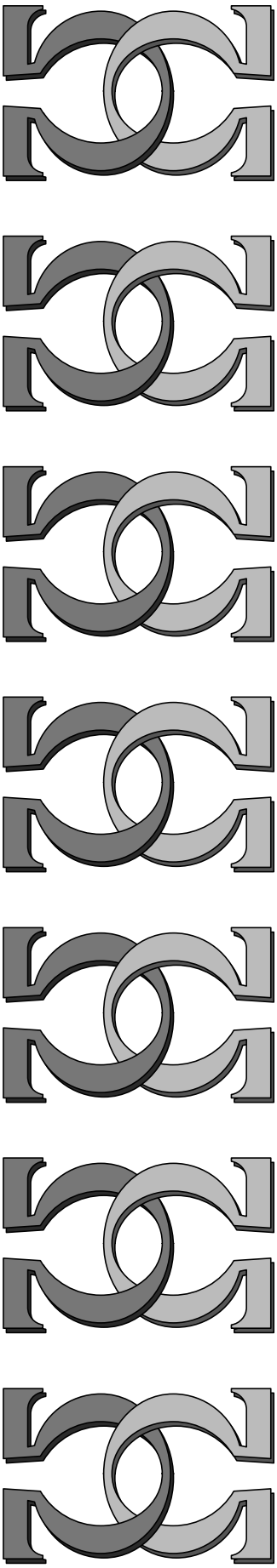

CDMTCS Research Report Series

Supplemental Papers for DLT'04

C. S. Calude[1], E. Calude[2] and M. J. Dinneen[1] (Editors)

[1]University of Auckland,
New Zealand

[2]Massey University at Albany,
New Zealand

CDMTCS-252
November 2004

Centre for Discrete Mathematics and
Theoretical Computer Science

Preface

These are the papers for the poster talks to be given at the Eighth *International Conference on Developments in Language Theory* (DLT'04) to be held at Auckland, New Zeland on December 13–17, 2004. The conference is jointly organized by Massey Universty at Albany and the CDMTCS (New Zealand). The conference, organised under the auspicies of the European Association for Theoretical Computer Science (EATCS), is supported by the New Zealand Royal Society.

On Regular Language Factorisation: A Complete Solution for Unary Case

Sergey Afonin[1,2], Elena Hazova[2], and Alexander Shundeev[1,2]

[1] Moscow State University, Institute of Mechanics
Moscow, Russia
[2] Center for Scientific Telecommunications,
Russian Academy of Sciences
Moscow, Russia
{serg, shundeev}@msu.ru

Abstract. In this paper the problem of regular language decomposition as a concatenation of given regular languages is considered. We prove that the problem is decidable in unary case when all the languages are in single-letter alphabet. This result may be used to give a negative answer in some cases of the general problem.

1 Introduction

Language factorisation problem, i.e. the problem of representing a given regular language as a concatenation of other (regular) languages has a long history. The classical results, such as Krohn-Rhodes decomposition theorem for transformation semigroups [4], or language factorisation into a finite number of stars and primes [5] guarantees the existence of such decompositions. These works, however, deal with "unrestricted" case of the problem, when factors are arbitrary languages of certain classes (i.e. stars and primes). The problem of interest is: how a given regular language can be represented in terms of a *fixed* set of arbitrary languages.

This problem was motivated by the semi-structured data processing problem. Semistructured data naturally arise in many areas, including integration of data from heterogeneous sources, processing text documents with semantic markup, etc [11, 17]. A powerful mathematical model for semi-structured data is an edge-labeled directed graph [16]. Nodes of the graph correspond to objects in the subject area, and edges are relations between them. In this model the relationships between objects are represented as *paths* in the database graph $\mathcal{B}$ and the following problem typically arise: for given regular language Q (query), find all the pairs (u, v) of $\mathcal{B}$ nodes such that there exist a labeled path between u and v in $\mathcal{B}$ and its labels comprise a word in Q [15, 1]. Although this problem has polynomial complexity, the whole graph $\mathcal{B}$ may need to be searched, which is inefficient.

One possible method towards increasing efficiency is using views [9, 7]. Let us assume that we know the search results for the queries, corresponding to regular languages $\mathcal{E} = \{E_1, \ldots, E_k\}$. The question is, can this data be used to help execution of an arbitrary query Q? This is the language substitution problem. As it was noted in [2], the efficiency of query processing (especially parallel processing) significantly depends on query structure and concatenation of E_i can be effectively evaluated.

The layout of the paper is as follows. Section 2 introduces the basic concepts and formulates the problem of "constrained" rewriting. A brief survey of related work

is given in section 3. In section 4 the problem is considered in its general case. A complete solution for the case of single-letter alphabet presented in section 5. The conclusion discusses the results and directions of future work.

2 Preliminaries

An *alphabet* is a finite non-empty set Σ of *symbols*. A finite sequence of symbols from Σ is called a *word* in Σ. The *empty* word is denoted ε.

Any set of words is called a *language* in Σ. Σ^* denotes the set of all words in a given alphabet, $\varnothing$ is an empty language (containing no words). The union of languages L_1 and L_2 is the language $L_1 + L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$. The language $L_1 L_2 = \{w \in \Sigma^* \mid \exists w_1 \in L_1, w_2 \in L_2 : w = w_1 w_2\}$ is called a concatenation of L_1 and L_2. $L^k = LL^{k-1}$ is L to the power k. By definition, L to the power zero is the empty word: $L^0 = \{\varepsilon\}$. The Kleene closure of L is the language $L^* = \cup_{k=0}^{\infty} L^k$. A language is *regular* if it can be obtained from letters of the alphabet, empty language, and $\{\varepsilon\}$ using a finite number of operations of concatenation, union and closure.

Let $\Sigma_{\mathcal{E}} = \{E_1, E_2, \ldots, E_k\}$ be a fixed set of regular languages in Σ, and $Q \subseteq \Sigma^*$. Let $\Sigma_{\mathcal{E}} = \{e_1, \ldots, e_k\}$ be a *view* alphabet, such that $\Sigma \cap \Sigma_{\mathcal{E}} = \varnothing$. Define the language mapping

$$L_\Sigma : \Sigma_{\mathcal{E}} \to \mathcal{P}(\Sigma^*)$$

as $L_\Sigma(e_i) = E_i$. The subscript index indicates the alphabet of image language. This mapping can be naturally extended to language morphism $L_\Sigma : \mathcal{P}(\Sigma_{\mathcal{E}}^*) \to \mathcal{P}(\Sigma^*)$. For (possibly non-regular) language $P \subseteq \Sigma_{\mathcal{E}}^*$, let

$$L_\Sigma(P) = \bigcup_{w \in P} L_\Sigma(w). \tag{1}$$

Definition 1. *Let $\mathcal{H}(A)$ be a class of regular languages in an alphabet A. Q' is an* $\mathcal{H}$-rewriting *of $Q \subseteq \Sigma^*$ with respect to $\mathcal{E}$ if (1) $Q' \in \mathcal{H}(\Sigma_{\mathcal{E}})$ and (2) $L_\Sigma(Q') = Q$.*

Let us denote the class of languages that consist of a single word as $\mathcal{H}_c(A)$, the class of languages that can be obtained from symbols of A using a finite number of concatenations and closures as $\mathcal{H}_*(A)$, and the class of arbitrary regular languages in A as $\mathcal{H}_R(A)$.

This work deals with a $\mathcal{H}_c$-rewriting. The main problem of interest is:

Problem 1. For a given regular language $Q \subseteq \Sigma^*$ and a fixed set $\mathcal{E} = \{E_1, \ldots, E_k\}$ of regular languages in Σ, find a **word** w in $\Sigma_{\mathcal{E}}$ such that $L_\Sigma(w) = Q$:

$$Q = E_{i_1} E_{i_2} \ldots E_{i_n} \tag{2}$$

3 Related works

There are several variations of the problem 1 considered in the literature. In general, they can be divided into two classes: (1) when the set $\mathcal{E}$ is fixed, and (2) when

elements of $\mathcal{E}$ are unknown. Star and primes decomposition [5] is an example of the problem with unknown elements of $\mathcal{E}$. For given language L one should find a word $w \in \Sigma_{\mathcal{E}}^*$ **and** the set $\mathcal{E}$ such that $L_\Sigma(w) = L$ and elements of $\mathcal{E}$ are either stars, or prime languages in Σ. Another example is the finite substitution problem [12]: for given languages $K \subseteq \Sigma_{\mathcal{E}}^*$ (the alphabet $\Sigma_{\mathcal{E}}$ is fixed) and $L \subseteq \Sigma^*$ find finite languages E_i, such that $L_\Sigma(K) = L$.

To our best knowledge the problem with fixed set $\mathcal{E}$ has been only considered for $\mathcal{H}_R$-rewritings [6, 8] and $\mathcal{H}_c$-rewriting of finite language [14]. In [6] the problem of finding a so-called *$\Sigma_{\mathcal{E}}$-maximal* $\mathcal{H}_R$-rewriting is solved. The language $Q' \subseteq \Sigma_{\mathcal{E}}^*$ is called a *$\Sigma_{\mathcal{E}}$-maximal* rewriting of $Q \subseteq \Sigma^*$, if $L_\Sigma(Q') \subseteq Q$ and Q' contains, as a language in $\Sigma_{\mathcal{E}}$, all other rewritings. That work propose an algorithm to find $\Sigma_{\mathcal{E}}$-maximal rewriting. Due to the fact that a $\Sigma_{\mathcal{E}}$-maximal rewriting Q' is not necessarily exact (i.e. $L_\Sigma(Q') \subseteq Q$), [6] proposes an algorithm to test whether a rewriting is exact. Both algorithms are shown to have exponential complexity.

The decidability of $\mathcal{H}_c$-rewriting for *finite languages*, which was considered in [14] follows from the fact that each component of decomposition (2) increase the length of the longest word, so the length of (2) is bounded by the length of the longest word in Q.

Another related area is language equations [13]. An interesting relation exists between language factorisation and language equations [13]. Consider the equation $L = EX$, where L and E are given regular languages. The questions are: whether a solution exists, whether it is regular if L and E are regular, whether a maximal solution exists and whether the equation is decidable. These questions were positively answered in [10]. It was also shown that if the equation $L = EX$ has a solution then it has at least one minimal solution (F is a minimal if no proper subset of F is a solution) but it is unclear how such a solution may be computed and how many minimal solutions exist.

The question that relates language decomposition and equation: Is it possible to build the decomposition by a number of "prefix removals"? If all the equations of the form $L = (E_{i_1} E_{i_2} \dots E_{i_m})X$ have a finite number of solutions then the problem 1 is decidable.

A recent work [3] proves the decidability of building an *unambiguous* decomposition of a language into $L = EX + Y$, i.e. each word in L can be uniquely decomposed in a right way. This is related to minimal solution for $L = EX$ since if a decomposition of the form $(X, \varnothing)$ exists then X is a minimal solution but in that paper only a "simple" case of one-word Y was considered.

4 Factorisation for arbitrary alphabet

Let us consider the problem of factorisation of a regular language into a concatenation of given ones in general case. Let us note that if none of the languages from $\mathcal{E}$ contain empty word the problem is decidable because the decomposition can not be longer than the shortest word in Q. It is also decidable when all of the languages from $\mathcal{E}$ are finite (with or without the empty word). The length of decomposition (2) of a (finite) language Q is limited because each component increase the length of

the longest word. If the languages E_i may contain empty word the question becomes "more complex". As it was noted in [14] only an exponential algorithm is known for the decomposition problem of finite language and we do not know whether it is decidable in the general case.

Reduction to star elimination problem. According to [6], it is decidable whether there exist the language $Q' \in \mathcal{H}_R(\Sigma_\mathcal{E})$ such that $L_\Sigma(Q') = Q$. It is well known that any regular language can be represented as *finite union* of languages in $\mathcal{H}_*$. Let

$$Q' = \bigcup_{i=1}^{d} R_i, \tag{3}$$

where $R_i \in \mathcal{H}_*(\Sigma_\mathcal{E})$ be such decomposition. A word $w \in \Sigma_\mathcal{E}^*$ such that $L_\Sigma(w) = Q$ exists if and only if $L_\Sigma(R_i) = Q$ for some i. Hence the original problem can be reformulated as follows.

Problem 2. Given a regular language $Q \in \Sigma^*$, a set of regular languages $\mathcal{E} = \{E_1, \ldots, E_k\}$ in Σ, and a regular language $R \in \mathcal{H}_*(\Sigma_\mathcal{E})$ such that $L_\Sigma(R) = Q$ one should decide whether there exist a word $w \in R$ such that $L_\Sigma(w) = Q$.

Example 1. Let $Q = a + aaa(a)^*$ and $\mathcal{E} = \{(aa)^*,\ a,\ (aaa)^*\}$. The maximal $\mathcal{H}_R$-rewriting of Q wrt $\mathcal{E}$ is the language

$$Q' = e_2 + (e_1 + e_3 + e_2e_2(e_1 + e_2 + e_3) + e_2(e_1 + e_3))(e_1 + e_2 + e_3)^*$$

One of the $\mathcal{H}_*(\Sigma_\mathcal{E})$-components of Q' is

$$R = e_1(e_1^*(e_3e_3^*e_1)^*)^*e_2(e_1^*e_3^*)^*$$

and $L_\sigma(R) = Q$. This language contains the word $w = e_1e_3e_1e_2$. It is easy to verify that $L_\Sigma(w) = Q$. □

Our solution is based on the *finite power property* of regular languages. For any language $L \subseteq \Sigma^*$ one can verify whether there exist $k \geqslant 0$ such that $L^* = L^k$. The minimal number k, $L^* = L^k$ is called the finite power of language L and denoted by $FPP(L)$. If no such number exists when $FPP(L) = \infty$.

It is evident what if all star languages from R has finite power property then all stars can be removed. Consider, for an instance, $(e_3e_3^*e_1)^*$ fragment of R from the above example. Since $L_\Sigma(e_3)$ is a star language then $e_3^* = e_3$. Thus, $(e_3e_3^*e_1)^* = (e_3e_3e_1)^*$. The language $e_3e_3e_1$ is a star also, so $(e_3e_3^*e_1)^* = (e_3e_3e_1)$. Unfortunately, there exist regular languages $L_1, L_2,$ and L_3 such that:

- $FPP(L_2) = \infty$;
- $L_1L_2^*L_3 = L_1L_2^kL_3$ for some k.

Let us consider one-letter alphabet $\Sigma = \{a\}$, and languages $L_1 = (aaa)^*$, $L_2 = (aa + \varepsilon)$, $L_3 = (aaaaa)^*$. The language $L_1L_2^*L_3$ contains all words over Σ, except a. Since L_2 is a finite language $FPP(L_2) = \infty$. From the other hand

$$L_1L_2^2L_3 = L_1L_2^*L_3.$$

This example demonstrates that finite power property can not be directly applied to the problem 2. We conject, however, that this generalised finite power property (then prefix and suffix languages are given) is crucial for language decomposition. More preciesly,

Conjecture 1. A regular language R of the form $R = L_1L_2^*L_3L_4^*L_5$ is equivalent to $L_1L_2^{k_2}L_3L_4^{k_4}L_5$ iff k_2 and k_4 satisfy the eqations $R = L_1L_2^{k_2}(L_3L_4^*L_5)$ and $R = (L_1L_2^*L_3)L_4^{k_4}L_5$, respectively.

It is worth noting that generalised finite power property is decidable using the same technique as for original one.

5 Unary case

In this section we prove what the problem 2 is decidable for arbitrary regular languages in $\Sigma = \{a\}$.

Let us first introduce the notation. First, since a language L in one-letter alphabet may contain only one word of length n, we will not distinguish a word and its length. Moreover, we will define a word and its length by the same symbol. Consequently, a set of words F is thought as the set of numbers.

Any regular language in $\Sigma = \{a\}$ can be represented as

$$L = F + \bigcup_{z \in Z} z_i(a^t)^*, \tag{4}$$

where F and Z are finite sets of words. The finiteness of Z follows form the structure of the deterministic automaton for L: each state has only one outgoing edge.

