Multiset Processing by means of Systems of Sequential Transducers

Gheorghe Păun
Institute of Mathematics of the Romanian Academy, Bucureşti, Romania

Gabriel Thierrin
Department of Computer Science,
University of Western Ontario, Canada

CDMTCS-101
April 1999

Multiset processing by means of systems of sequential transducers[1]

Gheorghe PĂUN
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
E-mail: gpaun@imar.ro

Gabriel THIERRIN
Department of Computer Science, University of Western Ontario
N6A 5B7 London, Ontario, Canada
E-mail: gab@csd.uwo.ca

Abstract. We introduce a computing mechanism of a biochemical inspiration (similar to a P system in the area of computing by membranes) which consists of a multiset of symbol-objects and a set of finite state sequential transducers. The transducers process symbols in the current multiset in the usual manner. A computation starts in an initial configuration and ends in halting configuration. The power of these mechanisms is investigated (the main result says that systems with two components generate all gsm images of all permutations of recursively enumerable languages), as well as the closure properties of the obtained family (which is shown to be a full AFL).

1 Introduction

The present paper can be seen as a contribution both to Natural Computing, in the area of computing with membranes (P systems, see [8], [6], [11], [12], or the survey in [9]), and to Distributed (Parallel) Computing, in the multiset rewriting area (see, [1], [2], [3], etc.).

We introduce here a computing mechanism of the following "biochemical" type. In a given space (a *membrane*) we have a multiset of objects, identified with symbols from a given alphabet. In the same space, we place several finite automata with outputs (generalized sequential machines). In a parallel manner, these automata take symbols available around and, depending on their states, change their states and produce new symbols. In this way, a new *configuration* of the system is obtained. A sequence of such transitions among configurations is a *computation*; a computation is complete if it *halts*, that is, no further move is possible in its last configuration. In this way, a mapping between the initial multiset of objects and the multiset present in the halting configuration is defined. We can also associate a set of strings with a computation, as in P systems: we distinguish a *terminal* set of symbols and construct

[1]Research supported by Grant OGP0007877 of the Natural Sciences and Engineering Research Council of Canada

the string of terminal symbols appearing during the computation, in the order they are produced; when several terminal symbols are introduced at the same time, then any ordering of them is accepted (thus, several strings are associated with the same computation).

Consequently, we have here a variant of a P system, with only one membrane (so, a particular case from this point of view), but with the *evolution rules* of a powerful form: finite state machines, which remember by their states some information about their previous work. Still, such machines are among the simplest we can consider. (This could make appropriate the term *colony* for our device, in the sense introduced in [7], of a collectivity of as simple as possible devices working together.)

Somewhat expected (this happens in general in distributed systems, P systems included), the power of our computing machinery is rather large: systems with only two components are able to generate all recursively enumerable languages modulo a permutation; even the gsm images of permutations of recursively enumerable languages can be obtained in this way. We do not know whether or not *all* recursively enumerable languages can be obtained. Anyway, the family of languages generated by our devices is a full AFL (Abstract Family of Languages), so we can appreciate it as being very large.

2 Prerequisites

For elements of formal language theory we shall use below we refer to [13]. We only specify some notations.

For an alphabet V, V^* is the free monoid generated by V, λ is the empty string, and $V^+ = V^* - \{\lambda\}$. The length of $x \in V^*$ is denoted by $|x|$, while $|x|_a$ is the number of occurrences of the symbol a in the string x. If $V = \{a_1, \ldots, a_n\}$ (the ordering is important), then $\Psi_V(x) = (|x|_{a_1}, \ldots, |x|_{a_n})$ is the *Parikh vector* of the string $x \in V^*$. For a language $L \subseteq V^*$, $\Psi_V(L) = \{\Psi_V(x) \mid x \in L\}$ is the *Parikh set* of L.

A Chomsky grammar is written in the form $G = (N, T, S, P)$, where N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, and P is the set of productions. We denote by RE the family of recursively enumerable languages.

In general, for a family FL of languages, we denote by pFL, FL^{one}, and FL^{bound} the families of permutations of languages in FL, of languages in FL over the one-letter alphabet, and of the bounded languages in FL, respectively (a language $L \subseteq V^*$ is bounded if there are $w_1, \ldots, w_n \in V^+$ such that $L \subseteq w_1^* \ldots w_n^*$).

A notion which will be very useful below is that of a *matrix grammar*. Such a grammar is a construct $G = (N, T, S, M, C)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \rightarrow x_1, \ldots, A_n \rightarrow x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and C is a set of occurrences of rules in M (N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called *matrices*).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \rightarrow x_1, \ldots, A_n \rightarrow x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w'_i A_i w''_i, w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \rightarrow x_i$ appears in C. (The rules of a matrix are applied in order, possibly skipping the rules in C if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) If $C = \emptyset$, then the grammar is said to be without appearance checking (and C is no longer mentioned).

We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$. The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When we use only grammars without appearance checking, then the obtained family is denoted by MAT.

A matrix grammar $G = (N, T, S, M, C)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \dagger\}$, with these three sets mutually disjoint, and that the matrices in M are of one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$,
3. $(X \to Y, A \to \dagger)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*$.

Moreover, there is only one matrix of type 1 and C consists exactly of all rules $A \to \dagger$ appearing in matrices of type 3. One sees that $\dagger$ is a trap-symbol; once introduced, it is never removed. A matrix of type 4 is used only once, at the last step of a derivation (clearly, matrices of forms 2 and 3 cannot be used at the last step of a derivation).

According to Lemma 1.3.7 in [5], for each matrix grammar there is an equivalent matrix grammar in the binary normal form.

A *multiset* over an alphabet $V = \{a_1, \dots, a_n\}$ is a mapping $\mu : V \longrightarrow \mathbf{N} \cup \{\infty\}$. A multiset can be given in the form $\{(a_1, \mu(a_1)), \dots, (a_n, \mu(a_n))\}$ or can be represented by any string $w \in V^*$ such that $\Psi_V(w) = (\mu(a_1), \dots, \mu(a_n))$. We shall make below an extensive use of the string representation of a multiset, almost using as synonymous the terms "string" and "multiset". For the sake of mathematical accuracy, the multiset $\{(a_1, |w|_{a_1}), \dots, (a_n, |w|_{a_n})\}$ represented by a string $w \in V^*$ is denoted by $\mu(w)$.

For $a_i \in V$ and a multiset μ over V, we say that a_i belongs to μ and we write $a_i \in \mu$ if $\mu(a_i) \geq 1$.

We say that the multiset μ_1 is included in the multiset μ_2, and write $\mu_1 \subseteq \mu_2$, if $\mu_1(a) \leq \mu_2(a)$ for all $a \in V$. The union of μ_1, μ_2 is the multiset defined by $(\mu_1 \cup \mu_2)(a) = \mu_1(a) + \mu_2(a)$, for all $a \in V$. The difference of two multisets, $\mu_1 - \mu_2$ is defined here only when $\mu_2 \subseteq \mu_1$, by $(\mu_1 - \mu_2)(a) = \mu_1(a) - \mu_2(a)$, for all $a \in V$.

3 P systems of transducers

We now introduce the computing mechanisms we investigate in this paper.

