Identifying P Rules from Membrane Structures with an Error-Correcting Approach*

José M. Sempere and Damián López

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
{jsempere,dlopez}@dsic.upv.es

Abstract. In this work we propose an error-correcting approach to solve the identification of P rules for membrane modifications based on the behavior of the P system. To this aim, we take the framework of inductive inference from (structural) positive examples. The algorithm that we propose is based on previous definitions of distances between membrane structures and multiset tree automata.

1 Introduction

Membrane structures are linked to P systems as the structural information associated to them in order to make calculations. The membrane structure can be represented by a tree in which the internal nodes denote regions which have inner regions inside. The root of the tree is always associated to the skin membrane of the P system.

The relation between regions and trees has been strengthened by Freund *et al.* [6]. The authors established that any recursively enumerable set of trees can be generated by a P system with active membranes and string objects. In such a framework, P systems can be viewed as tree generators.

In this work we use multiset tree automata to accept and handle the tree structures defined by P systems [19] and we use edit distances between trees and multiset tree automata [11]. Edit distances for membrane systems were also considered in [4] and [5]. Here, we propose an inferring method to obtain a multiset tree automaton from a (finite) set of trees.

The structure of this work is as follows: First we introduce basic definitions and notation about multisets, tree languages, and automata and P systems. Then, we define multiset tree automata, we define the relation of *mirroring* between trees and we show some results between tree automata, multiset tree automata, and mirroring trees. We establish the minimum editing distance between membrane structures. In Section 3, we propose an error-correcting approach to infer multiset tree automata from examples of membrane structures. Finally, we mention some conclusions and give some guidelines for future works.

* Work supported by the Spanish CICYT under contract TIC2003-09319-C03-02 and the Generalitat Valenciana GV06/068.

H. J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 507–520, 2006.

2 Notation and Definitions

In the sequel we will provide some concepts from formal language theory, membrane systems and multiset processing. We suggest the following books to the reader: [16], [14], and [2].

Multisets

First, we will provide some definitions from multiset theory as exposed in [20].

Definition 1. *Let D be a set. A multiset over D is a pair $\langle D, f\rangle$ where $f : D \longrightarrow \mathbb{N}$ is a function.*

Definition 2. *Let $A = \langle D, f\rangle$ be a multiset; we say that A is* empty *if for all $a \in D$, $f(a) = 0$.*

Definition 3. *Suppose that $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ are two multisets. Then*

1. *the removal of multiset B from A, denoted by $A \ominus B$, is the multiset $C = \langle D, h\rangle$ where for all $a \in D$ $h(a) = max(f(a) - g(a), 0)$,*
2. *their sum, denoted by $A \oplus B$, is the multiset $C = \langle D, h\rangle$, where for all $a \in D$ $h(a) = f(a) + g(a)$, and*
3. *we say that $A = B$ if the multiset $(A \ominus B) \oplus (B \ominus A)$ is empty.*

A multiset $M = \langle D, f\rangle$ whose size, $|M|$, defined by $\sum_{a\in D} f(a)$, is finite is said to be *bounded*. Formally, we will denote the set of all multisets $\langle D, f\rangle$ such that $\sum_{a\in D} f(a) = n$ by $\mathcal{M}_n(D)$.

A concept that is quite useful to work with sets and multisets is the *Parikh mapping*. Formally, a Parikh mapping can be viewed as the application $\Psi : D^* \rightarrow \mathbb{N}^n$ where $D = \{d_1, d_2, \cdots, d_n\}$. Given an element $x \in D^*$ we define $\Psi(x) = (\#_{d_1}(x), \cdots, \#_{d_n}(x))$ where $\#_{d_j}(x)$ denotes the number of occurrences of d_j in x.

P Systems

We introduce now some basic concepts from membrane systems taken from [14] A general P system of degree m is a construct

$$\Pi = (V, T, C, \mu, w_1, \cdots, w_m, (R_1, \rho_1), \cdots, (R_m, \rho_m), i_0),$$

where:

- V is an alphabet (the *objects*)
- $T \subseteq V$ (the *output alphabet*)
- $C \subseteq V$, $C \cap T = \emptyset$ (the *catalysts*)
- μ is a membrane structure consisting of m membranes
- w_i, $1 \leq i \leq m$, is a string representing a multiset over V associated with the region i

- R_i, $1 \leq i \leq m$, is a finite set of *evolution rules* over V associated with the ith region and ρ_i is a partial order relation over R_i specifying a *priority*. An evolution rule is a pair (u, v) (or $u \rightarrow v$) where u is a string over V and $v = v'$ or $v = v'\delta$ where v' is a string over

$$\{a_{here}, a_{out}, a_{in_j} \mid a \in V, 1 \leq j \leq m\}$$

and δ is an special symbol not in V (it defines the *membrane dissolving action*). From now on, we will denote the set $\{here, out, in_k \mid 1 \leq k \leq m\}$ by tar.
- i_0 is a number between 1 and m and it specifies the *output* membrane of Π (in the case that it equals to ∞ the output is read outside the system).

