# GENERALIZED P-SYSTEMS

Rudolf FREUND
Institut für Computersprachen, Technische Universität Wien,
Resselg. 3, A-1040 Wien, Austria
email: rudi@logic.at, tel.: ++43 1 58801 18542

**Abstract.** We consider a variant of P-systems, a new model for computations using membrane structures and recently introduced by Gheorghe Păun. Using the membranes as a kind of filter for specific objects when transferring them into an inner compartiment turns out to be a very powerful mechanism in combination with suitable rules to be applied within the membranes. The model of generalized P-systems, GP-systems for short, considered in this paper allows for the simulation of graph controlled grammars of arbitrary type based on productions working on single objects; for example, the general results we establish in this paper can immediately be applied to the graph controlled versions of context-free string grammars, $n$-dimensional #-context-free array grammars, and elementary graph grammars.

## 1 Introduction

One of the main ideas incorporated in the model of P-systems introduced in [7] is the membrane structure (for a chemical variant of this idea see [1]) consisting of membranes hierarchically embedded in the outermost *skin* membrane. Every membrane encloses a *region* possibly containing other membranes; the part delimited by the membrane labelled by $k$ and its inner membranes is called *compartiment* $k$. A region delimited by a membrane not only may enclose other membranes but also specific objects and operators, which in this paper are considered as multisets, as well as evolution rules, which in *generalized P-systems (GP-systems)* as introduced in this paper are evolution rules for the operators. Moreover, besides ground operators the most important kind of operators are transfer operators (simple rules of that kind are called travelling rules in [10]) allowing to transfer objects or operators (or even rules) either to the outer compartiment or to an inner compartiment delimited by a membrane of specific kind with also checking for some permitting and/or forbidding conditions on the objects to be transferred (in that way, the membranes act as a filter like in test tube systems, see [8]). In contrast to the original definition of P-systems we do not demand all objects to be affected in parallel by the rules; the proofs of the results established so far in various papers on P-systems, see [3], [6], and [9], show that only bounded parallelism is needed. Moreover, we also omit the feature of priority relations on the rules, because this feature can be captured in another way by using the transfer conditions in the transfer operators.

In the following section we shall give a general defintion of a grammar and then we define the notions of matrix grammars and graph controlled grammars in

this general setting. In the third section we introduce our model of GP-systems, and in the fourth section we establish our main results showing that GP-systems allow for the simulation of graph controlled grammars of arbitrary type based on productions working on single objects; we also elaborate how these results can be used to show that graph controlled context-free string grammars, graph controlled $n$-dimensional #-context-free array grammars, and graph controlled elementary graph grammars can be simulated by GP-systems using the corresponding type of objects and underlying productions. An outlook to future research topics concludes the paper.

## 2 Definitions

First, we recall some basic notions from the theory of formal languages (for more details, the reader is referred to [2]).

For an alphabet $V$, by $V^*$ we denote the free monoid generated by $V$ under the operation of concatenation; the *empty string* is denoted by $\lambda$, and $V^* \setminus \{\lambda\}$ is denoted by $V^+$. Any subset of $V^+$ is called a *$\lambda$-free (string) language*.

A *(string) grammar* is a quadruple $G = (V_N, V_T, P, S)$, where $V_N$ and $V_T$ are finite sets of *non-terminal* and *terminal symbols,* respectively, with $V_N \cap V_T = \emptyset$, $P$ is a finite set of *productions* $\alpha \to \beta$ with $\alpha \in V^+$ and $\beta \in V^*$, where $V = V_N \cup V_T$, and $S \in V_N$ is the *start symbol.* For $x, y \in V^*$ we say that *$y$ is directly derivable from $x$ in $G$*, denoted by $x \Longrightarrow_G y$, if and only if for some $\alpha \to \beta$ in $P$ and $u, v \in V^*$ we get $x = u\alpha v$ *and* $y = u\beta v$. Denoting the reflexive and transitive closure of the derivation relation $\Longrightarrow_G$ by $\Longrightarrow_G^*$, the *(string) language generated by $G$* is $L(G) = \{w \in V_T^* \mid S \Longrightarrow_G^* w\}$. A production $\alpha \to \beta$ is called *context-free,* if $\alpha \in V_N$.

In order to prove our results in a general setting, we use the following general notion of a grammar:

A *grammar* is a quadruple $G = (B, B_T, P, A)$, where $B$ and $B_T$ are sets of *objects* and *terminal objects,* respectively, with $B_T \subseteq B$, $P$ is a finite set of *productions,* and $A \in B$ is the axiom. A production $p$ in $P$ in general is a partial recursive relation $\subseteq B \times B$, where we also demand that the domain of $p$ is recursive (i.e., given $w \in B$ it is decidable if there exists some $v \in B$ with $(w, v) \in p$) and, moreover, that the range for every $w$ is finite, i.e., for any $w \in B$, $card\,(\{v \in B \mid (w, v) \in p\}) < \infty$. As for string grammars above, the productions in $P$ induce a derivation relation $\Longrightarrow_G$ on the objects in $B$ etc. The *language generated by $G$* is $L(G) = \{w \in B_T \mid A \Longrightarrow_G^* w\}$.