Let us first recall that a *gsm* (generalized sequential machine) is a construct $\gamma = (K, V_1, V_2, s_0, F, P)$, where K, V_1, V_2 are alphabets (set of states, input and output alphabets, respectively), $s_0 \in K$ (initial state), $F \subseteq K$ (final states), and P is a finite set of rewriting rules of the form $sa \to xs'$, for $s, s' \in K$ and $a \in V_1, x \in V_2^*$.

For $s, s' \in K, y_1, x \in V_2^*, y_2 \in V_1^*, a \in V_1$ we write $y_1 s a y_2 \Longrightarrow y_1 x s' y_2$ if $sa \to xs' \in P$. Then, for $w \in V_1^*$ we define $\gamma(w) = \{z \in V_2^* \mid s_0 w \Longrightarrow^* z s_f$, for some $s_f \in F\}$. For $L \subseteq V_1^*$, we define $\gamma(L) = \bigcup_{x \in L} \gamma(w)$. We say that $\gamma(L)$ is the image of L through the gsm γ.

For a family FL of languages, we denote by $gsm(FL)$ the family of gsm images of languages in FL.

Here we consider the gsm's not as strings (and languages) transducers, but as *operators* on multisets of symbols; in this case, the final states are no longer necessary.

A *system of transducers* (in order to remind the membrane computing, we say, shortly, a *PT system*) of degree $n, n \geq 1$, is a construct

$$\Pi = (V, T, w_0, \gamma_1, \dots, \gamma_n),$$

where:

- V is an alphabet (its elements are called *objects*),
- $T \subseteq V$ (the *terminal* alphabet),
- $w_0 \in V^*$ represents a multiset over V (the *initial* multiset),
- $\gamma_i = (K_i, V, s_{0,i}, P_i)$, $1 \leq i \leq n$, are gsm's without final states (and with identical input and output alphabets, namely equal to V); the rules in P_i are of the form $sa \rightarrow xs'$, for $a \in V - T, x \in V^*, s, s' \in K_i$ (each such gsm is called a *component* of Π).

Note that w_0 is a string over V representing a multiset and that the terminal symbols cannot be processed by the rules in the components of Π.

Any $(n+1)$-tuple $(w, s_1, \dots, s_n)$, with $w \in V^*, s_i \in K_i, 1 \leq i \leq n$, is called a *configuration* of Π; $(w_0, s_{0,1}, \dots, s_{0,n})$ is the *initial configuration* of Π.

For two configurations $(w, s_1, \dots, s_n)$, $(w', s'_1, \dots, s'_n)$ we write $(w, s_1, \dots, s_n) \Longrightarrow (w', s'_1, \dots, s'_n)$ (and we say that we have a *transition* between the two configurations) if the following conditions hold:

1. there is $k \geq 1$ and there are the indices $i_1, \dots, i_k \in \{1, 2, \dots, n\}$ such that
   - $a_{i_j} \in \mu(w), 1 \leq j \leq k$,
   - $\mu(a_{i_1} \dots a_{i_k}) \subseteq \mu(w)$ (multiset inclusion),
   - $s_j a_{i_j} \rightarrow x_j s'_j \in P_j$, for $1 \leq j \leq k$,
   - $\mu(w') = (\mu(w) - \mu(a_{i_1} \dots a_{i_n})) \cup (\mu(x_1) \cup \mu(x_2) \cup \dots \mu(\cup x_n))$ (multiset operations),
2. the set $\{i_1, \dots, i_k\}$ is maximal, in the sense that there is no transition $s_r a_r \rightarrow x_r s'_r \in P_r$ for some $r \in \{1, 2, \dots, n\} - \{i_1, \dots, i_k\}$, such that $a_r \in \mu(w) - \mu(a_{i_1} \dots a_{i_k})$ (no further object in the multiset $\mu(w)$ can be processed by a gsm different from those mentioned at the previous point, $\gamma_{i_1}, \dots, \gamma_{i_k}$).

A configuration $(w, s_1, \dots, s_n)$ is said to be a *halting* one of there is no configuration $(w', s'_1, \dots, s'_n)$ such that a transition $(w, s_1, \dots, s_n) \Longrightarrow (w', s'_1, \dots, s'_n)$ is possible.

As usual, we denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$. A sequence of transitions is called a (complete) *computation* if it starts in the initial configuration and ends in a halting configuration.

There are several possibilities of associating a *result* to a computation Δ : $(w_0, s_{0,1}, \dots, s_{0,n}) \Longrightarrow^* (w, s_1, \dots, s_n)$. First, we can consider that we get a mapping

$\Delta(w_0) = w$ (from multisets over V to multisets over V). In this way, a PT system is a mapping computing machine. Second, we can consider the Parikh vector associated with w, in two variants: $\Psi_V(w)$ and $\Psi_T(w)$ (in the latter case, one understands that the symbols which appear in w but are not terminals are ignored). In this way, we can associate with Π a set of tuples of natural numbers (a relation over $\mathbf{N}$).

Because we want to work here with languages, we choose a third possibility (also followed in the P systems area, see [8], [9]): we collect the terminal symbols, in the order in which they are introduced, and form a string; if several terminal symbols are introduced at the same time (by the same component of Π, using a rule $sa \to xs'$ with x being a string, or by several components), then all the orderings of those symbols are allowed, hence a set of strings is associated with the same computation. The set of strings of this form is the language *generated* by Π it is denoted by $L(\Pi)$.

Formally, this language is defined as follows. For two strings $w, w' \in V^*$ such that $\Psi_T(w) \leq \Psi_T(w')$ (componentwise), we denote by $L(w' - w)$ the set of words $x \in T^*$ such that $\Psi_T(w') = \Psi_T(w) + \Psi_T(x)$ (the set of strings over T composed of symbols which appear in w' and not in w). Then, for a halting computation $\Delta : (w_0, s_{0,1}, \ldots, s_{0,n}) \Longrightarrow (w_1, s_{1,1}, \ldots, s_{1,n}) \Longrightarrow \ldots \Longrightarrow (w_m, s_{m,1}, \ldots, s_{m,n})$, we define

$$L(\Delta) = L(w_1 - w_0)L(w_2 - w_1) \ldots L(w_m - w_{m-1}).$$

The language $L(\Pi)$ is the union of all languages $L(\Delta)$, for Δ being a halting computation with respect to Π.

We denote by PTL_n the family of languages generated by PT systems of degree less than or equal to $n, n \geq 1$; the union of all these families is denoted by PTL.

4 An example

In order to illustrate the definition and the work of a PT system, let us consider the system (of degree 3)

$$\Pi = (V, T, w_0, \gamma_1, \gamma_2, \gamma_3),$$

with

$$\begin{aligned} &V = \{a, a', a'', \bar{a}, b, c, d, e, f, g, h\},\\ &T = \{a\},\\ &w_0 = a'a'b^n d, \text{ for some } n \geq 1, \end{aligned}$$

and the following components:

$$\begin{aligned} \gamma_1 &= (\{s_{0,1}, s_{1,1}\}, V, s_{0,1}, P_1),\\ P_1 &= \{s_0 b \to c s_{1,1},\ s_{1,1} h \to s_{0,1}\},\\ \gamma_2 &= (\{s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}, s_{4,2}\}, V, s_{0,2}, P_2),\\ P_2 &= \{s_{0,2} a' \to a s_{4,2},\ s_{4,2} a' \to a s_{4,2},\ s_{4,2} b \to b s_{4,2},\\ &\quad s_{4,2} c \to c s_{4,2},\ s_{0,2} d \to d s_{0,2},\ s_{0,2} c \to s_{1,2},\\ &\quad s_{1,2} a' \to a'' a'' s_{1,2},\ s_{1,2} e \to f s_{2,2},\ s_{2,2} g \to s_{3,2},\\ &\quad s_{3,2} a'' \to a' s_{3,2},\ s_{3,2} e \to f s_{0,2}\}, \end{aligned}$$

$$\begin{aligned}
\gamma_3 &= (\{s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3}, s_{4,3}\}, V, s_{0,3}, P_3), \\
P_3 &= \{s_{0,3}d \to ds_{0,3},\ s_{0,3}d \to eds_{1,3},\ s_{1,3}a' \to \bar{a}s_{2,3}, \\
&\quad s_{2,3}\bar{a} \to \bar{a}s_{2,3},\ s_{1,3}f \to gs_{3,3},\ s_{3,3}d \to ds_{3,3}, \\
&\quad s_{3,3}d \to eds_{4,3},\ s_{4,3}a'' \to \bar{a}s_{2,3},\ s_{4,3}f \to hs_{0,3}\}.
\end{aligned}$$

