УДК 621

W. Korczyński, G. Wawrzoła*, S. Wawrzoła**
Wyższa Szkoła Umiejętności,
*Politechnika Wrocławska,
Wydział Elektroniki,
**Akademia Świętokrzyska,
Wydział Zarządzania i Administracji

ON A RECONSTRUCTION PROBLEM FOR MEMBRANE SYSTEMS

The problem of reconstruction of an object from some of their reduct is very important in many areas of life and science. It is one of the well known problems in the theory of partial orders. In the paper we describe and solve this problem in a very special case: the infrastructure of Paun's systems (called also membrane structures). The main result of the paper is an algorithm of reconstruction of the original membrane structure from two of its reducts.

1. Introduction. Paun's systems are widely used tool in analyzing of massively parallel computations and natural computing. Perhaps the most well known area connected with Paun's systems are DNA\ computations called also molecular computations. Let us recall some problems from the area of computational complexity theory.

The notion of NP-hard problem is based on the classical von Neuman paradigm of computing understood as a sequence of actions. This paradigm works very well and almost all the theory of complexity is based on it. The considering of a set of parallel computing devices like Turing machines do not improve the complexity of NP hard problems. A problem can be perhaps solved a bit more quickly, but it remains in the same class of complexity (namely in the class of NP hard problems). In this sense one can say that (theoretically only of course) adding new processors does not allow for really improvement of the solving process of the problem. Another paradigm of computing, called DNA computing, seems to be a tool which allows to transform an NP hard problems to the class of problems which can be solved in polynomial time. It can be made, roughly speaking, by a really massively parallelism. L.M. Adleman has shown in 1994 [1] that the well known salesman problem can be solved in polynomial time using the so called DNA computation. The theory of DNA computation is nowadays one of the most quickly developed area of theoretical computer science. Paun's systems endowned with a stochastic mechanism may be seen as a model of devices in which some DNA computations may be performed. This fact determines the role and the meaning of Paun's systems in the (theoretical) computer science. The interested reader can find details in [5, 6, 7, 10, 13].

Paun's systems are also used in other areas where one has to deal with hierarchical systems In this case Paun's systems can be seen as a model of hierarchical dynamical systems that means systems changing their (local) situations on different levels. So, they allow to consider the concurrence on various level of system description. A hierarchical system can be seen as a tree. In the vertices of the tree there may be allocated multisets of some objects. The way of working of the system consists of the changing of the allocation of these multisets object. It seems to be natural to allowed such a changing of allocation as a kind of diffusion of these objects trough the "membranes" delimiting the objects represented by the vertices of the tree. The way of working (the way of life) of such hierarchical systems is determined by the so called evolution rules describing in which situations the "diffusion"' of some objects through the membranes is possible. The definition of Paun's systems and some of their properties are described in part 2. This is – roughly speaking – the ideology of the way of working of Paun's systems. The interested reader can find more details in [11, 14, 17]. One of these evolution rules describes when, that means in which situation, a "region" (that means an object corresponding to a vertex of the tree mentioned above) can be deleted. The problem we solve in the paper is the following. Let us consider a Paun's system represented by a tree, say T. Let us assume that some vertices, say v_1 and v_2 have been deleted from this

tree by means of the above mentioned deleting rules. Now a very natural question arises. What for information is needed to reconstruct the original shape of the tree representing the corresponding Paun's system? We show that we need only information about the place (the region) in which the membranes mentioned above have been deleted. More precisely we show that having the information of two such places of deleting of membranes we can reconstruct the original tree. This problem can be seen as a variant of the well known reconstruction problem for partial orders more precisely for trees. The difference is that we assume in a sense more information that in the classical reconstruction problem. Namely we assume we know the parents which children (two) have been deleted. The nature of these informations that means "how to recognize" the parents which have lose the children is irrelevant in our consideration[1].

In the definition of P system, the notion of multiset there occurs. The most important difference between multisets and sets is that in a multiset the object are allowed to occur more than once. Let us define the notion formally. The set $\mathbb{N}_\infty$ is the set of natural numbers extended with the element ∞.

Definition. Let A be a set. A **multiset on** A is a function m: $A \to \mathbb{N}_\infty$. The class of multisets (on A) is denoted by **Mst** (**Mst**(A)). Let $a \in A$. We say that an element a of the set A **has** in a multiset M **multiplicity** n iff $M(a) = n$. We define $a \in M \Leftrightarrow M(a)>0$ and say that a **is an element of** M. The **multiset inclusion** relation is defined by $X \subseteq Y \Leftrightarrow \forall_{a \in A}\, Y(a) \geq X(a)$. We say that X is a **submultiset** of Y.

In the paper the standard mathematical terminology and notation is used. The terminology and notation of Paun's systems originates from [17]. We also use some additional notions which are specific for the so called "reconstruction" of partial orders ([8, 9, 15, 16]).

2. *P*-systems – a short description. In this section we define the notion of Paun's systems. The material and the "plan" of this section is based on [17].

The infrastructure of the Paun's systems are membrane structures. Membrane structures may be seen as Venn diagrams in which every two sets are either included or disjoint. Such structures may be models of many various objects; computational devices (see [11, 13, 17]), (biological) cell structures, hierarchical systems in economy or management and many, many others. Formally such structures may be defined in many ways. The membrane structures we define in this section are one of possibly descriptions of such structures. The approach we use in the paper is based on the paper [17] from which the definition, examples and many descriptions have been taken.

Roughly speaking, a membrane structure is a tree. The vertices of this tree are interpreted as objects containing some multisets of objects like directories in directories-tree of an operating system like UNIX or DOS. The similarity is really very strong except the fact that directories contain some **sets**, not multisets of objects. One of the applications of P-systems is the complexity theory, where formal languages are frequently used. This is one of the reasons for which P-systems are often defined as some "linguistic" objects. Probably the most popular presentation of membrane structures defines them as some structures of parentheses.

Definition *Let us consider the language* MS *over the alphabet {[,]}, whose strings are recursively defined as follows:*

- $[\,] \in MS$;
- if $\mu_1,\ldots,\mu_n \in MS$, $n \geq 1$, then $[\mu_1,\ldots,\mu_n] \in MS$;
- nothing else is in MS.

For each pair $(x, y) \in MS$, $x \sim y$ if and only if the two strings can be written in the form

$$x = [_1 \ldots [_k \ldots]_k \ldots [_{k+1} \ldots]_{k+1} \ldots]_1 \quad \text{and} \quad y = [_1 \ldots [_{k+1} \ldots]_{k+1} \ldots [_k \ldots]_k \ldots]_1 \qquad (1)$$

that is two pairs of parentheses which are neighbors at the same level are interchanged together with their contents. We denote by $\sim^*$ the reflexive and transitive closure of the relation $\sim$. Let $\overline{MS}$ denote the set of equivalence classes of MS with respect to this relation. The elements of $\overline{MS}$ are called **membrane structures**.

[1] One of the possibilities is to compare the contents of a region in two subsequent situations, but this is neither the only possibility nor the sufficient difference.