The language generated by Π in external mode ($i_0 = \infty$) is denoted by $L(\Pi)$ and it is defined as the set of strings that can be defined by collecting the objects that leave the system by arranging the leaving order (if several objects leave the system at the same time then permutations are allowed). The set of numbers that represent the objects in the output membrane i_0 will be denoted by $N(\Pi)$. Obviously, both sets $L(\Pi)$ and $N(\Pi)$ are defined only for *halting computations*.

One of the multiple variations of P systems is related to the creation, division, and modification of membrane structures. There have been several works in which these variants have been proposed (see, for example, [1,13,14,15]).

In the following, we enumerate some kinds of rules which are able to modify the membrane structure:

1. 2-division: $[_h a]_h \rightarrow [_{h'} b]_{h'} [_{h''} c]_{h''}$
2. Creation: $a \rightarrow [_h b]_h$
3. Dissolving: $[_h a]_h \rightarrow b$

The power of P systems with the previous operations and other ones (e.g., *exocytosis*, *endocytosis*, etc.) has been widely studied in the membrane computing area.

Tree Automata and Tree Languages

Now, we will introduce some concepts from tree languages and automata as exposed in [3,9]. First, let a *ranked alphabet* be the association of an alphabet V with a finite relation r in $V \times \mathbb{N}$. We denote by V_n the subset $\{\sigma \in V \mid (\sigma, n) \in r\}$. We will denote by $maxarity(V)$ the maximum integer n such that V_n is not empty.

The set V^T of trees over V, is defined inductively as follows:

$a \in V^T$ for every $a \in V_0$
$\sigma(t_1, \ldots, t_n) \in V^T$ whenever $\sigma \in V_n$ and $t_1, \ldots, t_n \in V^T$, $(n > 0)$

and let a *tree language* over V be defined as a subset of V^T.

Given the tuple $l = \langle 1, 2, \ldots, k \rangle$ we will denote the set of permutations of l by $perm(l)$. Let $t = \sigma(t_1, \ldots, t_n)$ be a tree over V^T, we will denote the set of

permutations of t at first level by $perm_1(t)$. Formally, $perm_1(t) = \{\sigma(t_{i_1}, \ldots, t_{i_n}) \mid \langle i_1, i_2, \ldots, i_n\rangle \in perm(\langle 1, 2, \ldots, n\rangle)\}$.

Let $\mathbb{N}^*$ be the set of finite strings of natural numbers, separated by dots, formed by using the catenation as the composition rule and the empty word λ as the identity. Let the prefix relation $\leq$ in $\mathbb{N}^*$ be defined by the condition that $u \leq v$ if and only if $u \cdot w = v$ for some $w \in \mathbb{N}^*$ $(u, v \in \mathbb{N}^*)$. A finite subset D of $\mathbb{N}^*$ is called a *tree domain* if:

$$u \leq v \text{ where } v \in D \text{ implies } u \in D, \text{ and}$$
$$u \cdot i \in D \text{ whenever } u \cdot j \in D \ (1 \leq i \leq j)$$

Each tree domain D could be seen as an unlabeled tree whose nodes correspond to the elements of D where the hierarchy relation is the prefix order. Thus, each tree t over V can be seen as an application $t : D \rightarrow V$. The set D is called the *domain of the tree* t, and denoted by $dom(t)$. The elements of the tree domain $dom(t)$ are called *positions* or *nodes* of the tree t. We denote by $t(x)$ the label of a given node x in $dom(t)$.

Let the level of $x \in dom(t)$ be $|x|$. Intuitively, the level of a node measures its distance from the root of the tree. Then, we can define the depth of a tree t as $depth(t) = max\{|x| \mid x \in dom(t)\}$. In the same way, for any tree t, we denote the size of the tree by $|t|$ and the set of subtrees of t (denoted with $Sub(t)$) as follows:

$$Sub(a) = \{a\} \text{ for all } a \in V_0$$
$$Sub(t) = \{t\} \cup \bigcup_{i=1,\ldots,n} Sub(t_i) \text{ for } t = \sigma(t_1, \ldots, t_n) \ (n > 0)$$

Given a tree $t = \sigma(t_1, \ldots, t_n)$, the root of t will be denoted as $root(t)$ and defined as $root(t) = \sigma$. If $t = a$ then $root(t) = a$. The successors of a tree $t = \sigma(t_1, \ldots, t_n)$ will be defined as $H^t = \langle root(t_1), \ldots, root(t_n)\rangle$. Finally, $leaves(t)$ will denote the set of leaves of the tree t.

