is a YES in w_s when g can be derived from the facts in A.

To place a NO into the skin at proper times requires a further observation that the upper bound on the number of steps Π_0 makes is $n+1$. Indeed, all rules in R_0 may be applied only once and $|R_0| = |R| + 1 = n + 1$. Thus, we may wait until all the rules in the system are exhausted. If after $n+1$ steps the symbol YES has not been produced, the system should produce a NO. In the following P system Π_1 we have implemented the timer:

$$
\begin{aligned}
\Pi_1 &= (U_1, [\ ]_s, w_s^{(1)}, R_1, s), \text{ where} \\
U_1 &= U_0 \cup \{t_i \mid 0 \le i \le n+1\} \cup \{\text{NO}\}, \\
w_s^{(1)} &= w_s^{(0)} \cdot t_0, \\
R_1 &= R_0 \cup \{t_i \to t_{i+1} \mid 0 \le i \le, n\} \cup \{t_{n+1} \to \text{NO}, \text{YES NO} \to \text{YES}\}.
\end{aligned}
$$

Π_1 will always stop in either $n+1$ steps if a NO has been produced, or in $n+2$ steps it a YES has been produced. To nondeterministically minimize the number of steps, one may consider $R_1' = R_1 \cup \{t_i \text{YES} \to \text{YES} \mid 1 \le i \le n\}$. This, however, does not guarantee that the system will stop in a small (constant) number of steps after a YES has been produced and, in the worst case, it possible that the whole chain of transformations of t_i will take place.

To assure that the system always stops when no rules are being applied, we can use rules with inhibitors. Consider the following P system:

$$
\begin{aligned}
\Pi_2 &= (U_2, [\ ]_s, w_s^{(2)}, R_2, s), \text{ where} \\
U_2 &= U_0 \cup \{t, p, \text{NO}\}, \\
w_s^{(2)} &= w_s^{(0)} \cdot tp, \\
R_2 &= \{\gamma_i V \to VWp|_{\neg \text{YES}} \mid (r_i : V \to W) \in R, 1 \le i \le n\} \cup \\
&\quad \cup \{p \to \lambda, t \to \text{NO}|_{\neg p}, gt \to \text{YES}\}.
\end{aligned}
$$

Any rule application produces an instance of p, which is immediately erased. While rules are still being applied, p is always present in the system and thus t cannot change into NO. When rules are not being applied any more, p is erased from the system and t evolves into NO. If a rule application adds the goal symbol g to the system, g consumes t and produces YES. Thus, when no more rules can be applied, the system always needs two more steps to produce a NO. When the goal fact is produced, the system always needs one more step to produce a YES.

The inhibitors are required when, for the problem $\langle A, R, g\rangle$, $\exists(V \to W) \in R$ such that $g \in V$. Once YES is present in the system, computations should stop.

The last problem to solve is cleaning up. This is pretty obvious:

$$\Pi_f^{(1)} = \Pi_3 = (U_2, [\ ]_s, w_s^{(2)}, R_3, s), \text{ where}$$
$$R_3 = R_2 \cup \{a \to \lambda|_{\neg p} \mid a \in U \cup \{\gamma_i \mid 1 \le i \le n\}\}.$$

When the system produces a YES, the application of rules derived from R stops and p is not produced any more. This allows the rules $a \to \lambda|_{\neg p}$ to clean everything in one extra step. When there are no more rules derived from R to apply, p is not produced as well, which triggers the clean-up. Note that the clean-up procedure does not considerably alter the number of steps Π_3 needs to solve the problem.

Π_3 takes advantage of the maximal parallelism and always applies as many rules as possible at the same time. However, if there are several ways to derive g from A, Π_3 may not always follow the most efficient strategy.

4 A Different Approach to Forward Chaining

We will now try to go beyond the most trivial translation of a decision problem to a P system. We will consider a P system $\Pi_f^{(2)}$ with a single membrane, and a set of rules of type $v \to w|_{\neg i;p}$ where i, p, v, w are objects of the alphabet.