For example, a string grammar $(V_N, V_T, P, S)$ in this general notion now is written as $\left((V_N \cup V_T)^*, V_T^*, P, S\right)$.

### 2.1 Control mechanisms

In the following, we give the necessary definitions of matrix and graph controlled grammars in our general setting. For detailed informations concerning these

control mechanisms as well as many other interesting results about regulated rewriting in the theory of string languages, the reader is referred to [2].

A *matrix grammar* is a construct $G_M = (B, B_T, (M, F), A)$ where $B$ and $B_T$ are sets of *objects* and *terminal objects*, respectively, with $B_T \subseteq B$, $A \in B$ is the axiom, $M$ is a finite set of matrices, $M = \{m_i \mid 1 \leq i \leq n\}$, where the matrices $m_i$ are sequences of the form $m_i = (m_{i,1}, \ldots, m_{i,n_i})$, $n_i \geq 1$, $1 \leq i \leq n$, and the $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, are productions over $B$, and $F$ is a subset of $\bigcup_{1 \leq i \leq n, 1 \leq j \leq n_i} \{m_{i,j}\}$.

For $m_i = (m_{i,1}, \ldots, m_{i,n_i})$ and $v, w \in B$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \ldots, w_{n_i} \in B$ such that $w_0 = v$, $w_{n_i} = w$, and for each $j, 1 \leq j \leq n_i$,

- **either** $w_j$ is the result of the application of $m_{i,j}$ to $w_{j-1}$,
- **or** $m_{i,j}$ is not applicable to $w_{j-1}$, $w_j = w_{j-1}$, and $m_{i,j} \in F$.

The language generated by $G_M$ is

$$L(G_M) = \{w \in B_T \mid A \Longrightarrow_{m_{i_1}} w_1 \ldots \Longrightarrow_{m_{i_k}} w_k = w, \; w_j \in B, \, m_{i_j} \in M \;\; \text{for } 1 \leq j \leq k\}.$$

If $F = \emptyset$ then $G_M$ is called a *matrix grammar without appearance checking.* $G_M$ is said to be of type $X$ if the corresponding underlying grammar $G = (B, B_T, P, A)$, where $P$ exactly contains every production occuring in some matrix in $M$, is of type $X$.

A *graph controlled grammar* is a construct $G_C = (B, B_T, (R, L_{in}, L_{fin}), A)$; $B$ and $B_T$ are sets of *objects* and *terminal objects*, respectively, with $B_T \subseteq B$, $A \in B$ is the axiom; $R$ is a finite set of rules $r$ of the form $(l(r) : p(l(r)), \sigma(l(r)), \varphi(l(r)))$, where $l(r) \in Lab(G_C)$, $Lab(G_C)$ being a set of labels associated (in a one-to-one manner) to the rules $r$ in $R$, $p(l(r))$ is a production over $B$, $\sigma(l(r)) \subseteq Lab(G_C)$ is the *success field* of the rule $r$, and $\varphi(l(r))$ is the *failure field* of the rule $r$; $L_{in} \subseteq Lab(G_C)$ is the set of initial labels, and $L_{fin} \subseteq Lab(G_C)$ is the set of final labels. For $r = (l(r) : p(l(r)), \sigma(l(r)), \varphi(l(r)))$ and $v, w \in B$ we define $(v, l(r)) \Longrightarrow_{G_C} (w, k)$ if and only if

- **either** $p(l(r))$ is applicable to $v$, the result of the application of the production $p(l(r))$ to $v$ is $w$, and $k \in \sigma(l(r))$,
- **or** $p(l(r))$ is not applicable to $v$, $w = v$, and $k \in \varphi(l(r))$.

The language generated by $G_C$ is

$$L(G_P) = \{w \in B_T \mid (A, l_0) \Longrightarrow_{G_C} (w_1, l_1) \Longrightarrow_{G_C} \ldots (w_k, l_k), \; k \geq 1, \; w_j \in B \;\text{ and }\; l_j \in Lab(G_C) \;\; \text{for } 0 \leq j \leq k, \; w_k = w, \; l_0 \in L_{in}, \; l_k \in L_{fin}\}.$$

If the failure fields $\varphi(l(r))$ are empty for all $r \in R$, then $G_C$ is called a *graph controlled grammar without appearance checking.* $G_C$ is said to be of type $X$ if the corresponding underlying grammar $G = (B, B_T, P, A)$, where $P = \{p(q) \mid q \in Lab\}$, is of type $X$.