For the reader convenience, the components of this PT system are represented graphically in Figure 1.

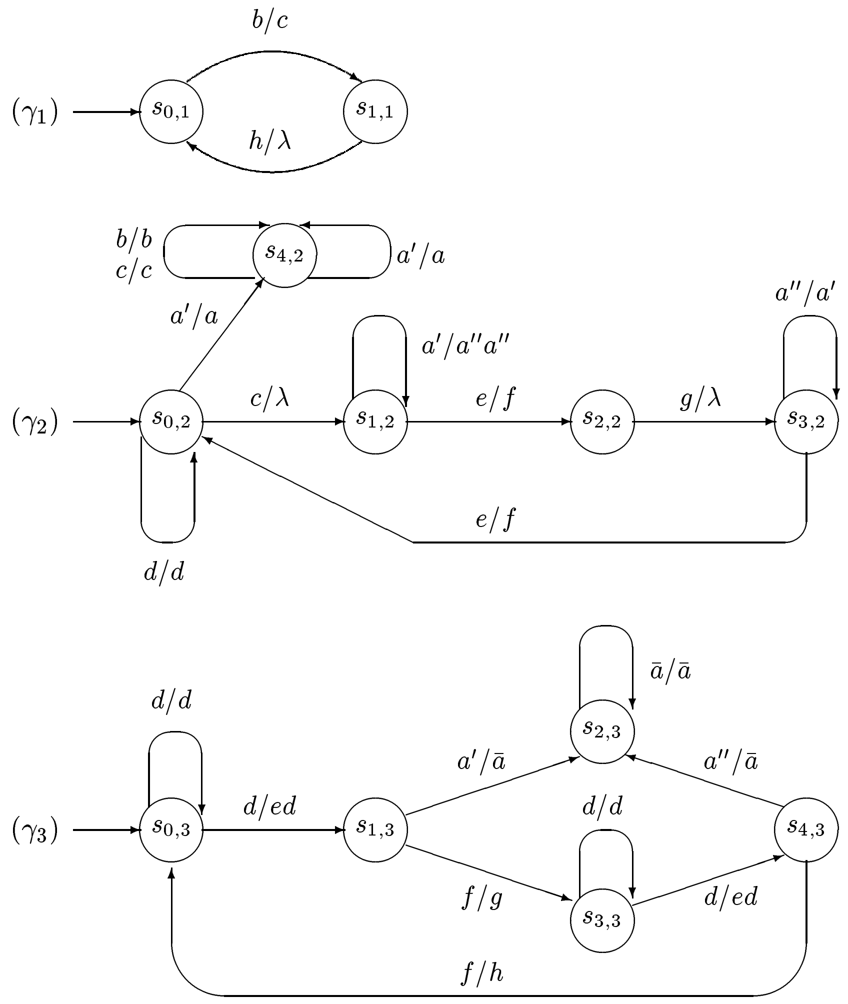

Figure 1: Example of a PT system.

The system works as follows (and halts in a configuration which contains 2^n copies of the symbol a).

If in state $s_{0,2}$ the component γ_2 chooses to go to state $s_{4,2}$, then we never come back to $s_{0,2}$. Assume that at the first step γ_2 uses the rule $s_{0,2}d \to ds_{0,2}$. Simultaneously, γ_1 transforms one occurrence of b in c and γ_3 either remains in the initial state, or it passes to $s_{1,3}$. Assume that γ_3 remains in $s_{0,3}$. At the next step, γ_2 can pass to $s_{1,2}$, by using the rule

$s_{0,1}c \to s_{1,2}$, making use of the symbol c introduced by the first component. γ_1 will wait in state $s_{1,1}$ until a symbol h is produced.

In state $s_{1,2}$, we can replace each a' by two copies of a''. During this time, γ_3 remains in state $s_{0,3}$. Component γ_2 cannot leave $s_{1,2}$ before having produced a copy of e in γ_3. At any moment, γ_3 can use the rule $s_{0,3}d \to eds_{1,3}$. If in the current multiset we still have occurrences of a', then γ_3 will now introduce $\bar{a}$, passing to state $s_{2,3}$ (this is obligatory, because the symbol f is not available). The computation will never stop, because of the rule $s_{2,3}\bar{a} \to \bar{a}s_{2,3}$ which can be used for ever. No output is obtained in such a case. Therefore, the rule $s_{0,3}d \to eds_{1,3}$ in γ_3 should be used after transforming all symbols a' into a'' (doubling them).

After using $s_{1,2}e \to fs_{2,2}$ in γ_2, we can use $s_{1,3}f \to gs_{3,3}$ in γ_3. At the next step, γ_2 can use g in order to pass to state $s_{3,2}$ while γ_3 stays in $s_{3,3}$ any number of steps. During this time, γ_2 replaces each a'' by a' (in state $s_{3,2}$). Again we can control whether or not this process is complete, by means of symbols $d, e, \bar{a}$: if γ_3 uses the rule $s_{3,3}d \to eds_{4,3}$ while symbols a'' are still present, at the next step γ_3 will introduce the trap-symbol $\bar{a}$ (we cannot use the rule $s_{4,3}f \to hs_{0,3}$, because no symbol f is available).

If all symbols a'' were replaced by a', then γ_3 introduces the symbol h and returns to its initial state; note that γ_2 is already in the initial state. Using the symbol h, also γ_1 can return to its initial state. In the current multiset, the number of occurrences of a' is doubled in comparison with the number of such symbols in the previous configuration. During this time, a copy of g has been transformed in c.

When γ_2 enters the state $s_{4,2}$, we have two possibilities. If any copy of b or of c is present, then the computation will continue for ever. If no copy of b or of c is present, then the computation can continue only until all symbols a' are replaced by a and then it stops: the component γ_3 can proceed further only using occurrences of the symbol f and such symbols are produced only by γ_2, which is no longer able to introduce f.

In conclusion, we can double n times the number of occurrences of a, that is, we stop in a configuration which contains 2^n copies of a. We may say that the system Π above computes the function $f(n) = 2^n, n \geq 1$.

One can modify this system in order to generate the language $\{a^{2^n} \mid n \geq 1\}$, but we leave this task to the reader.

5 The power of PT systems; The main result

We pass now to investigating the generative power of PT systems. The main result in this sense is the next one, showing that our mechanisms are very powerful.

Theorem 1. $gsm(pRE) \subseteq PTL_2$.