We will translate a rule $r_i \equiv u_1 u_2 \dots u_n \to v$ from R into n rules: $\rho_{ij} \equiv r_{ij} \to r_{ij+1}|_{\neg v;u_j}$ for $j \in \{1, \dots, n-1\}$ and $\rho_{in} \equiv r_{in} \to v|_{\neg v;u_j}$. Note that we require that the right-hand side of every rule in R should contain exactly one fact. We will also add the following rules to implement the timer:

$$\{t_k \to t_{k+1}|_{\neg g;t_k} \mid 1 \le k \le l-1\} \cup \{t_k \to \text{YES}|_{\neg \text{NO};g} \mid 0 \le k \le l\}$$
$$\cup \{t_l \to \text{NO}|_{\neg g;t_l}\}$$

The following rules will clean up the regions of the system:

$$\{a \to \lambda|_{\neg \text{NO};\text{YES}} \mid a \in \Gamma \backslash \{\text{YES}\}\} \cup \{b \to \lambda|_{\neg \text{YES};\text{NO}} \mid b \in \Gamma \backslash \{\text{NO}\}\}$$

Here g is the goal fact and Γ is the alphabet of the P system. l is the sum of the lengths of the left-hand sides of all rules in R. In other words, l is the maximal number of steps $\Pi_f^{(2)}$ has to go through to try all rules from R.

The alphabet contains all the atoms from U, the symbols $\{\text{YES}, \text{NO}\}$, all t_k, $1 \le k \le l$, and all the r_{ij} where i is the index of the corresponding rule from R and j is the index of an atom in the LHS of the rule r_i.

In the initial configuration the skin membrane contains all objects from A, an object t_0, and all objects r_{i1}, $1 \le i \le |R|$.

The rules ρ_{ij} are meant to check whether all left-hand-side symbols of the rule r_i are present in the system. If this condition is satisfied, the right-hand side of the rule r_i is added. In parallel with the application of rules ρ_{ij} the timer symbols

t_k evolve from t_1 to t_l. If the goal symbol g is produced, the rule $t_k \to \text{YES}|_{\neg\text{NO};g}$ produces YES. This will lead to the eventual erasure of all other symbols. If, however, the goal symbol is not produced before t_l appears in the system, a NO is produced and forces the erasure of all other symbols.

Example 1. Consider the tuple $\langle A, R, d\rangle$ with $A = \{a, b\}$ and $R = \{r_1 \equiv ab \to c, r_2 \equiv bc \to d\}$. From this deduction problem we can construct the P system $\Pi = (\Gamma, w, R_f)$ where

- the alphabet is $\Gamma = \{a, b, c, d, r_{11}, r_{12}, r_{21}, r_{22}, t_1, t_2, t_3, t_4, \text{YES}, \text{NO}\}$;
- the initial multiset in the unique membrane is $w = abr_{11}r_{21}t_0$;
- the rules ρ_{ij} are:

$$\rho_{11} \equiv r_{11} \to r_{12}|_{\neg c;a} \qquad \rho_{21} \equiv r_{21} \to r_{22}|_{\neg d;b}$$
$$\rho_{12} \equiv r_{12} \to c|_{\neg c;b} \qquad \rho_{22} \equiv r_{22} \to d|_{\neg d;c}$$

In the initial configuration $C_0 = [abr_{11}r_{21}t_0]$ rules ρ_{11} and ρ_{21} can be applied, which yields the configuration $C_1 = [abr_{12}r_{22}t_1]$. In C_1 the rule ρ_{12} can be applied and we obtain $C_2 = [abcr_{22}t_2]$. Now, by applying ρ_{22} we obtain $C_3 = [abcdt_3]$. Since the goal fact has appeared in the system, t_3 will evolve into a YES: $C_4 = [abcd\text{YES}]$. In the next step all symbols but YES will be erased: $C_5 = [\text{YES}]$.