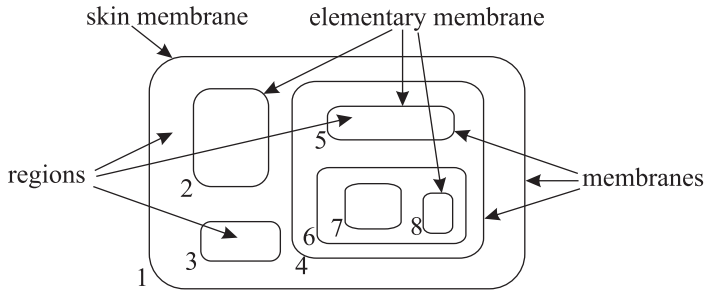
By a *membrane* it is meant any pair of matching parentheses [,]. The membrane consisting of external parenthesis is called the *skin*, and membranes containing no other membranes are called *elementary membranes*. By the *degree* of μ denoted by $deg(\mu)$ it is meant the number of membranes in the membrane structure μ. By the *depth* of a membrane structure μ, denoted by $dep(\mu)$, we mean the function defined recursively as follows:

- if $\mu = [\ ]$, then $dep(\mu) = 1$;
- if $\mu = [\mu_1,...,\mu_n]$, then $dep(\mu) = \max\{\text{dep}(\mu_i)\colon 1 \le i \le \text{n}\}+1$ for some $\mu_1,...,\mu_n \in MS$.

The notion of depth of an membrane system corresponds to the standard notion of the depth of a tree.

To each membrane we associate a *region*. In the graphical presentation of an *MS* it can be seen as a space boarded by this membrane, that means the membrane is "the topological border" of this region.

In the picture below the skin of the membrane structure is labeled by the number 1, elementary membranes are 2, 3, 5, 7 and 8.



Having a finite set U, of *objects*, a membrane structure $\mu = [_1...[_n...]_n...]_1$ of degree n, $n \ge 1$, with the membranes labeled in a one to one manner with the numbers $\{1,...,n\}$, one can identify the regions with the numbers $\{1,...,n\}$. Let us associate with each region i of μ, $1 \le i \le n$, a *multiset* M_i on U.

Definition *By a super-cell we mean any*

$$\text{SC} = \{([\mu, M_1,...,M_n)\colon \mu \in \overline{MS} \text{ and } deg(\mu) = n\}$$

where each $M_i\colon U \to \mathbb{N}$ is the multiset associated with region i, $1 \le i \le n$.

The multiset corresponding to a region of a super-cell is called its *contents*. If a membrane *m'* is placed in a membrane *m* and there isn't any membrane *m''* such that *m'* is placed in *m''* and *m''* is placed in *m*, then all objects placed in the region *m* (i.e. the objects are immediately outside *m'* and inside membrane *m*) are said to be *adjacent* to membrane *m'*. By the *support* of a super-cell π, denoted by $supp(\pi)$, it is meant the set of all objects appearing in π at least once.

Definition *A membrane system (or* P *system) of degree* n*,* $\text{n} \ge 1$*, is a construct*

$$\Pi = (V, \mu, M_1,...,M_n, R_1,...,R_n, i_0),$$

where:

- V is an alphabet which elements are called objects;
- μ is a membrane structure consisting of n membranes, with the membranes and the regions labeled in a one-to-one manner with the elements of a given set; we will use the labels 1,2,...,n;
- M_i, $1 \le i \le n$, are strings representing multisets over V associated with the regions 1,...,n of μ;
- R_i, $1 \le i \le n$, are finite sets of evolution rules over V associated with the regions 1,...,n of μ; An evolution rule is a pair (u, v), usually written in the form $u \to v$, where $u \in V^+$ and $v = v'$ or $v = v'\delta$, where v' is a string over

$$(V \times \{\text{here, out}\}) \cup (V \times \{in_j\colon 1 \le j \le n\}),$$

and δ is a special symbol not in V. The length of the string u is called the radius of the rule $u \to v$.

- $i_0 \in \{1,2,...,n\} \cup \{\infty\}$. If i_0 is a number between 1 and n then it specifies the **output membrane** of Π. If $i_0 = \infty$, or if it is omitted, then output is read in the outer region.

The evolution rules are interpreted as a kind of moving of objects between regions of a membrane systems. All rules which do not contain the symbol δ can be interpreted in this way. The rules containing this symbol correspond to deleting of some membranes.

A *configuration* of the system Π is a sequence $(\mu', M'_{i_1}, ..., M'_{i_k})$ where μ' is obtained from μ by removing all membranes different from $i_1,...,i_k$ (the skin membrane is not removed), M'_j, $1 \le j \le k$, are strings over V, and $\{i_1,...,i_k\} \subseteq \{1,2,...,n\}$.

The system can change the configuration by using the evolution rules in parallel to all objects which can evolve. For two configurations

$$C_1 = (\mu', M'_{i_1}, ..., M'_{i_k}), C_2 = (\mu'', M''_{j_1}, ..., M''_{j_k})$$

of Π, a *transition* $C_1 \Rightarrow C_2$ is the passing from C_1 to C_2 by applying the evolution rules appearing in $R_{i_1}, ..., R_{i_k}$ in the following way:

Consider a rule of the form $u \to v$ in a set of rules R_{i_h}. If in the region associated with the membrane i_h all the objects in u, with the multiplicities specified in u, appear in $m(M'_{i_h})$, then we apply the rule $u \to v$. All objects to which the rule *can* be applied *must* be the subject of the rule application: all objects in u are "consumed", that is, the multiset identified by u is subtracted from the multiset $m(M'_{i_h})$. We examine the rules and assign objects to them. A rule can be used only if there are copies of the objects needed to apply such rule.

If in the rule $u \to v$ occurs the symbol δ in v, then the membrane i_h is dissolved. Simultaneously the set of rules R_{i_h} is removed. The objects from dissolved membrane are added to the region immediately external to dissolved membrane.

The rules associated to regions may be used concurrently, independent in all regions. It can be that at the same time two or more membranes are dissolving. This isn't a cause for any contradiction.

Thank the notion of configuration we can speak about a computing device: one can define the *computation* with respect to P system Π as a sequence of transitions $C_0 \Rightarrow C_1 \Rightarrow ... \Rightarrow C_m$, $m \ge 0$, where C_0 is the initial configuration. If in the configuration C_m there isn't a rule which can be applied to the objects in C_m, then the computation *halts*, and we say that it is *successful*. Otherwise we call the computation *unsuccessful*. This is in the case when there is a rule in C_m which allows to evolve to a configuration C_{m+1}, that is $C_m \Rightarrow C_{m+1}$. The configuration C_{m+1} can be equal to C_m.

If the output membrane $i_0 \in \{1,...,n\}$ belongs to the set of membranes of the system, then we say that system Π works in *internal mode* and it can be seen as a computing device which generates various objects:

- multisets – the multiset generated by system Π is the multiset contained in the output membrane in the configuration on which the system halts. This is the case when a successful computation has been occurred
- numbers – the number generated by the system is the sum of multiplicities of all objects in membrane i_0
- relations – each k-tuple $(n_1,..., n_k)$ is an element of a relation generated by system Π if for some initial objects $a_{i_1}, ..., a_{i_k}$ there is in the output membrane in the end of a successful computation n_1 copies of a_{i_1},..., n_k copies of a_{i_k} – in other words n_1 is the multiplicity of a_{i_1},..., n_k is the multiplicity of a_{i_k} in the output membrane after the successful computation.
- A *P*-system can also be seen as a device *recognizing* a multiset or a number or a relation. In this case a *P*-system works similar to a Turing machine or an automaton..

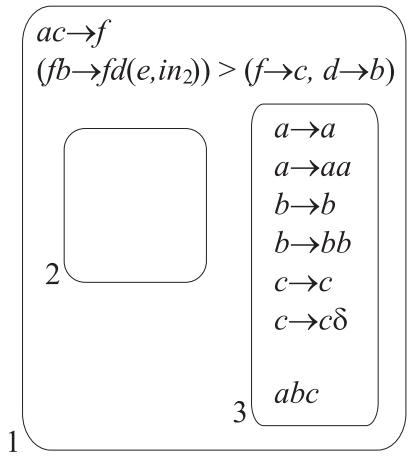
If $i_0 = \infty$, that means the "output membrane" is outside of the system, then it works in *external mode*: the object expelled through the skin membrane ordered in the order they are ejected, form a string. Some objects can be expelled at the same time. In such a case one take any permutation of them. So one can associate to any *P*-system a language. It will be denoted by $L(\Pi)$.