Proof. It is known that $RE = MAT_{ac}$; this implies that $pRE = pMAT_{ac}$. Consider a matrix grammar with appearance checking $G = (N, T, S, M, C)$ and a gsm $\gamma = (K, T, V_2, q_0, F, P)$. Assume that G is in the binary normal form (hence $N = N_1 \cup N_2 \cup \{S, \dagger\}$) and that it contains k matrices of the form $m_i : (X_i \to \alpha_i, A_i \to x_i)$, $1 \leq i \leq k$, for some $X_i \in N_1, \alpha_i \in N_1 \cup \{\lambda\}$, $A_i \in N_2, x \in (N_2 \cup T)^*$, and n matrices of the form $m_j : (X_j \to Y_j, A_j \to \dagger)$, $k+1 \leq j \leq k+n$, for some $X_j, Y_j \in N_1, A_j \in N_2$. Let XA be the symbols introduced by the unique initial matrix, $(S \to XA), X \in N_1, A \in N_2$.

For a string $x \in (N_2 \cup T)^*$ we denote by $\bar{x}$ the string obtained by replacing each terminal symbol a which appears in x by $\bar{a}$ (the nonterminal symbols remain unchanged).

We construct the PT system (of degree 2)

$$\Pi = (V, V_2, w_0, \gamma_1, \gamma_2),$$

with

$$\begin{aligned} V &= N_1 \cup N_2 \cup V_2 \cup \{\dagger, c, d, e\} \cup \{\bar{a} \mid a \in T\}, \\ w_0 &= XAcc, \end{aligned}$$

and the following components:

$$\begin{aligned}
\gamma_1 &= (K_1, V, s_{0,1}, P_1), \\
K_1 &= \{s_{i,1} \mid 0 \le i \le k+n+1\} \\
&\cup \{s'_{i,1} \mid 1 \le i \le k\} \\
&\cup \{[q,r,i] \mid r : qa \to zq' \in P, q, q' \in K, a \in T, z \in V_2^*, 1 \le i \le |z|, |z| \ge 2\} \\
&\cup K \cup \{q_f\}, \\
P_1 &= \{s_{0,1}X_i \to \alpha_i s_{i,1},\ s_{i,1}A \to \bar{x}s'_{i,1}, \\
&\quad s'_{i,1}c \to cs_{0,1},\ s_{i,1}c \to cs_{k+n+1,1} \mid 1 \le i \le k\} \\
&\cup \{s_{0,1}X_j \to Y_j s_{j,1},\ s_{j,1}d \to es_{0,1}, \\
&\quad s_{j,1}A_j \to cs_{k+n+1,1} \mid k+1 \le j \le k+n\} \\
&\cup \{s_{0,1}c \to cq_0\} \\
&\cup \{qc \to cq \mid q \in K\} \\
&\cup \{q\bar{a} \to \alpha q' \mid qa \to \alpha q' \in P, \alpha \in V_2 \cup \{\lambda\}\} \\
&\cup \{q\bar{a} \to a_1[q,r,1],\ [q,r,1]c \to ca_2[q,r,2], \ldots, \\
&\quad [q,r,t-2]c \to ca_{t-1}[q,r,t-1],\ [q,r,t-1]c \to ca_t q' \mid \\
&\quad \text{for } r : qa \to zq' \in P, z = a_1a_2\ldots a_t, t \ge 2, a_i \in V_2, 1 \le i \le t\} \\
&\cup \{qc \to cq_f \mid q \in F\} \\
&\cup \{q_f\alpha \to \alpha q_f \mid \alpha \in N_1 \cup N_2 \cup \{\bar{a} \mid a \in T\}\}, \\
\gamma_2 &= (K_2, V, s_{0,2}, P_2), \\
K_2 &= \{s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}\}, \\
P_2 &= \{s_{0,2}c \to cs_{1,2},\ s_{1,2}c \to ds_{2,2},\ s_{2,2}e \to cs_{0,2}, \\
&\quad s_{2,2}d \to cs_{0,2},\ s_{0,2}c \to cs_{3,2}\} \\
&\cup \{s_{3,2}\alpha \to \alpha s_{3,2} \mid \alpha \in N_1 \cup N_2\}.
\end{aligned}$$

The system is schematically represented in Figure 2 (γ_1 is only partially given).

Let us examine the work of this system.

We start from the multiset represented by $XAcc$; as long as a nonterminal symbol of G is present, the component γ_2 cannot stop: we either cycle in states $s_{0,2}, s_{1,2}, s_{2,2}, s_{0,2}$, or we cycle in state $s_{3,2}$. Therefore, we can halt only when no nonterminal symbol is present.

If γ_1 moves from $s_{0,1}$ to q_0 (the initial state of the gsm γ), then it never returns to $s_{0,1}$. From q_0, we simulate the work of γ, in the following way.

First, note that in each state $q \in K$ we can work for ever using the rule $qc \to cq$. Thus, we have to reach the state q_f. Also in this state we can work for ever if a nonterminal symbol of G is present or any symbol of the form $\bar{a}$, for $a \in T$, is present. Such barred symbols are introduced by the rules which simulate matrices in M (see below). Consequently, after entering the state q_0, we can finish the work of γ_1 only if no nonterminal is present and all terminals which are present (in the barred form) are correctly parsed by the gsm γ.

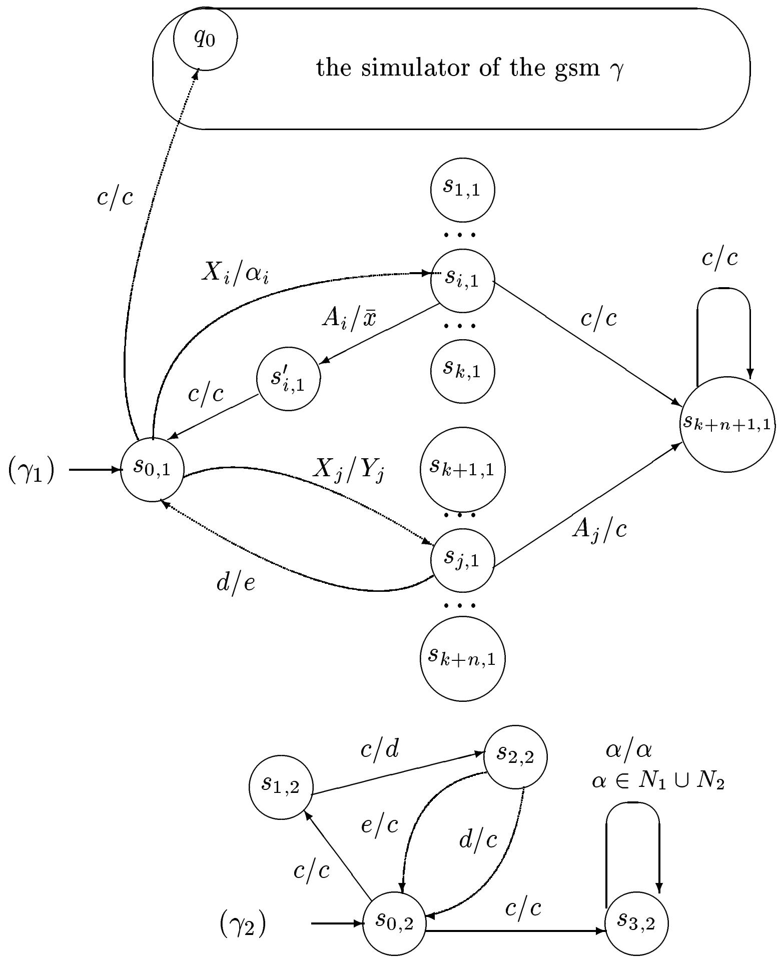

Figure 2: The PT system in the proof of the main theorem.