Definition 4. *A* finite deterministic tree automaton *is defined by the tuple* $A = (Q, V, \delta, F)$ *where* Q *is a finite set of states,* V *is a ranked alphabet,* $Q \cap V = \emptyset$, $F \subseteq Q$ *is the set of final states and* $\delta = \bigcup_{i:V_i \neq \emptyset} \delta_i$ *is a set of transitions defined as follows:*

$$\delta_n : (V_n \times (Q \cup V_0)^n) \rightarrow Q \qquad n = 1, \ldots, m$$
$$\delta_0(a) = a \qquad \forall a \in V_0$$

Given the state $q \in Q$, we define the *ancestors* of the state q, denoted by $Ant(q)$, as the set of strings

$$Ant(q) = \{p_1 \cdots p_n \mid p_i \in Q \cup V_0 \wedge \delta_n(\sigma, p_1, \ldots, p_n) = q\}$$

From now on, we will refer to finite deterministic tree automata simply as *tree automata*. We suggest [3,9] for other definitions on tree automata.

The transition function δ is extended to a function $\delta : V^T \to Q \cup V_0$ on trees as follows:

$$\delta(a) = a \text{ for any } a \in V_0$$
$$\delta(t) = \delta_n(\sigma, \delta(t_1), \ldots, \delta(t_n)) \text{ for } t = \sigma(t_1, \ldots, t_n) \ (n > 0)$$

Note that the symbol δ denotes both the set of transition functions of the automaton and the extension of these functions to operate on trees. In addition, you can observe that the tree automaton A cannot accept any tree of depth zero.

Given a finite set of trees T, let the *subtree automaton* for T be defined as $AB_T = (Q, V, \delta, F)$, where:

$$Q = Sub(T)$$
$$F = T$$
$$\delta_n(\sigma, u_1, \ldots, u_n) = \sigma(u_1, \ldots, u_n) \qquad \sigma(u_1, \ldots, u_n) \in Q$$
$$\delta_0(a) = a \qquad a \in V_0$$

Multiset Tree Automata and Mirrored Trees

We will extend over multisets some definitions of tree automata and tree languages. We will introduce the concept of multiset tree automata and then we will characterize the set of trees that it accepts.

Given any tree automaton $A = (Q, V, \delta, F)$ and $\delta_n(\sigma, p_1, p_2, \ldots, p_n) \in \delta$, we can associate to δ_n the multiset $\langle Q \cup V_0, f \rangle \in \mathcal{M}_n(Q \cup V_0)$ where f is defined by $\Psi(p_1 p_2 \ldots p_n)$. The multiset defined in such way will be denoted by $M_\Psi(\delta_n)$. Alternatively, we can define $M_\Psi(\delta_n)$ as $M_\Psi(p_1) \oplus M_\Psi(p_2) \oplus \cdots \oplus M_\Psi(p_n)$ where $\forall 1 \leq i \leq n \ M_\Psi(p_i) \in \mathcal{M}_1(Q \cup V_0)$. Observe that if $\delta_n(\sigma, p_1, p_2, \ldots, p_n) \in \delta$, $\delta'_n(\sigma, p'_1, p'_2, \ldots, p'_n) \in \delta$ and $M_\Psi(\delta_n) = M_\Psi(\delta'_n)$ then δ_n and δ'_n are defined over the same set of states and symbols but in different order (that is the multiset induced by $\langle p_1, p_2, \ldots, p_n \rangle$ equals to the one induced by $\langle p'_1 p'_2 \ldots p'_n \rangle$).

Now, we can define a *multiset tree automaton* that performs a bottom-up parsing as in the tree automaton case.

Definition 5. *A* multiset tree automaton *is a tuple* $MA = (Q, V, \delta, F)$, *where* Q *is a finite set of states,* V *is a ranked alphabet with maxarity*$(V) = n$, $Q \cap V = \emptyset$, $F \subseteq Q$ *is a set of final states and* δ *is a set of transitions defined as follows:*

$$\delta = \bigcup_{\substack{1 \leq i \leq n \\ i \,:\, V_i \neq \emptyset}} \delta_i$$

$$\delta_i : (V_i \times \mathcal{M}_i(Q \cup V_0)) \to \mathcal{P}(\mathcal{M}_1(Q)) \qquad i = 1, \ldots, n$$
$$\delta_0(a) = M_\Psi(a) \in \mathcal{M}_1(Q \cup V_0) \qquad \forall a \in V_0$$

We can take notice that every tree automaton A defines a multiset tree automaton MA as follows.

Definition 6. *Let $A = (Q, V, \delta, F)$ be a tree automaton. The multiset tree automaton induced by A is defined by the tuple $MA = (Q, V, \delta', F)$ where each δ' is defined as follows: $M_\Psi(r) \in \delta'_n(\sigma, M)$ if $\delta_n(\sigma, p_1, p_2, \ldots, p_n) = r$ and $M_\Psi(\delta_n) = M$.*

Observe that, in the general case, the multiset tree automaton induced by A is nondeterministic.