One of the main differences between the usual semantics in P systems and the semantics in propositional logic is that the application of a rule in membrane computing consumes the objects in the LHS of the rule. This is undesirable from the point of view of propositional logic, since the validity of an atom does not change if the atom is used in a derivation. This drawback is avoided by using new auxiliary objects r_{ij} which are consumed instead of atoms.

Using these auxiliary objects has other positive effects as well. In propositional logic, once the rule $a \to b$ has been used to derive b, the rule will not be used any more. Or rather, further applications of this rule make no difference, since in propositional logic rules operate on sets of facts. This property needs to be treated specially in P systems since we use maximal parallelism and if a rule can be applied multiple times it will be applied so. By consuming the objects r_{ij} we avoid multiple applications of rules.

Finally, the use of inhibitors stops the production of an object (the derivation of an atom) if this object has been previously produced by another rule.

Theorem 1. *$\Pi_f^{(2)}$ solves the reasoning problem it was designed for using forward chaining.*

Proof. $\Pi_f^{(2)}$ works by transforming the symbols r_{i1} into the corresponding right-hand sides of rules $(r_i \equiv V_i \to a_i) \in R$, $a_i \in U$ if all the symbols in the left-hand side of rule r_i are present in the skin region. Thus the system never produces the right-hand side of a rule if not all of the symbols in the left-hand side of the rule are present in the skin region. This means that the set of facts the system derives

is always a subset of the set of facts that can be derived from A by Generalized Modus Ponens.

On the other hand, independently of the order in which the symbols in the left-hand side of the rule appear in the system, if all of the symbols in the left-hand side of the rule r_i are present in the skin region, the promoters of the rules ρ_{ij}, $1 \leq j \leq |V|$ guarantee that r_{i1} is transformed into a_i. This means that the system produces at least all facts that can be derived from A by Generalized Modus Ponens.

The conclusion is that the system always properly constructs the set of facts which can be derived from A by R.

If g can at all be derived from A, it is guaranteed to be produced at at most l-th step. At this point there will be t_l and g in the skin region and t_l will deterministically evolve into a YES. If, however, the symbol g is never produced, t_l will (correctly) evolve into a NO.

5 Backward Chaining

Backward chaining, along with forward chaining, is one of the two most commonly used methods of reasoning with inference rules. Backward chaining is also based on the modus ponens inference rule and is usually implemented by SLD resolution. Given a goal clause:

$$\neg L_1 \vee \neg L_2 \vee \cdots \vee \neg L_i \vee \cdots \vee \neg L_n$$

with selected literal $\neg L_i$ and an input definite clause

$$L \vee (\neg K_1 \vee \neg K_2 \vee \cdots \vee \neg K_3)$$

in which the atom L unifies with the atom L_i, SLD resolution derives another goal clause, in which the selected literal is replaced by the negative literals of the input clause and the unifying substitution θ is applied:

$$\text{SUBST}(\theta, \neg L_1 \vee \neg L_2 \vee \cdots \vee (\neg K_1 \vee \neg K_2 \vee \cdots \vee K_n) \vee \cdots \vee \neg L_n)$$

As in the previous section, we will only consider zero-order logic in this section. In this case we may treat unification as equality. In this section we will take advantage of the possibility to only allow rules with exactly one symbol in the right-hand side.

In the case of zero-order logic, backward chaining is more complex than forward chaining, as are P systems doing backward chaining.

The description of the backward chaining algorithm is similar to depth-first search in a state space. In artificial intelligence backward chaining is often perceived in this way and a lot of considerations are built on top of this representation.

As usual, when implementing backward chaining in P systems, we would like to take as much advantage as possible of the parallelism offered by these devices.

The obvious way to exploit parallelism is exploring the branches of the deduction tree in parallel. More concretely, if, for a certain value of $Targets$, several rules $(V \to w) \in R$, $w \in Targets$ are found, the system should start investigating each of these branches in parallel. This approach is not equivalent to investigating all possible deduction branches in parallel.

Since we would like to explore a number of branches in parallel and since these branches are completely independent of one another, it would be natural to investigate each branch in a separate region. Because we would like to decide at each certain state how many new parallel explorations to start, membrane division would suit us greatly. This brings us to the conclusion that P systems with active membranes is what we need.