Lemma 1. *A regular language $L \neq \{\varepsilon\}$ in one-letter alphabet $\Sigma = \{a\}$ has finite power property iff (1) L is infinite, and (2) $\varepsilon \in L$.*

Proof. $\Rightarrow$. If $\varepsilon \notin L$ then $\varepsilon \notin L^k$ for any $k > 0$ and L has no finite power property because $\varepsilon \in L^*$. If L is finite, then L^k is finite for any $k > 0$, but L^* is infinite so L has no finite power property.

$\Leftarrow$. Consider the representation (4) for the language L. Without loss of generality let us assume that $F = \{f\}$ and $Z = \{z\}$. Let w be a word in L^*. The length of w may be represented as

$$|w| = jf + pz + it, \tag{5}$$

where $j, p,$ and i are natural numbers and $i \neq 0$ only if $p \neq 0$. Note that $w \in L^{j+p}$. We prove now that there exist bounded numbers $\widehat{j}$ and $\widehat{p}$ such that for any $j, p,$ and i

$$jf + pz + it = \widehat{j}f + \widehat{p}z + \widehat{i}t, \tag{6}$$

thus $w \in L^{\widehat{j}+\widehat{p}}$.

Let $\mathrm{lcm}(n, m)$ be the least common multiple for n and m, i.e. $\mathrm{lcm}(n, m) = \min_k\{k \mod n = 0 \text{ and } k \mod m = 0\}$.

$$jf = \left(\frac{\mathrm{lcm}(f,z)}{f}\widetilde{j} + \widehat{j}\right) f = \mathrm{lcm}(f,z)\widetilde{j} + \widehat{j}f = p'z + \widehat{j}f.$$

Choose $\widetilde{j}$ and $\widehat{j}$ from the condition:

$$0 \leqslant \widehat{j} < \frac{\mathrm{lcm}(f,z)}{f}. \tag{7}$$

When (5) may be rewritten as:

$$|w| = \widetilde{j}f + (p+p')z + it. \tag{8}$$

Now,

$$(p+p')z = \left(\frac{\mathrm{lcm}(z,t)}{z}\widetilde{p} + \widehat{p}\right) z = \widehat{p}z + \mathrm{lcm}(z,t)\widetilde{p} = \widehat{p}z + ti'.$$

Choosing $\widetilde{p}$ and $\widehat{p}$ from the condition:

$$0 \leqslant \widehat{p} < \frac{\mathrm{lcm}(z,t)}{z} \tag{9}$$

we get:

$$|w| = \widehat{j}f + \widehat{p}z + (i+i')t, \tag{10}$$

where $\widehat{j}, \widehat{p}$ are bounded by 7 and 9, and i' may be arbitrary large. Note that if both p and p' are equals to 0, then $i' = 0$. Thus, $w \in L^k$, where $k = \widehat{j} + \widehat{p} = \frac{\mathrm{lcm}(f,z)}{f} + \frac{\mathrm{lcm}(z,t)}{z}$.

If $|F| > 1$ then

$$FPP(L) \leqslant \max_{f \in F}\left[\frac{\mathrm{lcm}(f,z)}{f} + \frac{\mathrm{lcm}(z,t)}{z}\right]. \tag{11}$$

□

Lemma 2. *For any regular languages $L \subseteq \Sigma^*$ and $M \subseteq \Sigma^*$ in one-letter alphabet Σ the equation*

$$L^*M^* = L^{k_1}M^{k_2}, \tag{12}$$

holds for some natural k_1 and k_2 iff:

1. *$\varepsilon \in M$, $\varepsilon \in L$;*
2. *one of the languages L or M is infinite.*

Proof. The proof of if part is the same as for the lemma 1.

If both M and L are infinite the statement immediately follows from the lemma 1. Assume that M is finite. Note, that if $\varepsilon \in M$, $\varepsilon \in L$; then

$$L^*M^* = (L+M)^*.$$

The infinite language $L + M$ has finite power property, $k = FPP(L+M)$. It is evident that $L^*M^* = L^kM^k$. □

Corollary 1. *For any system of regular languages* $\{L_l\}_{l=1}^n$ *in one-letter alphabet* Σ *there exists* $k_1 \neq 0, k_2 \neq 0, ..., k_n \neq 0$ *such that:*

$$L_1^* L_2^* ... L_n^* = L_1^{k_1} L_2^{k_2} ... L_n^{k_n}, \tag{13}$$

iff:

1. $\varepsilon \in L_l$, *for all* $l = 1, 2, ..., n$;
2. *at least one of the languages* L_l *is infinite.*

Now consider the case when one of the languages has no star operation: LM^*. If M is infinite then according to the lemma 1 LM^* equals to LM^k for some k. If both languages are finite then there is no finite power property. The following lemma deals with the case when L is infinite and M is finite.

Lemma 3. *Let* $L = G + \cup_{i=1}^r z_i(a^t)^*$ *be an infinite language, and* F *be a finite language in* $\Sigma = \{a\}$. *The equation*

$$LF^* = LF^k \tag{14}$$

holds for some $k \geqslant 0$ *iff it holds for*

$$k_0 = (\max\{t \cup F\})^2 + \sum_{f \in F} \frac{\operatorname{lcm}(f, t)}{f}. \tag{15}$$

Proof. Let $F = \{f_1, f_2, \ldots, f_n\}$. The general form for (the length of) a word $w \in LF^*$ is

$$\begin{array}{ll} \text{either 1)} & |w| = (g_s) + (j_1 f_1 + j_2 f_2 + \ldots + j_n f_n), \\ \text{or 2)} & |w| = (z_s) + it + (j_1 f_1 + j_2 f_2 + \ldots + j_n f_n). \end{array} \tag{16}$$

The sum of j_q equals to the power of F. We have to prove that there exist $\widehat{j_1}, \widehat{j_2}, \ldots, \widehat{j_n}$ such that $\sum_{q=1}^n \widehat{j_q} \leqslant k_0$ and for all $w \in LM^*$

$$\begin{array}{l} |w| = g_s + \widehat{j_1} f_1 + \widehat{j_2} f_2 + \ldots + \widehat{j_n} f_n), \text{ or} \\ |w| = z_s + it + \widehat{j_1} f_1 + \widehat{j_2} f_2 + \ldots + \widehat{j_n} f_n. \end{array} \tag{17}$$

For all words $w \in LF^*$ of the form 16.2 we can apply the technique from the lemma 1, so we have

$$j_l < \frac{1}{f_l} \operatorname{lcm}(f_l, t) \text{ for all } l = 1, \ldots, n. \tag{18}$$

In order to prove the statement in the case 16.1 consider the equation

$$x_0 t + (x_1 - j_1) f_1 + (x_2 - j_2) f_2 + \ldots + (x_n - j_n) f_n = g_s - z_m. \tag{19}$$

We prove now that $LF^* = LF^k$ for some k if and only if for all $g_s \in G$ there exists $z_m \in Z$ such that this equation has a solution in $\mathbb{Z}^+$.

Suppose that $LF^* = LF^k$ for some k but the equation (19) has no integer solution for some $g_s \in G$. Consider a word $w_1 w_2 \in LM^*$ such that:

(1) $w_1 = g_s$, and
(2) $|w_1 w_2| > \max G + k \max F$.
These conditions guarantee that $w_1 w_2$ has no representation of the form $w_1' w_2'$, where $w_1' \in G$ and $w_2' \in F^k$. But $w_1 w_2 \in LF^k$ by assumption, so

$$|w_1 w_2| = z_m + it + \widehat{j_1} f_1 + \widehat{j_2} f_2 + \ldots + \widehat{j_n} f_n \tag{20}$$

for some $\widehat{j_1}, \widehat{j_2}, \ldots, \widehat{j_n}$, and $(i, \widehat{j_1}, \widehat{j_2}, \ldots, \widehat{j_n})$ is a solution for (19). A contradiction.

Now suppose that for all g_s there exist z_m such that (19) has a solution. Rewrite (19) as

$$x_0 t + x_1 f_1 + x_2 f_2 + \ldots + x_n f_n = \gcd(t, f_1, \ldots, f_n) W, \tag{21}$$

$$\text{where } W = \frac{g_s - z_m}{\gcd(t, f_1, \ldots, f_n)} + \frac{j_1 f_1 + j_2 f_2 + \ldots + j_n f_n}{\gcd(t, f_1, \ldots, f_n)}.$$

Both parts of (21) can be reduced by $\gcd(t, f_1, \ldots, f_n)$, so (19) is equivalent to

$$x_0 \widehat{t} + x_1 \widehat{f_1} + x_2 \widehat{f_2} + \ldots + x_n \widehat{f_n} = W. \tag{22}$$

The equation (22) has a solution, say $(x_0^0, x_1^0, \ldots, x_n^0)$, because (19) has. Then for any $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}$ the set

$$(x_0^0 - \alpha_1 \widehat{f_1} - \alpha_2 \widehat{f_2} - \ldots - \alpha_n \widehat{f_n}, x_1^0 + \alpha_1 \widehat{t}, x_2^0 + \alpha_2 \widehat{t}, \ldots, x_n^0 + \alpha_n \widehat{t})$$

is also a solution of (22). Let us choose $\alpha_1, \ldots, \alpha_n$ from the conditions

$$0 \leqslant x_l^0 + \alpha_l \widehat{t} \leqslant t, \quad l = 1, \ldots, n.$$

Then $x_0^0 - \alpha_1 \widehat{f_1} - \alpha_2 \widehat{f_2} - \ldots - \alpha_n \widehat{f_n} > 0$ if $W > \widehat{t}(\widehat{f_1} + \widehat{f_2} + \ldots + \widehat{f_n})$. The later condition holds if

$$j_1 f_1 + j_2 f_2 + \ldots + j_n f_n > \left(\max(t, f_1, f_2, \ldots, f_n)\right)^2. \tag{23}$$

Summing up (23) and (18) we get (15). □

Now consider the problem 1.

Theorem 1. *Let Q and $\mathcal{E} = \{E_1, \ldots, E_k\}$ be regular languages in $\Sigma = \{a\}$. The problem of regular language factorisation for Q and $\mathcal{E}$ is decidable.*

Proof. Let Q' be the $\Sigma_{\mathcal{E}}$-maximal rewriting for Q (see 3). If $L_\Sigma(Q') \neq Q$, than Q can not be represented in the form 2. Suppose that $L_\Sigma(Q') = Q$.

Consider the decomposition (3) for Q'. Let R be a component of (3) such that $L_\Sigma(R) = Q$. If no such R exist than there is no solution for Q and $\mathcal{E}$. Since $(L_1^* L_2)^* = L_1^* L_2^*$ for any languages in Σ we have to consider only the case when R is a language of star height 1.

Due to commutative property of languages in one-letter alphabet and the equality $L_1^* L_2^* = (L_1 + L_2)^*$ the general form of R is LM^*, where M is the union of all star languages in R and L is the concatenation of all "non-star" languages in R. If M is infinite then according to the lemma 1 $LM^* = LM^k$ for some k. If M is finite than the solution exists only if $LM^* = LM^{k_0}$, where k_0 satisfy the condition of the lemma 3. □

6 Conclusion

In this paper the problem of representing a regular language as a concatenation of given ones was considered. A complete solution for a specific case of one-letter alphabet was developed. This result may be used to give a negative answer in some cases of the general problem. Consider a language morphism $f : \Sigma \to \{a\}$ and reduce the problem into a one-letter case, when only lengths of words are taken into account. If corresponding problem for one-letter alphabet has no solution then the original problem has no solution also because the regular languages have lengths of words that do not allow $\mathcal{H}_c$-decomposition. The problem in the general case remains open.

References

1. Abiteboul, S., and Vianu, V. Regular path queries with constraints. In *Proc. of the sixteenth ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS 97)* (1997), pp. 122–133.
2. Afonin, S., Shundeev, A., and Roganov, V. Semistructured data search using dynamic parallelisation technology. In *Proceedings of the 26th International Convention MIPRO-2003, Opatija, Croatia* (2003), pp. 152–157.
3. Anselmo, M. A non-ambiguous decomposition of regular languages and factorizing codes. *J. Discrete Applied Mathematics 126*, 2-3 (2003), 129–165.
4. Arbib, M. *Algebraic Theory of Machines, Languages, and Semigroups.* Academic Press, 1968.
5. Brzozowski, J. A., and Cohen, R. On decompositions of regular events. *Journal of the ACM 16*, 1 (Jan. 1969), 132–144.
6. Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Rewriting of regular expressions and regular path queries. *Journal of Computer and System Sciences 64* (May 2002), 443–465.
7. Calvanese, D., Giacomo, G. D., Lenzerini, M., and Vardi, M. Y. Answering regular path queries using views. In *ICDE* (2000), pp. 389–398.
8. Conway, J. *Regular Algebra and Finite Machines.* Chapman and Hall, 1971.
9. Halevy, A. Y. Theory of answering queries using views. *SIGMOD Record (ACM Special Interest Group on Management of Data) 29*, 4 (2000), 40–47.
10. Kari, L., and Thierrin, G. Maximal and minimal solutions to language equations. *Journal of Computer and System Sciences 53* (December 1996), 487–496.
11. Karvounarakis, G., Magganaraki, A., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M., and Tolle, K. Querying the semantic web with rql. *Comput. Networks 42*, 5 (2003), 617–640.
12. Kirsten, D. Desert automata i. a burnside problem and its solution. In *Proceedings of the 21st International Symposium on Theoretical Aspects of Computer Science STACS-2004* (2004).
13. Leiss, E. *Language Equations.* Springer-Verlag, 1999.
14. Mateescu, A., Salomaa, A., and Yu, S. On the decomposition of finite languages. Tech. Rep. TUCS-TR-222, 8, 1998.
15. Mendelzon, A. O., and Wood, P. T. Finding regular simple paths in graph databases. In *Proceedings of the 15th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Amsterdam* (1989).
16. Quass, D., Widom, J., Goldman, R., Haas, K., Luo, Q., McHugh, J., Nestorov, S., Rajaraman, A., Rivero, H., Abiteboul, S., Ullman, J. D., and Wiener, J. L. LORE: A Lightweight Object REpository for semistructured data. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (Montreal, Quebec, Canada, 4–6 June 1996), H. V. Jagadish and I. S. Mumick, Eds., p. 549.
17. Vasenin, V. A., and Afonin, S. A. To the problem of building an integrated system of university distributed information resources. In *Proceedings of the Finnish Data Processing Week Conference FDPW-2001* (2001), pp. 152–177.

Communication Complexity Classes for Distributed Generation of Languages

Liliana Cojocaru

Rovira i Virgili University of Tarragona
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
liliana.cojocaru@estudiants.urv.es

Abstract. In this paper we present new insights into the nature of languages generated by Cooperating Distributed Grammar Systems (CDGS) with regular, linear and context-free components. The generative power of these systems is investigated within the framework of the communication complexity theory. We obtain several communication complexity classes, depending on the types of the system components, modes of derivation, weak and strong fairness conditions. We deal with trades-off between time, space and communication complexity in order to characterize the generative process of languages.

1 Introduction

Usually, complexity theory is concerned with the complexity measurement of computations in terms of time and space. In 1979, Yao [20] introduced another method of measuring the computation based on the communication within the system. To evaluate a process, by ignoring the number of steps (time) and the size of the memory (space) used during the computation, he introduced a model in which two players (processors or abstract computers) each having access to a partial part of the input, try to collaboratively evaluate a function with the least amount of communication between them. The *communication complexity* measure was defined as the minimum number of bits of information exchanged between the two players at any input. Works in communication complexity were motivated by the design and analysis of VLSI (Very Large Scale Integrated) circuits and distributed computation. It turned out to be an interesting tool for proving lower bounds in the study of small complexity classes.

Approaches to classes of languages from Chomsky hierarchy within the framework of communication complexity can be found in [17], [16], [15] and [10]. In [15] it is shown that regular languages have small (constant) communication complexity, while context-free languages need linear communication complexity. In [10] and [16] it is proved that there exist non-recursively enumerable languages that are recognizable within 0 communication complexity, so that hard languages according to the Chomsky hierarchy can be simple according to communication complexity. Lower bounds in communication complexity for several particular languages of Chomsky hierarchy are described in [17], while in [10] and [16] several area-time trades-off results for Chomsky hierarchy are approached within the framework of VLSI communication complexity.