As in the case of tree automata, δ' could also be extended to operate on trees. Here, the automaton carries out a bottom-up parsing where the tuples of states and/or symbols are transformed by using the Parikh mapping Ψ to obtain the multisets in $\mathcal{M}_n(Q \cup V_0)$. If the analysis is completed and δ' returns a multiset with at least one final state, then the input tree is accepted. So, δ' can be extended as follows:

$$\delta'(a) = M_\Psi(a) \text{ for any } a \in V_0,$$

$$\delta'(t) = \{M \in \delta'_n(\sigma, M_1 \oplus \cdots \oplus M_n) \mid M_i \in \delta'(t_i)\ 1 \leq i \leq n\}$$
$$\text{for } t = \sigma(t_1, \ldots, t_n)\ (n > 0).$$

Formally, every multiset tree automaton MA accepts the following language

$$L(MA) = \{t \in V^T \mid M_\Psi(q) \in \delta'(t), q \in F\}.$$

Another extension which will be useful is the one related to the ancestors of every state. So, we define $Ant_\Psi(q) = \{M \mid M_\Psi(q) \in \delta_n(\sigma, M)\}$.

Theorem 1. *(Sempere and López, [19]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A and $t = \sigma(t_1, \ldots, t_n) \in V^T$. If $\delta(t) = q$, then $M_\Psi(q) \in \delta'(t)$.*

Corollary 1. *(Sempere and López, [19]) Let $A = (Q, V, \delta, F)$ be a tree automaton and $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A. If $t \in L(A)$, then $t \in L(MA)$.*

We will introduce the concept of *mirroring* in tree structures as exposed in [19]. Informally speaking, two trees will be related by mirroring if some permutations at the structural level hold. We propose a definition that relates all the trees with this mirroring property.

Definition 7. *Let t and s be two trees from V^T. We will say that t and s are* mirror equivalent, *denoted by $t \bowtie s$, if one of the following conditions holds:*

1. $t = s = a \in V_0$,
2. $t \in perm_1(s)$,
3. $t = \sigma(t_1, \ldots, t_n)$, $s = \sigma(s_1, \ldots, s_n)$ *and there exists* $< s^1, s^2, \ldots, s^k > \in perm(< s_1, s_2, \ldots, s_n >)$ *such that* $\forall 1 \leq i \leq n\ t_i \bowtie s^i$.

Theorem 2. *(Sempere and López, [19]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $t = \sigma(t_1, \ldots, t_n) \in V^T$ and $s = \sigma(s_1, \ldots, s_n) \in V^T$. Let $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A. If $t \bowtie s$, then $\delta'(t) = \delta'(s)$.*

Corollary 2. *(Sempere and López, [19]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $MA = (Q, V, \delta', F)$ the multiset tree automaton induced by A and $t \in V^T$. If $t \in L(MA)$, then, for any $s \in V^T$ such that $t \bowtie s$, $s \in L(MA)$.*

The last results were useful to propose an algorithm to determine whether two trees are mirror equivalent or not [19]. So, given two trees s and t, we can establish in time $\mathcal{O}((min\{|t|, |s|\})^2)$ whether or not $t \bowtie s$.

Editing Distances Between Membrane Structures

The initial order of a membrane structure can be fixed. Anyway, whenever the system evolves (membrane dissolving, division, creation, etc.) this order can be at least somehow ambiguous. Furthermore, the initial order of a P system is only a naming convention given that the membrane structure of any P system can be renamed without changing its behavior due to the parallelism ingredient (observe that if this mechanism was sequential, then the ordering could be important for the final output).

The representation by trees could be essential for the analysis of the dynamic behavior of P systems Whenever we work with trees to represent the membrane structure of a given P system, we can find a *mirroring effect*

Example 1. Let us take the three membrane structures of Figure 1. Then, the associated trees are the following (in top-down order):

$\sigma(\sigma(a, a), \sigma(a, \sigma(a)))$

$\sigma(\sigma(\sigma(a), \sigma(a, a)), \sigma(a, a))$

$\sigma(\sigma(\sigma(a, \sigma(a), a), a), \sigma(a, a))$

Observe that the symbol a denotes the names for elementary regions instead of objects.

We proposed a method to establish if two membrane structures μ and $\mu\prime$ are identical [19], while in [11] we proposed an algorithm to obtain the minimum set of membrane rule applications needed to transform μ into $\mu\prime$ (or vice versa). The last algorithm was based on multiset tree automata and the tree representation for membrane structures Note that the target of that algorithm was to force the automaton to accept the tree. The algorithm employed edit operations for substitution (reduction) of a tree to a state of the automaton, deletion of a (sub)tree and insertion of a state. Intuitively, the substitution of a tree by a state of the automaton could be seen as the substitution of the tree by the nearest tree that could be reduced to the state.