The biochemical interpretation of the previous description is the following.

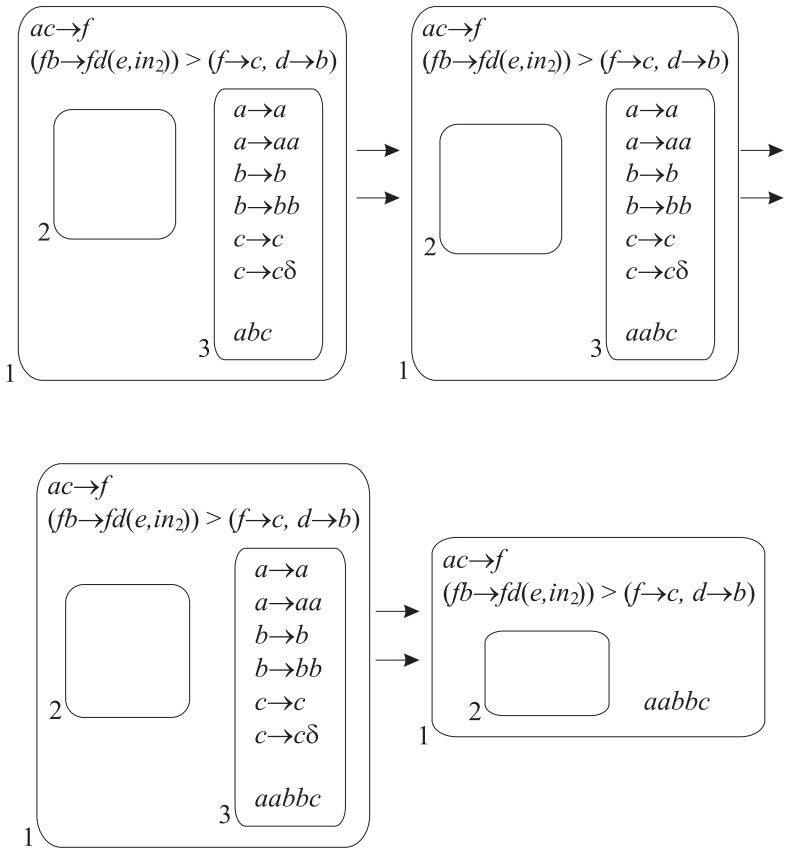
We have a cell, delimited by a skin (the cell membrane). Inside, we have cell-organs and free molecules, organized in a hierarchical way. The molecules and the organs float randomly in the "cytoplasmic liquid" of each membrane.

Under specific conditions, the molecules evolve (alone or with the help of certain catalysts). This is done in parallel, synchronously for all molecules (a universal clock is assumed to exist). The new molecules can remain in the same region where they have appeared, or can pass through the membranes delimiting this space, selectively. Some reactions modify not only the molecules but the membranes too; in particular certain chemical are produced which dissolve the membrane. When a membrane is broken, the molecules previously placed inside it will remain free in the larger space newly created, but the evolution rules of the former membrane are lost. If the external membrane is broken, then the cell ceases to exist.

An example of *P*-system given below comes from [11].



This system can evaluate for example as follows:



3. Preliminaries. Let T be a tree. By the labeling of T it is meant any function of the form l: $T \to A$ where A is a set. By an isomorphism of a tree $T = (T, \leq)$ into a tree $T' = (T', \leq')$ it is meant any function f:$T \to T'$ transforming the relation $\leq$ into $\leq'$. By an isomorphism of labeled tree $T = (T, \leq, l: T \to A)$ into $T' = (T', \leq', l': T' \to A)$ it is meant any isomorphism from $T = (T, \leq)$ into $T = (T, \leq)$ respecting the labeling of the trees that means $l'(f(x)) = l(x)$ for any $x \in T$. The notion of card of a labeled tree is defined analogously.

We except the standard terminology and notation for trees. Particularly for any element $a \in T$ the set of children of a is the set

$$children(a) = \{x \in T: x \angle a\}$$

and the set of parent of a is one-element set

$$parent(a) = \{ x \in T: a \angle x \}$$

where $\angle = \leq \setminus \leq^2$

Definition. The tree T_1 is longer (deeper) than T_2 if there exists a sequence $a_1 \angle a_2 \angle ... \angle a_n$ where $a_1,...,a_n \in T_1$ such that for all sequences of the form $b_1 \angle b_2 \angle ... \angle b_k$ where $b_1,...,b_k \in T_2$ it holds $n>k$.

By a labeled card we mean the card, say T_a, such that the parent of the element a has the information of the deleted element. By a labeled card we mean any card of the form $T_a = (T \setminus \{a\}, \leq | T \setminus \{a\}, parent(a))$. In what follows by a card we always mean a labeled card. The problem we solve in the paper is the reconstruction of a tree from two of its cards.

Remark The above notion may be seen a little artificial. Formally such a card is simple a pointed tree that means the tree with distinguished element. Similarly to the notions of pointed graph or pointed set. The only difference is that we do not consider the category of pointed trees, i.e. we do not need consider mappings of such trees respecting the points of them. We exploit this "pointing" in order to fix an information about the place, where the element of the original tree has been deleted.

By a part of a tree $(T, \leq)$ we mean any partial order of the form $(T', \leq | T')$ with $T' \subseteq T$.

Let T be a tree, $a \in T$. By $(a]$ we denote the ideal generated by the element a, that means the set $(a] = \{x \in T: x \leq a\}$. For any tree $\mathbb{T}$ and $a \in T$ by T_{-a} we mean the tree $\mathbb{T}_{-a} = (T \setminus (a], \leq | T \setminus (a])$. Let us note that T_{-a} is really a tree. For any tree T and an element $a \notin T$ the order $\mathbb{T}^{+a} = (T \cup \{a\}, \leq \cup \{(x, a): x \in T\})$. For any trees T, T' and any element $x \in T$ by $T[x]T'$ we mean the partial order $(T \cup (T' \setminus root(T')), \leq \cup (T' \setminus root(T')) \times \{x\})$.

Fact. For any trees T, T' and an element $x \in T$ the structure $T[x]T'$ is a tree.

By a forest F we mean a set of trees. Two forests F_1 and F_2 are isomorphic, denoted $F_1 \cong F_2$ if there exists a bijection f: $F_1 \to F_2$ such that for each $T \in F_1$ it holds $T \cong f(T)$, that means corresponding trees are isomorphic.

In the paper we use the notion of *level* and *level of an element*. The second one can be defined recursively.

1. $level(\text{root}) = 0$
2. $level(x) = level(parent(x))+1$

The *level* is the set of elements x, for which the function $level(x)$ has the same value. The elements in the first level, that means the children of root, will be called *antiatoms*. For each element $x \neq$ root we can find its antiatom, that is such an antiatom y, that $x \leq y$, thus the function $antiatom(x)$ is well defined for all element of the tree not equal to the root.