Investigations related to the communication complexity of distributed grammar systems can be found in several papers, e.g. [11], [12], [13], [14]. The authors of them

focus on the communication complexity of *Parallel Communicating Grammar Systems* ($PCGS$). They considered two kinds of communication complexity measures. The first one is the *communication structure of PCGS*, i.e. the shape of the communication graph, consisting of directed communication links between the grammars, while the second one is a *communication complexity measure*, i.e. the number of exchanged messages during the computational process. Several hierarchies of classes are obtained through the above complexity measures in [11] and [12], along with several lower bound results. In [14] is shown that $k+1$ communications are more powerful than k communications, so that an infinite hierarchy of constant communication complexity for PCGS without any restrictions on their communication graph, exists. The price of obtaining non-regular languages, over one letter alphabet, is payed with $\Omega(log\ n)$ communication complexity.

This paper is devoted to the communication complexity of Cooperating Distributed Grammar Systems, with regular, linear and context-free components, but concerns also other computational resources used by the system, such as time and space. We deal with trades-off between these measures in order to control the generative process tackled by CDGS.

For the case of CDGS we propose two types of communication structures. The *first structure* is determined by the *communication graph* of CDGS, which is a directed graph where the vertices are labeled by the CDGS components and the directed edges correspond to pairs $(G_a,\ G_b)$, $a \neq b$, of grammars that communicate with each other. The communication is done through those nonterminals that appear on the right side of a production of G_a and on the left side of a production of G_b, according to the protocol of cooperation used by the system. We refer to these nonterminals as *communicational nonterminals*. The *second structure* is a protocol tree determined by the interconnection between the system components, i.e. the way in which they bring consecutive contributions on the sentential form during the language generation process. For each language L generated by a CDGS we define a new kind of control language, called *communicational Szilard language* viewed as the set of all *communicational control words* of L. If γ_w^c is a *communicational control word* of a certain word $w \in L$, then the derivation tree of γ_w^c is the *communicational protocol tree* attached to w. A *communication complexity measure*, i.e., how many times the system components communicate with each other using a minimal number of communicational nonterminals, is defined and studied depending on modes of derivation, weak and strong fairness conditions.

2 Preliminaries

Grammar Systems have been introduced in [3] and [4], as a mathematical formalization of the blackboard model of problem solving. CDGS are sets of grammars that work sequencially on a common sentential form, according to a specified protocol of cooperation. At each moment only one grammar is active. Which component of the system is active at a given moment, and when a grammar stops to be active, is decided by the protocol of cooperation. This protocol consists in stop conditions such as modes of derivation (how many times a rewriting rule of the same component can

be applied), in weak fairness conditions (each component has to be activated almost the same number of times) or in strong fairness conditions (each component has to be activated almost the same number of times, by taking into account the number of internal productions that are applied for each grammar). For more results the reader is reffered to [5] and [9]. Formally a CDGS is defined as follows:

Definition 1 A **Cooperating Distributed Grammar System** of degree r, $r \geq 1$ is a construct of the form: $\Gamma = (N, T, S, P_1, \ldots, P_r)$, where the sets N and T are disjoint finite alphabets, the *nonterminal* and the *terminal* alphabet, respectively. $S \in N$ is the system axiom, and $P_1, P_2, \ldots, P_r$ are finite sets of rewriting rules over $N \cup T$.

A CDGS can be equivalently rewritten as $\Gamma = (N, T, S, G_1, \ldots, G_r)$ in which $G_i = (N, T, S, P_i)$, for all i, $1 \leq i \leq r$, are Chomsky grammars, called the *components* of Γ. For $X \in \{REG, LIN, CF\}$ we denote by CDGS$_r$X, $r \geq 1$, CD grammar systems with r components, that have regular, linear and context-free components, respectively. The language generated by these systems depends on the way in which the internal rules of each component bring their own contribution on the sentential form. This can be done with respect to several modes of derivation, recalled below:

Definition 2 Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$ be a CDGS, $x, y \in (N \cup T)^*$, and $i \in \{1, ..., r\}$. The **terminating derivation** (denoted by $\Rightarrow^t_{P_i}$), the **k-steps derivation** (denoted by $\Rightarrow^{=k}_{P_i}$), **at most k-steps derivation** (denoted by $\Rightarrow^{\leq k}_{P_i}$), **at least k-steps derivation** (denoted by $\Rightarrow^{\geq k}_{P_i}$), and the ***-mode of derivation** (denoted by $\Rightarrow^*_{P_i}$), represent modes of derivations that allow for each component P_i to consecutively activate each rule: as many times as possible, exactly k times, at most k times, at least k times, and arbitrarily many times, respectively.

Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$ be a CDGS and $M = \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$.

Definition 3 The language generated by Γ in *f-mode*, $f \in M$ is defined as:
$L_f(\Gamma) = \{w \in T^* | S = w_0 \Rightarrow^f_{P_{i_1}} ... \Rightarrow^f_{P_{i_m}} w_q = w, m \geq 1, 1 \leq i_j \leq r, 1 \leq j \leq m\}$.

For $X \in \{REG, LIN, CF\}$, $f \in M$ we denote by $CD_rX(f)$, $r \geq 1$, the family of languages generated by CDGS with r components, that have only regular, linear, and context-free rules activated in the f-mode of derivation.

Besides modes of derivation other restrictions that control the generative process are given by fairness conditions. Informally, these conditions require that all components of the system have approximately the same contribution on the common sentential form. They have been introduced in [7], in order to control and to increase the generative capacity of grammar systems. Formally they are defined as follows:

Definition 4 Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$ be a CDGS, and

$$\text{D: } S = w_0 \Rightarrow^{=n_1}_{P_{i_1}} w_1 \Rightarrow^{=n_2}_{P_{i_2}} w_2 ... \Rightarrow^{=n_q}_{P_{i_m}} w_q = w$$

be a derivation in f-mode, where P_{i_j} performs n_j steps, $1 \leq j \leq m$. For any $1 \leq p \leq r$, we set

$$\psi_D(p) = \sum_{i_j = p} 1 \quad \text{and} \quad \varphi_D(p) = \sum_{i_j = p} n_j$$

- *the weak maximal difference between the contribution of two components involved in the derivation* D is defined as:

$$dw(D) = max\{|\psi_D(i) - \psi_D(j)| \; |1 \leq i, j \leq r\},$$

- *the strong maximal difference between the contribution of two components* is:

$$ds(D) = max\{|\varphi_D(i) - \varphi_D(j)| \; |1 \leq i, j \leq r\}.$$

Let $u \in \{w, s\}$, $x \in (N \cup T)^*$, $f \in M$ and

$$du(x, f) = min\{du(D)| \text{ where D is a derivation of } x \text{ in } f\text{-mode}\},$$

for a fixed natural number $q \geq 0$,

- the **weakly q-fair** language generated by Γ in the f-mode is defined as:

$$L_f(\Gamma, w - q) = \{x | x \in L_f(\Gamma) \text{ and } dw(x, f) \leq q\}$$

- the **strongly q-fair** language generated by Γ in the f-mode as:

$$L_f(\Gamma, s - q) = \{x | x \in L_f(\Gamma) \text{ and } ds(x, f) \leq q\}.$$

For $X \in \{REG, LIN, CF\}$ and $f \in M$, $M = \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$ we denote by $CD_rX(f, w - q)$ and $CD_rX(f, s - q)$, $r \geq 1$, the family of weakly and strongly q-fair languages, respectively, generated by CDGS with r components, that have regular, linear, and context-free components in the f-mode of derivation.

3 Communication versus protocols of collaboration

In this section a mathematical formalization of the communicational process performed during the distributed generation of languages, for the case of CDGS, is presented. In order to investigate the communicational phenomenon that is going on during the generative process, we propose a new kind of Szilard language, called *communicational Szilard language*, and two communication structures. The first structure is given by the *communication graph* of the CDGS, while the second structure, called *communicational protocol tree*, depends on the *protocol of collaboration* between the system components, and it is strictly related to the structure of the communicational Szilard language. Another measure deals with the number of communicational steps spent during the computational process. We call it *communication complexity*. We use these measures in order to divide the families of languages generated by these systems into classes of communication. Let $\Gamma = (N, T, S, G_1, \ldots, G_r)$ be a CDGS and $M = \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$.

Definition 5 We say that two grammars G_a and G_b, $a \neq b$, communicate with each other, during the generative process of a word $w \in L_f(\Gamma)$, $f \in M$, if there exists at least one nonterminal that appears on the right side of a production of G_a and of the left side of a production of G_b, rewritten at least one time during the derivational process in the f-mode. We call these nonterminals *communicational nonterminals*. The rules through which the communication is performed, i.e. rules that have on their right side at least one communicational nonterminal, are called *communicational rules*.

Definition 6 Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$ be a CDGS. The **control word** of w, with respect to the system components applied in f-mode, $f \in M$, for a terminal derivation: $S = w_0 \Rightarrow^f_{P_{i_1}} w_1 \Rightarrow^f_{P_{i_2}} w_2 ... \Rightarrow^f_{P_{i_m}} w_q = w$ is defined as

$\gamma_w = P_{i_1}P_{i_2}...P_{i_m}$. The **Szilard language** associated to the derivation in the f-mode, in Γ is: $Sz(\Gamma, f) = \{\gamma_w | w \in L_f(\Gamma), f \in M\}$.

We denote by $SZ(f)$ the *family of Szilard languages* $Sz(\Gamma, f)$ for any grammar system Γ, in the f-mode of derivation, $f \in M$. For more properties of these languages the reader is referred to [8].

Definition 7 Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$ be a CDGS. The **communicational control word** of w, that is a control word built with respect to the communicational nonterminals used during a terminal derivation of the form $S = w_0 \Rightarrow^f_{P_{i_1}} w_1 \Rightarrow^f_{P_{i_2}} w_2... \Rightarrow^f_{P_{i_m}} w_q = w$, in f-mode, $f \in M$, is defined as $\gamma^c_w = P^{n_1}_{i_1}P^{n_2}_{i_2}...P^{n_q}_{i_m}$, where n_j, is the *number of communicational nonterminals* rewritten during the application of rules of the component P_{i_j}, $1 \leq j \leq m$, during a particular step of communication. The **communicational Szilard language** associated to a terminal derivation in the f-mode, in Γ is defined as:

$$Szc(\Gamma, f) = \{\gamma^c_w | w \in L_f(\Gamma), f \in M\}.$$

We denote by $SZC(f)$ the *family of communicational Szilard languages* $Szc(\Gamma, f)$ for any grammar system Γ, in f-mode of derivation. Note that in the case of grammar systems with regular and linear components the languages $Sz(\Gamma, f)$ and $Szc(\Gamma, f)$ are equal. Furthermore, the same property takes place in the case of CDGS with non-linear context-free rules for which each communicational rule has only one communicational nonterminal. They can be different only in the case of grammar systems that contain at least one non-linear communicational rule having on its left side at least two communicational nonterminals activated in $= k$, $\geq k$, where $k \geq 2$, or t mode of derivation.

Definition 8 The **communication graph** of a language $L_f(\Gamma)$ generated by a CD grammar system Γ in the f-mode of derivation, $f \in M$, is a directed graph in which the vertices are labeled by the CDGS components that communicate with each other. Each directed edge, from a node labeled by G_a to another node labeled by G_b, $a \neq b$, corresponds to a communication step from the component G_a to the component G_b, done during the derivational process, i.e., there exists at least one nonterminal that appears on the right side of a production from G_a and of the left side of a production from G_b, rewritten at least one time during the generative process in the f-mode.

Definition 9 The **communicational protocol tree** attached to a word $w \in L_f(\Gamma)$, $f \in M$ is the derivation tree attached to the communicational control word of w, i.e. γ^c_w, in the f-mode.

Note that the number of sons of a given node in the communicational protocol tree depends on the type of the rule through which the communication is performed. In the case of regular or linear rules, a grammar G_a communicates with another grammar G_b through only one nonterminal, so that the corresponding protocol tree will be a simple tree (each node has only one son). In the case of non-linear

context-free rules the number of sons equals the number of communicational nonterminals from the right side of the communicational rule. Consequently, the shape of the communicational protocol tree depends not only on modes of derivation, or on the type of the rule. It depends also on the number of communicational nonterminals that exist on the right side of a communicational rule. Therefore, there can be grammar systems with non-linear context-free rules for which each communicational (context-free) rule has only one communicational nonterminal. In this case the protocol tree will be a simple tree, too. The communicational protocol tree might not be a simple tree, only in the case of CDGS for which there exists at least one non-linear context-free rule that has, on its right side, at least two communicational nonterminals.

Next a *communication complexity* measure that represents the number of communications between different components, during the generative process, by using a minimal number of communicational nonterminals in a specified mode of derivation, is defined. Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$ be a CDGS, and D a derivation in Γ, such that $D : S \Rightarrow^f_{P_{i_1}} w_1 \Rightarrow^f_{P_{i_2}} w_2 ... \Rightarrow^f_{P_{i_m}} w_q = w$.

Definition 10 We denote by $Com(D) = \sum_{p=1}^{r} \psi_D(p)$, where $\psi_D(p) = \sum_{i_j=p} 1$, the number of communication steps used during the derivation D.
The **communication complexity of a word** w, $w \in L_f(\Gamma)$ is defined as:

$$Com(w, \Gamma) = min\{Com(D) | D : S \Rightarrow^* w\}.$$

The **communication complexity of** Γ over all words of length n is:

$$Com_\Gamma(n) = sup\{Com(w, \Gamma) | w \in L_f(\Gamma), |w| = n\}.$$

The **class of languages that can be generated within communication g** by a CDGS, is defined as: $COM(g) = \bigcup_\Gamma \{L_f(\Gamma) | Com_\Gamma = O(g)\}$.

Let Γ be a CDGS, and D be a (minimal) terminal derivation of w, where $w \in L_f(\Gamma)$, $f \in M$, $M = \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$. We denote by $|\gamma_w(D)|$ and $|\gamma_w^c(D)|$ the length of the derivation of the control word $|\gamma_w|$, and of the communicational control word $|\gamma_w^c|$, associated to w, respectively. In terms of trees, the length of a derivation represents the number of internal nodes of the derivation tree. Informally, we use the Szilard language to control the number of communication steps performed during the generative process, while the communicational Szilard language is used to control the growth of the length of the generated word between two communication steps done during the derivational process. Formally, the connection between these two languages is presented in the next theorem, proved in [1].

Theorem 1 For each grammar system Γ that has only useful components[1] and $w \in L_f(\Gamma)$, we have:
1. $|\gamma_w(D)| = Com(w, \Gamma)$,
2. there exist two positive constants a and b such that $a|\gamma_w^c(D)| \leq |w| \leq b|\gamma_w^c(D)|$.

[1] Each component brings contributions on the sentential form, directly through terminal symbols (in the case of regular or linear rules), or indirectly through non-terminal symbols (in the case of context-free components).

4 Trades-off between time, space, and communication complexity

4.1 CDGS with regular and linear components

Theorem 2 For each grammar system $CDGS_rX$, with $X \in \{REG, LIN\}$ and $r \geq 2$, there exists a $CDGS$ with only one component that will generate the same language, independently of modes of derivation.

Corollary 1 $CD_*X = X$, for $X \in \{REG, LIN\}$, independently of modes of derivation.

Corollary 2 The communication complexity of $CD_*X(f)$, $X \in \{REG, LIN\}$, $f \in \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$ is 0.

Corollary 3 $LIN \subseteq COM(0)$.

It is well known that the communication complexity divides languages into small complexity classes. The above results show that the communicational process of CDGS is a lazy one. So that the process of communication in these systems is not so powerful as it has been proved to be for the case of PCGS, where several hierarchies of very (small) complexity classes have been found. Due to the fact that fairness conditions increase the generative power of a grammar system, the above theorem and corollaries do not hold for the case of q-fair languages. A CDGS with arbitrary number of components cannot be "compressed" into a single grammar that generates the same q-fair language, by preserving the mode of derivation, too. Even for these types of languages in the case of a constant communication, the class of weakly q-fair languages generated by CDGS with regular or linear components coincides with the languages generated by the same grammar without any weak fairness condition, so that due to Corollary1 we have:

Corollary 4 $CD_{r,c}X(f, w-q) \subseteq COM(0)$ and $CD_{r,c}X(f, s-q) \subseteq COM(c)$ where $X \in \{REG, LIN\}$ and c is the constant number of communicational steps performed during the derivation.