The parsing through γ can be simulated at any time; in particular, we can do that after eliminating all the nonterminals (this means that a derivation in G is completely simulated, see below). This is ensured by the fact that in each state $q \in K$ we can wait as much as we need, by using the rule $qc \to cq$. In this way, we have at our disposal all the terminal symbols, hence we can process them in any order we want. Otherwise stated, any permutation of a string generated by G is available and we can translate it. Note also the important fact that the rules of the form $q\bar{a} \to a_1[q,r,1]$, $[q,r,1]c \to ca_2[q,r,2], \ldots, [q,r,t-1]c \to ca_tq'$, corresponding to a rule $r : qa \to a_1 \ldots a_tq'$ from P, introduce the symbols $a_1, \ldots, a_t$ one by one, hence in the order imposed by the rule r (if all these symbols were produced at the same time, then any permutation of them has to be considered, which is not correct).

What remains is to show that each derivation with respect to G can be simulated in Π.

Assume that we are in a configuration $(w, s_{0,1}, s_{0,2})$ (initially, $w = XAcc$).

If we use a rule $s_{0,1}X_i \to \alpha_i s_{i,1}$ in γ_1, for $m_i : (X_i \to \alpha_i, A_i \to x_i)$ in M (at this time, γ_2 passes to state $s_{1,2}$), then at the next step we have to use the rule $s_{i,1}A_i \to \bar{x}s'_{i,1}$. Indeed, if we pass to state $s_{k+n+1,1}$, then the computation never stops. This means that the use of the matrix m_i is correctly simulated, both its rules were used. We return to $s_{0,1}$ at the same time when γ_2 returns to $s_{0,2}$ (after using the rules $s_{1,2}c \to ds_{2,2}, s_{2,2}d \to cs_{0,2}$). The process can be iterated.

If in the configuration $(w, s_{0,1}, s_{0,2})$ we use a rule $s_{0,1}X_j \to Y_j s_{j,1}$ for some matrix m_j : $(X_j \to Y_j, A_j \to \dagger), k+1 \leq j \leq k+n$, from M (at the same step, γ_2 passes to state $s_{1,2}$), in state $s_{j,1}$ we have two possibilities.

If the symbol A_j is present in the current configuration, then we have to use the rule $s_{j,1}A_j \to cs_{k+n+1,1}$ and the computation will never finish. If the symbol A_j is not present, then γ_1 cannot work, we remain in state $s_{j,1}$. The component γ_2 uses the rule $s_{1,2}c \to ds_{2,2}$. After that, again we have two possibilities for the next step. If the symbol d is used by γ_2, then γ_2 returns to $s_{0,2}$ (and the symbol c is reproduced), while γ_1 should wait until an occurrence of d will be produced. Thus, nothing happens in this case. If the symbol d introduced by γ_2 is used by γ_1, then γ_1 returns to its initial state and the symbol e is introduced. At the next step, this symbol will be used by γ_2 in order to return to its initial state, too.

One can see that again we simulate correctly a matrix in M, namely one with a rule used in the appearance checking manner.

When γ_2 uses the rule $s_{2,2}e \to cs_{0,2}$ (in this way, the symbol c is reintroduced), γ_1 can either start simulating a matrix of type $m_i, 1 \leq i \leq k$, or a matrix $m_j, k+1 \leq j \leq k+n$. In the first case, no interference appears between the two components. In the latter case, γ_1 will arrive in a state $s_{j,1}$ with $k+1 \leq j \leq k+n$ and either it will go to $s_{k+n+1,1}$ (which leads to a non-halting computation), or it waits in state $s_{j,1}$ until the symbol d is introduced by component γ_2.

In all cases, we get computations which can halt only when we correctly simulate the matrices of G. As we have seen above, when the derivation in G which is simulated by Π is terminal, and only in this case, we can also terminate the computation, reaching a halting configuration. In conclusion, $L(\Pi)$ consists exactly of all strings $w \in \gamma(z)$, for z being a permutation of a string in $L(G)$. This concludes the proof. □

The previous theorem has a series of interesting consequences.

On the one-letter alphabet the permutation of a language is equal to the language, so $RE^{one} \subseteq PTL_2^{one}$. The inclusion $PTL \subseteq RE$ follows from Church-Turing thesis (or can be proved in a direct, constructive, manner). Consequently, we get:

Corollary 1. $RE^{one} = PTL_2^{one}$.

If we start the construction in the proof of Theorem 1 from a matrix grammar without appearance checking, then component γ_2 is useless, which implies the next result:

Corollary 2. $gsm(pMAT) \subseteq PTL_1$.

Consider the language

$$D_4 = \{w \in \{a_1, a_2, b_1, b_2\}^* \mid |w|_{a_i} = |w|_{b_i}, i = 1, 2\}$$

(the number of occurrences of a_1, a_2 is equal to the number of occurrences of b_1, b_2, respectively). This language is a generator of the family of context-free languages, CF (see [4], pag. ??), hence each context-free language L can be written in the form $L = \gamma(D_4)$, for a gsm γ. The language D_4 is context-free and permutation closed, therefore it belongs to $pMAT$. Because $gsm(pMAT) \subseteq PTL_1$, we obtain

Corollary 3. $CF \subset PTL_1$.

We do not know whether or not the inclusion $RE \subseteq PTL_2$ (or $RE \subseteq PTL$) holds. For bounded languages, such a relation is true.

Corollary 4. $RE^{bound} = PTL_2^{bound}$.

Proof. Consider a language $L \subseteq V^*, L \subseteq w_1^* \dots w_n^*$, for some $w_1, \dots, w_n \in V^+$. Take the new symbols $a_1, \dots, a_n$ and define the morphism $h : \{a_1, \dots, a_n\}^* \longrightarrow V^*$ by $h(a_i) = w_i, 1 \leq i \leq n$. Denote by $p(M)$ the permutation of a language M. If $L \in RE$, then $h^{-1}(L) \in RE, h^{-1}(L) \subseteq a_1^* \dots a_n^*$. From Theorem 1 it follows that $p(h^{-1}(L)) \in PTL_2$. We can write

$$L = h(p(h^{-1}(L)) \cap a_1^* \dots a_n^*).$$

A morphism and an intersection with a regular language can be realized by a gsm; again from Theorem 1, we get $L \in PTL_2$. Therefore, $RE^{bound} \subseteq PTL_2^{bound}$. The converse inclusion follows from Church-Turing thesis (or can be directly proved). □

6 Another collapsing hierarchy

Theorem 1 shows that the hierarchy $PTL_n, n \geq 1$, collapses at the second level. The component γ_1 of the PT system in the proof of Theorem 1 has a number of states which depends on the gsm γ and the starting matrix grammar G. We do not see a way to avoid the dependence on γ. However, if we do not look for gsm images of permutations of languages in RE, then we can avoid the dependence on the starting grammar G: the hierarchy on the maximal number of states in the components of PT systems which generate languages in pRE collapses at the fourth level:

Theorem 2. *For each language $L \in pRE$ there is a PT system Π such that $L = L(\Pi)$ and each component of Π has at most four states.*

Proof. Starting from a matrix grammar $G = (N, T, S, M, C)$ in the binary normal form, with k matrices of the form $m_i : (X_i \to \alpha_i, A_i \to x_i), 1 \leq i \leq k$, and n matrices of the form $m_j : (X_j \to Y_j, A_j \to \dagger), k+1 \leq j \leq k+n$, we construct a PT system as follows:

$$\Pi = (V, T, w_0, \gamma_1, \dots, \gamma_{k+n}, \gamma_{k+n+1}, \gamma_{k+n+2}),$$

with

$$\begin{aligned} V &= N_1 \cup N_2 \cup T \cup \{\dagger, c, d, e\} \cup \{\bar{a} \mid a \in T\}, \\ w_0 &= XAcc, \end{aligned}$$

and the following components:

$$\gamma_i \;\; = \;\; (K_i, V, s_{0,i}, P_i), \text{ for } 1 \leq i \leq k, \text{ with}$$

$$
\begin{aligned}
K_i &= \{s_{0,i}, s_{1,i}, s'_{1,i}, s_{2,i}\},\\
P_i &= \{s_{0,i}X_i \to s_{1,i},\ s_{1,i}A \to \bar{x}s'_{1,i},\\
&\quad s'_{1,i}c \to c\alpha_i s_{0,i},\ s_{1,i}c \to cs_{2,i},\ s_{2,i}c \to cs2,i\},\\
\gamma_j &= (K_j, V, s_{0,j}, P_j),\ \text{for } k+1 \le j \le k+n,\ \text{with}\\
K_j &= \{s_{0,j}, s_{1,j}, s_{2,j}\},\\
P_j &= \{s_{0,j}X_j \to s_{1,j},\ s_{1,j}d \to eY_j s_{0,j},\\
&\quad s_{1,j}A_j \to cs_{2,j},\ s_{2,j}c \to cs_{2,j},\ s_{2,j}c \to cs_{2,j}\},\\
\gamma_{k+n+1} &= (K_{k+n+1}, V, s_{0,k+n+1}, P_{k+n+1}),\\
K_{k+n+1} &= \{s_{0,k+n+1}, s_{1,k+n+1}, s_{2,k+n+1}, s_{3,k+n+1}\},\\
P_{k+n+1} &= \{s_{0,k+n+1}c \to cs_{1,k+n+1},\ s_{1,k+n+1}c \to ds_{2,k+n+1},\\
&\quad s_{2,k+n+1}e \to cs_{0,k+n+1},\ s_{2,k+n+1}d \to cs_{0,k+n+1},\\
&\quad s_{0,k+n+1}c \to cs_{3,k+n+1}\}\\
&\cup\ \{s_{3,k+n+1}\alpha \to \alpha s_{3,k+n+1} \mid \alpha \in N_1 \cup N_2\},\\
\gamma_{k+n+2} &= (K_{k+n+2}, V, s_{0,k+n+2}, P_{k+n+2}),\\
K_{k+n+2} &= \{s_{0,k+n+2}, s_{1,k+n+2}\},\\
P_{k+n+2} &= \{s_{0,k+n+2}\bar{a} \to \bar{a}s_{0,k+n+2},\ s_{0,k+n+2}\bar{a} \to as_{0,k+n+2} \mid a \in T\}\\
&\cup\ \{s_{0,k+n+2}c \to s_{1,k+n+2}\}\\
&\cup\ \{s_{1,k+n+2}\alpha \to \alpha s_{1,k+n+2} \mid \alpha \in N_1 \cup N_2 \cup \{\bar{a} \mid a \in T\}.
\end{aligned}
$$

This system works in a way similar to that in the proof of Theorem 1. The components $\gamma_i, 1 \le i \le k$, simulate the matrices in M which do not involve rules used in the appearance checking mode. The components $\gamma_j, k+1 \le j \le k+n$, simulate the matrices which contain rules used in the appearance checking mode. Note that in each moment only one of these components can work, because only one occurrence of a symbol from N_1 is present in the current multiset; moreover, the use of a rule $X_j \to Y_j$, for anu i, is completed only when the simulation of the matrix in which this rule appears is completed (the symbol Y_i is introduced by a rule which returns to the initial state of the component γ_i). The component γ_{k+n+1} is used, as in the proof of Theorem 1, for ensuring the correct simulation of the matrices which contain rules used in the appearance checking manner.

The component γ_{k+n+2} is used for permuting the terminal symbols, at the end of a computation, in such a way to obtain all permutations of strings in $L(G)$ (we can wait in state $s_{0,k+n+2}$ as long as we need). Moreover, this component checks whether or not the derivation is terminal: if any nonterminal of G is present in the configuration, then we can cycle in state $s_{1,k+n+2}$.

In conclusion, $L(\Pi) = p(L(G))$. □

Corollary 5. *Each language in pMAT can be generated by a PT system whose components have at most three states each.*

Proof. If we start the construction in the previous proof from a matrix grammar without appearance checking, then the component γ_{k+n+1} is no longer necessary (of course, also the components $\gamma_j, k+1 \le j \le k+n$, are missing); moreover, we can also avoid using the states $s'_{1,i}$ in components $\gamma_i, 1 \le i \le k$. Consequently, each component contains three states, with the exception of the component γ_{k+1} – the former component γ_{k+n+2} – which has two states. This completes the proof. □

The number of components of systems constructed in the proofs of Theorem 2 and Corollary 5 depends on the starting grammar.

In all the results from this and the previous section, the length of the string x in rules $sa \to xs'$ of the components of the PT systems we use can be bounded by two: start from a matrix grammar in the binary normal form having the string z in matrices $(X \to \alpha, A \to z)$ of length at most two (this can be arranged – see [5]). One can see from the constructions that the obtained PT system has the desired property.

7 Closure properties

A way to estimate the size of a family of languages is to consider its closure properties. From this point of view, the family PTL seems rather large:

Theorem 3. *The family PTL is a full AFL.*

Proof. It is easy to see that the family PTL is closed under arbitrary gsm mappings; this implies the closure under arbitrary morphisms, intersection with regular languages, and inverse morphisms.

Union. Consider two PT systems $\Pi_i = (V_i, T_i, w_{0,i}, \gamma_{1,i}, \ldots, \gamma_{n_i,i}), i = 1, 2$. Without loss of generality we may assume that the states used by components of Π_1 are different from those used by the components of Π_2 and that also $V_1 - T_1$ is disjoint of $V_2 - T_2$. We construct the system

$$\Pi = (V_1 \cup V_2 \cup \{d, d_1, d_2\}, T_1 \cup T_2, dw_{0,1}w_{0,2}, \gamma_0, \gamma'_{1,1}, \ldots, \gamma'_{n_1,1}, \gamma'_{1,2}, \ldots, \gamma'_{n_2,2}),$$

with

$$\begin{aligned}
\gamma_0 &= (\{s_0\}, V_1 \cup V_2 \cup \{d, d_1, d_2\}, s_0, \{s_0 d \to d_1^{n_1} s_0,\ s_0 d \to d_2^{n_2} s_0\}),\\
\gamma'_{i,j} &= (K_{i,j} \cup \{s'_{0,i,j}\}, V_j, s'_{0,i,j}, P_{i,j} \cup \{s'_{0,i,j} d_j \to s_{0,i,j}),\\
&\quad \text{for all } 1 \le i \le n_j, j = 1, 2.
\end{aligned}$$

One can easily see that we first work in the new component, γ_0, introducing either n_1 occurrences of d_1 or n_2 occurrences of d_2. In this way, all components of Π_1 or all components of Π_2 pass simultaneously to their initial states. From now on, these components work exactly as they are doing in the initial system. Because we have only one occurrence of d, γ_1 can work only once, hence only the components of one of Π_1, Π_2 are activated. Consequently, we get $L(\Pi) = L(\Pi_1) \cup L(\Pi_2)$.

