SEQUENTIAL P-SYSTEMS

Rudolf FREUND
Institut für Computersprachen, Technische Universität Wien,
Karlsplatz 13, A-1040 Wien, Austria
email: rudi@logic.at, tel.: ++43 1 58801 18542

Abstract. We consider sequential variants of P-systems, the new model for computations using membrane structures and recently introduced by Gheorghe Păun. Using the permeabilty of the membranes for specific objects as a kind of filter turns out to be a very powerful mechanism in combination with suitable rules to be applied within the membranes. Generalized P-systems, GP-systems for short, constitute the most general model of sequential P-systems, considered in this paper. GP-systems allow for the simulation of graph controlled grammars of arbitrary type based on productions working on single objects; for example, the general results we state 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. Moreover, we consider GP-systems as molecular computing devices using splicing or cutting and recombination of strings. Various variants of such systems have universal computational power, too, e.g., test tube systems based on splicing or cutting and recombination of strings can be simulated by the corresponding GP-systems.

1 Introduction

One of the main ideas incorporated in the model of P-systems introduced in [15] is the membrane structure (for a chemical variant of this idea see [2]) 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)* 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 [18]) 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 [16]). 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 [6], [14], and [17], show that only bounded parallelism is needed. Moreover, in GP-systems 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; moreover, we define molecular systems and the molecular operations on strings, i.e., splicing as well as cutting and recombination. In the third section we describe the model of GP-systems as introduced in [12], and in the fourth section we show that GP-systems allow for the simulation of graph controlled grammars of arbitrary type based on productions working on single objects; 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. In the fifth section we consider GP-systems with splicing and cutting/recombination and show their universal computional power with respect to recursively enumerable string languages.

2 Definitions

First, we recall some basic notions from the theory of formal languages (for more details, the reader is referred to [5]).

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 λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. Any subset of V^+ is called a *λ-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 $((V_N \cup V_T)^*, V_T^*, P, S)$.

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 [5].

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.

2.2 Molecular systems

A *molecular system* is a quadruple $\sigma = (B, B_T, P, A)$, 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 is a (finite) multiset of axioms from B. A production p in P in general is a partial recursive relation $\subseteq B^k \times B^m$ for some $k, m \geq 1$, where we also demand that the domain of p is recursive (i.e., given $w \in B^k$ it is decidable if there exists some $v \in B^m$ with $(w, v) \in p$) and, moreover, that the range for every w is finite, i.e., for any $w \in B^k$, $card(\{v \in B \mid (w, v) \in p\}) < \infty$. For any two multisets L and L' over B, we say that L' is computable from L by a production p if and only if

$$L' = (L - (w_1 + ... + w_k)) + (v_1 + ... + v_m)$$

for some $(w_1, ..., w_k) \in B^k$ and $(v_1, ..., v_m) \in B^m$ with $(w_1, ..., w_k, v_1, ..., v_m) \in p$; we also write $L \Longrightarrow_p L'$ and $L \Longrightarrow_\sigma L'$. A computation in σ is a sequence

$$L_0, ..., L_n$$

such that the L_i, $0 \leq i \leq n$, $n \geq 0$, are multisets over B as well as $L_i \Longrightarrow_\sigma L_{i+1}$, $1 \leq i \leq n$; in this case we also write $L_0 \Longrightarrow_\sigma^n L_n$, and moreover, we write $L_0 \Longrightarrow_\sigma^* L_n$ if $L_0 \Longrightarrow_\sigma^n L_n$ for some $n \geq 0$. The *language generated by* σ is

$$L(\sigma) = \{w \in B_T \mid A \Longrightarrow_\sigma^* L, (L, w) \geq 1\}.$$

2.3 Splicing systems and cutting/recombination systems

The special string productions we shall consider in the following are the splicing as well as the cutting and recombination operations:

A *splicing scheme* (an *H-system* for short) is a pair (V, P), where V is an alphabet and $P \subseteq V^*\#V^*\$V^*\#V^*$; $\#, \$$ are special symbols not in V; P is the set of *splicing rules.* For $x, y, z \in V^+$ and $r = u_1\#u_2\$u_3\#u_4$ in P we define $(x, y) \Longrightarrow_r z$ if and only if $x = x_1u_1u_2x_2$, $y = y_1u_3u_4y_2$, and $z = x_1u_1u_4y_2$ for some $x_1, x_2, y_1, y_2 \in V^*$. For any language $L \subseteq V^+$, $\sigma(L)$ denotes the language obtained from L by any single application of rules from σ to a string from L. We also define $\sigma^0(L) = L$ and $\sigma^{i+1}(L) = \sigma(\sigma^i(L))$ for all $i \geq 0$, as well as $\sigma^{(0)}(L) = L$ and $\sigma^{(i+1)}(L) = \sigma^{(i)}(L) \cup \sigma\left(\sigma^{(i)}(L)\right)$ for all $i \geq 0$; moreover, we denote $\sigma^*(L) = \bigcup_{i=0}^{\infty} \sigma^{(i)}(L)$. An *extended mH-system* (or *extended splicing system with multisets*) is a molecular system σ, $\sigma = (V^*, V_T^*, P, A)$, where $V_T \subseteq V$, P is a set of splicing rules $p \subseteq (V^* \times V^*) \times V^*$, and A is the multiset of axioms.