Corollary 5 The communication complexity of weakly/strongly q-fair languages, $L_f(\Gamma, w-q)/L_f(\Gamma, s-q)$, $f \in \{t\} \cup \{\leq k, = k, \geq k | k \geq 1\}$, generated by $CDGS_*X$, $X \in \{REG, LIN\}$, for which the communication graph is a tree or a dag is 0/constant.

Nevertheless the above results do not hold for the case of strongly q-fair languages generated by CDGS with non-constant communication. Next we show that if the communication is not constant then the communication complexity of q-fair languages cannot be more than linear.

Theorem 3 $CD_rX(f, w-q) \cup CD_rX(f, s-q) \in COM(n)$.

Proof. In the case of grammar systems with regular and linear rules the Szilard and the communicational Szilard languages are equal. With respect to Theorem 1, for any word w that belongs to the weakly or strongly q-fair language, we have: $|\gamma_w(D)| = Com(w,\Gamma) = |\gamma_w^c(D)|$, where D is a minimal derivation of w. Hence, $sup\{|\gamma_w(D)||\, w \in L_f(\Gamma, u-q), |w| = n\} = sup\{Com(w,\Gamma)|\, w \in L_f(\Gamma, u-q), |w| = n\} = Com_\Gamma(n) = sup\{|\gamma_w^c(D)||w \in L_f(\Gamma, u-q), |w| = n\} = O(n)$, where $u \in \{w, s\}$. □

In [1] and [2] we proved that fairness conditions can be checked in linear time and space by a multitape Turing machine, and in linear space and quadratic time by one tape Turing machine, see Theorem 4 and 5. For the definitions of one tape or multitape Turing machine, the reader is referred to [19].

Theorem 4 Let Γ be a CDGS$_r$X, for $X \in \{REG, LIN\}$. The weakly q-fair language generated by Γ, i.e., $L_f(\Gamma, w-q)$, can be accepted by a nondeterministic Turing machine with $r+1$ tapes in linear time and space. Moreover the next relation hold:

$$Space_T(n) \in O(Com_\Gamma), \quad Time_T \in O(Com_\Gamma).$$

Theorem 5 Let Γ be a CDGS$_r$X, for $X \in \{REG, LIN\}$. The weakly *q-fair* language generated by Γ, i.e. $L_f(\Gamma, w-q)$, can be accepted by a nondeterministic Turing machine with one tape in linear space and quadratic time, i.e.

$$Space_T \in O(n), \quad Time_T \in O(n^2).$$

4.2 CDGS with context-free components

Results related to the generative power of CDGS with arbitrarily components and context-free rules can be found in [5]. In the next theorem we succinctly present them[2].

Theorem 6 1. $CD_*CF(f) = CF$, for $f \in \{*, = 1, \geq 1\} \cup \{\leq k | k \geq 1\}$,
2. $CD_3CF(t) = CD_*CF(t) = ETOL$,
3. $CD_1CF(f_1) \subset CD_2CF(f_1) \subseteq CD_rCF(f_1) \subseteq MAT$, where $r \geq 3$ and $f_1 \in \{= k, \geq k | k \geq 2\}$.

Corollary 6 $CF \subseteq COM(0)$; $CD_{r,c}CF(f, w-q) \in COM(0)$, where $f \in \{*, = 1, \geq 1\} \cup \{\leq k | k \geq 1\}$, and c is the constant number of communication steps spent during the computation.

Corollary 7 The communication complexity of $L_f(\Gamma, w-q)$, $f \in \{*, = 1, \geq 1\} \cup \{\leq k | k \geq 1\}$ generated by $CDGS_*CF$, for which the communication graph is a tree or a dag is 0.

Corollary 8 $CD_*LIN(f) \cup CD_*CF(f_1) \cup CD_{*,c}LIN/CF(f/f_1, w-q) \subset COM(0)$, $CD_{*,c}CF(f_2, w-q) \cup CD_{*,c}X(f, s-q) \subset COM(c)$, $X \in \{REG, LIN, CF\}$, for $f \in \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$, $f_1 \in \{*, = 1, \geq 1\} \cup \{\leq k | k \geq 1\}$ and $f_2 \in \{t\} \cup \{= k, \geq k | k \geq 1\}$.

[2] For the definition of ETOL and MAT languages the reader is referred to [18].

For the case of non-constant communication we have.

Theorem 7 For each grammar system Γ with context-free components, and $w \in L_f(\Gamma)$, there exists a bijection $h : N \to N$ such that $|\gamma_w^c(D)| = h(|\gamma_w(D)|)$, where D is the the (minimal) terminal derivation of w. $\gamma_w(D)$ and $\gamma_w^c(D)$ are the length of the derivation of the control word γ_w, and of the communicational control word γ_w^c, associated to w, respectively.

Proof. Let $\gamma_w^c = P_{i_1}^{n_1} P_{i_2}^{n_2} ... P_{i_k}^{n_k}$ be the communicational control word attached to w. The derivation tree of this word is the communicational protocol tree attached to w. The number of sons at each level in this tree depends recursively on the number of sons of the previous levels, because communicational rules from different components can be applied recursively, by using each time the same type of rules, that increases (linearly or exponentially) the number of communicational nonterminals used during the derivation. Consequently, at the end of the generative process the sum $n_1 + n_2 + ... + n_k$, will be a linear, polynomial or exponential function that depends on the length of the generated string. □

Next we describe several situations in which the function h is linear, polynomial or exponential.

In the case of CDGS for which all communicational rules contain only one communicational nonterminal, or in the case of CDGS that have at least one communicational rule with at least two communicational nonterminals, activated in $= 1, \geq 1, *, \leq k$ modes of derivation, the function h is the identity function (because the Szilard language and the communicational Szilard language are equal).

In the case of CDGS that have at least one communicational rule with at least two communicational nonterminals, each of these rules being recursively activated by only one communicational nonterminal, in the t mode of derivation or $= k$, $\geq k$ ($k \geq 2$) mode of derivation, the function h is a polynomial function.

In the case of CDGS for which all communicational rules contain at least two communicational nonterminals, each of the communicational nonterminals being recursively activated by the same rule during the generative process the function h will be an exponential function. This is the case of t modes of derivation, for instance.

To illustrate the above remarks, next we briefly present several examples that deal with these situations.

Example 1 Let $\Gamma_1 = (\{S, S', A, A', B, B'\}, \{a, b, c\}, S, P_1, P_2, P_3)$ be a CDGS with the components: $P_1 = \{S \to S', S' \to AB, B' \to B\}$,
$P_2 = \{A \to aS'b, B \to cB'\}$,
$P_3 = \{A \to ab, B \to c\}$.

The language generated by Γ_1, in the f mode of derivation, where $f \in \{= 1, \geq 1, *\}$ is: $L_f(\Gamma_1) = \{a^n b c^{m_1} b c^{m_2} b c^{m_n} | n \geq 1, m_i \geq 1, 1 \leq i \leq n\}$. The Szilard and the communicational Szilard languages are $Sz(\Gamma_1) = Sz(\Gamma_1) =$
$\{P_1 (P_2 P_1)^n P_3 (P_2 P_1)^{m_1 - 1} P_3 (P_2 P_1)^{m_2 - 1} P_3 ... (P_2 P_1)^{m_n - 1} P_3 | n \geq 1, m_i \geq 1, 1 \leq i \leq n\}$. In this case $h(x) = x$, i.e. the identity function.

The language generated by Γ_1, in the f mode of derivation, where $f \in \{t, = 2, \geq 2\}$ is: $L_f(\Gamma_1) = \{a^n bcbc^2...bc^{n-1} | n \geq 1\}$. The Szilard language is $Sz(\Gamma_1) = \{(P_1P_2)^{n-1}P_1P_3 | n \geq 1\}$, while the communicational Szilard language is $Szc(\Gamma_1) = \{P_1P_2^2P_1^2P_2^3P_1^3...P_2^{n-1}P_1^{n-1}P_3^n | n \geq 1\}$. For a certain word $w \in L_f(\Gamma_1)$ the corresponding control word γ_w from $Sz(\Gamma_1)$ has the length of the derivation $|\gamma_w(D)| = n+1$, while the length of the derivation of γ_w^c from $Szc(\Gamma_1)$ is $|\gamma_w^c(D)| = n^2 - 1$. So that $h(x) = x^2 - 1$.

Example 2 Let $\Gamma_2 = (\{S, S', A, A', B, B'\}, \{a\}, S, P_1, P_2, P_3)$ be a CDGS with the components: $P_1 = \{S \to S', S' \to AB\}$,
$P_2 = \{A \to aS'a, B \to aS'\}$,
$P_3 = \{A \to aa, B \to a\}$.

The language generated by Γ_2, in the f mode of derivation, where $f \in \{t, = 2, \geq 2\}$ is: $L_f(\Gamma_2) = \{a^{3(2^n-1)} | n \geq 1\}$. The Szilard language is $Sz(\Gamma_2) = \{P_1(P_2P_1)^{n-1}P_3 \mid n \geq 1\}$, while the communicational Szilard language is
$Szc(\Gamma_2) = \{P_1(P_2^{2^1}P_1^{2^1})(P_2^{2^2}P_1^{2^2})...(P_2^{2^{n-1}}P_1^{2^{n-1}})P_3^{2^n} | n \geq 1\}$.
For a certain word $w \in L_f(\Gamma_2)$ the corresponding control word γ_w from $Sz(\Gamma_1)$ has the length of the derivation $|\gamma_w(D)| = 2n$, while the length of the derivation of the communicational control word $\gamma_w^c \in Szc(\Gamma_1)$ is $|\gamma_w^c(D)| = 2(2^0+2^1+2^2+...+2^{n-1})+2^n - 1 = 3(2^n - 1)$, so that $h(x) = 3(2^{x/2} - 1)$.

In the sequel we will refer to the function h as the characterization function of the communicational Szilard language $Szc(\Gamma, f)$. The next theorem shows how this function can be used to obtain communication complexity classes for the case of non-constant communication.

Theorem 8 The class of languages generated by CDGS$_r$CF in f-mode of derivation, $f \in \{t\} \cup \{= k, \geq k | k \geq 2\}$, for which the characterization function of the $Szc(\Gamma, f)$ language is linear, polynomial of rank p or exponential with the base p, has the communication complexity in $O(n)$, $O(\sqrt[p]{n})$, or $O(log_p n)$, respectively.

Proof. With respect to Theorem 1, for any word $w \in L_f(\Gamma)$, we have
$$|\gamma_w(D)| = Com(w, \Gamma) \text{ and } |\gamma_w^c(D)| = O(|w|).$$
With respect to Theorem 7, there exists a generative bijection $h : N \to N$, such that $|\gamma_w^c(D)| = h(|\gamma_w(D)|)$. Consequently, for a word $w \in L_f(\Gamma)$ of length n we have $O(n) = h(Com_\Gamma(n))$, so that $Com_\Gamma(n) = h^{-1}(O(n))$. Therefore, if h is a linear function then $Com_\Gamma(n) \in O(n)$, in the case that h is a polynomial function of rank p, then $Com_\Gamma(n) \in O(\sqrt[p]{n})$, while in the case that h is an exponential function of base p, then $Com_\Gamma(n) \in O(log_p n)$. □

As a direct consequence of Theorem 6 and Theorem 8, we have :

Corollary 9 ETOL and MAT languages can be generated within $O(\sqrt[p]{n})$ and $O(log_p n)$ communication complexity.

Furthermore, next results have been proved in [1] and [2].

Theorem 9 The class of languages generated by a CDGS$_r$CF in f-mode of derivation, $f \in \{t\} \cup \{=k, \geq k | k \geq 2\}$, are generated by a nondeterministic Turing machine, with $r+1$ tapes, within $Space_T \in O(Com_\Gamma)$ and $Time_T \in \Theta(n)$.

Corollary 10 The class of languages generated by a CDGS$_r$CF in f-mode of derivation, $f \in \{t\} \cup \{=k, \geq k | k \geq 2\}$ for which the characterization function of the $Szc(\Gamma, f)$ is linear, polynomial of rank p, or exponential with the base k, are recognizable by a nondeterministic Turing machine, with $r+1$ tapes, within $Space_T$ $O(n)$, $O(\sqrt[p]{n})$, or $O(log_p n)$, respectively, and in $Time_T \in \Theta(n)$.

Theorem 10 The class of q-fair languages generated by a CDGS$_r$CF in f-mode of derivation, $f \in \{t\} \cup \{=k, \geq k | k \geq 1\}$ for which the characterization function of $Szc(\Gamma, f)$ is linear, polynomial of rank p, or exponential with the base p, are recognizable by a $(k+1)$-tape nondeterministic Turing machine in $Space_T \in O(n)$, $O(\sqrt[p]{n})$, or $O(log_p n)$, respectively, and $Time_T \in \Theta(n)$.

5 Conclusion

In this paper several results obtained so far in the domain of the communication complexity of distributed generation of languages have been presented. We conclude that in the case of CDGS with regular and linear components the communication between the system components can be neglected, it has no efficiency in building communication complexity classes. This is not the case for CDGS with regular or linear components, with fairness conditions. In the case of non-constant communication, weakly and strongly q-fair languages, requires linear communication complexity. The communication fails too, in the case of languages generated by CDGS with context-free components, in the f mode of derivation, $f \in \{*, =1, \geq 1\} \cup \{\leq k | k \geq 1\}$, or in the case of weakly q-fair languages generated by CDGS with constant communication. The communication is preserved in the case of CDGS with context-free rules, non-constant communication, weakly and strongly q-fair condition and in $f \in \{t\} \cup \{=k, \geq k | k \geq 1\}$ mode of derivation. Moreover, in this last situation several communication complexity classes have been reached depending on the characterization function of the communicational Szilard language associated to a CDGS.

References

1. L. Cojocaru. On the Time, Space, and Communication Complexity of Cooperating Distributed Grammar Systems. In *Proceedings of Grammar Systems Week 2004*, E. Csuhaj-Varjú and Gy. Vaszil Ed., pp. 101-113, Budapest, Hungary, July 5-9, 2004.
2. L. Cojocaru. On the Time, Space, and Communication Complexity of q-fair Languages. In *Preproceedings of the 6^{th} Descriptional Complexity of Formal Systems Workshop*, DCFS 2004, pp. 154-163, London, Ontario, Canada, July 26-28, 2004.
3. E. Csuhaj-Varjú, J. Dassow. On cooperating/distributed grammar systems. *Journal of Information Processing and Cybernetics EIK* 26, 49-63, 1990.
4. E. Csuhaj-Varjú, J. Kelemen. Cooperating grammar systems: a syntactical framework for the blackboard model of problem solving. In *Proceedings AICSR'89*, I. Plander, Ed. North Holland, Amsterdam, 121-127, 1989.

5. E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation,* Gordon and Breach, 1994.
6. J. Dassow, Gh. Păun, G. Rozenberg, Grammar Systems. *Handbook of Formal Languages*, Volume II, Chapter 4, edited by G. Rozenberg and A. Salomaa, Springer, Berlin, 155-213, 1997.
7. J. Dassow, V. Mitrana. Fairness in Grammar Systems. Acta Cybernetica 12, 331-345, 1996.
8. J. Dassow, V. Mitrana, Gh. Păun. Szilard Languages Associated to Cooperating Distributed Grammar Systems. *Studii si Cercetari Matematice*, 45, 403-413, 1993.
9. M. Gheorghe, Gh. Păun. Further remarks on cooperating distributed grammar system. *Bull. Math. Soc. Sci. Math. Roumanie*. 34(82), 232- 245, 1990.
10. J. Hromkovič. Relation between Chomsky hierarchy and communication complexity hierarchy. *Acta Mathematica* University Comenian. 48-49, 311-317, 1988.
11. J. Hromkovič, J. Kari, L. Kari, D. Pardubská. Two lower bounds on distributive generation of languages. *Fundamenta Informatica*, 25, 271-284, 1996.
12. J. Hromkovič, J. Kari, L. Kari, Some hierarchies for the communication complexity measures of cooperating grammar systems. *Theoretical Computer Science*, 127, 123-147, 1994.
13. D. Pardubská, On the power of communication structure for distributed generation of languages. *In Developments in Language Theory at the Crossroads of Mathematics, Computer Science and Biology*, Editors G. Rozenberg and A. Salomaa, Turku, Finland, 12-15, July, 1993.
14. D. Pardubská, The Communication complexity hierarchy of parallel parallel communicating systems. IMYC'92, 1992.
15. J. Hromkovik. Communication Complexity and Parallel Computing. *Springer-Verlag Berlin Heidelberg*, 1997.
16. G. Jiràskovà. Chomsky Hierarchy and Communication Complexity. *Journal of Information Processing and Cybernetics: EIK* 25, 4, 157-164.
17. E. Kushilevitz, N. Nisan. Communication Complexity. *Cambridge University Press*, 1997.
18. Handbook of Formal Languages, Vol. 1, Words, Language, Grammar. G. Rozenberg, A. Salomaa Ed., *Springer-Verlag Berlin Heidelberg*, 1997.
19. I. Sudkamp. Languages and Machines. An Introduction to the Theory of Computer Science. Addison-Wesley Ed., 1997.
20. A. C. Yao. Some complexity questions related to distributed computing. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, 209-213, 1979.