## 3 Generalized P-systems (GP-systems)

In this section we quite informally describe the model of generalized P-systems discussed in this paper. Only the features not captured by the original model of P-systems as described in [6] and [7] will be defined in more details.

The basic part of a (G)P-system is a *membrane structure* consisting of several membranes placed within one unique surrounding membrane, the so-called skin membrane. All the membranes can be labelled in a one-to-one manner by natural numbers; the outermost membrane (skin membrane) always is labelled by 0. In that way, a membrane structure can uniquely be described by a string of correctly matching parentheses, where each pair corresponds to a membrane. For example, the membrane structure depicted in Figure 1, which within the skin membrane contains two inner membranes labelled by 1 and 2, is described by $[_0[_1]_1[_2]_2]_0$. Figure 1 also shows that a membrane structure graphically can be represented by a Venn diagram, where two sets can either be disjoint or one set be the subset of the other one. In this representation, every membrane encloses a *region* possibly containing other membranes; the part delimited by the membrane labelled by $k$ and its inner membranes is called *compartiment* $k$ in the following. The space outside the skin membrane is called *outer region.*

Informally, in [6] and [7] *P-systems* were defined as membrane structures containing multisets of objects in the compartiments $k$ as well as evolution rules for the objects. A priority relation on the evolution rules guarded the application of the evolution rules to the objects, which had to be affected in parallel (if possible according to the priority relation). The output was obtained in a designated compartiment from a halting configuration (i.e., a configuration of the system where no rules can be applied any more).

A *generalized P-system (GP-system) of type* $X$ is a construct $G_P$ of the following form:

$$G_P = (B, B_T, P, A, \mu, I, O, R, f)$$

where

- $(B, B_T, P, A)$ is a grammar of type $X$;
- $\mu$ is a membrane structure (with the membranes labelled by natural numbers $0, ..., n$);
- $I = (I_0, ..., I_n)$, where $I_k$ is the initial contents of compartiment $k$ containing a finite multiset of objects from $B$ as well as a finite multiset of operators from $O$ and of rules from $R$; we shall assume $A \in I_0$ in the following;
- $O$ is a finite set of operators (which will be described in detail below);
- $R$ is a finite set of (evolution) rules of the form $(op_1, ..., op_k; op'_1, ..., op'_m)$ with $k \geq 1$ and $m \geq 0$, where $op_1$, ..., $op_k$, $op'_1$, ..., $op'_m$ are operators from $O$;
- $f \in \{1, ..., n\}$ is the label of the final compartiment; we shall always assume $I_f = \emptyset$ and $R_f = \emptyset$.

The main power of GP-systems lies in the operators, which can be of the following types:

- $P \subseteq O$, i.e., the productions working on the objects from $B$ are operators;
- $O_0 \subseteq O$, where $O_0$ is a finite set of special symbols, which are called *ground operators*;
- $Tr_{in} \subseteq O$, where $Tr_{in}$ is a finite set of transfer operators on objects from $B$ of the form $(\tau_{in,k}, E, F)$, $1 \leq k \leq n$, $E \subseteq P$, $F \subseteq P$; the operator $(\tau_{in,k}, E, F)$ transfers an object $w$ from $B$ being in compartiment $m$ into compartiment $k$ provided
  1. region $m$ contains membrane $k$,
  2. every production from $E$ could be applied to $w$ (hence, $E$ is also called the permitting transfer condition),
  3. no production from $F$ can be applied to $w$ (hence, $F$ is also called the forbidding transfer condition);
- $Tr_{out} \subseteq O$, where $Tr_{out}$ is a finite set of transfer operators on objects from $B$ of the form $(\tau_{out}, E, F)$, $1 \leq k \leq n$, $E \subseteq P$, $F \subseteq P$; the operator $(\tau_{out}, E, F)$ transfers an object $w$ from $B$ being in compartiment $m$ into compartiment $k$ provided
  1. region $k$ contains membrane $m$,
  2. every production from $E$ could be applied to $w$,
  3. no production from $F$ can be applied to $w$;
- $Tr'_{in} \subseteq O$, where $Tr'_{in}$ is a finite set of transfer operators working on operators from $P$, $O_0$, $Tr_{in}$, and $Tr_{out}$ or even on rules from $R$; a transfer operator $\tau_{in,k}$ moves such an element in compartiment $m$ into compartiment $k$ provided region $m$ contains membrane $k$;
- $Tr'_{out} \subseteq O$, where $Tr'_{out}$ is a finite set of transfer operators working on operators from $P$, $O_0$, $Tr_{in}$, and $Tr_{out}$ or even on rules from $R$; a transfer operator $\tau_{out}$ transfers such an element in compartiment $m$ into the surrounding compartiment.