However, in P systems with active membranes one can only divide a membrane into two children membranes. A way to avoid this limitation would be to demand the set $R_w = \{V \to w \mid (V \to w) \in R\}$ to have no more than two elements. Any set of rules R can be transformed to satisfy this constraint by substituting every set of rules $R_w = \{V_1 \to w, V_2 \to w, \ldots, V_n \to w\}$, $n > 2$ with

$$\{V_1 \to z_1, V_2 \to z_1\} \cup \{z_{i-1} \to z_i, V_{i+1} \to z_i \mid 2 \leq i \leq n-2\} \cup \{z_{n-2} \to w, V_n \to w\}$$

The corresponding symbols z_i, $1 \leq i \leq n-2$ should be added to the new set of facts U'.

Note that this is not the most efficient transformation.

We introduce two morphisms $(')$ and $('')$ on a multiset $W = w_1 w_2 \ldots w_n$, $W \in U^+$:

$$W' = w_1' w_2' \ldots w_n'$$
$$W'' = w_1'' w_2'' \ldots w_n''$$

We also consider the corresponding specialization of these morphisms for sets.

Given that R satisfies the constraint specified above, consider the following P system with active membranes:

$$\Pi_b = (U_b, \{k, s\}, [\ [\ ]_k\ ]_s, w_k, w_s, R_b, s), \text{ where}$$

$$U_b = U \cup U' \cup U'' \cup \{\rho_i \mid (r_i : V_i \to w_i) \in R\} \cup \{f_0, f_1, f, c_0, c_1, c_2, c_3, c, l, p, q, \$_1, \$_2, \#\} \cup \cup\{t_i | 0 \le i \le 7\} \cup \{\text{YES}, \text{NO}\},$$

$$\begin{aligned} R_b = {} & \{[w'']_k^0 \to [\rho_i]_k^+ [\rho_j]_k^+ \mid \exists\{V_i \to w, V_j \to w\} \subseteq R, V_i \ne V_j\} \cup \\ & \cup\{[w'']_k^0 \to [\rho_i]_k^+ [\#]_k^+ \mid \exists(V_i \to w) \in R, \nexists(\alpha \to w) \in R, \alpha \ne V_i\} \cup \\ & \cup\{[qa \to a'a'']_k^0, [aa' \to a']_k^+, [aa \to a]_k^+ \mid a \in U\} \cup \\ & \cup\{[\rho_i \to qc_0 f_0 V_i]_k^+ \mid \exists(V_i \to \alpha) \in R, \alpha \subseteq U\} \cup \\ & \cup\{[f_0 \to f_1]_k^+, [l]_k^+ \to []_k^0 l, [l]_s^0 \to []_s^+ l, [l \to \lambda]_s^+\} \cup \\ & \cup\{[f_1 a \to a l]_k^+ \mid a \in U\} \cup \\ & \cup\{[c_{i-1} \to c_i]_k^+ \mid 1 \le i \le 3\} \cup \{[c_3 \to \lambda]_k^0, [c_3]_k^+ \to []_k^+ \$_0\} \cup \\ & \cup\{[\$_0 \to \$_1 \text{YES}]_s^+, [\$_0 \to \text{YES}]_s^0, [\text{YES NO} \to \text{YES}]_s^+, [\$_1]_s^0 \to []_s^+ \$_1\} \cup \\ & \cup\{[t_{i-1} \to t_i]_s^0 \mid 1 \le i \le 6\} \cup \\ & \cup\{[t_6 \to p t_7]_s^0, [t_7 \to t_2]_s^+, [t_7 \to \lambda]_s^0, [p \to \text{NO}]_s^0, [p]_s^+ \to []_s^0 p\}, \end{aligned}$$

$$w_k = qgA',$$

$$w_s = t_0.$$

This system decides, using backward chaining, whether g can be derived from A according to the rules in R. Handling cycles in inference rules ($\{a \to b, b \to a\}$) is a matter of further research.