A Full Range of Continuum-Many Non-Context-free Languages with Strong Iteration

Sándor Horváth and Manfred Kudlek

[1] Department of Computer Science, Eötvös Loránd University,
Budapest, Hungary
email : horvath@cs.elte.hu
[2] Fachbereich Informatik, Universität Hamburg, Germany
email : kudlek@informatik.uni-hamburg.de

Abstract. We show that the cardinality of languages fulfilling all strong iteration lemmata for context-free languages, and having either an exponential, polynomial, or super-polynomial but sub-exponential density, is the cardinality of the continuum. In the last case we could prove this only for non-erasing iteration so far.

1 Introduction

In [1, 3, 4] it has been shown that there are continuously many languages fulfilling various iteration lemmata for context-free languages. However, languages with exponential density were given only. Exponential density is maximal. It is interesting to investigate also the cardinality of languages fulfilling iteration lemmata for context-free languages for polynomial and super-polynomial (but sub-exponential) densities. The question can be put also for polynomials of arbitrary but fixed degree k (there exist recursively enumerably many), as well as for arbitrary but fixed super-polynomial functions (there exist continuously many).

2 Definitions

Let $\mathbf{N} = \{0, 1, 2, \cdots\}$ denote the set of non-negative natural numbers. Let X be any alphabet, and $L \subseteq X^*$ a language over X.

The density $\nu_L : X^* \longrightarrow \mathbf{N}$ is defined by $\nu_L(s) = |L \cap X^s|$ for $s \in \mathbf{N}$.

In the sequel we shall consider only the alphabet $X = \{a, b\}$ since other alphabets can easily be encoded in binary.

Consider an arbitrary set of numbers $H \subseteq \mathbf{N}$.

Strong iteration lemmata for context-free languages are such with *distinguished* and *excluded* positions [2], more exactly as follows.

Let $\delta(z)$ denote the number of *distinguished* positions in a word z, and $\epsilon(z)$ the number of *excluded* positions in z. Note that an excluded position might also be a distinguished one. Then the following strong iteration lemmata hold.

Proposition 1 : (Generalized Bader Moura Lemma)

Let L be a context-free language. Then there exists an integer $n = n(L) \geq 2$, depending only on the language L, such that for any $z \in L$ with $\delta(z) > n^{\epsilon(z)+1}$ there exist u, v, w, x, y such that $z = uvwxy$ with

(1) $\epsilon(vx) = 0$ and
    either $\delta(u) > 0$, $\delta(v) > 0$, $\delta(w) > 0$
    or $\delta(w) > 0$, $\delta(x) > 0$, $\delta(y) > 0$,
(2) $\delta(vwx) \leq n^{\epsilon(w)+1}$
(3) $\forall i \geq 0 : uv^iwx^iy \in L$.

Proposition 2 : Let L be a context-free language. Then there exists an integer $n = n(L) \geq 2$, depending only on the language L, such that for any $z \in L$ with $\delta(z) > n \cdot max(\epsilon(z), 1)$ there exist u, v, w, x, y such that $z = uvwxy$ with

(1) $\epsilon(vx) = 0$ and
    either $\delta(u) > 0$, $\delta(v) > 0$, $\delta(w) > 0$
    or $\delta(w) > 0$, $\delta(x) > 0$, $\delta(y) > 0$
(2) $\delta(vwx) \leq n \cdot (\epsilon(w) + 1)$
(3) $\forall i \geq 0 : uv^iwx^iy \in L$.

2.1 Exponential Density

Define $L'_H = \{(ab)^n | n \in H\}$, $L = \{a, b\}^* \cdot \{aa, bb\} \cdot \{a, b\}^*$, and $L_H = L'_H \cup L$.

Then the density of L_H is $\nu_{L_H}(s) = \Omega(2^s)$,

i.e. $\exists c > 0\ \exists s_0 \in \mathbf{N}\ \forall s \geq s_0 : \nu_{L_H}(s) \geq c \cdot 2^s$.

This follows immediately from $|\{w \in L \mid |w| = s\}| \geq \frac{1}{4}2^s = 2^{s-2}$ for $s \geq 2$.

To fulfill iteration lemmata, e.g. with distinguished positions, consider some $z = z_0c_1z_1c_2 \cdots c_mz_m \in L_H$ with $m \geq 1$, $|z|$ long enough, $z_j \neq \lambda$ for $1 \leq j \leq m$, and $c_j \in \{a, b\}$. There will always be sufficiently many distinguished positions among the c_j.

If $z \in L'_H$ then erasing and iteration of a distinguished position gives $z' \in L$ since either aa or bb is produced.

For $z \in L$ the same holds, for in that case a distinguished position not in the part $\{aa, bb\}$ can be chosen.

$H_1 \neq H_2$ implies $L_{H_1} \neq L_{H_2}$, and conversely.

Clearly, if $H_1 \neq H_2$ then $L'_{H_1} \neq L'_{H_2}$, and therefore $L_{H_1} \neq L_{H_2}$.

On the other hand, if $L_{H_1} \neq L_{H_2}$ then $L'_{H_1} \neq L'_{H_2}$, and therefore $H_1 \neq H_2$.

Therefore we conclude

Theorem 1 : The cardinality of languages with exponential density, fulfilling all strong iteration lemmata for context-free languages, is the cardinality of the continuum, $2^{\aleph_0}$.

2.2 Polynomial Density

Consider two linearly independent vectors, e.g. $\langle 1, 2\rangle$, $\langle 2, 1\rangle$, and an arbitrary set $H \subseteq \mathbf{N}$. Define the language

$L_H = \{ab^n ab^{2n}a \mid n \in \mathbf{N}\} \cup \{ab^{k+2m}ab^{m+2k}a \mid k \in H, m \in \mathbf{N} \setminus \{0\}\}$.

Note that the two parts of L_H are disjoint.

L_H represents the set $\mathbf{N} \cdot \{\langle 1,2\rangle\} \cup (H \cdot \{\langle 1,2\rangle\} + (\mathbf{N} \setminus \{0\}) \cdot \{\langle 2,1\rangle\})$.

Since the two vectors $\langle 1,2\rangle$, $\langle 2,1\rangle$ are linearly independent, any element $\langle p,q\rangle \in \{k_1\langle 1,2\rangle + k_2\langle 2,1\rangle \mid k_1, k_2 \in \mathbf{N}\}$ has a unique representation of the form $k_1\langle 1,2\rangle + k_2\langle 2,1\rangle$.

Therefore, if $H_1 \neq H_2$ then there is e.g. $\langle k_1, k_2\rangle \in H_2 \setminus H_1$ with unique representation. But then $ab^{k_1+2k_2}ab^{k_2+2k_1} \in L_{H_2} \setminus L_{H_1}$.

Thus $H_1 \neq H_2$ implies $L_{H_1} \neq L_{H_2}$, and conversely.

It is obvious that the density is at most linear :

$\nu_{L_H}(s) = O(s)$, i.e. $\nu_{L_H}(s) \leq c \cdot s$ for some $c > 0$. This follows from $|z| = 3 \cdot (n+1)$ or $|z| = 3 \cdot (k+m+1)$ for $z \in L_H$, and that there are only s possibilities $\langle k,m\rangle$ with $s = 3 \cdot (k+m+1)$, and only one possibility with $s = 3 \cdot (n+1)$.

Let L be an arbitrary language with density ν_L at least linear. Define the language $L'_H = \{a\} \cdot L \cup \{b\} \cdot L_H$. L'_H has a density $\nu_{L'_H}$ of the same order as ν_L.

Furthermore, $\{a\} \cdot L \cap \{b\} \cdot L_H = \emptyset$, and $|az|_{ab} \leq |z|_{ab} + 1$, $|az|_{ba} = |z|_{ba}$ for $z \in L$, and $|bz|_{ab} = |z|_{ab}$, $|bz|_{ba} = |z|_{ba} + 1$ for $z \in L_H$.

Therefore also $L'_{H_1} \neq L'_{H_2}$ iff $H_1 \neq H_2$.

In the sequel we shall define languages L_d with polynomial densities ν_{L_d}. of arbitrary degree d.

Let $L_d = \{z \in \{a,b\}^+ \mid |z|_{ab} + |z|_{ba} \leq d\}$. Then it is easily seen that the density is

$$\nu_{L_d}(s) = 2 \cdot \sum_{j=0}^{d} \binom{s-1}{j}$$

for $s > 0$. This is a polynomial in s of degree d.

Now, for arbitrary but fixed d, consider the language L_d instead of L, and $L_{Hd} = \{a\} \cdot L_d \cup \{b\} \cdot L_H$.

To fulfill iteration lemmata, e.g. with distinguished positions, consider some $z = z_0c_1z_1c_2 \cdots c_mz_m \in L_{Hd}$ with $m \geq 1$, $|z|$ long enough, $z_j \neq \lambda$ for $0 \leq j \leq m$, and $c_j \in \{a,b\}$. There will always be sufficiently many distinguished positions among the c_j.

If $z \in \{a\} \cdot L_d$ then erasing or iteration does not increase the number of subwords ab and ba, yielding $z' \in \{a\} \cdot L_d$.

If $z \in \{b\} \cdot L_H$ then there are sufficiently many distinguished positions b in z, either in the second block or in the third.

If $z \in \{bab^n ab^{2n}a \mid n \in \mathbf{N}\}$ then erase or iterate with (b, b^2) in the first and second block of b, yielding $z' \in \{bab^n ab^{2n}a \mid n \in \mathbf{N}\}$.

If $z \in \{bab^{k+2m}ab^{m+2k}a \mid k \in H, m \in \mathbf{N} \setminus \{0\}\}$ then iterating with (b^2, b) yields $z' \in \{bab^{k+2m}ab^{m+2k}a \mid k \in H, m \in \mathbf{N} \setminus \{0\}\}$. Erasing with (b^2, b) either yields $z' \in \{bab^{k+2m}ab^{m+2k}a \mid k \in H, m \in \mathbf{N} \setminus \{0\}\}$ or $z' \in \{bab^n ab^{2n}a \mid n \in \mathbf{N}\}$.

Thus we get

Theorem 2 : The cardinality of languages with polynomial density, fulfilling all strong iteration lemmata for context-free languages, is the cardinality of the continuum, $2^{\aleph_0}$.

2.3 Super-polynomial and Sub-exponential Density

In the following density functions being super-polynomial and sub-exponential will be considered. The function *log* stands for logarithm with base 2.

Such a function e.g. is $s^{log(s)}$, since

$$\lim_{s\to\infty}\frac{s^{log(s)}}{s^d}=\lim_{s\to\infty}s^{log(s)-d}=\lim_{s\to\infty}2^{log(s)(log(s)-d)}=\infty$$

$$\text{and}\qquad \lim_{s\to\infty}\frac{2^s}{s^{log(s)}}=\lim_{s\to\infty}2^{s-(log(s))^2}=\infty\ .$$

This follows from

$$\lim_{s\to\infty}(log(s)(log(s)-d)=\infty\ ,\ \lim_{s\to\infty}(s-(log(s))^2)=\infty\ .$$

Now consider the functions f, g given by $f(s)=\lfloor log(s)\rfloor$ and $g(s)=\lceil log(s)\rceil$. Since $log(s)$ is monotone in s, with $lim_{s\to\infty}log(s)=\infty$, both functions are monotone step functions, with the property that their values are increased exactly by 1 at 2^i ($i\geq 1$).

Clearly, $log(s)-1\leq f(s)\leq g(s)\leq log(s)+1$.

Let $K\subseteq\mathbf{N}$. Define new step functions f_K, g_K by

$f_K(s)=f-1$ for $2^i\leq s<2^{i+1}$ if $i\in K$, $f_K(s)=i$ for $2^i\leq s<2^{i+1}$ if $i\notin K$, and $g_K(s)=i+2$ for $2^i\leq s<2^{i+1}$ if $i\in K$, $g_K(s)=i+1$ for $2^i\leq s<2^{i+1}$ if $i\notin K$.

By this definition the number of functions f_K, g_K has the cardinality of the continuum, $2^{\aleph_0}$.

Obviously, all functions f_K, g_K are monotone, with $\lim_{s\to\infty}f_K(s)=\infty$, and fulfilling $log(s)-2\leq f_K(s)\leq g_K(s)\leq log(s)+2$.

Other super-polynomial and sub-exponential functions are, e.g.

$s^{\sqrt{s}}$, $s^{s^{\frac{1}{k}}}$, $log^k s$, $(log(s))^k$ with $log^1(s)=log(s), log^{k+1}(s)=log(log^k(s))$, and $k\geq 1$.

For these analogous step functions can be defined.

Now consider the language L_H defined in the poynomial case. It has linear density. Instead of L_d consider languages

$L_f=\{z\in\{a,b\}^*\mid |z|_{ab}+|z|_{ba}\leq f_K(|z|)\}$ for some $K\subseteq\mathbf{N}$, e.g. $K=\emptyset$.

Then the density is

$$\nu_{L_f}(s)=2\cdot\sum_{j=0}^{f(s)}\binom{s-1}{j}\ .$$

$$\text{Since}\qquad \left(\frac{s-k}{k}\right)^k\leq\frac{(s-k)^k}{k!}\leq\sum_{j=0}^{k}\frac{1}{j!}(s-j)^j\leq\sum_{j=0}^{k}\binom{s-1}{j}$$

and $$\sum_{j=0}^{k} \binom{s-1}{j} \leq \sum_{j=0}^{k} \frac{1}{k!}(s-1)^j \leq (s-1)^k \cdot \sum_{j=0}^{k} \frac{1}{j!} \leq 3 \cdot (s-1)^k$$

it follows that $2 \cdot \left(\frac{s-f(s)}{f(s)}\right)^{f(s)} \leq \nu_{L_f}(s) \leq 6 \cdot (s-1)^{f(s)}$.

Now

$$\lim_{s\to\infty} \frac{2 \cdot \left(\frac{s-f(s)}{f(s)}\right)^{f(s)}}{s^d} \geq 2 \cdot \lim_{s\to\infty} \frac{\left(\frac{s-f(s)}{f(s)}\right)^{f(s)}}{s^{f(s)}} = 2 \cdot \lim_{s\to\infty} \left(\frac{s-f(s)}{sf(s)}\right)^{f(s)}$$

$$= 2 \cdot \lim_{s\to\infty} \left(\frac{s}{f(s)} - \frac{f(s)}{s}\right)^{f(s)} = \infty$$

since

$$\lim_{s\to\infty} f(s) = \infty \ , \ \lim_{s\to\infty} \frac{s}{f(s)} = \infty \ , \ \lim_{s\to\infty} \frac{f(s)}{s} = 0 \ ,$$

and

$$\lim_{s\to\infty} \frac{2^s}{6 \cdot (s-1)^{f(s)}} \geq \lim_{s\to\infty} \frac{1}{6} \cdot \frac{2^s}{s^{f(s)}} = \lim_{s\to\infty} \frac{1}{6} \cdot \frac{2^s}{2^{log(s)f(s)}}$$

$$\geq \lim_{s\to\infty} \frac{1}{6} \cdot \frac{2^s}{2^{log(s)(log(s)+1)}} = \lim_{s\to\infty} \frac{1}{6} \cdot 2^{s-log(s)-(log(s))^2} = \infty \ ,$$

since

$$\lim_{s\to\infty} (s - log(s) - (log(s))^2) = \infty \ .$$

Therefore, ν_{L_f} is super-polynomial and sub-exponential.