3 Identification with an Error-Correcting Approach

Here, we address the problem of inferring some operators that regulate the behavior of a P system: given a set of membrane structures generated by an arbitrary

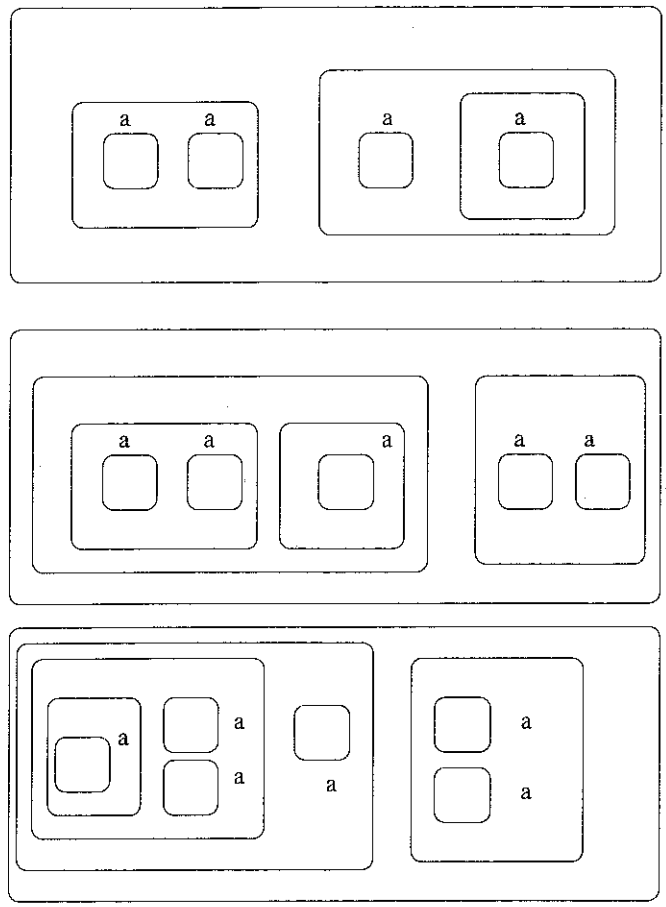

Fig. 1. Three membrane structures

P system, the problem consists of defining the set of rules needed to generate the membrane structures of the set and (possibly) some others that do not belong to it.

This problem could be approached as a *Grammatical Inference* problem [18]. Here, the set of membrane structures can be represented by a set of trees and the set of rules that regulate the P system behavior is deduced from a multiset tree automata. So, we will solve the problem by using inference methods for tree languages.

The problem of inferring tree languages has been widely approached in the grammatical inference literature. So, in [7] a method to infer k-testable tree sets from sample data is proposed. In [8] a method to infer recognizable tree languages from finite information is proposed. Finally, in [12] a method to infer reversible tree languages from samples was proposed.

Here, we will use an inferring method based on error-correcting techniques. The method that we propose is based on a preliminary technique for tree automata inference [10] based on the *Error-Correcting Grammatical Inference* method (ECGI) [17]. Basically, the method constructs the set of transitions of the automaton such that the editing distances from the sample data to the automaton is minimal.

In what follows, we propose an adaptation of such method. We will use multiset tree automata instead of tree automata and the editing operations together with the minimal editing distance is based on our previous work [11]. So, we propose Algorithm 1 as a method to infer multiset tree automata from a finite

sample of trees. Algorithm 1 uses a subroutine showed as Algorithm 2 which calculates the minimal editing distance for every tree to the current multiset tree automaton and adds the new transitions needed to converge to the minimal cost tree automaton.

Algorithm 1. Error correcting multiset tree automata inference algorithm.

Input:

- A set of trees $S = \{t_1, t_2, \ldots t_n\}$ (Membrane representations)

Output:

- A multiset tree automaton $A = (Q, V, \delta, F)$ that, at least, recognizes the set of trees $\{s : \exists t \in S,\ t \bowtie s\}$

Method:

1. Obtain the initial automaton
   - $V = \{\sigma\} \cup leaves(t_1)$
   - $\forall s \in Sub(t_1)$ in postorder
     if $s = \sigma(u_1, u_2, \ldots, u_p)$ then $Q = Q \cup \{< u_1 u_2 \ldots u_p >\}$
     else /*$s \in leaves(t_1)$*/ $Q = Q \cup s$
     End$\forall$
   - $F = \{< u_1 u_2 \ldots u_k >\}$ where $t_1 = \sigma(u_1, u_2, \ldots, u_k)$
   - $\delta(a) = a\ \forall a \in leaves(t_1)$
     if $u_1, u_2, \ldots u_k, < u_1 u_2 \ldots u_k > \in Q$ then add to the automaton the transition $\delta(\sigma, u_1 u_2 \ldots u_k) =< u_1 u_2 \ldots u_k >$
2. $\forall t_i \in S$ with $2 \leq i \leq n$
   - $A = Expand(A, t_i)$

   End$\forall$
3. $Return(A)$

EndMethod.