In sum, $O$ is the disjoint union of $P$, $O_0$, and $Tr$, where $Tr$ itself is the (disjoint) union of the sets of transfer operators $Tr_{in}$, $Tr_{out}$, $Tr'_{in}$, and $Tr'_{out}$. In the following we shall assume that the transfer operators in $Tr'_{in}$, and $Tr'_{out}$ do not work on rules from $R$; hence, the distribution of the evolution rules is static and given by $I$. If in all transfer operators the permitting and the forbidding sets are empty, then $G_P$ is called a GP-system without transfer checking.

A *computation* in $G_P$ starts with the initial configuration with $I_k$ being the contents of compartiment $k$. A transition from one configuration to another one is performed by evaluating one evolution rule $(op_1, ..., op_k; op'_1, ..., op'_m)$ in some compartiment $k$, which means that the operators $op_1$, ..., $op_k$, are applied to suitable elements in compartiment $k$ in the multiset sense (i.e., they are "consumed" by the usage of the rule; observe that ground operators have no arguments and are simply consumed in that way); thus we may obtain a new object by the application of a production and/or we may move elements out or into inner compartiments by the corresponding transfer operators; yet we also obtain the operators $op'_1$, ..., $op'_m$ (in the multiset sense) in compartiment $k$.

The language generated by $G_P$ is the set of all terminal objects $w \in B_T$ obtained in the terminal compartiment $f$ by some computation in $G_P$.

To give a first impression of the facilities offered by GP-systems we consider a simple example of a GP-system generating the non-context-free string language $L_1 = \{a^n b^n c^n \mid n \geq 1\}$ :

*Example 1.* The string language $L_1$ can be generated by the regular matrix grammar with axiom (without ac)

$G_M = \left(\{A, B, C, a, b, c\}^*, \{a, b, c\}^*, (\{m, m'\}, \emptyset), ABC\right)$ with

$m = [p_1, p_2, p_3]$, $p_1 = A \to aA$, $p_2 = B \to bB$, $p_3 = C \to cC$,

$m' = [p'_1, p'_2, p'_3]$, $p'_1 = A \to a$, $p'_2 = B \to b$, $p'_3 = C \to c$.

Obviously, for any $n \geq 1$, from the axiom $ABC$ we obtain $a^{n-1}Ab^{n-1}Bc^{n-1}C$ by applying $(n-1)$ times the matrix $m$ and finally $a^n b^n c^n$ by once applying matrix $m'$. This regular matrix grammar with axiom can easily be simulated by a GP-system whose main ingredients are depicted in Figure 1:

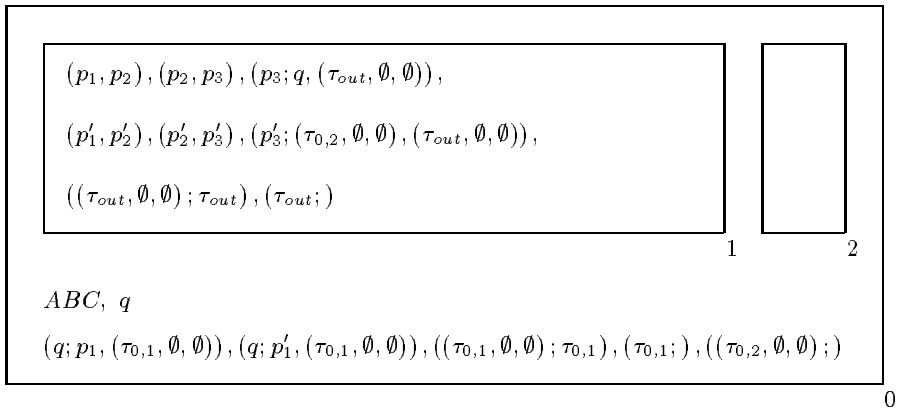


**Fig. 1.** Membrane structure with rules and initial objects and operators

Within the skin membrane we start with the axiom $ABC$ as the initial object and the ground operator $q$; we now can either choose $(q; p_1, (\tau_{0,1}, \emptyset, \emptyset))$ or $(q; p'_1, (\tau_{0,1}, \emptyset, \emptyset))$ from the rules available in the compartiment delimited by the skin membrane labelled by 0. Using $(q; p_1, (\tau_{0,1}, \emptyset, \emptyset))$ yields the transfer operator $(\tau_{0,1}, \emptyset, \emptyset)$, which allows us to transfer the current string, in general of the form $a^{n-1}Ab^{n-1}Bc^{n-1}C$ for some $n \geq 1$, into the compartiment surrounded by the membrane labelled by 1. By applying $(\tau_{0,1}, \emptyset, \emptyset)$ we now gain the transfer operator $\tau_{0,1}$ which allows us to transfer the production $p_1$ into this compartiment 1, too, by applying the rule $(\tau_{0,1};)$. In compartiment 1, by using the rules $(p_1, p_2)$, $(p_2, p_3)$, and $(p_3; q, (\tau_{out}, \emptyset, \emptyset))$ sequentially we obtain the string $a^n Ab^n Bc^n C$ as well as the ground operator $q$ and the transfer operator $(\tau_{out}, \emptyset, \emptyset)$, which then transfers the string out into the skin compartiment through the rule $((\tau_{out}, \emptyset, \emptyset); \tau_{out})$, simultaneously yielding the tranfer operator $\tau_{out}$, which finally transfers $q$, too, by the rule $(\tau_{out};)$.