Now define $L'_H = \{a\} \cdot L_f \cup \{b\} \cdot L_H$. L'_H has the same density as L_f. Then the same arguments as in the case for polynomial density can be applied, except that erasing may yield words $z \notin L'_H$.

Theorem 3 : The cardinality of languages with super-polynomial and sub-exponential density, fulfilling all strong iteration lemmata for context-free languages, but without erasing, is the cardinality of the continuum, $2^{\aleph_0}$.

Furthermore, for continuum many such functions there exist continuum many languages, fulfilling all strong iteration lemmata for context-free languages, but without erasing.

References

1. L. Boasson, S. Horváth : *On Languages Satisfying Ogden's Lemma.* RAIRO Informatique Théorique, **12**, pp. 201-202, 1978.
2. P. Dömösi, M. Kudlek : *Strong Iteration Lemmata for Regular, Linear, Context-free, and Linear Indexed Languages.* LNCS 1684, pp 226-233, 1999.
3. S. Horváth : *The Family of Languages Satisfying Bar-Hillel's Lemma.* RAIRO Informatique Théorique, **12**, pp. 193-199, 1978.
4. S. Horváth : *A Comparison of Iteration Conditions on Formal Languages.* In : Proc. *Colloq. Algebra, Combinatorics and Logic in Comp. Sci.* (Győr, Hungary, 1983), Colloq. Math. Soc. J. Bolyai. **42**, J. Bolyai Math. Soc., Budapest, and North-Holland, Amsterdam, pp. 193-199, 1986.

On Black Hole Languages

Manfred Kudlek and Roxana Melinte

[1] Fachbereich Informatik, Universität Hamburg
email : kudlek@informatik.uni-hamburg.de
[2] Fachbereich Informatik, Universität Hamburg
email : melinte@informatik.uni-hamburg.de

Abstract. A new method of generating languages from rewriting systems, inspired by home states of reachability sets of Petri nets, is introduced. The generative power of, and decidability problems for various rewriting systems are investigated.

1 Introduction

The concept of *home state* has been introduced for Petri nets in [3]. A home state or *home marking* is a marking reachable from any reachable marking. A *home space* is a set of reachable markings such that from any reachable marking a marking in the home space can be reached (see also [8, 2, 7]).

We generalize this idea to arbitrary binary rewriting systems with reachability interpreted as derivability, considering the set of all home states, and introducing in this way a new method to generate languages from rewriting systems. Thus we consider *Right Regular, Semi-Thue, Normal*, and *Lindenmayer* systems as word rewriting, as well as *Multiset* rewriting systems being equivalent to *Vector Addition systems*. The latter are just another way to describe the reachability sets of P/T nets where the basic operation is *addition* instead of *catenation*.

Such home markings are unavoidable objects, with respect to derivation. We call the unique set of all such objects a *black hole* since the term *unavoidable* is already used in another sense.

In this article we investigate the generative power of this new generative method for different rewriting systems, the relation of corresponding language families to other ones, as well as decidability problems related to them, like membership, emptiness, finiteness, inclusion, and equivalence.

2 Definitions

In the sequel we consider various rewriting systems and use a uniform notation to classify them. $\mathbf{REG}, \mathbf{CF}, \mathbf{CS}, \mathbf{RE}$ denote the families of *regular*, *context-free*, *context-sensitive*, and *recursively enumerable* languages, respectively. For all basic definitions of formal languages see [10]. Letters u, v stand for variables. taking values from Σ^*.

A *(Right) Regular System* is a rewriting system $G = (\Sigma, \{\omega\}, P)$ with productions of the form $\alpha u \rightarrow \beta u$, $(\alpha, \beta) \in \Sigma^+ \times \Sigma^*$. Such a system is denoted by IR where I means *interaction*. If $|\alpha| \leq |\beta|$ holds for all productions, then the system is called *monotone* or *propagating*, denoted by the letter P, giving a system PIR. If $|\alpha| = 1$

for all productions, then the system is called *context-independent* or *context-free* and is denoted by O, giving systems OR or POR.

Corresponding families of *sentential form languages* are denoted by $\mathbf{POR}, \mathbf{OR}$, $\mathbf{IR}, \mathbf{PIR}$. Note that $\mathbf{IR} \subset \mathbf{REG}$ holds.

A *Semi-Thue System* is a rewriting system $G = (\Sigma, \{\omega\}, P)$ with productions of the form $u\alpha v \rightarrow u\beta v$, $(\alpha, \beta) \in \Sigma^+ \times \Sigma^*$. Analogous to Right Regular systems notations POS, OS, PIS, IS are introduced, as well as for corresponding language families $\mathbf{POS}, \mathbf{OS}, \mathbf{PIS}, \mathbf{IS}$. Note that $\mathbf{OS} \subset \mathbf{CF}$ and $\mathbf{IS} \subset \mathbf{RE}$ hold.

A *(context-independent) Lindenmayer System* is a rewriting system $G = (\Sigma, \{\omega\}, P)$ with productions $(a, \alpha) \in \Sigma \times \Sigma^*$. A rewriting step $x = a_1 \cdots a_n \rightarrow \beta_1 \cdots \beta_n = y$ is defined by $(a_i, \beta_i) \in P$ for $1 \leq i \leq n$. Analogous to Right Regular systems notations POL, OL are introduced, as well as for corresponding language families $\mathbf{POL}, \mathbf{OL}$.

A *(context-independent) Table Lindenmayer System* is a rewriting system of the form $G = (\Sigma, \{\omega\}, \mathcal{T})$ where $\mathcal{T} = \{P_1, \cdots, P_k\}$ is the set of *tables*, such that each of $G_j = (\Sigma, \{\omega\}, P_j)$ is an OL system. A rewriting step $x = a_1 \cdots a_n \rightarrow \beta_1 \cdots \beta_n = y$ is defined by $(a_i, \beta_i) \in P_j$ for $1 \leq i \leq n$ and some $1 \leq j \leq k$ (i. e. one table is used only). In this case notations $PTOL, TOL$ are introduced, and also $\mathbf{PTOL}, \mathbf{TOL}$. For more information on L systems see [9].

Multiset rewriting systems are defined in the following way (see also [4–6]). A *multiset* μ is just a vector $\mu = (m_1, \cdots, m_s) \in I\!N^s$.

If $\mu = (m_1, \cdots, m_s)$, $\mu' = (m'_1, \cdots, m'_s)$ then $\mu + \mu' = (m_1 + m'_1, \cdots, m_s + m'_s)$, $\mu \sqsubseteq \mu'$ iff $m_j \leq m'_j$ for $1 \leq j \leq s$. If $\mu \sqsubseteq \mu'$ then $\mu' - \mu = (m'_1 - m_1, \cdots, m'_s - m_s)$. The *length* or *norm* of a multiset μ is defined by $|\mu| = \sum_{j=1}^{s} m_j$.

Vector addition plays the role of word catenation. Note that the set of all multisets is a commutative monoid since addition is commutative, with neutral element $\mathbf{0} = (0, \cdots, 0)$.

Multisets can be represented by words over an alphabet $\Sigma = \{a_1, \cdots, a_s\}$ in the form $a_1^{m_1} \cdots a_s^{m_s}$. In other words, we deal with the commutative monoid $\Sigma^{\oplus}$ (see [4]).

Analogous to word rewriting systems multiset rewriting systems can be defined, in particular multiset grammars analogous to Semi Thue systems, using productions (α, β) ($\alpha, \beta \in I\!N^s$) for rewriting : $\mu \rightarrow \mu'$ iff $\alpha \sqsubseteq \mu$ and $\mu' = (\mu - \alpha) + \beta$. In this way *context-free*, *monotone*, and *arbitrary* multiset grammars can be defined (see [4]), denoted by $mPOS, mOS, mPIS, mIS$, respectively. Corresponding families of multiset languages are denoted by **mPOS**, **mOS**, **mPIS**, and **mIS**. Note that $\mathbf{mOS} \subseteq \mathbf{mCF} = \mathbf{SLin}$ where **SLin** denotes the family of *semilinear* sets.

In all cases the set of *sentential forms* is defined by $S(G) = \{x \in \Sigma^* \mid \omega \stackrel{*}{\rightarrow} x\}$.

If a subset $\Delta \subseteq \Sigma$ of *nonterminal* symbols is specified the *language* generated by G is defined by $L(G) = S(G) \cap \Delta^*$. In this case the letter $\mathbf{E}$ is used to denote language classes, e.g. $\mathbf{EOL}, \mathbf{ETOL}$.

A *black hole* is a $z \in S(G)$ with the property $\forall x \in S(G) : x \stackrel{*}{\rightarrow} z$.

The set of *black holes* is defined by

$B(G) = \{z \in S(G) \mid \forall x \in S(G) : x \overset{*}{\rightarrow} z\}$.
Trivially, $B(G) \subseteq S(G)$.

We shall use the letter **B** to denote families of languages defined by black holes, yielding **BOR, BIR, BOS, BIS, BON, BIN, BOL, BTOL, mBOS, mBIS**.

Define a relation $\overset{G}{\sim}$ by $(x, y) \in \overset{G}{\sim} \Leftrightarrow (x \overset{*}{\rightarrow} y \ \wedge \ y \overset{*}{\rightarrow} x)$. Obviously, then $\overset{G}{\sim}$ is an equivalence relation, and $B(G)$ is one of the equivalence classes.

A *trap* T is an equivalence class of $\overset{G}{\sim}$ with $T \subseteq S(G)$.

3 Basic Results

Lemma 31 *: If $z \overset{*}{\rightarrow} y$ for $z \in B(G)$ then $y \in B(G)$.*

Proof. Let $z \in B(G)$ and $z \overset{*}{\rightarrow} y$. Then for any $x \in S(G)$ holds $x \overset{*}{\rightarrow} z$, and therefore also $x \overset{*}{\rightarrow} y$, implying $y \in B(G)$.

Corollary 31 *: $B(G)$ is a complete graph with respect to $\overset{*}{\rightarrow}$, and it is an equivalence class of $\overset{G}{\sim}$.*

□

Lemma 32 *: If $B(G) \neq \emptyset$ then $\overset{*}{\rightarrow}$ is confluent on $S(G)$.*

Proof. Let $x, y \in S(G)$. Then $\omega \overset{*}{\rightarrow} x$ and $\omega \overset{*}{\rightarrow} y$. By the definition of $B(G)$ it follows that for any $z \in B(G)$ holds $x \overset{*}{\rightarrow} z$ and $y \overset{*}{\rightarrow} z$. Therefore, $\overset{*}{\rightarrow}$ is confluent on $S(G)$.

Lemma 33 *: If $\overset{*}{\rightarrow}$ is Noetherian on $S(G)$ then $|B(G)| \leq 1$.*

Proof. If $|B(G)| > 1$ then there exist $z_1, z_2 \in B(G)$ with $z_1 \neq z_2$. Since $B(G)$ is a complete graph there would be an infinite sequence with $\overset{*}{\rightarrow}$, a contradiction.

Lemma 34 *: If $\lambda \in S(G)$ then either $B(G) = \emptyset$ or $B(G) = \{\lambda\}$.*

Proof. Obviously $\lambda \overset{*}{\rightarrow} \lambda$. Either $x \overset{*}{\rightarrow} \lambda$ for all $x \in S(G)$ implying $B(G) = \{\lambda\}$, or there exists a $x \in S(G)$ with $\neg(x \overset{*}{\rightarrow} \lambda)$. In that case $B(G) = \emptyset$ since $\neg(\lambda \overset{*}{\rightarrow} y)$ for $y \neq \lambda$.

4 Regular Systems

In this section we consider rewriting systems of regular type with context-independent, monotone, and arbitrary rewriting productions.

First we present some examples of OR systems yielding sets $S(G)$ and $B(G)$ of different cardinalities.

Example 41 *: $G = (\{a, b\}, \{b\}, \{b \rightarrow ba\}) \in POR$.*
$S(G) = \{b\}\{a\}^*$, $B(G) = \emptyset$.

This example also shows that the number of equivalence classes of $\overset{G}{\sim}$ is infinite, since $ba^k \not\sim ba^m$ for $k \neq m$.

Example 42 *: $G = (\{a\}, \{a\}, \{a \to \lambda, a \to aa\})$.*
$S(G) = \{a\}^*$, $B(G) = \{\lambda\}$.

Example 43 *: $G = (\{a, b, c, d, e, f\}, \{f\}, \{f \to a, a \to ba, b \to bb, b \to \lambda, a \to c,$
$c \to d, d \to e, e \to c\}) \in OR$.*
$S(G) = \{b\}^*\{a\} \cup \{c, d, e, f\}$, $B(G) = \{c, d, e\}$.

Example 44 *: $G = (\{a, b\}, \{b\}, \{b \to ab, a \to aa, a \to \lambda\}) \in OR$.*
$S(G) = B(G) = \{a\}^*\{b\}$.

Example 45 *: $G = (\{a, b, c, d\}, \{d\}, \{d \to ab, a \to aa, a \to \lambda, d \to cb, c \to cc,$
$c \to a, b \to ab\}) \in OR$.*
$S(G) = \{d\} \cup \{a, \}^*\{c\}^+\{b\} \cup \{a\}^*\{b\}$, $B(G) = \{a\}^*\{b\}$.

Example 46 *: $G = (\{a, b, c\}, \{a\}, \{a \to aa, a \to bc, b \to bb, b \to \lambda, c \to c\}) \in OR$*
$S(G) = \{a\}^+ \cup \bigcup_{k=0}^{\infty}\{b\}^*\{ca^k\}$, $B(G) = \emptyset$.
There are infinitely many infinite traps $\{b\}^*\{ca^k\}$, $k \geq 0$.

Lemma 41 *: If $G = (\Sigma, \{\omega\}, P)$ is regular then $B(G)$ is a regular set. More precisely, $B(G) \in$ **OR**, hence **BOR** $\subseteq$ **OR**.*

Proof. Let $G = (\Sigma, \{\omega\}, P)$ be a regular rewriting system. If $B(G) = \emptyset$ then $B(G)$ is regular. In case $B(G) \neq \emptyset$ let $z \in B(G)$. Define a new regular rewriting system $G' = (\Sigma, \{z\}, P)$. Since $B(G)$ is a complete graph with respect to $\overset{*}{\to}$ it follows that $S(G') = B(G)$, and therefore $B(G) \in$ **OR**.

Lemma 42 *:* **POR** $\not\subseteq$ **BOR**.

Proof. Consider $L = \{a\}^+\{b\}$. Clearly, $L = L(G) \in$ **POR** by the regular system $G = (\{a, b\}, \{ab\}, \{a \to aa\})$. But $L \in$ **BOR** is possible only if $a^m b \overset{*}{\to} ab$ which can be achieved only by a production $a \to \lambda$, also yielding $b \in L$, a contradiction.

Lemma 43 *:* **PIR** $\not\subseteq$ **BIR**.

Proof. Consider $L = \{a\}^+\{b\} \cup \{a\}\{c\}^+$. Clearly, $L = L(G) \in$ **PIR** by the system $G = (\{a, b, c\}, \{ab\}, \{ab \to ac, ab \to aab, aa \to aaa, ac \to acc\})$. But $L \notin$ **BIR**.

Lemma 44 *:* **OR** $\not\subseteq$ **BIR**.

Proof. Consider $L = \{a\}^* \in$ **OR** by $G = (\{a\}, \{a\}, \{a \to \lambda, a \to aa\})$. But $L \notin$ **BIR**.

Lemma 45 *:* **BIR** $\not\subseteq$ **OR**.

Proof. Consider the language $L = \{a\}^+\{b\}\{a\}^*$. $L \in$ **BIR** by the regular system $G = (\{a, b\}, \{ab\}, \{a \to aa, aa \to a, ab \to aba, aba \to ab\})$. Clearly, $L \notin$ **OR**.

In the sequel let $G = (\Sigma, \{\omega\}, P)$ be an OR system. Define $\Sigma_0 = \{a \in \Sigma \mid a \xrightarrow{*} \lambda\}$ and $\Sigma_1 = \Sigma \setminus \Sigma_0$. Furthermore, define the function ρ_1 on Σ^* by $\rho_1(y) = by_2$ if $y = y_1 b y_2$ with $y_1 \in \Sigma_0^*$, $b \in \Sigma_1$, and $y_2 \in \Sigma^*$. Note that this is well defined.