The output of the Algorithm 1 is a multiset tree automaton that recognizes all the trees given as input at the minimum editing cost together with the mirrored trees of the input. Observe, that this automaton could recognize some other trees that have not been supplied as input. The correctness of the algorithm and its complexity, which is polynomial with the size of the tree set, is deduced from our previous work on editing distance [11].

We will discuss a complete example of the algorithm running in order to clarify its behavior.

Example 2. Let us consider the following training set:

$$\left\{ \begin{array}{c} \sigma(\sigma(a,a),\sigma(a,\sigma(a))), \\ \sigma(\sigma(\sigma(a),\sigma(a,a)),\sigma(a,a)), \\ \sigma(\sigma(\sigma(a,\sigma(a),a),a),\sigma(a,a)) \end{array} \right\}$$

which corresponds to the membrane structures showed in Figure 1.

Algorithm 2. Automaton modification subroutine $Expand(A, t)$.

Input:

- A multiset tree automaton $A = (Q, V, \delta, \{q_f\})$ and a tree $t = \sigma(t_1, t_2, \ldots, t_m)$ (membrane representation)

Output:

- A multiset tree automaton $A = (Q, V, \delta, F)$ that accepts, at least, $L(A) \cup \{s : s \bowtie t\}$

Method:

1. Consider the input automaton A and perform an error correcting analysis for t according to [11]
2. From the previous step, obtain the accepting minimum cost path Δ_t. Distinguish those non-error transitions Δ_t^N from the error transitions Δ_t^E
3. $\forall s \in Sub(t)$
   if $s = (\sigma, s_1, \ldots, s_k)$ then
   consider τ_s the transition $\delta(\sigma, q_1 \ldots q_k) = p_s \in \Delta_t$
   and set $Red[s] = p_s$
   else /* $s \in leaves(t)$ */ set $Red[s] = s$
   End$\forall$
4. $\forall s = \sigma(s_1, \ldots, s_k) \in Sub(t)$ /* in postorder */
   if $\tau_s \in \Delta_t^E$ or $Red[s] = \emptyset$ then
   add a new state q_N to Q
   add the transition $\delta(\sigma, Red[s_1] \ldots Red[s_k]) = q_N$
   set $Red[s] = q_N$
   else if $\exists \tau_{s_i} \in \Delta_t^E$ then add the transition $\delta(\sigma, Red[s_1] \ldots Red[s_k]) = q_N$
   End$\forall$
5. add the transition $\delta(\sigma, Red[t_1]Red[t_2] \ldots Red[t_m]) = q_f$
6. $Return(A)$

EndMethod.

The first step in Algorithm 1 considers the empty automaton and the first tree, thus obtaining the initial multiset tree automaton with the following transitions:

$$\begin{aligned} \delta(\sigma, aa) &= q_1 \\ \delta(\sigma, a) &= q_2 \\ \delta(\sigma, aq_2) &= q_3 \\ \delta(\sigma, q_1 q_3) &= q_4 \in F \end{aligned}$$

The second inference step performs an error correcting analysis of the second tree in the training set and the previous automaton. The distance matrix is shown in Table 1. Figure 2 summarizes the error correcting analysis.

Table 1. Distance table of each subtree (in postorder) and each state of the automaton. Minimum cost path is boldmarked. Second postorder subtree is deleted in the analysis, therefore its row contains no bold figure. Note that it is not necessary to obtain all the distances for the root node because only distances to final states will be considered.

	q_1	q_2	q_3	q_4
s_1	1	**0**	2	8
s_2	0	1	3	9
s_3	7	6	4	2
s_3	**0**	1	3	7
s_5	–	–	–	**4**