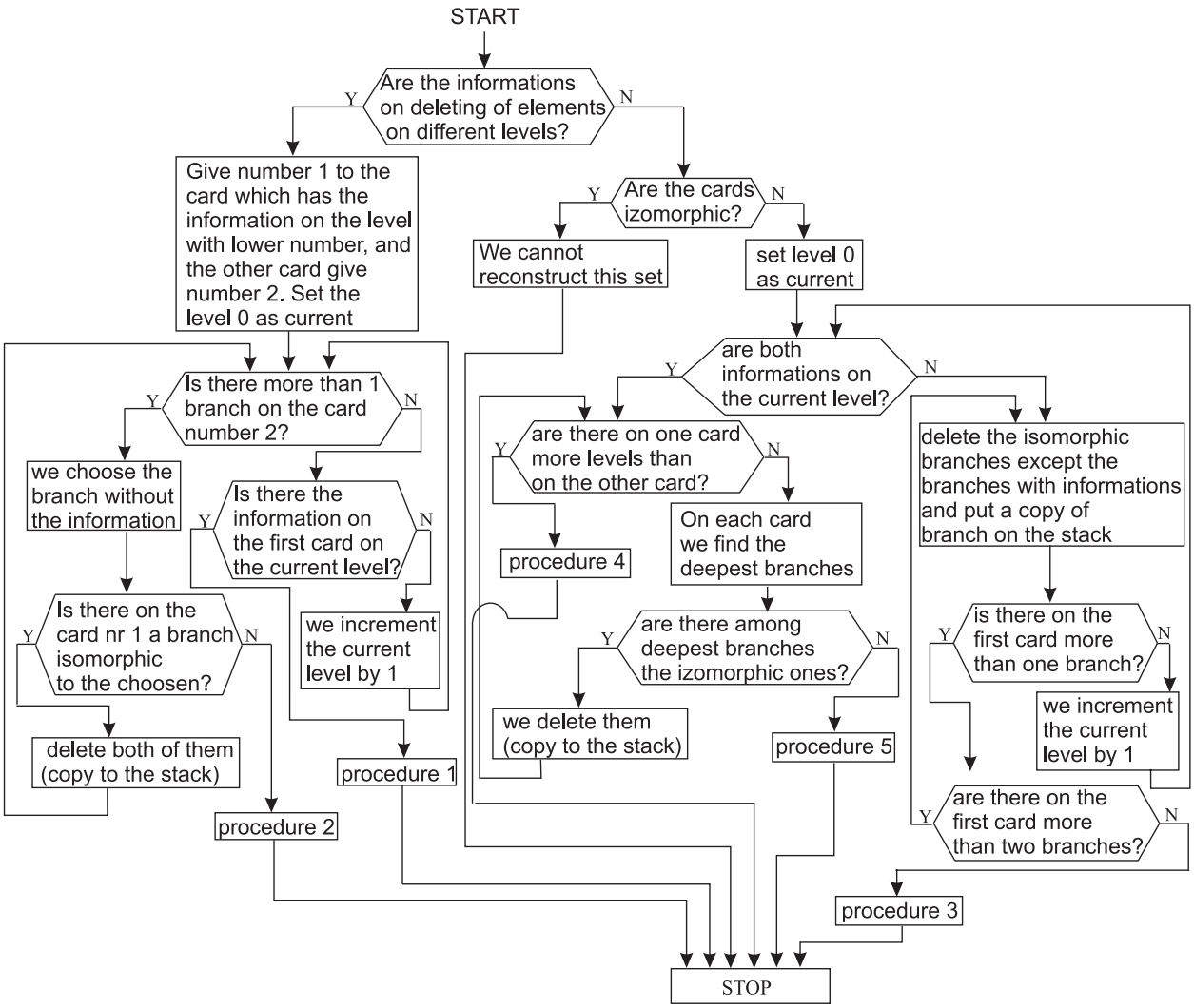
4. A reconstruction of a Paun's system from two its states. We prove that any structure of Paun's system can be reconstructed from their two cards. In other words we show that double using of the rule of the form $u \to x\delta y$ (δ is the special symbol which denotes dissolving of a membrane) allow to reconstruct the original state of a P-system.

The main result of the paper is the following proposition.

Proposition Let $T = (T, \leq)$ be a tree and $a, b \in T$. Given non-isomorphic[2] cards T_a and T_b we can reconstruct the original tree T.

The sketch of the proof

The proof is effective. The block scheme of the corresponding algorithm is given below.



In what follows the card T_a and T_b will be numbered; roughly speaking the card T_x ($x = a, b$) obtains the number 1 if x is nearer the root then the second element of the set $\{a, b\}$. In other words T_a obtains number 1 and T_b number 2 if level a is higher then level of b. If both a and b are on the same level, the numbering of cards is irrelevant. Let us describe the procedures used in the algorithm.

5. Procedures descriptions

5.1. Procedure 1 (used in the case when the deleted elements were on different levels, and there is only one branch on the card nr 2)

This is the case in which both deleted elements were comparable. In the first step we find and delete (the copy we put on the stack) the isomorphic branches on both trees. The procedure below describes the searching of isomorphic branches and putting one of them on the stack. B and C are variables running the set of branches, that means the ideals generated by antiatoms. T_1 and T_2 are the trees from the first and the second card respectively. s is the stack on which we put the branches.

[2] The case when the cards are isomorphic is the case in which actually we have to reconstruct the tree from one card. We can do it uniquely when the information is in a leaf. When it isn't – the reconstruction is not unique – there can exists many non-isomorphic trees which are "reconstructions"

procedure *find_isomorphic_branches* (T_1, T_2, s)
while exist $B \le T_1$ & $C \le T_2$ & $B \cong C$ do
begin
$T_1 := T_1 \setminus B$;
$T_2 := T_2 \setminus C$;
put (s, B)
end

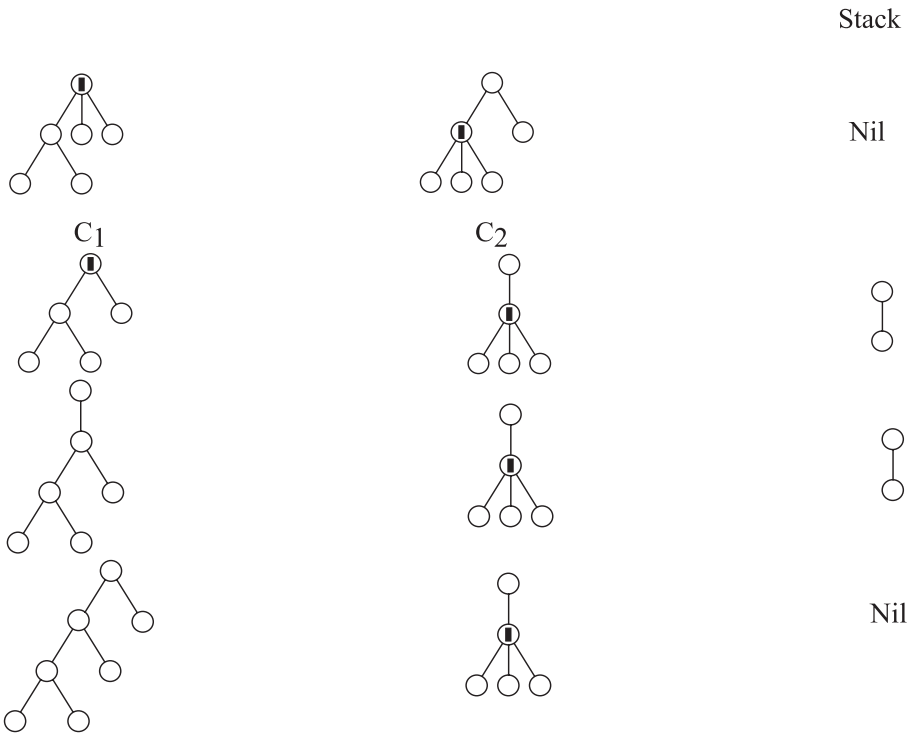
When the procedure stops, on the tree T_2 there is only one antiatom and almost whole tree (almost – because without a root) is the ideal generated by this antiatom. In the next step we add a new element to the tree T_1 from the first card. Let i be the element with information on the tree T_1 and $x \notin T_1$ a new element. We add the element x to the tree T_1 in following way: $\mathbb{T}_1^0 := (T_1 \cup \{x\}, \le \cup (y, x) \cup (x, \text{root}))$ for each element $y \ne$ root. $\mathbb{T}_1^0$ is the tree obtained in this way from the tree $\mathbb{T}_1$.