Lemma 46 *: Let $B(G) \neq \emptyset$ and $B(G) \neq \{\lambda\}$. Then there exists a $k > 0$ such that $x = x_1 a x_2$ with $x_1 \in \Sigma_0^*$, $a \in \Sigma_1$, and $x_2 \in \Sigma^{k-1}$ for all $x \in B(G)$.*

Proof. Since $B(G) \neq \emptyset$ there exists an $x_0 \in B(G)$ of minimal length, say $|x_0| = k$. Assume that there is a $y \in B(G)$ with $y = y_1 \rho_1(y)$, $y_1 \in \Sigma_0^*$, $b \in \Sigma_1$, and $|\rho_1(y)| > k$. Now $y \xrightarrow{*} \rho_1(y) \xrightarrow{*} x_0$. But since $|\rho_1(y)| > k$ it follows that $b \xrightarrow{*} \lambda$, a contradiction.

Lemma 47 *: Let $B(G) \neq \emptyset$ and $B(G) \neq \{\lambda\}$. Then $y = y_1 b y_2$ with $y_1 \in \Sigma_0^*$, $b \in \Sigma_1$, and $|y_2| < k$ for all $y \in S(G)$ where k is the constant from Lemma 46.*

Proof. Assume $y = y_1 b y_2$ with $y_1 \in \Sigma_0^*$, $b \in \Sigma_1$, and $|by_2| > k$. Again, $y \xrightarrow{*} \rho_1(y) \xrightarrow{*} x_0$ must hold, yielding the same contradiction as in Lemma 46.

Lemma 48 *: Let $B(G) \neq \emptyset$ and $B(G) \neq \{\lambda\}$. If N is the constant from the iteration lemma for $S(G)$ (note that $S(G) \in$ **REG**), then $k \leq N$.*

Proof. Assume $k > N$. By Lemma 46 there exists an $x_0 \in B(G)$ with $x_0 = ax_1$, $a \in \Sigma_1$, and $|ax_1| = k$. Since $k > N$ the iteration lemma for $S(G)$ yields infinitely many $y \in S(G)$ with $y = by_2$, $b \in \Sigma_1$, and $y_1 \in \Sigma^*$. Thus there exists a $y \in S(G)$ with $y = by_2$ with $b \in \Sigma_1$, $y_2 \in \Sigma^*$ and $|by_2| > k$, a contradiction to Lemma 47.

Theorem 41 *: The emptiness problem for* **BOR** *is decidable, i.e. for any $G \in OR$ it is decidable whether $B(G) = \emptyset$.*

Proof. Since the membersip problem for **OR** $\subset$ **REG** is decidable, $\lambda \in S(G)$ is decidable. If $\lambda \in S(G)$ then $B(G) = \{\lambda\}$. Thus assume $\lambda \notin S(G)$.

Construct the set $D = \{x \in S(G) \cap \Sigma_1 \Sigma^* \mid |x| \leq N\}$. Since the membership problem for **OR** is decidable this can be done effectively. Obviously, $|D| < \infty$.

Now, for all $x \in D$ it is decidable whether $y \xrightarrow{*} x$ for all $y \in D$ since the reachability problem for **OR** $\subset$ **REG** is decidable. Let $C = \{x \in D \mid \forall y \in D : y \xrightarrow{*} x\}$. Clearly, C can be constructed effectively.

Since $S(G) \cap \Sigma_1 \Sigma^* \in$ **REG** it is decidable whether there exists an $x \in S(G) \cap \Sigma_1 \Sigma^*$ with $|x| > N$. This follows by the decidability of the finiteness problem for $S(G) \cap \Sigma_1 \Sigma^*$.

If there exists an $x \in S(G) \cap \Sigma_1 \Sigma^*$ with $|x| > N$ then $B(G) = \emptyset$ by Lemma 48 since the iteration lemma for $S(G)$ yields infinitely many $y \in S(G) \cap \Sigma_1 \Sigma^*$.

Thus assume $S(G) \cap \Sigma_1 \Sigma^* = D$. Now, for all $y \in S(G)$ holds $y \xrightarrow{*} \rho_1(y) \in D$. If $B(G) \neq \emptyset$ then $\rho_1(x) \in C$ for $x \in B(G)$ since $y \xrightarrow{*} x \xrightarrow{*} \rho_1(x)$ for all $y \in D$. This implies $C \subseteq B(G)$ and $C \neq \emptyset$.

On the other hand, if $C \neq \emptyset$ then obviously $C \subseteq B(G)$ since any $x \in C$ is reachable from any $y \in S(G)$ by $y \xrightarrow{*} \rho_1(y) \xrightarrow{*} x$. This implies $B(G) \neq \emptyset$.

Hence $C = \emptyset$ iff $B(G) = \emptyset$.

Therefore the emptiness problem for **BOR** is decidable.

Theorem 42 *: The membership problem for* **BOR** *is decidable.*

Proof. By Theorem 41 an OR system $G' = (\Sigma, \{x\}, P)$ with $S(G') = B(G)$ can be constructed effectively, taking some $x \in C$ in the case $B(G) \neq \emptyset$. Since $B(G) \in \mathbf{REG}$ the membership problem is decidable.

Theorem 43 *: The finiteness problem for* **BOR** *is decidable.*

Proof. As in Theorem 42 construct an OR system G' with $S(G') = B(G)$. Since the finiteness problem for **REG** is decidable, it is decidable for **BOR**, too.

Theorem 44 *: The equivalence problem for* **BOR** *is decidable.*

Proof. By Theorem 42 for $G_1, G_2 \in OR$ systems $G_1', G_2' \in OR$ can be constructed such that $S(G_1') = B(G_1)$ and $S(G_2') = B(G_2)$. Then the statement follows from the decidability of the equivalence problem for **REG**.

Theorem 45 *: For any $G = (\Sigma, \{\omega\}, P) \in OR$ with $B(G) \neq \emptyset$*
a $G' = (\Sigma, \{x\}, P) \in OR$ can be constructed effectively such that $S(G') = B(G') = B(G)$.

Proof. This follows from Theorem 42.

5 Semi-Thue Systems

In this section we consider rewriting systems of Semi-Thue type with context-independent, monotone, and arbitrary productions with *catenation* as underlying operation.

Lemma 51 *: If $G = (\Sigma, \{\omega\}, P)$ is context-free then $B(G)$ is context-free. More precisely, $B(G) \in$* **OS***, hence* **BOS** $\subseteq$ **OS**.

Proof. Let $G = (\Sigma, \{\omega\}, P)$ be a context-free rewriting system. If $B(G) = \emptyset$ then $B(G)$ is context-free. In case $B(G) \neq \emptyset$ let $z \in B(G)$. Define the context-free rewriting system $G' = (\Sigma, \{z\}, P)$. Since $B(G)$ is a complete graph with respect to $\overset{*}{\rightarrow}$ it follows that $S(G') = B(G)$, and therefore $B(G) \in$ **OS**.

Note that the rewriting system G' in the previous lemma has not been constructed but only its existence has been proved.

Lemma 52 *If $G = (\Sigma, \{\omega\}, P)$ is monotone then $B(G)$ is finite.*

Proof. Assume $|B(G)| = \infty$. Then there exist $z_1, z_2 \in B(G)$ with $|z_1| < |z_2|$. But $z_2 \overset{*}{\rightarrow} z_1$ contradicts the monotonicity of G. Therefore $|B(G)| < \infty$.

Lemma 53 *: If $G = (\Sigma, \{\omega\}, P)$ is arbitrary then $B(G)$ is arbitrary, too. More precisely, $B(G) \in$* **IS***, hence* **BIS** $\subseteq$ **IS**.

Proof. Let $G = (\Sigma, \{\omega\}, P)$ be an arbitrary rewriting system. If $B(G) = \emptyset$ then $B(G)$ is context-free. In case $B(G) \neq \emptyset$ let $z \in B(G)$. Define the arbitrary rewriting system $G' = (\Sigma, \{z\}, P)$. Since $B(G)$ is a complete graph with respect to $\stackrel{*}{\rightarrow}$ it follows that $S(G') = B(G)$, and therefore $B(G) \in$ **IS**.

Note that the rewriting system G' in the previous lemma too has not been constructed but only its existence has been proved.

The following examples exhibit OS systems with different cardinalities for $S(G)$ and $B(G)$.

Example 51 *:* $G = (\{a, b\}, \{b\}, \{b \rightarrow ba\}) \in OS$.
$S(G) = \{b\}\{a\}^*$, $B(G) = \emptyset$.

Example 52 *:* $G = (\{a, b, c, d, e\}, \{a\}, \{a \rightarrow bc, b \rightarrow bb, b \rightarrow \lambda, a \rightarrow c, c \rightarrow d,$
$d \rightarrow e, e \rightarrow c\}) \in OS$.
$S(G) = \{a\} \cup \{b\}^*\{c, d, e\}$, $B(G) = \{c, d, e\}$.

Example 53 *:* $G = (\{a, b, c, d\}, \{d\}, \{d \rightarrow bc, d \rightarrow ba, b \rightarrow ba, c \rightarrow a, c \rightarrow cc,$
$a \rightarrow \lambda\}) \in OS$.
$S(G) = \{d\} \cup \{b\}\{a, c\}^* \cup \{b\}\{a\}^*$, $B(G) = \{b\}\{a\}^*$.

Example 54 *:* $G = (\{a, b, c, d, e, f, g, h, k, p, q, r, s\}, \{s\},$
$\{s \rightarrow abc, ab \rightarrow adeb, eb \rightarrow be, ec \rightarrow fcc, bf \rightarrow fb, af \rightarrow aa,$
$df \rightarrow fb, abb \rightarrow ghb, hb \rightarrow bh, hc \rightarrow k, bk \rightarrow kb, gk \rightarrow \lambda$
$s \rightarrow pqr, q \rightarrow qq, q \rightarrow \lambda, pr \rightarrow abc\}) \in IS$
$S(G) \cap \{a\}^*\{b\}^*\{c\}^* = B(G) \cap \{a\}^*\{b\}^*\{c\}^* = \{a^n b^n c^n | n > 0\}$.

Lemma 54 *:* **BOS** $\subset$ **CF**.

Proof. **BOS** $\subset$ **CF** follows from Lemma 51 since **OS** $\subset$ **CF**.

Lemma 55 *:* **REG** $\not\subseteq$ **BIS**

Proof. Consider $L = \{a\}^+\{b\} \cup \{a\}\{c\}^+$ as in Lemma 43.

Lemma 56 *:* **POS** $\not\subseteq$ **BIS**.

Proof. Consider $L = \{d\} \cup \{a\}^+\{b\} \cup \{a\}\{c\}^+$. Clearly, $L \in$ **POS** by the system $G = (\{a, b, c, d\}, \{d\}, \{d \rightarrow ab, d \rightarrow ac, b \rightarrow ab, c \rightarrow cc\})$. But $L \notin$ **BIS** since there must hold $ac^m \stackrel{*}{\rightarrow} d$ yielding also $ac^{k+m} \stackrel{*}{\rightarrow} dc^k \in L$, a contradiction.

Theorem 51 *: For $G \in IS$ it is undecidable whether $B(G) = \emptyset$, i.e. the emptiness problem for* **BIS** *is undecidable.*

Proof. Consider a Post correspondence problem (**PCP**) on $\{a, b\}$ given by the pairs $\{(\alpha_j, \beta_j) \in \{a, b\}^* \times \{a, b\}^* | 1 \leq j \leq n\}$. It may be assumed that all α_j, β_j are non-empty.

Construct the IS system $G = (\{a, b, c, d, e, f, \$\}, \{\$c\$\}, P)$ with productions
$P = \{c \rightarrow \alpha_j c \beta_j^R \ (1 \leq j \leq n), xcx \rightarrow d, xdx \rightarrow d, ydz \rightarrow e \ (y \neq z), yd\$ \rightarrow e\$,$
$\$dy \rightarrow \$e, ycz \rightarrow e \ (y \neq z), xe \rightarrow e, ex \rightarrow e, \$e\$ \rightarrow \$c\$, \ \$d\$ \rightarrow \$fdf\$,$
$fdf \rightarrow ffdfff, \ (x, y, z \in \{a, b\})\}$
where u^R denotes the mirror image of u.
Then the PCP has a solution iff $B(G) = \emptyset$.

Lemma 57 *: For given $G \in IS$ and $w \in \Sigma^*$ it is undecidable whether $w \in B(G)$, i.e. the membership problem for* **BIS** *is undecidable.*

Proof. Consider the system in Theorem 51. Obviously, $\$c\$ \in B(G)$ iff the PCP has no solution, implying the undecidability of the word problem for **BIS**.

Lemma 58 *: If $G \in BPIS$ then the following holds :*
$\forall x, y \in B(G) : |x| = |y|$.

Proof. Assume that there exist $x, y \in B(G)$ with $|x| < |y|$. Since $y \stackrel{*}{\rightarrow} x$ there must exist a production $\alpha \rightarrow \beta \in P$ with $|\alpha| > |\beta|$, a contradiction.

In the sequel let $G = (\Sigma, \{\omega\}, P)$ be an OS system. Define $\Sigma_0 = \{a \in \Sigma \mid a \stackrel{*}{\rightarrow} \lambda\}$ and $\Sigma_1 = \Sigma \setminus \Sigma_0$. Furthermore, let $\pi_1 : \Sigma^* \rightarrow \Sigma_1^*$ denote the projection on Σ_1, defined by $\pi_1(y) = a_1 \cdots a_m$ if $y = y_0 a_1 y_1 \cdots a_m y_m$ with $y_i \in \Sigma_0^*$ ($0 \leq i \leq m$) and $a_j \in \Sigma_1$ ($1 \leq j \leq m$).

Lemma 59 *: Let $B(G) \neq \emptyset$. Then there exists a $k > 0$ such that $|x|_{\Sigma_1} = k$ for all $x \in B(G)$.*

Proof. Since $B(G) \neq \emptyset$ there exists an $x_0 \in B(G)$ of minimal length, say $|x_0| = k$. Now assume that there is a $y \in B(G)$ with $|y|_{\Sigma_1} > k$, i.e. $|\pi_1(y)| > k$. Now $y \stackrel{*}{\rightarrow} \pi(y) \stackrel{*}{\rightarrow} x_0$. But since $y = a_1 \cdots a_m$ this implies that $a_j \stackrel{*}{\rightarrow} \lambda$ for some j, a contradiction to the minimality of k.

Lemma 510 *: Let $B(G) \neq \emptyset$. Then $|y|_{\Sigma_1} \leq k$ for all $y \in S(G)$ where k is the constant from Lemma 59.*

Proof. Assume $|y|_{\Sigma_1} > k$ for some $y \in S(G)$. Again, $y \stackrel{*}{\rightarrow} \pi_1(y) \stackrel{*}{\rightarrow} x_0$ must hold, yielding the same contradiction as in Lemma 58.

Lemma 511 *: Let $B(G) \neq \emptyset$. If N is the constant from the iteration lemma for $S(G)$ (note that $S(G) \in$* **CF** *), then $k \leq N$.*

Proof. Assume $k > N$. By Lemma 59 there exists an $x_0 \in B(G)$ with $x_0 \in \Sigma_1^*$ and $|x_0| = k$. Since $k > N$ the iteration lemma for $S(G)$ yields infinitely many $y \in S(G)$ with $y \in \Sigma_1^*$. Thus there exists a $y \in S(G)$ with $|y|_{\Sigma_1} > k$, a contradiction to Lemma 59.

Theorem 52 *: The emptiness problem for* **BOS** *is decidable, i.e. for any $G \in OS$ it is decidable whether $B(G) = \emptyset$.*

Proof. Construct the set $D = \{x \in S(G) \cap \Sigma_1^+ \mid |x| \leq N\}$. Since the membership problem for **OS** is decidable this can be done effectively. Obviously, $|D| < \infty$.

Now, for all $x \in D$ it is decidable whether $y \xrightarrow{*} x$ for all $y \in D$ since the reachability problem for $\mathbf{OS} \subset \mathbf{CF}$ is decidable. Let $C = \{x \in D | \ \forall y \in D : y \xrightarrow{*} x\}$. Clearly, C can be constructed effectively.