In a similar way, after applying $(q; p'_1, (\tau_{0,1}, \emptyset, \emptyset))$ in the skin compartiment, in the inner compartiment 1 we obtain $a^n b^n c^n$ from $a^{n-1}Ab^{n-1}Bc^{n-1}C$ by sequentially using the rules $(p'_1, p'_2)$, $(p'_2, p'_3)$, and $(p'_3; (\tau_{0,2}, \emptyset, \emptyset), (\tau_{out}, \emptyset, \emptyset))$. Instead of the ground operator $q$ now the transfer operator $(\tau_{0,2}, \emptyset, \emptyset)$ is generated and moved out into compartiment 0, where according to the rule $((\tau_{0,2}, \emptyset, \emptyset);)$ it transfers the terminal string $a^n b^n c^n$ into the terminal compartiment 2.

The example given above already indicates how matrix grammars could be simulated by GP-systems. In general, the productions in the matrices need not all be different as in this example, hence, usually we will have to use different compartiments for the application of different matrices. In the proof of Theorem 2 we will exploit this idea for showing how GP-systems can simulate graph controlled grammars when using only rules of the form $(op_1; op_2, ..., op_k)$ for some $k \geq 1$, which corresponds to some kind of context-freeness of rules, because $op_1$ is applied without checking for the applicability of other operators.

## 4 Results

The main result of this paper is to show how graph controlled grammars of arbitrary type can be simulated by GP-systems of the same type. The following result covers the case where also a kind of "context-sensitive" rules is taken into account.

**Theorem 1.** *Any graph controlled grammar of arbitrary type can be simulated by a GP-system of the same type with the simple membrane structure* $[_0[_1]_1[_2]_2]_0$.

*Proof.* Let $G_C = (B, B_T, (R, L_{in}, L_{fin}), A)$ be a graph controlled grammar of type $X$ and $G = (B, B_T, P, A)$ be the corresponding underlying grammar of type $X$ with $P = \{p(q) \mid q \in Lab\}$. The main ingredients of the GP-system $G_P$ of type $X$ generating the same language as $G_C$ can be described in the following way (the complete formal description of $G_P$ is obvious and therefore omitted):

- For each $q \in Lab$, in compartiment 0 we take the following rules:
  For the success case, we use the rules $\left(q; q^{(1)}, (\tau_{in,1}, \emptyset, \emptyset)\right)$, $((\tau_{in,1}, \emptyset, \emptyset); \tau_{in,1})$ and $(\tau_{in,1};)$ to transfer the current sentential form as well as the ground operator $q^{(1)}$, which represents the actual node in the control graph, into compartiment 1.
  For the failure case, where $p(q)$ is not applicable to the current sentential form, we use $(q; q^*, (\tau_{in,1}, \emptyset, \{p(q)\}))$ as well as $((\tau_{in,1}, \emptyset, \{p(q)\}); \tau_{in,1})$ (and again $(\tau_{in,1};)$) for the transfer of the current sentential form and the ground operator $q^*$. The non-applicability of $p(q)$ is checked by the forbidding condition $\{p(q)\}$ in the transfer rule $(\tau_{in,1}, \emptyset, \{p(q)\})$.
- In compartiment 1 we take the following rules for each $q \in Lab$:
  The rules $\left(q^{(1)}; q^{(2)}, p(q)\right)$ and $\left(q^{(2)}, p(q); \alpha, (\tau_{out}, \emptyset, \emptyset)\right)$ with $\alpha \in \sigma(q)$ guarantee the simulation of the application of the production $p(q)$ to the underlying sentential form as well as the proceeding in the control graph to a node

in the success field of $q$; the rules $((\tau_{out}, \emptyset, \emptyset); \tau_{out})$ and $(\tau_{out};)$ then transfer the result of the application of $p(q)$ as well as $\alpha$ into the skin compartiment again.
In the failure case, we only have to proceed in the control graph to a node in the failure field of $q$, which is accomplished by one of the rules $(q^*; \beta, (\tau_{out}, \emptyset, \emptyset))$ for $\beta \in \varphi(q)$; the transfer back into compartiment 0 again is performed by the rules $((\tau_{out}, \emptyset, \emptyset); \tau_{out})$ and $(\tau_{out};)$.