In the last step of the procedure 1 we add the branches deleted as isomorphic from the file s to the tree:

while *stack_pointer* ≠ EOF do $T_1^0 := T_1^0 + get(s)$.

T_1^0 and all trees isomorphic to this tree are now the reconstruction of the tree.

An example of using procedure 1 illustrates a figure below



5.2. Procedure 2. In the first step we find and delete (copying one of them to a file) the isomorphic branches like in procedure 1. The corresponding procedure is the procedure *find_isomorphic_branches*. Let us note that when we start the searching of the isomorphic\ branches, we should start from the card number 2. However the conjunction "&" is commutative, but in the second card there is the original number of branches (that means the same number like in the tree we try to reconstruct) and in the tree on the first card there can be more than the original number of branches and we can find more than one branch without the information which hasn't the corresponding isomorphic branch on the second card. Fortunately the computer executes the conjunction sequentially and it can first get a branch from the second card and next search for the branch isomorphic to it on the first card. After the procedure *find_isomorphic_branches* on the tree from the second card there should be only two branches: one with an element with information and the second without such an element.

Let i be the element with information on the card *C1* and j the element with information on card *C2*. Let *level*(x) denotes the level on which there is the element x. In the next step we consider forests of the form (a] \ {a}, where (a] is an ideal generated by element a.

1) delete the branch (*antiatom*(j)]

2) mark the elements x on the card *C2* for which *level*(x) = *level*(i)

3) procedure *isomorphic forests* (F_2, x_k, X)

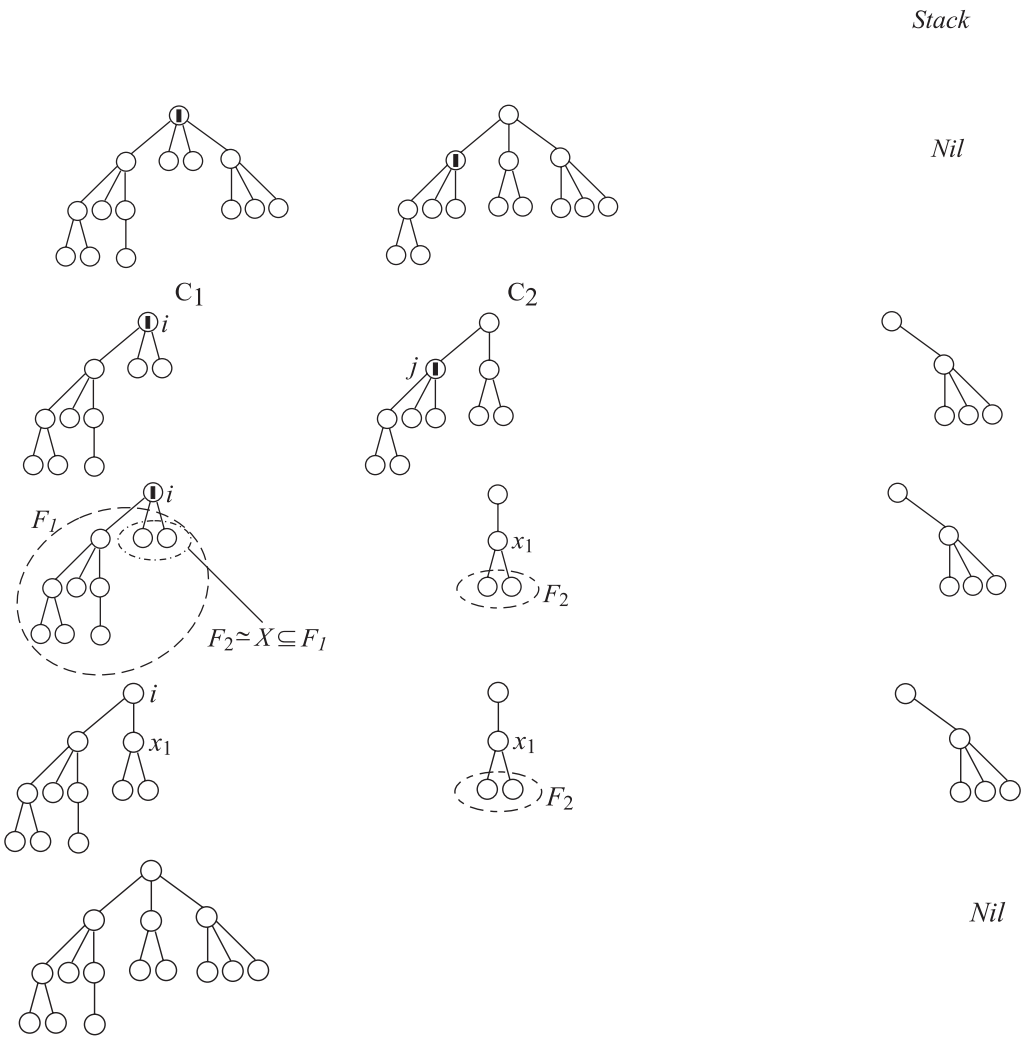
```
begin
        F1:=(i] \ {i};
                repeat
                        k:= 1;
                        F2:= (xk] \ {xk};
                        k:= k+1;
                until F2 \simeq ⊆ F1;
end
```

4) T_1:= T_{1} \ X+(x_k]↑ i, where (x_k]↑ i means that we add to the relation ∠ on the tree T_1 the pair (x_k, i) and all elements belonging to x_k.

5) we add the branches deleted as isomorphic from the file s to the tree T_1 like in last step in procedure 1.

The tree T_1 and each isomorphic to it is now the reconstruction of the tree.

On the picture below one can see the case in which procedure 2 is executed



5.3. Procedure 3. This procedure is used in the case when the elements with information are on the same level but not on zero level – the informations are not in roots. The loop occurring in block scheme leads to the situation, we have to consider two trees or subtrees on cards C_1 and C_2 (the cards are numbered arbitrarily in this case) in which we have two branches: one with the information and the other without it. In the loop the word "branch" should be understood as a branch in the subtree where the root is the element on the current level. The loop uses procedure *find_isomorphic_branches* described above. Let us mark the two new cards C_1^0 and C_2^0 and the corresponding trees T_1^0 and T_2^0. Let us mark the element with information on the tree T_1^0 by *i* and the element with information on the tree T_2^0 by *j*. Next we construct the tree:

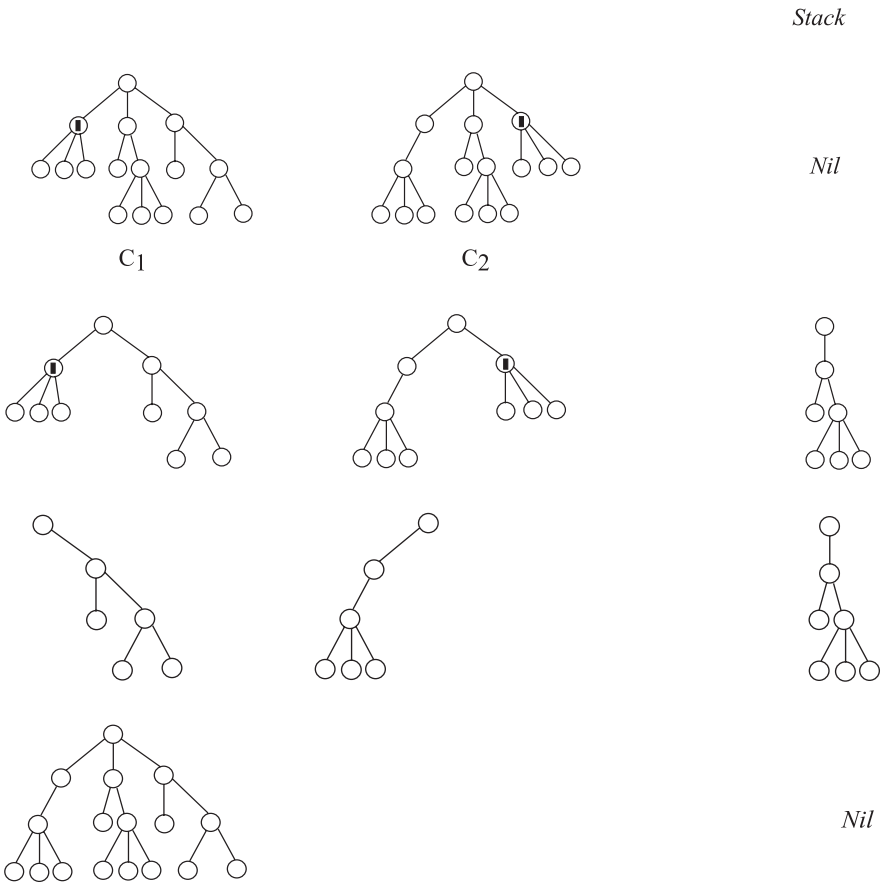
$T_1^0 := T_1^0 \setminus (antiatom(i)]$;

$T_2^0 := T_2^0 \setminus (antiatom(j)]$;

$T_1^0 := T_1^0 + T_2^0$

The last operation + is a simple sum of trees. The added trees T_1^0 and T_2^0 have different roots. We identify the roots (they are stickled together) but don't identify any other element of the trees. In the next step we add the deleted branches. The procedure which gets the branches from a file can be executed more than once in the sense that the branches can be remembered in more than one file: it can be that the branches are added not only to the root of the tree but also to the roots of subtrees considered during running of the loop from block scheme.

The procedure 3 is used in situation like in the example below



5.4. Procedure 4. The loop defined in the description of the procedure 3 can be executed before the execution of the procedure 4. Thus the notions of root and branch are relative, in the sense that the root or a branch can be the toot or a branch of a subtree, not necessarily of the whole tree. When we start with

procedure 4, we have two cards C_1^0 and C_2^0 and the corresponding trees T_1^0 and T_2^0 on the cards. One of the tree is deeper and this will be called T_1^0 and the second one T_2^0.

The elements with informations are on the same level on both cards and the informations can be in roots (of considered trees – maybe subtrees of whole trees).

1) find the longest branch on the tree T_1^0. By a longest branch it is meant a subtree constructed as follows. Search the longest sequence of the form $\$a_1 \angle a_2 \angle \ldots \angle a_n$ where $a_1, a_2, \ldots a_n \in T_1^0$. The longest branch is $(a_n]$ – the ideal generated by a_n.

2) the longest branch in T_1^0 is original (in T_1^0, because the whole branch in which, probably, the longest branch in T_1^0 is a subtree, may be not original. The element with information can be in this whole branch). Delete all other branches from T_1^0.

$T_1^0 := (antiatom(a_n)] \cup \text{root}(T_1^0)$

3) Let us assume that i is the element with information in the tree T_2^0. There should be only one element, say β, on the tree T_1^0 on the level $level(i)+1$ such that after deleting this element β the tree T_1^0 get shorter. Find such an element β. In order to find this element we can for example execute a procedure like this:

$x := a_n$ (we understand a_n like in point 1) and 2))
while $level(x) > level(i)+1$ do
begin $x := parent(x)$ end
$\beta := x$

4) delete the element β and set $T_1^1 := T_1^0{}_\beta$.

5) check if $F_1 = (\text{root}(T_1^1)] \setminus \{\text{root}(T_1^1)\} \cong X \subseteq F_2 = (\text{root}(T_2^0)] \setminus \{\text{root}(T_2^0)\}$

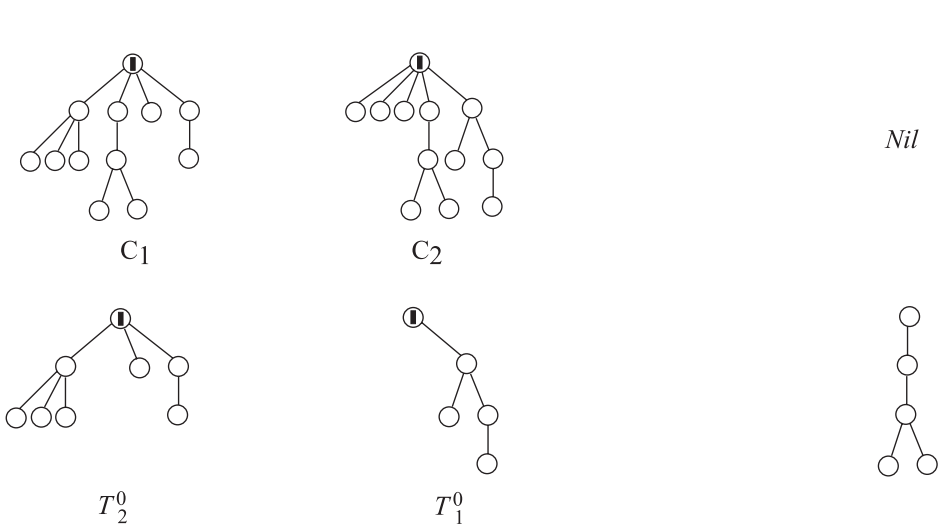
6) if so, $T_2^0 := T_2^0 \setminus X + T_1^0$ where operation + is like in procedure 3). The tree T_1^0 is with element β. Among the trees constructed in this procedure only the tree T_1^1 is without this element (in fact the tree T_2^0 before the last operation was without the element β. β is the failed element in card C_2).