Π_b works as follows. It starts with a single worker membrane with the label k, which contains the goal symbol g and A'. All primed symbols in the worker membrane are the symbols which have already been reduced once. The system will never reduce the same symbol twice in a worker membrane. The rule $[qa \to a'a'']_k^0$ marks g as reduced and creates a double primed copy of it. Double primed symbols are the symbols which are meant to be reduced.

The system includes two types of operations for reducing symbols: $[w'']_k^0 \to [\rho_i]_k^+ [\rho_j]_k^+$ and $[w'']_k^0 \to [\rho_i]_k^+ [\#]_k^+$. The first operation is done in the cases when $|R_w| = 2$. It creates two new worker membranes with polarization + for each of the two left-hand sides of the rules used for reduction. The second operation is performed when $|R_w| = 1$. This operation creates two worker membranes, but one of them contains # which will not allow the membrane to evolve further. It is important to realize that, even if $|R_w| = 1$, the corresponding P system rule need employ a membrane to avoid attempts to multiply apply rules.

In any of the newly created membranes the rule $[\rho_i \to qc_0 f_0 V_i]_k^+$ introduces the actual left-hand side of the corresponding inference rule, as well as several service symbols. In the next step the rule $[aa' \to a']_k^+$ removes any of the new symbols which have already been reduced. At the same time f_0 evolves into f_1 and c_0 into c_1. In the next step f_1 verifies whether not primed symbols are still present in the membrane. If there indeed are such symbols, an l is produced, which eventually re-polarizes the worker membrane to 0 and thus re-launching the

process of application of inference rules. The role of the symbol q is to assure that only one not yet reduced symbol is reduced.

In parallel with the rule $[aa' \to a']_k^+$, the rule $[aa \to a]_k^+$ is applied. It removes the duplicates which appear in situations when the worker membrane contains a and b and reduces b by the rule $V \to b$, $a \in V$. Note that the rule removing the already reduced symbols and the rule cannot be applicable to the same symbol at the same time, which removes concurrency effects.

If the worker membrane contains no more symbols which have not yet been reduced, Π_b concludes that we have discovered a resolution of g to the set of know atoms A. Since the rule $[f_1 a \to al]_k^+$ does not produce the symbol l, the symbols c_i evolve until c_3 ejects a $\$_0$ in the skin region. This symbol will eventually produce YES.

The skin membrane contains symbols t_i which check whether there still is some activity in the system. Each application of an inference rule (or two inference rules, when $|R_w| = 2$) takes 6 steps. At the very beginning, the symbol t_0 evolves into t_6. If $\exists(V \to g) \in R$, when t_6 is produced in the skin region, the skin region will also contain an l. t_6 evolves into pt_7 and, at the same time, l polarizes the skin membrane to $+$. This makes t_7 evolve into t_2, thus restarting the t_i loop, while p resets the skin polarization to 0.

If, however, no worker membrane produces an l any more, when p is produced in the skin, the skin polarization stays at 0. This forces p to produce a NO and erases t_7, thus breaking the t_i loop.

The symbol $\$_0$ will always appear in the skin region at the same time as pt_7. Two situations are possible: if there has just been at least one l in the skin, the skin will have polarization $+$. In this case t_2 is produced and p is erased. At the same time, $\$_0$ produces a $\$_1$ and YES. $\$_1$ polarizes the skin to $+$, thus stopping the t_i loop. In the other situation no instances of l are produced and, when pt_7 is produced in the skin, the polarization of this membrane is 0. In this case p will produce a NO, while t_7 will be erased, thus breaking the t_i loop. At the same time $\$_0$ will produce a YES, which will erase NO in the next step.

We remark that Π_b always stops, because any application of an inference rule necessarily leads to an eventual extension of the set of primed symbols within a worker membrane. This means that the time Π_b works in can be estimated as $O(|U|)$.

Π_b is a P system with active membranes with two polarizations and cooperative rules of type (a). It is unfourtunately necessary to use cooperation, too, in this context because we need to apply every rule only once. While it should be possible to implement backward chaining using purely non-cooperative rules, such approach would hurt the clarity of the solution and it is highly possible that the parallelism of P systems would not be fully used.