- The initial configuration starts with the axiom $A$ and a special ground operator $q_0$, $q_0 \notin Lab$, in compartiment 0. By using one of the rules $(q_0; q)$ for $q \in L_{in}$ we can proceed to a starting node in the control graph.
- For any $q \in L_{fin}$ we take the rule $(q; (\tau_{in,2}, \emptyset, \emptyset))$; the application of the rule $((\tau_{in,2}, \emptyset, \emptyset);)$ finally transfers terminal objects into the terminal compartiment 2.

The explanations given above show how the GP-system $G_P$ can simulate derivations in the graph controlled grammar $G_C$; a more detailed proof is left to the reader. □

As it is obvious from the proof given above, the use of forbidding transfer conditions in the operators moving the underlying objects is only needed for simulating the ac-case; hence we immediately infer the following result:

**Corollary 1.** *Any graph controlled grammar without ac of arbitrary type can be simulated by a GP-system without transfer checking of the same type with the simple membrane structure* $[_0[_1]_1[_2]_2]_0$.

When we only allow "context-free" rules of the form $(op_1; op_2, ..., op_k)$ for some $k \geq 1$, then we need a seperate compartiment for every node of the control graph:

**Theorem 2.** *Any graph controlled grammar of arbitrary type with the control graph containing n nodes can be simulated by a GP-system of the same type with the membrane structure* $[_0[_1]_1 \cdots [_n]_n[_{n+1}]_{n+1}]_0$ *and rules of the forms* $(op_1;)$, $(op_1; op_2)$, *and* $(op_1; op_2, op_3)$.

*Proof.* The main ingredients of the GP-system $G_P$ of type $X$ generating the same language as the graph controlled grammar $G_C$, $G_C = (B, B_T, (R, L_{in}, L_{fin}), A)$, of type $X$ with $card(Lab) = n$ are described as follows:

- For each $q \in Lab$, in compartiment 0 we take the following rules:
  For the success case, we use the rules $\left(q; q^{(1)}, (\tau_{in,q}, \emptyset, \emptyset)\right)$, $((\tau_{in,q}, \emptyset, \emptyset); \tau_{in,q})$ and $(\tau_{in,q};)$ to transfer the current sentential form as well as the ground operator $q^{(1)}$ into the corresponding compartiment $q$. For the failure case, we now use $(q; q^*, (\tau_{in,q}, \emptyset, \{p(q)\}))$ as well as $((\tau_{in,q}, \emptyset, \{p(q)\}); \tau_{in,q})$ (and again $(\tau_{in,q};)$) for the transfer of the current sentential form and the ground operator $q^*$ into compartiment $q$. The forbidding condition $\{p(q)\}$ in the transfer rule $(\tau_{in,q}, \emptyset, \{p(q)\})$ checks for the non-applicability of $p(q)$.

– For each $q \in Lab$, in compartiment $q$ we take the following rules:
The rules $\left(q^{(1)}; p\,(q)\right)$ and $(p\,(q)\,; \alpha, (\tau_{out}, \emptyset, \emptyset))$ with $\alpha \in \sigma\,(q)$ allow for the simulation of the application of the production $p\,(q)$ to the underlying sentential form as well as to proceed to a ground operator $\alpha$ representing a node in the success field of $q$ in the control graph; the rules $((\tau_{out}, \emptyset, \emptyset)\,; \tau_{out})$ and $(\tau_{out};)$ then transfer the result of the application of $p\,(q)$ as well as $\alpha$ back into the skin compartiment. The failure case again is accomplished by one of the rules $(q^*; \beta, (\tau_{out}, \emptyset, \emptyset))$ for $\beta \in \varphi\,(q)$ (as well as by the rules $((\tau_{out}, \emptyset, \emptyset)\,; \tau_{out})$ and $(\tau_{out};)$).
– The axiom $A$ and the special ground operator $q_0$, $q_0 \notin Lab$, constitute the initial configuration in compartiment 0. The rules $(q_0; q)$ for $q \in L_{in}$ allow us to proceed to a starting node in the control graph.
– For any $q \in L_{fin}$, the rules $(q; (\tau_{in,n+1}, \emptyset, \emptyset))$ and $((\tau_{in,n+1}, \emptyset, \emptyset)\,;)$ finally transfer terminal objects into the terminal compartiment $n+1$.

In contrast to the GP-system constructed in the preceding theorem, now the GP-system $G_P$ can simulate the application of a production $p\,(q)$ at the node $q$ in the control graph of $G_C$ only by using a specific compartiment $q$ for each $q$.□

**Corollary 2.** *Any graph controlled grammar without ac of arbitrary type with the control graph containing $n$ nodes can be simulated by a GP-system without transfer checking of the same type with the membrane structure $[_0[_1]_1 \ldots [_n[_{n+1}]_{n+1}]_0$ and rules of the forms $(op_1;)$, $(op_1; op_2)$, and $(op_1; op_2, op_3)$.*

The general results proved above immediately apply for the string case, but as well for the objects being $d$-dimensional arrays and (directed) graphs, which will be elaborated in the following subsections. Moreover, in all the cases considered there, the graph controlled garmmars can be constructed in such a way that only terminal objects from $B_T$ have been derived when in a derivation a final node from $L_{fin}$ is reached; hence, the results obtained in the terminal compartiments of the GP-systems constructed in the proofs above are from $B_T$ only, i.e., in this case no final intersection with $B_T$ is necessary any more.