Concatenation. Start from two systems $\Pi_i, i = 1, 2$, as above, with disjoint sets of states and sets of non-terminal symbols, and construct a new system as follows. Instead of a formal (highly cumbersome) construction, we indicate it in Figure 3 and describe it informally.

As one can see in Figure 3, we have two new components, γ_0 and γ'_0, and modified variants of all the components of Π_1 and Π_2. In particular, for each $\gamma_i, 1 \le i \le n_1$, we consider $\gamma'_{i,1}$, which "contains" $\gamma_{i,1}$ as well as a modified copy of $\gamma_{i,1}$ with all states in the form $\bar{s}$. From each state s in the copy of $\gamma_{i,1}$ to the corresponding state $\bar{s}$ in the modified copy of $\gamma_{i,1}$ we have a transition, via a rule $sd_2 \to \bar{s}$. Moreover, for each move $sa \to xs'$ from $\gamma_{i,1}$, in the modified copy we introduce the rule $\bar{s}a \to Xx\bar{s}'$.

The initial multiset is again $dw_{0,1}w_{0,2}$. At the first step, only the new component γ_0 can work; it introduces n_1 copies of d_1, which make possible the activation of all components of Π_1. These components (their copy from $\gamma'_{i,1}$) reaches their initial state and work as they work in Π_1. During this time, γ_0 can stay in state s_1, using the rule $s_1d \to ds_1$, while all other components of the system (γ'_0 and $\gamma'_{i,2}$, for $1 \le i \le n_2$) are doing nothing, because they do not have symbols to process.

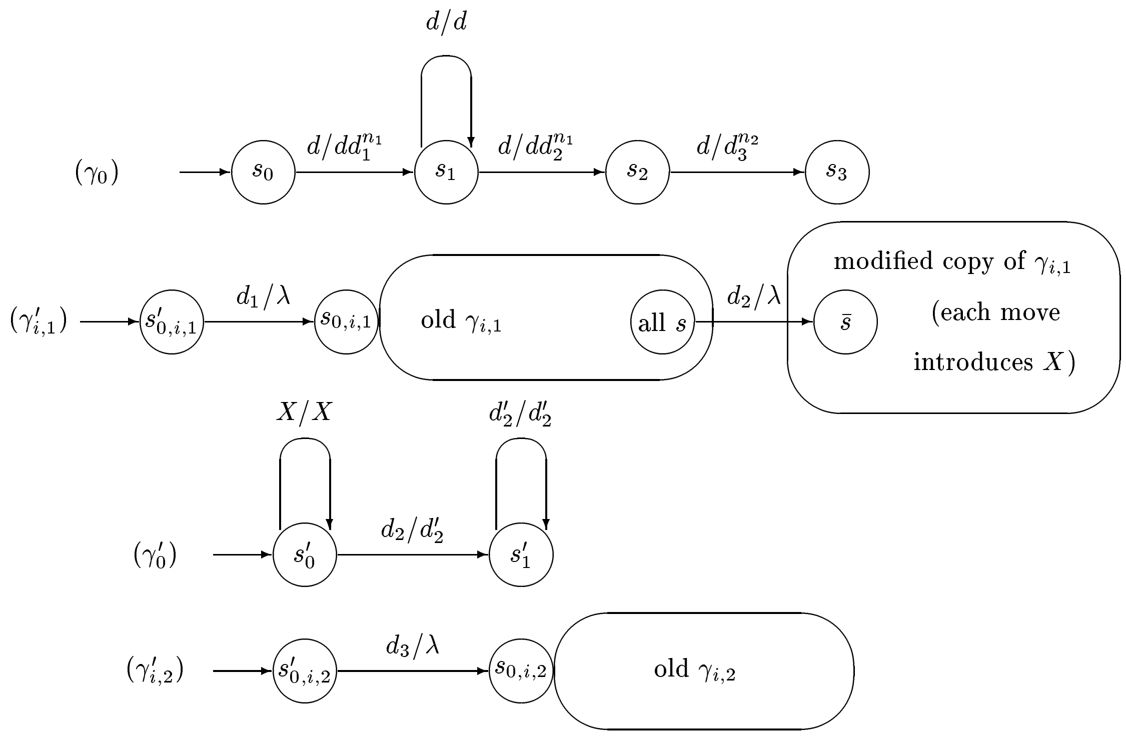

Figure 3: The construction for concatenation.

At any moment, γ_0 can introduce n_1 copies of d_2 and pass to state s_2. This is the moment when we want to finish the work of Π_1, to check whether or not this is done correctly (we have a halting configuration from the point of view of Π_1), and to pass to also simulate Π_2. This is ensured by the "controller" component γ'_0. After having introduced n_1 copies of d_2, all these copies must be used at the next step by the components $\gamma'_{i,1}, 1 \le i \le n_1$, otherwise the component γ'_0 can use such a symbol and enter the cycle $s'_1d'_2 \to d'_2s'_1$, hence the computation will never finish. A symbol d_2 can be used by a component $\gamma'_{i,1}$ by the rule $sd_2 \to \bar{s}$, where s is the current state of $\gamma_{i,1}$ reached in $\gamma'_{i,1}$ and $\bar{s}$ is a copy of it. Note that rules of the form $sd_2 \to \bar{s}$ are introduced for all states s, but only one per component can be used, because we are in a given state. If we are in a halting configuration of $\gamma_{i,1}$ (it is possible that such a configuration has been reached at a previous step and we have just waited for the symbol

d_2 to be introduced), then we also move to a halting configuration of the copy of $\gamma_{i,1}$, that which uses barred states. If this is not the case, that is at least a further transition can be performed in $\gamma_{i,1}$, then this is also possible in the copy of $\gamma_{i,1}$ now activated in $\gamma'_{i,1}$. Because each rule of this copy introduces an occurrence of the symbol X, the computation is again lost: the "controller" component will use this symbol X, working for ever.

In conclusion, when the state s_2 is reached in γ_0, we can continue the computation without entering a cycle if and only if a halting configuration was reached in Π_1.

The component γ_0 introduces now n_2 copies of d_3, which activate the components $\gamma'_{i,2}, 1 \leq i \leq n_2$; in this way, we continue working as in Π_2, hence we obtain the concatenation of the languages $L(\Pi_1)$ and $L(\Pi_2)$.

Kleene closure. Consider a PT system $\Pi = (V, T, w_0, \gamma_1, \ldots, \gamma_n)$. We proceed as above, indicating the construction by a picture – Figure 4 – and then discussing it informally.

The new component γ_0 controls the iteration of using the system Π (in the new form, where the components γ_i were modified to $\gamma'_i, 1 \leq i \leq n$, in a way similar to that in the proof of the closure under concatenation), and γ'_0 is again the "controller" of the correct termination of a computation in Π before starting another computation.