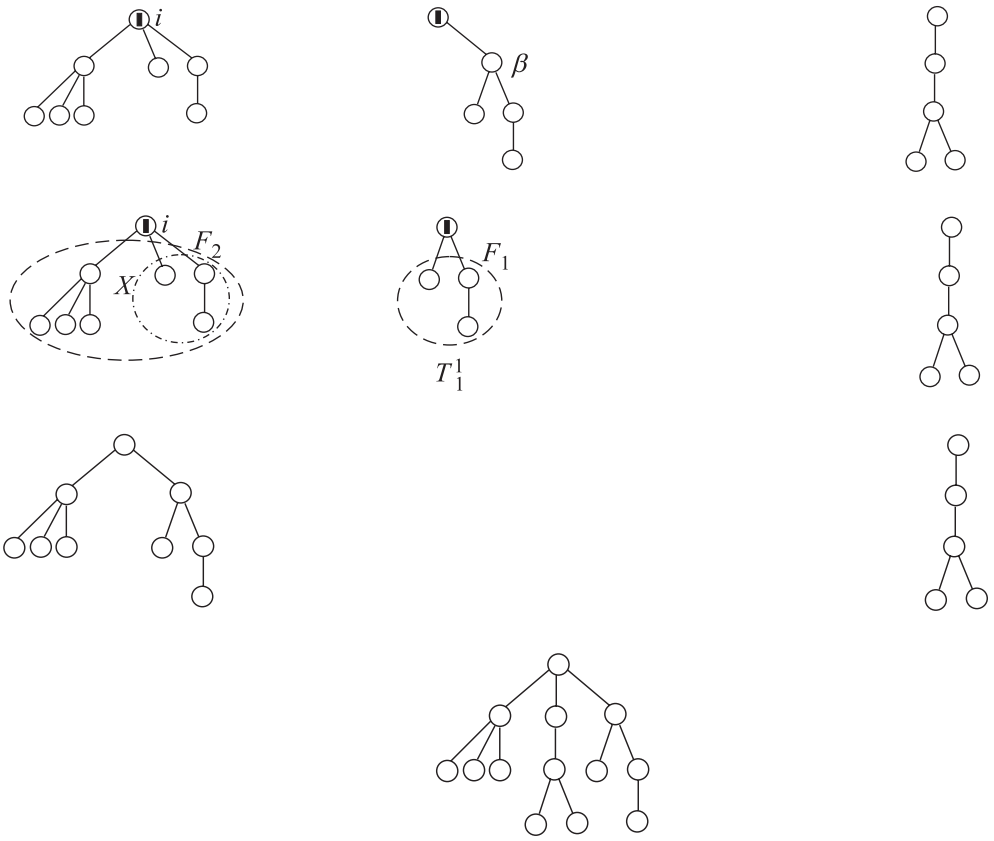
7) if not – it is an error.

8) we add all the branches deleted as isomorphic – like in procedure 3)

9) T_2^0 and all trees isomorphic to them should be the wanted reconstruction.

An example of using procedure 4 is illustrated in a figure below



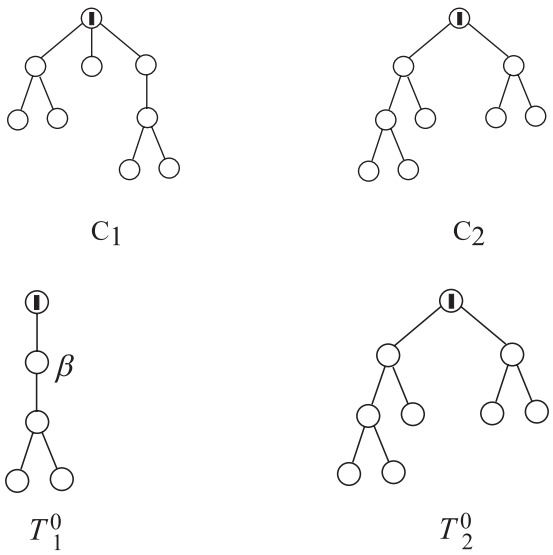


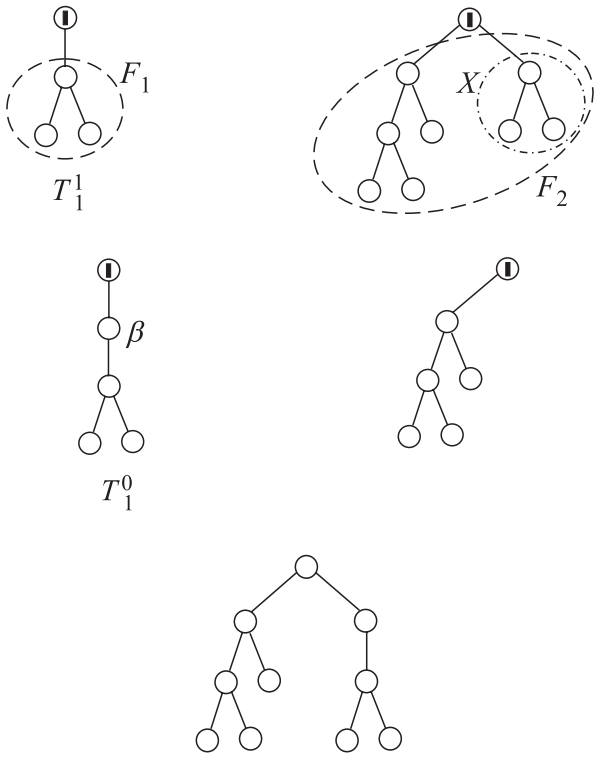
5.5. Procedure 5. Before using the procedure 5 we can use the procedure *find_isomorphic_branches*, but the set which is run by variables *B* and *C* is the set of branches of maximal length (the set can change during executing of corresponding loop). Corresponding loop is illustrated on the block scheme. This operation is not yet in proper procedure 5.

In the first step of procedure 5 we find on one of cards the deepest branch. Let us denote the element on the deepest level by a_n like in procedure 4.

We construct the tree T_1^0= (antiatom(a_n)] ∪ root(T_1). The further steps are like in procedure 4 (points 3) – 8)). We can see it on the picture-example.

The example of using procedure 5 is illustrated in the figure below





6. Sketches of proof in various cases

6.1. The case of procedure 4. If the tree on one of the cards is deeper (has more levels than the other tree) than the tree on the other card, then the difference between the depths is one. If the difference was greater than one, then we would have not the cards, that would mean the two trees hadn't been obtained from the same tree by deleting one element. When we delete one element the tree can get shorter and the number of levels decreases at one or the depth of tree can stay not changed – when deleted element was not on the "longest path". Since we consider a subtree where the root is the element with information (it can be the whole tree, if the information is in the level zero, that means in the root of the tree), the situation is like this, when we delete an antiatom (element on the first level) from the longest branch and in this case the number of levels of the tree get decreases at one, thus the longest branch on the card with the deeper tree is original (the branch is also in the original tree). If it wasn't so, that would mean, that this branch originates by deleting an atom in considered subtree and that would mean that in original tree existed a branch longer at least two levels than the tree on the second card. It is impossible, because the second card is originated from the original tree by deleting exactly one element and such a deleting can make the tree at most one level shorter (less deep).

6.2. When the deleted elements are comparable. I. It is sufficient to consider the case, when one of information is on the zero-level (in the root) and the second on the level of greater number. If one of the information is on the non-zero level, say level k and the second on the level l where $l > k$ then we consider a subtree, in which the root is this element on the level k in which we have the information. The remaining part of the tree is not changed.

II. On the card with number two the levels 0,1,...,l are not changed – they are like in original tree, because the zero level cannot change (one of foundation of these consideration) and when something is changed on the level i, we have the information on the level i-1, and the information is not before level l.

1. The branches without information on the second card are like in original tree, thus the branches on the first card isomorphic to them are like in original tree, too.

2. The subtree (or fragment – for example a tree $(T \setminus (a] \setminus (b] \setminus \ldots$ for some elements a, b,...), which remain on the first card after deleting isomorphic branches on both cards, has one level less than in original

tree, because the deleted element is an atom: if it had some elements under, then now they are all one level higher (the level with number one less than their original level), else on the first card we have only the root (in this case).

3. In order to return to the original tree we have to add an atom, and all the elements from the first card, except the root, should go one level lower.

4. On the first card, after the deleting of isomorphic branches, there aren't, except the root, elements which are on their original level, especially the elements on the first level are not original atoms: if they were, they would be also on the second card, where weren't changes on the first level. But now, after the deleting of isomorphic branches, on the first level in the second card is only one element, and this element is deleted in the first card – we know that, because we have an information in the root.

6.3. Other cases. I. After the deleting of isomorphic branches remain two branches on the first and two branches on the second card (the deleted elements are not comparable), their lowest upper bound is the root or an element different from the root – in the last case we consider a subtree, which root is this lowest upper bound.

This can be in the case, when elements are deleted on the same level or in different levels. This proof is not correct for the case, when at least one of elements of the first level (one of antiatoms) is deleted.