### 4.1 String languages

As shown in [2], context-free graph controlled string grammars can generate any recursively enumerable string language. The theorems proved above show that GP-systems using these context-free string productions can generate any recursively enumerable string language, too.

### 4.2 Array languages

Let $Z$ denote the set of integers and let $d \in N$. Then a *$d$-dimensional array* $\mathcal{A}$ over an alphabet $V$ is a function $\mathcal{A} : Z^d \to V \cup \{\#\}$, where $shape(\mathcal{A}) = \{\, v \in W \mid \mathcal{A}(v) \neq \# \,\}$ is finite and $\# \notin V$ is called the *background* or *blank symbol*. We

usually shall write $\mathcal{A} = \{(v, \mathcal{A}(v)) \mid v \in shape(\mathcal{A})\}$. The set of all $d$-dimensional arrays over $V$ is denoted by $V^{*d}$. The *empty array* in $V^{*d}$ with empty shape is denoted by $\Lambda_d$. Moreover, we define $V^{+d} = V^{*d} \setminus \{\Lambda_d\}$. Any subset of $V^{+d}$ is called a $\Lambda$-free $d$-dimensional array language.

The *translation* $\tau_v : Z^d \to Z^d$ is defined by $\tau_v(w) = w + v$ for all $w \in Z^d$, and for any array $\mathcal{A} \in V^{*d}$ we define $\tau_v(\mathcal{A})$, the corresponding $d$-dimensional array translated by $v$, by

$$(\tau_v(\mathcal{A}))(w) = \mathcal{A}(w - v) \quad \text{for all} \quad w \in Z^d.$$

A *$d$-dimensional array production* $p$ over $V$ is a triple $(W, \mathcal{A}_1, \mathcal{A}_2)$, where $W \subseteq Z^d$ is a finite set and $\mathcal{A}_1$ and $\mathcal{A}_2$ are mappings from $W$ to $V \cup \{\#\}$ such that $shape(\mathcal{A}_1) \neq \emptyset$; $p$ is called *#-context-free*, if $card(shape(\mathcal{A}_1)) = 1$. We say that the array $\mathcal{C}_2 \in V^{*d}$ is *directly derivable* from the array $\mathcal{C}_1 \in V^{*d}$ by the $d$-dimensional array production $(W, \mathcal{A}_1, \mathcal{A}_2)$ if and only if there exists a vector $v \in Z^d$ such that $\mathcal{C}_1(w) = \mathcal{C}_2(w)$ for all $w \in Z^d \setminus \tau_v(W)$ as well as $\mathcal{C}_1(w) = \mathcal{A}_1(\tau_{-v}(w))$ and $\mathcal{C}_2(w) = \mathcal{A}_2(\tau_{-v}(w))$ for all $w \in \tau_v(W)$, i.e., the sub-array of $\mathcal{C}_1$ corresponding to $\mathcal{A}_1$ is replaced by $\mathcal{A}_2$, thus yielding $\mathcal{C}_2$.

Based on these definitions of $d$-dimensional array productions we can define $d$-dimensional array grammars, graph controlled $d$-dimensional array grammars etc. As was shown in [5], any recursively enumerable two-dimensional array language can even be generated by a graph controlled #-context-free two-dimensional array grammar without ac. Hence, Corollaries 1 and 2 apply showing that any recursively enumerable two-dimensional array language can even be generated by a GP-system without transfer checking using #-context-free two--dimensional array productions. The same result holds true for dimensions 1 and 3, too. For $d \geq 4$, we only know that graph controlled #-context-free $d$-dimensional array grammars can generate any recursively enumerable $d$-dimensional array language; hence, at least Theorems 1 and 2 are valid showing that recursively enumerable $d$-dimensional array languages can be generated by specific GP-systems using #-context-free $d$-dimensional array productions.

### 4.3 Graph languages

As shown in [4], any recursively enumerable graph language can be generated by graph controlled graph grammars using only the following elementary graph productions:

1. add a new node with label $K$;
2. change the label of a node labelled by $K$ to $L$;
3. delete a node with label $K$;
4. add a new edge labelled by $a$ between two nodes labelled by $K$ and $M$, respectively;
5. change the label $a$ of an edge between two nodes labelled by $K$ and $M$, respectively, to $b$;
6. delete an edge labelled by $a$ between two nodes labelled by $K$ and $M$, respectively.