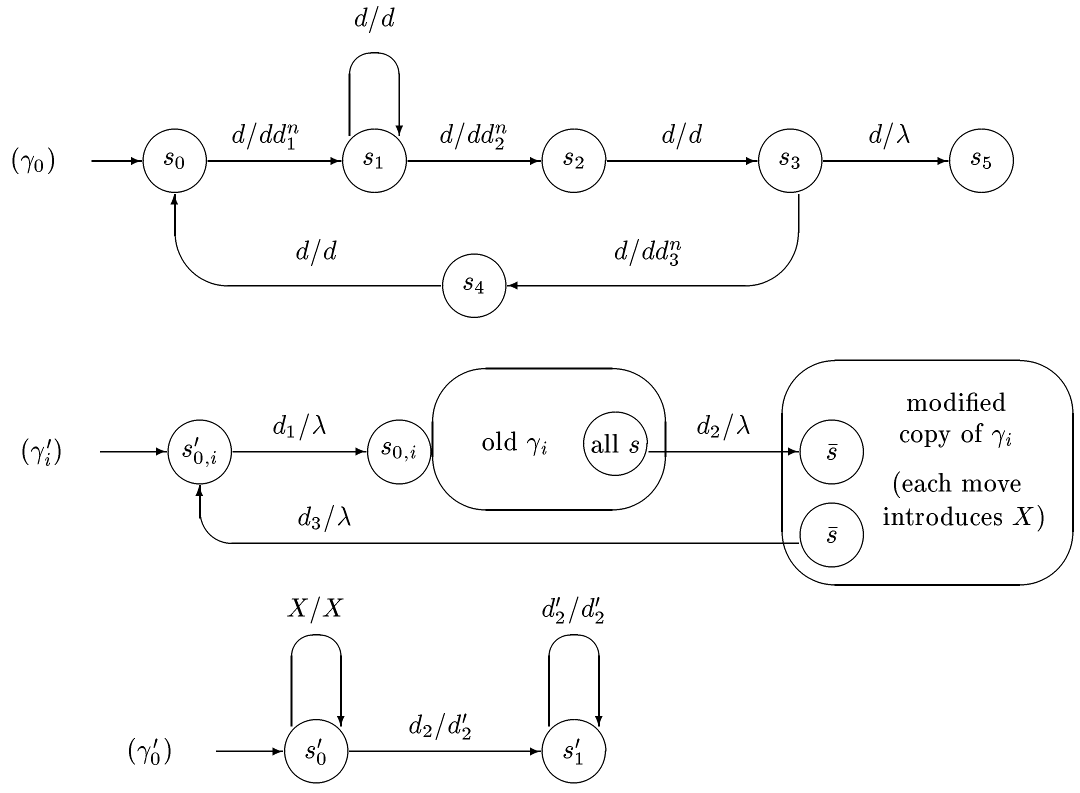

Figure 4: The construction for Kleene +.

The initial multiset is dw_0. At the first step we can work only in γ_0, where n occurrences of d_1 are introduced. Now, each γ'_i can start working. We enter the initial states of each γ_i

and then we work as in γ_i (at this time γ_0 stays in state s_1 and γ_0' stays in its initial state). At any time, γ_0 can introduce the symbol d_2 (again, n copies). The copies of d_2 should be used for passing from the current states of each γ_i to the barred version of that state in the modified copy of γ_i included in γ_i' (otherwise the computation will never stop, because γ_0' cycles in s_1'). As in the case of concatenation, if the computation in Π is not completed, then the symbol X is introduced and the computation will continue for ever in γ_0'. At this time, γ_0 passes to state s_3. At the next step, each component of the system returns to its initial state, by rules of the form $\bar{s}d_3 \rightarrow s_{0,i}'$, made active by the introduction of n copies of d_3 by γ_0. At the same step, also γ_0 returns to its initial state.

In this way, we can get the concatenation of any number of strings in $L(\Pi)$. After producing any number of strings in $L(\Pi)$ (maybe only one), we can pass to state s_5 of γ_0, which ends the computation.

In conclusion, we produce $L(\Pi)^+$, which concludes the proof of the closure under Kleene $+$ and the proof of the theorem, too. □

8 Final remarks

We have here introduced a computing model which belongs both to natural computing area (computing with membranes, [8], [9]) and to multiset processing ([1], [2], [3], etc.): several finite automata with outputs (gsm's) swim in a space where a multiset of symbols is present and process these symbols in a parallel manner. We prove that such machines are rather powerful: they can generate at least all gsm images of permutations of recursively enumerable languages.

We do not know whether or not we cover in this way all recursively enumerable languages, or whether the above result is the best we can obtain (whether or not languages which are not gsm images of permutations of recursively enumerable languages can be generated by our systems).

Several other problems remain to be investigated. For instance, we have considered here non-deterministic gsm's. What about using only deterministic components in our systems? Actually, we have here two types of non-determinism, one at the level of components (using non-deterministic gsm's) and one at the level of the whole system: in a given configuration, the component which takes a copy of a symbol and processes it is non-deterministically choosen among those which can do it. For instance, if we have n copies of the symbol a and $n+1$ components can take this symbol at that moment, only n of them will work on a; the remaining component will either wait, or will use a symbol different from a, if this is possible.

A way to diminish the non-determinism at the level of the system is to consider a priority relation among components: in each moment, the components are enabled in the decreasing order of their priority.

However, even a total ordering of components does not remove completely the non-determinism: if in a given state of a gsm we can read both a and b, and these symbols are present in the current multiset, then we may choose one of the two possible steps (note that this does not appear when gsm's translate strings, because in that case at each moment only one symbol is scanned by the read head).

On the other hand, systems which are *completely deterministic* (at each moment, only one next configuration is possible) do not seem to be of much interest: they can proceed along a unique computation, which either stops (hence the generated language is at most finite), or continues for ever (hence the language is empty).

The study of determinism in PT systems, at various levels, deserves a further investigation.

References

[1] J. P. Banâtre, A. Coutant, D. Le Metayer, A parallel machine for multiset transformation and its programming style, *Future Generation Computer Systems*, 4 (1988), 133–144.

[2] J. P. Banâtre, D. Le Metayer, Programming by multiset transformation, *Communications of the ACM*, 36 (1993), 98–111.

[3] G. Berry, G. Boudol, The chemical abstract machine, *Theoretical Computer Sci.*, 96 (1992), 217–248.

[4] J. Berstel, *Transductions and Context-Free Languages*, Teubner, Stuttgart, 1979.

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

[6] J. Dassow, Gh. Păun, On the power of membrane computing, *J. Univ. Computer Sci.*, 5, 2 (1999), 33–49. (see also *TUCS Research Report* No. 217, December 1998, http://www.tucs.fi).

[7] A. Kelemenova, J. Kelemen, Colonies, ???

[8] Gh. Păun, Computing with membranes, submitted, 1998 (see also *TUCS Research Report* No. 208, November 1998, http://www.tucs.fi).

[9] Gh. Păun, Computing with membranes. An introduction, *Bulletin of the EATCS*, 67 (1999), 139–152.

[10] Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Berlin, 1998.

[11] Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, submitted, 1998 (see also *TUCS Research Report* No. 218, December 1998, http://www.tucs.fi).

[12] I. Petre, A normal form for P systems, *Bulletin of the EATCS*, 67 (1999), 165–172.

[13] G. Rozenberg, A. Salomaa, Eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, Heidelberg, 1997.