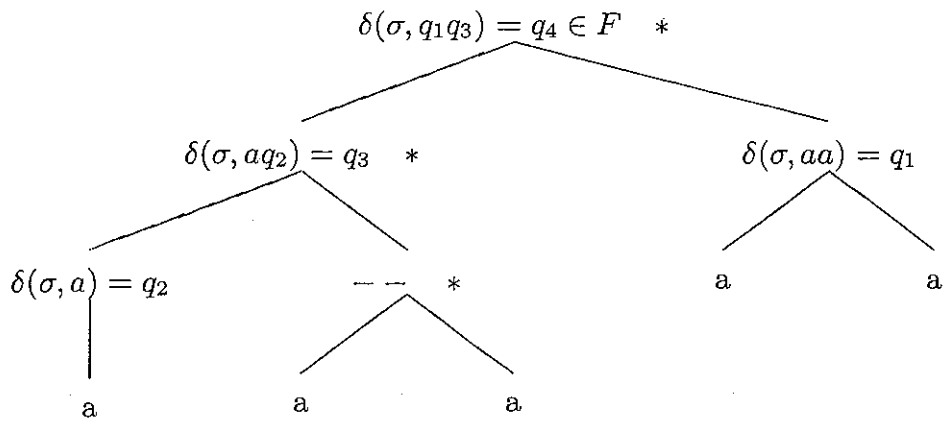

Fig. 2. Error correcting parsing of the tree $\sigma(\sigma(\sigma(a), \sigma(a, a)), \sigma(a, a))$. Error productions are marked with an asterisk. Note that a subtree deletion also occurred. This edit operation is also considered as an error transition.

The error correcting analysis obtains three error transitions. The automaton is then modified to accept the sample. Three new transitions are added (marked with an asterisk) in the following way:

$$\begin{array}{rr} & \delta(\sigma, aa) = q_1 \\ & \delta(\sigma, a) = q_2 \\ & \delta(\sigma, aq_2) = q_3 \\ & \delta(\sigma, q_1q_3) = q_4 \in F \\ (*) & \delta(\sigma, aa) = q_5 \\ (*) & \delta(\sigma, q_2q_5) = q_6 \\ (*) & \delta(\sigma, q_6q_1) = q_4 \in F \end{array}$$

Note that the inference process adds a transition which is equal to an already exiting one. This case can be run-time detected, using the existing transition instead of creating a new one. This automaton is considered in the following inference step. This step considers the following tree

$$\sigma(\sigma(\sigma(a, \sigma(a), a), a), \sigma(a, a))$$

The distance matrix is shown in Table 2. Figure 3 summarizes the error correcting analysis.

Table 2. Distance table of each subtree (in postorder) and each state of the automaton. Minimum cost path is boldmarked.

	q_1	q_2	q_3	q_4	q_5	q_6
s_1	1	0	2	8	1	6
s_2	2	**3**	1	7	2	5
s_3	6	5	**4**	6	6	6
s_3	**0**	1	3	9	0	7
s_5	–	–	–	–	–	**4**

The final automaton is the following:

$$
\begin{aligned}
\delta(\sigma, aa) &= q_1 \\
\delta(\sigma, a) &= q_2 \\
\delta(\sigma, aq_2) &= q_3 \\
\delta(\sigma, q_1q_3) &= q_4 \in F \\
\delta(\sigma, aa) &= q_5 \\
\delta(\sigma, q_2q_5) &= q_6 \\
\delta(\sigma, q_6q_1) &= q_4 \in F \\
(*)\quad \delta(\sigma, a) &= q_7 \\
(*)\quad \delta(\sigma, aq_7a) &= q_8 \\
(*)\quad \delta(\sigma, aq_8) &= q_3
\end{aligned}
$$

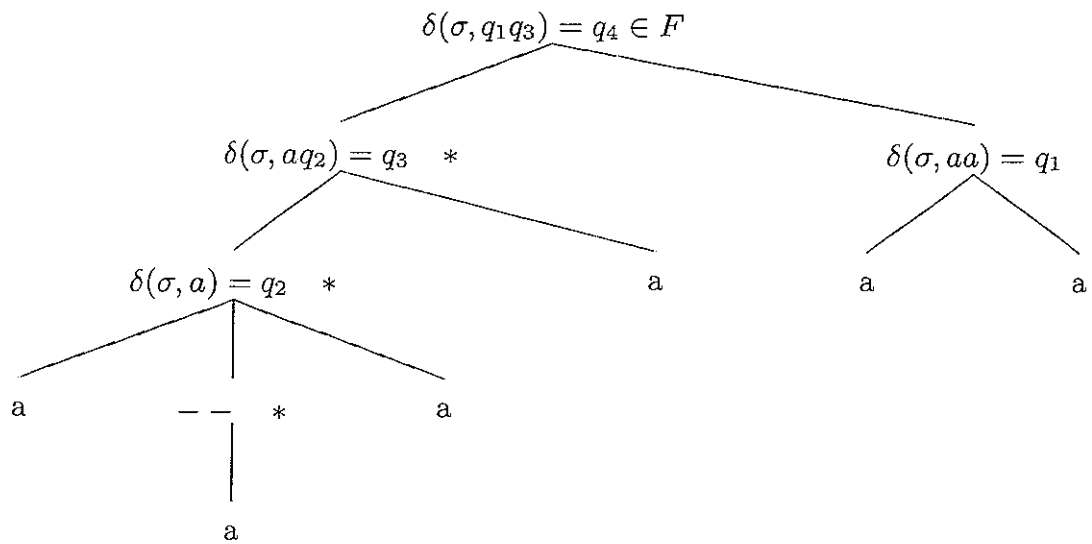

Fig. 3. Error correcting parsing of the tree $\sigma(\sigma(a, \sigma(a, \sigma(a), a), a), \sigma(a, a))$. Error productions are marked with an asterisk. Note that the transition used to reduce the third postorder subtree is marked as error. This transition does not add error cost but one of its descendants is an error transition and therefore a new transition must be created.