A *cutting/recombination scheme* (a *CR-scheme* for short) is a quadruple (V, M, C, R), where V is a finite alphabet; M is a finite set of *markers; V* and M are disjoint sets; C is a set of *cutting rules* of the form $u\#l\$m\#v$, where $u \in V^* \cup MV^*$,

$v \in V^* \cup V^*M$, and $m, l \in M$, and $\#, \$$ are special symbols not in $V \cup M$; $R \subseteq M \times M$ is the recombination relation representing the *recombination rules*. Cutting and recombination rules are applied to objects from $O(V, M)$, where we define

$$O(V, M) = V^+ \cup MV^* \cup V^*M \cup MV^*M.$$

For $x, y, z \in O(V, M)$ and a cutting rule $c = u\#l\$m\#v$ we define $x \Longrightarrow_c (y, z)$ if and only if for some $\alpha \in V^* \cup MV^*$ and $\beta \in V^* \cup V^*M$ we have $x = \alpha uv\beta$ and $y = \alpha ul$, $z = mv\beta$. For $x, y, z \in O(V, M)$ and a recombination rule $r = (l, m)$ from R we define $(x, y) \Longrightarrow_r z$ if and only if for some $\alpha \in V^* \cup MV^*$ and $\beta \in V^* \cup V^*M$ we have $x = \alpha l$, $y = m\beta$, and $z = \alpha\beta$. For a CR-scheme $\sigma = (V, M, C, R)$ and any language $L \subseteq O(V, M)$, $\sigma(L)$ then denotes the set of all objects obtained by applying one cutting or one recombination rule to objects from L; $\sigma^i(L)$, $\sigma^{(i)}(L)$, and $\sigma^*(L)$ are defined as above for H-systems. An *extended mCR-system* is a molecular system σ, $\sigma = (O(V, M), O(V_T, M_T), P, A)$, where $V_T \subseteq V$ is the set of terminal symbols, $M_T \subseteq M$ is the set of terminal markers, A is the multiset of axioms, P is the union of the relations (productions) defined by the cutting rules from C ($\subseteq O(V, M) \times O(V, M)^2$) and the recombination rules from R ($\subseteq O(V, M) \times O(V, M)$), and (V, M, C, R) is the underlying CR-scheme.

In [3] and [10] it was proved that extended mH-systems with a finite multiset of axioms and a finite number of splicing rules have the computational power of arbitrary grammars and Turing machines, repectively. Similar results for CR-systems were proved in [11].

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 [14] and [15] 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. 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 [14] and [15] *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;
- μ 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 (τ_{out}, E, F), $1 \leq k \leq n$, $E \subseteq P$, $F \subseteq P$; the operator (τ_{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 τ_{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.

4 P-systems and graph controlled grammars

The main result of this section is the simulation of graph controlled grammars of arbitrary type 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 4.1 *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 α 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. ◇

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 4.1 *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 4.2 *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* $[_0[_1]_1...[_n]_n[_{n+1}]_{n+1}]_0$ *and rules of the forms* $(op_1;)$, $(op_1; op_2)$, *and* $(op_1; op_2, op_3)$.

Proof see [12]. ◇

Corollary 4.2 *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* $[_0[_1]_1 ... [_n]_n [_{n+1}]_{n+1}]_0$ *and rules of the forms* $(op_1;)$, $(op_1; op_2)$, *and* $(op_1; op_2, op_3)$.

The general results proved above immediately apply for the string case, but as well for the objects being d-dimensional arrays and (directed) graphs.

4.1 String languages

As shown in [5], 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 Λ_d. Moreover, we define $V^{+d} = V^{*d} \setminus \{\Lambda_d\}$. Any subset of V^{+d} is called a Λ-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 [9], 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 4.1 and 4.2 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 4.1 and 4.2 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 [7], 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 GP-systems and molecular systems

Molecular systems of type X working in the multiset manner can be simulated by GP-systems of the corresponding types.