According to the theorems proved above, GP-systems using these elementary graph productions can generate any recursively enumerable graph language, too.

## 5 Summary and Future Research

The model of GP-systems investigated in the preceding sections has revealed great generative power with respect to arbitrary types of productions working on single objects. Yet among others, the following questions remained open:

- Is the hierarchy with respect to the number of inner membranes needed in the proof of Theorem 2 infinite or is there another proof showing that it collapses at a certain level? This question can be asked in general or for special types of grammars, e.g., for context-free string grammars.
- The effect of the permitting conditions in transfer operators corresponds with the permitting context condition in random context grammars (see [2]). Hence, in a similar way as in the proofs of Theorems 1 and 2 we can show that permitting random context grammars can be simulated by GP-systems with permitting transfer conditions of the same type.
- What is the generative power of GP-systems without ground operators and transfer operators working on them, i.e., when the only operators we allow are productions on objects and transfer operators to be applied to objects? Again we can ask this question in general or for special types of grammars. For example, in the string case (with at least context-free productions) the ground operators representing control variables can be encoded within the sentential forms; hence, at least in combination with the permitting and forbidding conditions in transfer operators moving objects GP-systems can reach the power of graph controlled string grammars again. A similar result can be established for graph controlled #-context-free $d$-dimensional array grammars and graph controlled elementary graph grammars, too. In all these cases, the proofs are much more complicated than the pure structural proofs of Theorems 1 and 2.

As already pointed out in [7], the idea of membrane structures offers a nearly unlimited variety of variants. Hence, this paper can be seen as a starting point for further investigations in this field of generalizing the idea of P-systems and the investigations of their modelling power. Let us mention just a few ideas to be considered in the future:

- In the proofs given above we have not made use of the possibilty to transfer productions working on the underlying objects or even transfer operators themselves as we did in Example 1, i.e., we only transferred ground operators and objects. Yet it might be interesting to use these features in general, too, and to investigate how the usage of other characteristic features can be reduced while still getting remarkable generative power.
- Additional interesting features to be considered with GP-systems, which were already discussed for P-systems in [7], are the deletion and also the generation of membranes.

– How can GP-systems using productions of a more complicated structure like splicing rules be defined in an adequate way? A splicing rule on strings over an alphabet $V$ usually (e.g., see [8]) is written as $u_1\#v_1\$u_2\#v_2$ and its application to two strings over $V$ of the form $x_1u_1v_1y_1$ and $x_2u_2v_2y_2$ yields the two strings $x_1u_1v_2y_2$ and $x_2u_2v_1y_1$, which are the result of cutting the two given strings at the sites $u_1v_1$ and $u_2v_2$ and immediately recombining the cut pieces in a crosswise way. Obviously, it is no problem just to use splicing rules instead of other string productions by interpreting a splicing rule as a partial recursive relation $\subseteq V^* \times V^* \to V^* \times V^*$. Yet a suitable modelling of splicing systems (H-systems) and related systems requires some more sophisticated definitions and therefore is left to a forthcoming paper.

The formal investigation of the generative power of all these (and many other) variants of GP-systems and their complexity for simulating specific other generating devices remains for future research.

## Acknowledgements

I gratefully acknowledge fruitful discussions with Grzegorz Rozenberg and especially with Gheorghe Păun, whose ideas inspired the model of GP-systems presented in this paper.

## References

1. G. Berry and G. Boudol, *The chemical abstract machine,* Theoretical Computer Science 96 (1992), pp. 217-248.
2. J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory* (Springer, Berlin, 1989).
3. J. Dassow and Gh. Păun, *On the power of membrane computing,* TUCS Research Report No. 217 (Dec. 1998).
4. R. Freund and B. Haberstroh, *Attributed Elementary Programmed Graph Grammars,* Proceedings 17th International Workshop on Graph-Theoretic Concepts in Computer Science (LNCS 570, Springer-Verlag, 1991), pp. 75–84.
5. R. Freund, *Control mechanisms on #-context-free array grammars,* Mathematical Aspects of Natural and Formal Languages (Gh. Păun, ed.), World Scientific, Singapore (1994), pp. 97-137.
6. Gh. Păun, *Computing with Membranes,* TUCS Research Report No. 208 (Nov. 1998).
7. Gh. Păun, *Computing with Membranes: An Introduction,* Bulletin EATCS **67** (Febr. 1999), pp. 139-152.
8. Gh. Păun, G. Rozenberg, and A. Salomaa, *DNA Computing: New Computing paradigms* (Springer-Verlag, Berlin, 1998).
9. Gh. Păun, G. Rozenberg, and A. Salomaa, *Membrane computing with external output,* TUCS Research Report No. 218 (Dec. 1998).
10. I. Petre, *A normal form for P-systems,* Bulletin EATCS **67** (Febr. 1999), pp. 165-172.