Π_b is notably more complicated than any of the P systems doing forward chaining. The reason for this situation is that we have only focused on zero-order logic so far, in which unification degenerates into equality and forward chaining becomes very straightforward to implement. In conventional programming, however, back-

ward chaining is sometimes preferred due to the fact that it is easier to satisfy memory restrictions.

6 Conclusion

In this paper we continued exploring the possibilities of solving reasoning problems with P systems, a topic which was started in [2]. The computing devices of P systems look appealing in this context due to two main reasons: the similarity of inference rules and multiset rewriting rules and the maximal parallelism. The similarity of the two types of rules allows a relatively natural transformation of reasoning problems into P systems and a rather efficient exploitation of the maximal parallelism.

We have only focused on zero-order logic in this paper, which resulted in quite simple P systems for forward chaining problems and more complicated devices for backward chaining. The difference in complexity appears because of the inherently recursive nature of backward chaining; and since one of our goals was to exploit the maximal parallelism, in the case of backward chaining we needed to branch off parallel explorations for each of the possibilities arousing at every reasoning step (after every SLD resolution).

This paper does not attempt to be exhaustive. One of the most evident questions is whether the P systems suggested in this paper can be optimized in the number of rules or control symbols. Another optimization criterion is the speed of the system. Although it is remarked in the paper that all P systems have the time complexity $O(|R|)$, Π_b is about five times slower on average than $\Pi_f^{(1)}$. In designing the P systems in this paper we tried to translate the reasoning tuple as intuitively as possible; maybe less intuitive transformations would operate in better time. A concrete question in this domain is whether it possible to design a general algorithm for constructing P systems which would solve reasoning problems in sublinear time (no matter which chaining algorithm is used).

A very important research question is how to handle the situations when R includes rules which form cycles. The presence of such cycles does not necessarily disrupt the functionality of the system, but may make it produce a falsely positive result.

Another relevant direction to explore is first-order logic. In zero-order logic we could comfortably translate facts to symbols and build relatively simple P systems. First-order logic poses serious questions, however, among which one of the most important ones is how to encode predicates in P systems and how to implement unification.

A problem we have only superficially talked of is universality. Because our focus was on intuitive transformations from a reasoning tuple to a P system, we didn't pay much attention to designing a P system which would solve all or a subset of reasoning problems using either chaining algorithm. This should be a relevant topic of theoretical research and could reveal further similarities between reasoning problems and certain kinds of P systems.

Acknowledgements

AA gratefully acknowledges the project RetroNet by the Lombardy Region of Italy under the ASTIL Program (regional decree 6119, 20100618).

MAGN acknowledges the support of the projects TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence of the Junta de Andalucía, grant P08-TIC-04200.

References

1. Jago, M.: Formal Logic, *Humanities-Ebooks LLP*, 2007, ISBN 978-1-84760-041-7.
2. Gutiérrez-Naranjo, M. A., Rogozhin, V., Deductive databases and P systems, *Computer Science Journal of Moldova*, vol. 12, no. 1(34), 2004.
3. Apt, K. R.: *Logic Programming*, Handbook of Theoretical Computer Science. Elsevier Science Publishers B.V., 1990.
4. Bratko, I.: *PROLOG Programming for Artificial Intelligence*, Third Edition. Addison-Wesley, 2001.
5. Krishna S. N., Rama R.: A Variant of P Systems with Active Membranes: Solving NP-Complete Problems. *Romanian Journal of Information Science and Technology*, 2, 4 (1999), pp. 357-367.
6. Lloyd, J. W.: *Foundations of Logic Programming*, (2nd ed.) Springer, Berlin, 1987.
7. Păun, Gh., *Membrane Computing. An Introduction.* Springer-Verlag, 2002.
8. Păun, Gh., Rosenberg, G., Salomaa, A., Eds: *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2009.
9. The P systems web page. `http://ppage.psystems.eu/`