In the last example we obtain a multiset tree automaton which is consistent with the input data. Observe that this automaton induces a set of P rules for a P system as was showed in [19] and [11].

4 Conclusions and Future Work

We have proposed a method to infer a multiset tree automaton from a finite set of membrane structures. This multiset tree automaton employs the minimum number of tree editing operations to accept the input set. From this multiset tree automaton we can obtain a set of P rules that regulates the behavior of a P system which generates the membrane structures given as input.

Here, we have used a grammatical inference technique based on an error-correcting approach. In the future we will explore other options to infer new models (i.e., reversible tree languages, locally testable tree languages, etc.) So, a new question arises: Given a set of trees of a tree language class (reversible, locally testable, etc.), what is the common characteristics of the P systems that generate them? Such a characteristic will introduce a characterization of P systems based on the kind of membrane structures that they generate.

References

1. A. Alhazov, T.O Ishdorj. Membrane operations in P systems with active membranes. In *Proc. Second Brainstorming Week on Membrane Computing*, TR 01/04 of RGNC, Sevilla University, 2004, 37–44.
2. C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, *Multiset Processing*. LNCS 2235, Springer, Berlin, 2001.
3. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi. *Tree automata techniques and applications*. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997, release October, 1rst 2002.
4. A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. Weak metrics on configurations of a P system. In *Proc. Second Brainstorming Week on Membrane Computing*, RGNC TR 01/04 of RGNC, Sevilla University, 2004, 139–151.
5. E. Csuhaj-Varjú, A. Di Nola, Gh. Păun, M.J. Pérez-Jiménez, G. Vaszil. Editing configurations of P systems. In *3rd Brainstorming Week on Membrane Computing 2005*, TR 01/05 of RGNC, Sevilla University (M.A. Gutiérrez Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, eds.), Fénix Editora, Sevilla, 2005, 131–155.
6. R. Freund, M. Oswald, A. Păun. P systems generating trees. In *Pre-proceedings of Fifth Workshop on Membrane Computing (WMC5)* (G. Mauri, Gh. Păun, C. Zandron, eds.), MolCoNet project IST-2001-32008, 2004, 221–232.
7. P. García. *Learning k-Testable Tree Sets from Positive Data*. Informe técnico DSIC-II/46/93, 1993.
8. P. García, J. Oncina. *Inference of Recognizable Tree Sets*. Informe técnico DSIC-II/47/93, 1993.
9. F. Gécseg, M. Steinby. Tree languages. In *Handbook of Formal Languages*, volume 3 (G. Rozenberg, A. Salomaa, eds.), Springer-Verlag, Berlin, 1997, 1–69.

10. D. López, S. España. Error correcting tree language inference. *Pattern Recognition Letters*, 23, 1-3 (2002), 1–12.
11. D. López, J.M. Sempere. Editing distances between membrane structures. In *Proceedings of the 6th International Workshop, WMC 2005* (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 3850, Springer, Berlin, 2006, 326–341.
12. D. López, J.M. Sempere, P. García. Inference of reversible tree languages. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, 34, 4 (2004), 1658–1665.
13. A. Păun. On P systems with active membranes. In *Proc. of the First Conference on Unconventionals Models of Computation*, 2000, 187–201.
14. Gh. Păun. *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
15. Gh. Păun, Y. Suzuki, H. Tanaka, T. Yokomori. On the power of membrane division on P systems. *Theoretical Computer Science* 324, 1 (2004), 61–85.
16. G. Rozenberg, A. Salomaa, eds. *Handbook of Formal Languages* Springer, Berlin, 1997.
17. H.M. Rulot. *ECGI. Un algoritmo de inferencia gramatical mediante corrección de errores*. PhD Dissertation, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia 1992 (in Spanish).
18. Y. Sakakibara. Recent advances of grammatical inference. *Theoretical Computer Science*, 185 (1997), 15–45.
19. J.M. Sempere, D. López. Recognizing membrane structures with tree automata. In *3rd Brainstorming Week on Membrane Computing 2005*, TR 01/05 of RGNC, Sevilla University (M.A. Gutiérrez Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, eds.), Fénix Editora, Sevilla, 2005, 305–316.
20. A. Syropoulos. Mathematics of multisets. In [2], 347–358.