1. If the antiatoms are not deleted, then on the first, and on the second card also, is the same number of branches like in original tree.

2. On each card only one branch is changed, because only one element is deleted, then on each card only one of branches can be different from original. When we have the information on the first or next levels we know which branch is changed (namely this one in which is the information).

3. One of the two branches, which remain after the deleting of isomorphic branches, on the first card is original, the same situation is on the second card.

4. The branch which is changed on the first card is this which is original on the second card. If it wasn't so, the same branch would have to be changed in both cards. The others would be not changed, thus isomorphic on both cards. But we have on each card two branches, which haven't isomorphic branch on the other card.

II. Not comparable elements from different levels, among them one from the first level, are deleted

1. After the deleting of isomorphic branches from both cards, on the second card (this is the card, on which the information is on the non-zero level) remain two branches, among them one without the information and this is an original branch.

2. On the first card there isn't the branch isomorphic to the original branch from the second card, but there is a fragment (usually it isn't a subtree, it is a branch or a few branches, we can see it as a forest) which originates after deleting an antiatom from the original branch from the second card.

3. When we delete this fragment, in the first card will remain one branch. It is original. If it wasn't original, that means if one element (antiatom) was deleted from it, then the original branch from the second card should have the isomorphic branch in the first card, because only one element was deleted. But there isn't such an isomorphic branch.

4. We know, which branches are original (one from the first card, and one from the second). We add the branches and the branches from the stack to root and we have the reconstruction of the tree.

III. The elements are deleted from the same level and they have the same parent. This case includes the situation when the parent is the root and the situation when the parent is an element different from the root – we consider then the subtree in which the root is this parent, because the remaining part of the tree is not changed

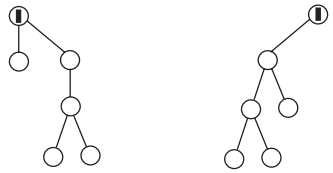
1. In the case, when on the cards are different numbers of levels, we prove the reasoning like in case of procedure 4.

2. When on both cards is the same number of levels, we check in turn the branches which have the greatest depth. If we find for a deepest (or one of deepest) branch from card 1 an isomorphic branch on the card 2, the branches are original, because if one of them originates by deleting an atom, then on the other card should be a branch one level deeper.

3. If we have a branch which has the greatest number of levels, say on the card *A*, and there isn't any branch isomorphic to its on the other card (say *B*), then this branch is original, and on the second card (*B*) should be a fragment (usually branch or a few branches) isomorphic to the fragment (a branch or few branches also) which we obtain when delete the atom from this original branch. Then we delete this fragment from the card *B* and we put into the place the original branch from card *A*.

4. If all the deepest branches on the first card have branches isomorphic to them on the second card, we delete them (copy to the stack) and again search the branches which now have the greatest number of levels. The tree is finite, thus we should find finally a branch which hasn't isomorphic branch on the other card. If we don't find such a branch, that means the card are isomorphic and we cannot uniquely reconstruct the tree.

There is a question: why in the last case we start the searching with deepest branches? If we deleted all isomorphic branches, we could go to bad results. The cause is among other that some branches which are in cards, aren't in original tree. Some of these branches can be isomorphic to original branches on the other cards and we can delete an original branch in one card and an isomorphic, but not original branch in the other card. In the picture illustrating the using of procedure 5 we cannot delete the isomorphic branches. If we deleted them, we obtain two trees like below



It is probably very difficult to reconstruct the original tree using the above cards. But if we start with situation like in the first picture and if we start with the deepest branches, we can reconstruct the tree without problems.

7. An implementation. In this section a way of using of the program (build by the second of the authors) will be, very roughly, described. The program reads data containing a description of two cards and produces the original tree. The best way to use this program is to prepare a file with data (two cards) and specify it as an input of the executing file (the name of executing file is *treerec.exe*)

>treerec data (> means prompt)

The *data* file is the text file where we prepare the two cards. It should be done as follows:

number of elements of the card

(empty line)

card1 (name of card 1)

number-of-vertex [space] number-of-its-parent (if root then doesn't occur) [space] i (only by the vertex with information – one vertex in one card)

We explain the way of using of the program by an example. The data file can see as follows:

```
6

Card1
0
1 0
2 0
3 0 i
4 2
5 2

Card2
0
```

1 0
2 0 i
3 0
4 2
5 3

The last line must be empty, like the lines which separate the cards and the number of elements and the cards. The program will print the reconstruction tree (if in data file are really the non-isomorphic cards) on the screen in the same form as in data file. The program one can get from

http://www.pu.kielce.pl/~sylwaw.

8. Concluding remarks. The problem of reconstruction of an object from information about a kind of its reducts occurs very often in many areas of life. Probably the most well known is the problem of reconstruction of scenario of a crime act solved by every investigation done by the police or other similar institution. But the problem of reconstruction is also of higher important in many branches of biology or medicine where we try to inference some properties of the actual state of a patient from some partial informations on his state in the past. We don't know the biological or medical "roots" of such investigations but we know that they have been taken up. This paper originates from simple notification that the well known reconstruction problem for partial order has a very easy and natural interpretation in the theory of Paun's systems. Some other consequences of these comparison will be described in further papers.

1. Adleman L.M. Molecular computation of solutions to combinatorial problems. Science, 226 (1994). – S. 1021–1024. 2. Beaver D. Factoring: The DNA Solution, ASIACRYPT 1994. – S. 419–423. 3. Beaver D., Computing with DNA, Journal of Computational Biology 2(1). 1–7 (1995). 4. Bondy J.A., Hemminger R.L. Graph reconstruction – a survey, Journal of Graph Theory 1 (1977). – P. 227–268. 5. Boneh D., Dunworth C., Lipton R., Sgall J. On the computational power of DNA, In Discrete Applied Mathematics, Special Issue on Computational Molecular Biology. Vol. 71 (1996). – P. 79–94. 6. Boneh D., Dunworth C., Lipton R. Breaking DES using a molecular computer, In Proceedings of DIMACS workshop on DNA computing, 1995, published by the AMS. 7. Boneh D., Dunworth C., Lipton R. Making DNA computers error resistant, In proceedings of second annual DIMACS conference on DNA computing. – 1996. 8. Kelly P.J. A congruence theorem for trees, Pacific Journal of Matematics 7. – 1957. – P. 961–968. 9. Kratsch D., Rampon J.-X. Towards the Reconstruction of Posets}, Order 11, 317-341, Kluwer Academic Publishers. – 1994. 10. Lipton R.J., Speeding up computations via molecular biology, draft, Princeton University. – December 1994. 11. Madhu M. Studies of P Systems as a model of cellular computing, Department of Computer Science and Engineering, Indian Institute of Technology Madras, Ph. D. Thesis, 2003. 12. van Oostrom, V., Confluence for Abstract and Higher-Order Rewriting, Academic Proefschrift, Vrije Universiteit te Amsterdam, Ph.D. Thesis. 13. Păun, Gh., Computing with Bio-Molecules, Springer Verlag, 1998. 14. Păun, Gh., Computing with membranes, Journal of Computer and System Sciences, 61, 1, 108–143, 2000. 15. Rampon J.-X. Reconstruction of finite ordered sets, "Ordered Sets", Warsaw, 19–31 July 1999. 16. Schröder B.S.W., Examples on ordered set reconstruction, Order 19, nr 3, 283–294, Kluwer Academic Publishers, 2002. 17. Zandron, C., A Model for Moloecular Computing: Membrane Systems, Università degli Studi di Milano, Dipartimento di Informatica, Dottorato di Ricerca in Informatica – XIIICiclo, Ph.D. Thesis