Theorem 5.1 *Any molecular system of type X working in the multiset manner can be simulated by a GP-system of the same type with the simplest membrane structure* $[_0]_0$.

Proof. Let $\sigma = (B, B_T, P, A)$ be a molecular system of type X. In a depictive way, the corresponding GP-system $(B, B_T, P, A, [_0]_0, I, P, R, 0)$, $I = A \cup \{(p, \infty) \mid p \in P\}$, $R = \{(p, p) \mid p \in P\}$ is illustrated in Figure 1. The productions are used by the evolution rules in a catalytic way only, i.e., they are not affected when being applied according to the evolution rule (p, p). Hence, we would obtain the desired result even with $I = A \cup \{(p, 1) \mid p \in P\}$. ◇

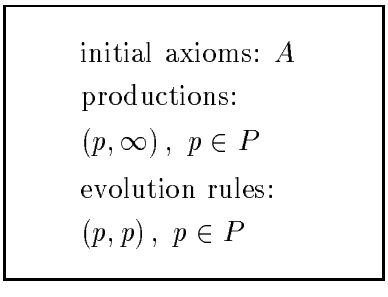

Figure 1: Simulation of molecular system by GP-system

Obviously, this theorem applies to extended mH-systems as well as to extended mCR-systems. In these cases it is even sufficient that at each time only one production is present in order to be evaluated by an evolution rule, which then has to generate the next production to be applied, which is stated in the following corollary:

Corollary 5.1 *Any recursively enumerable string language can be generated by a GP-system* $(B, B_T, P, A, [_0]_0, I, P, R, 0)$ *with the simplest membrane structure* $[_0]_0$ *and splicing or cutting/recombination rules such that* $I = A \cup \{(p_0, 1)\}$ *and* $R = \{(p, q) \mid p, q \in P\}$.

Acknowledgements

I gratefully acknowledge all the fruitful and interesting discussions with Gheorghe Păun on P-systems.

References

[1] L. M. Adleman, *Molecular computation of solutions to combinatorial problems*, Science, 226 (Nov. 1994), pp. 1021-1024.

[2] G. Berry and G. Boudol, The chemical abstract machine, *Theoretical Computer Science* **96** (1992), pp. 217–248.

[3] E. Csuhaj-Varjú, R. Freund,L. Kari, and Gh. Păun, *DNA computing based on splicing: Universality results,* in: L. Hunter and T. Klein (eds.), Pacific Symposium on Biocomputing'96, World Scientific (1996), pp. 179-190.

[4] E. Csuhaj-Varjú, L. Kari, and Gh. Păun, *Test tube distributed systems based on splicing*, Computers and Artificial Intelligence, Vol. 15 (2) (1996), pp. 211-232.

[5] J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory* (Springer, Berlin, 1989).

[6] J. Dassow and Gh. Păun, On the power of membrane computing, *Journal of Universal Computer Science* **5**, 2 (1999), pp. 33–49 (http://www.iicm.edu/jucs).

[7] 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.

[8] R Freund, *Splicing systems on graphs,* International IEEE Symposium on Intelligence in Neural and Biological Systems, Herndon, VA, USA, May 1995, pp. 189 – 194.

[9] 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.

[10] R. Freund, L. Kari, and Gh. Păun, *DNA computing based on splicing: The existence of universal computers*, Theory of Computing Systems, Vol. 32, (1999), pp. 69-112.

[11] R. Freund and F. Wachtler, *Universal systems with operations related to splicing,* Computers and Artificial Intelligence, Vol. 15 (4) (1996), pp. 273-294.

[12] R. Freund, Generalized P-systems, *Fundamentals of Computation Theory, FCT'99,* Iasi, 1999, (G. Ciobanu, Gh. Păun, eds.), *LNCS* 1684, Springer-Verlag (1999), pp. 281–292.

[13] R. Freund, Generalized P-systems with Splicing and Cutting/Recombination, *Grammars* (2000).

[14] Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences* **61** (2000).

[15] Gh. Păun, *Computing with Membranes: An Introduction,* Bulletin EATCS **67** (Febr. 1999), pp. 139-152.

[16] Gh. Păun, G. Rozenberg, and A. Salomaa, *DNA Computing: New Computing paradigms* (Springer-Verlag, Berlin, 1998).

[17] Gh. Păun, G. Rozenberg, and A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae* **41**, 3 (2000), pp. 259–266.

[18] I. Petre, *A normal form for P-systems,* Bulletin EATCS **67** (Febr. 1999), pp. 165–172.

[19] D. Pixton, Splicing in abstract families of languages, *Theoretical Computer Science* **234** (2000), pp. 135–166.