Since $S(G) \cap \Sigma_1^* \in \mathbf{CF}$ it is decidable whether there exists an $x \in S(G) \cap \Sigma_1^*$ with $|x| > N$. This follows by the decidability of the finiteness problem for $S(G) \cap \Sigma_1^*$.

If there exists an $x \in S(G) \cap \Sigma_1^*$ with $|x| > N$ then $B(G) = \emptyset$ by Lemma 510 since the iteration lemma for (G) yields infinitely many $y \in S(G) \cap \Sigma_1^*$.

Thus assume $S(G) \cap \Sigma_1^* = D$. Now, for all $y \in S(G)$ holds $y \xrightarrow{*} \pi_1(y) \in D$. If $B(G) \neq \emptyset$ then $\pi_1(x) \in C$ for $x \in B(G)$ since $y \xrightarrow{*} x \xrightarrow{*} \pi_1(x)$ for all $y \in D$. This implies $C \subseteq B(G)$ and $C \neq \emptyset$.

On the other hand, if $C \neq \emptyset$ then obviously $C \subseteq B(G)$ since any $x \in C$ is reachable from any $y \in S(G)$ by $y \xrightarrow{*} \pi_1(y) \xrightarrow{*} x$. This implies $B(G) \neq \emptyset$.

Hence $C = \emptyset$ iff $B(G) = \emptyset$.

Therefore the emptiness problem for **BOS** is decidable.

Theorem 53 *: The membership problem for* **BOS** *is decidable.*

Proof. By Theorem 52 an OS system $G' = (\Sigma, \{x\}, P)$ with $S(G') = B(G)$ can be constructed effectively, taking some $x \in C$ in the case $B(G) \neq \emptyset$. Since $B(G) \in \mathbf{CF}$ the membership problem is decidable.

Theorem 54 *: The finiteness problem for* **BOS** *is decidable.*

Proof. As in Theorem 53 construct an OS system G' with $S(G') = B(G)$. Since the finiteness problem for **CF** is decidable, it is decidable for **BOS**, too.

Theorem 55 *: For any $G = (\Sigma, \{\omega\}, P) \in OS$ with $B(G) \neq \emptyset$ a $G' = (\Sigma, x, P) \in OS$ can be constructed effectively such that $S(G') = B(G') = B(G)$.*

Proof. This follows from Theorem 53.

6 Lindenmayer Systems

Also here we just present some examples yielding infinite black hole sets.

Example 61 *: Let $G = (\{a, b, c, d\}, \{b\}, \{a \to dd, a \to \lambda, b \to ca, c \to b, d \to a\}) \in OL$.*

Then $S(G) = B(G) = \{b\}\{dd\}^* \cup \{ca\}\{aa\}^*$ since by induction we have $b \to ca \to b$ and $b \xrightarrow{*} bd^{2k} \to ca^{2k+1} = ca^k a^{k+1} \to bd^{2k+1} \to ca^{2k+3} \xrightarrow{*} b$.

Example 62 *: Let $G = (\{a, b\}, \{ba\}, \{\{a \to aa, b \to b\}, \{a \to \lambda, b \to ba\}\}) \in TOL$.*

Then $S(G) = B(G) = \{ba^{2^k} \mid \ k \geq 0\}$ since $ba \xrightarrow{*} ba^{2^k} \to ba$.

Example 63 *: Let* $G = (\{a,b\}, \{ba\}, \{\{a \to aa, b \to b\},$
$\{a \to a^3, b \to b\}, \{a \to \lambda, b \to ba\}\}) \in TOL$.
Then $S(G) = B(G) = \{ba^{2^k 3^m} \mid k,m \geq 0\}$ *since* $ba \Rightarrow ba^{2^k} \Rightarrow ba^{2^k 3^m} \to ba$. *Note that* $B(G) \notin$ **EOL**.

Lemma 61 *:* $\mathbf{POL} \not\subseteq \mathbf{BTOL}$

Proof. Consider $G = (\{a\}, \{a\}, \{a \to aa\})$.
Obviously $L = S(G) = \{a^{2^k} \mid k \geq 0\} \in$ **POL**.
Assume $L = B(G') \in$ **BTOL** with $G' = (\}a\}, \{\omega\}, T)$. Then there exists a table $P_j \in T$ with $a \to \lambda \in P_j$. Otherwise $B(G') = \emptyset$. But this implies $\lambda \in S(G')$ from which follows $B(G') = \{\lambda\}$, a contradiction.

Lemma 62 *:* $\mathbf{POR} \not\subseteq \mathbf{BTOL}$ *,* $\mathbf{POS} \not\subseteq \mathbf{BTOL}$

Proof. Consider $G = (\{a\}, \{a\}, \{a \to aa\}) \in POR$ (or $\in POS$).
Obviously $L = S(G) = \{a^k \mid k > 0\} \in$ **POR** (or $\in$ **POS**).
Assume $L = B(G') \in$ **BTOL** with $G' = (\}a\}, \{\omega\}, T)$. Then there exists a table $P_j \in T$ with $a \to \lambda \in P_j$. Otherwise $B(G') = \emptyset$. But this implies $\lambda \in S(G')$ from which follows $B(G') = \{\lambda\}$, a contradiction.

Lemma 63 *:* $\mathbf{BTOL} \not\subseteq \mathbf{EOL}$.

Proof. Consider G from Example 63.

7 Multiset Rewriting Systems

In this section we present some results on generative power and decidability.

First we give an example, actually the commutative version of example 5.3.

Example 71 *:* $G = (\{a,b\}, \{b\}, \{b \to ba\}) \in mOS$.
$S(G) = \{b\} \oplus \{a\}^{\oplus}$, $B(G) = \emptyset$.

Example 72 *:* $G = (\{a,b,c,d\}, \{d\}, \{d \to bc, d \to ba, b \to ba, c \to a, c \to cc,$
$a \to \lambda\}) \in mOS$.
$S(G) = \{d\} \cup \{b\} \oplus \{c\} \oplus \{a,c\}^{\oplus} \cup \{b\} \oplus \{a\}^{\oplus}$, $B(G) = \{b\} \oplus \{a\}^{\oplus}$.

Lemma 71 *: If* $G \in mBPIS$ *then the following holds :*
$\forall x, y \in B(G) : |x| = |y|$.

Proof. This is shown exactly as in Lemma 57.

Theorem 71 *: For any* $G \in mIS$ *holds* $B(G) \in$ **mCF**, *i.e.* $\mathbf{mBIS} \subseteq \mathbf{mCF}$.

Proof. This follows from [1], Corollary 4.11, stating that each equivalence class (there called strongly connected component) of $\overset{G}{\sim}$ is semilinear.

It should be remarked that the proof of the previous theorem is not constructive for the semilinear set.

Analogous to Lemmata 58 to 510, and Theorems 52 to 54, we can state the following lemma and theorem, where the proofs are similar to those from Section 5, words replaced by multisets. The definitions of Σ_0, Σ_1, and π_1 are analogous to those for words. For the context-independent case the proofs are alternatives to those given below for the general case.

Lemma 72 *: Let $B(G) \neq \emptyset$. Then the following facts hold :*

1. There exists a $k > 0$ such that $|x|_{\Sigma_1} = k$ for all $x \in B(G)$.

2. $|y|_{\Sigma_1} \leq k$ for all $y \in S(G)$ where k is the constant from 1.

3. If N is the constant from the iteration lemma for $S(G)$ (note that $S(G) \in$ **mCF** *), then $k \leq N$.*

□

Theorem 72 *:*

1. The emptiness problem for **mBOS** *is decidable, i.e. for any $G \in mOS$ it is decidable whether $B(G) = \emptyset$.*

2. The membership problem for **mBOS** *is decidable.*

3. The finiteness problem for **mBOS** *is decidable.*

□

The following results have been proved in [1]. There $S(G)$ is denoted by $R_N(M)$, the reachability set of the P/T net N with initial marking M, and $B(G)$ by $C_N(M)$.

Theorem 73 *: The membership problem for* **mBIS** *is decidable, i.e. for given $G \in mIS$ and $y \in \Sigma^{\oplus}$ it is decidable whether $y \in B(G)$.*

Proof. This is stated in [1], Corollary 4.12. Another proof is in [2] as a corollary of Theorem 2.

Theorem 74 *: 1. The finiteness problem for* **mBIS** *is decidable, i.e. for given $G \in mIS$ it is decidable whether $|B(G)| < \infty$.*

2. The inclusion problem for **mBIS** *is decidable, i.e. for given $G, G' \in mIS$ it is decidable whether $B(G) \subseteq B(G')$.*

3. The equivalence problem for **mBIS** *is decidable, i.e. for given systems $G, G' \in mIS$ it is decidable whether $B(G) = B(G')$.*

Proof. This is stated in [1], Corollary 4.12.

Theorem 75 *: For given $G \in mIS$ it is decidable whether $B(G) = S(G)$.*

Proof. This is [1], Theorem 5.3.

Theorem 76 *: The emptiness problem for* **mBIS** *is decidable, i.e. for given $G \in IS$ it is decidable whether $B(G) = \emptyset$.*

Proof. This follows from Theorem **??** with $B(G') = \emptyset$.

Theorem 77 *: For $G \in mIS$ with $B(G) \neq \emptyset$ a $G' \in mIS$ can be constructed effectively, such that $S(G') = B(G') = B(G)$.*

Proof. This is [1], Theorem 5.4.

8 Open Problems

More results on Normal and Lindenmayer systems, as well as closure properties of black hole classes, will be presented in a forthcoming article. Open are characterization and decidability properties for IR and IS systems, in particular to construct a $G' \in mIS$ with $S(G') = B(G') = B(G)$ for given $G \in mOS$. We also conjecture that **BOS** $\subseteq$ **REG**.

Acknowledgement

We thank **Matthias Jantzen** for fruitful discussions on the topic.

References

1. T. Araki, T. Kasami : Decidable Problems on the Strong Connectivity of Petri Net Reachability Sets. **TCS 4**, pp 99-119, 1977.
2. D. Frutos Escrig, C. Johnen : Decidability of Home Space Property. Technical Report LRI 503, 1989.
3. M. Hack : Analysis of Production Schemata by Petri Nets. MIT, Project MAC, TR-94, 1972. Corrections to 'Analysis of Production Schemata by Petri nets'. MIT, Project MAC, Computation Structure Note No. 17, 1974.
4. M. Kudlek, C. Martín Vide, Gh. Păun : Toward FMT (Formal Macroset Theory). *Multiset Processing*, eds. C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, LNCS 2235, pp 123-133, 2001.
5. M. Kudlek, V. Mitrana : Normal Forms of Grammars, Finite Automata, Abstract Families, and Closure Properties of Macrosets. *Multiset Processing*, eds. C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, LNCS 2235, pp 135-146, 2001.
6. M. Kudlek, V. Mitrana : Closure Properties of Multiset Language Families. **FI 49** (1-3), pp 191-203, 2002.
7. R. Melinte, O. Oanea, I. Olga, F. L. Ţiplea : The Home Marking Problem and Some Related Concepts. Acta Cybernetica **15** 3, pp 467-478, 2002, and Proc. PROMISE'2002 LNI P-21, ed. J. Desel, pp 104-115, Springer, 2002.
8. G. Memmi, J. Vautherin : Analysing Nets by the Invariant Method. *Advances in Petri Nets 1986*, part 1, eds. W. Brauer, W. Reisig, G. Rozenberg, pp 300-336, 1987.
9. G. Rozenberg, A. Salomaa : The Mathematical Theory of L Systems. Academic Press, 1980.
10. A. Salomaa : Formal Languages. Academic Press, New York, London, 1973.

Restoration of Punctured Languages

Gerhard Lischke

Institute of Informatics, Faculty of Mathematics and Informatics
Friedrich Schiller University Jena, D-07743 Jena, Germany
email: lischke@minet.uni-jena.de

Abstract. Punctured languages are languages whose words are partial words in such sense that the letters at some positions are unknown. Considering such languages is motivated by molecular biology of nucleic acids. We investigate to which extent restorations of punctured languages is possible if the number of unknown positions or the proportion of unknown positions per word, respectively, is bounded, and we study their relationships for different boundings.

1 Introduction

The concept of partial words have been introduced by Berstel and Boasson in 1998 [2]. It was motivated of molecular biology of nucleic acids. DNA molecules which are the carriers of the genetic information in almost all organisms can be seen as finite strings over a 4-element alphabet namely the nucleotides adenine, cytosine, guanine, and thymine. Processes in molecular biology can be seen as operations on such strings [6]. Thereby in nature often occures that the strings are imperfect and mismatches may result. But alignment of genes or strings is still possible if the mismatches are not very frequent. To study their influence the positions in question are regarded as unknown or holes, and to speak about partial words.

Berstel and Boasson ([2], see also [3]) introduce a partial word w of length n over an alphabet X as a partial function w from $\{1, ..., n\}$ into X. If $D(w)$ denotes the domain of w, then $Hol(w) =_{Df} \{1, ..., n\} \setminus D(w)$ is the *set of holes* of w. To each partial word w of length n over X is associated its *companion* $w_\Diamond$ which is the following total function from $\{1, ..., n\}$ into the augmented alphabet $X_\Diamond =_{Df} X \cup \{\Diamond\}$:

$$w_\Diamond(i) =_{Df} \begin{cases} w(i) & \text{if } i \in D(w) \\ \Diamond & \text{if } i \in Hol(w) \end{cases} .$$

The new symbol $\Diamond \notin X$ is viewed as a "do not know" symbol.

Because of $w \mapsto w_\Diamond$ is a bijection, and for simplifying our considerations, in the following we identify a partial word w and its companion $w_\Diamond$ with the ordinary, total word $w_\Diamond(1)w_\Diamond(2)\cdots w_\Diamond(n)$ over $X_\Diamond$, and we shall not use the functional approach to partial words (see Definition 1 below).

The extent of influence of gene defects and the ability to restore defect genes are of invaluable importance to modern medicine. If, for instance, a set of DNA words fulfilling a certain property, has changed a little bit after some time or under some influence, it is important to know whether the desired property still holds. The original set may be seen as punctured with holes like in partial words, and the question is, to a language of which kind it may be restored. In Section 2, after recalling the most important notions for words and languages, we introduce *punctured*

languages over X as sets of partial words over X. A *puncturing* over X is a function from X^* into $X_\Diamond^*$ which is length preserving and "puts the holes into the words". A (conventional) language L is called a *restoration* of a punctured language L' if there exists a puncturing f such that $f(L) = L'$. We are interested in the relationship between classes of languages and possibilities of restorations after puncturing these languages. Thereby we restrict ourselves mainly to the classes of the Chomsky hierarchy. It plays an important role whether and in which way the number or the proportion of holes per word is bounded. In Section 2 we also define several classes of punctured languages which are worth considering, and we define their restoration classes. With six observations we'll state some general properties.

Section 3 summarizes some useful lemmata. In Section 4 we show that there exist punctured regular languages with at most k holes per word which cannot be restored to languages which are restored from punctured enumerable sets with at most k' holes per word where $k' < k$. Calling two languages to be *k-similar* if, simply spoken, their words differ by at most k letters this means, that there exist languages which are k-similar to regular languages but not k'-similar to enumerable languages. From this we conclude that the restorations of the punctured languages from each class of the Chomsky hierarchy create a strict hierarchy with respect to the maximal number of holes per word. We also show that each context-free language is a restoration of a punctured regular language with unbounded number of holes but not for bounded number of holes. In contrast to this, there exist context-sensitive languages which are not restorations of punctured context-free languages, and not all enumerable languages are restorations of punctured context-sensitive languages, even with unbounded number of holes.

In Section 5 we consider punctured languages where the number of holes per word is not bounded by a constant but the proportion of holes per word is bounded. The most theorems from Section 4 remain valid with modified proofs if the ratio of holes is smaller than $\frac{1}{2}$. If this ratio is $\geq \frac{1}{2}$ we only can prove some weaker results and some open problems still remain. The weaker results include consideration of slender languages, that are languages for which the number of words of the same length is bounded from above by a constant.

2 Notation, definitions, and general observations

Because it is not meaningful to consider punctured languages over a one-letter alphabet (any "unknown" symbol would not really be unknown in this case), for the rest of the paper we fix an alphabet X of having at least two symbols, furthermore we can assume $X = \{a, b\}$. As usual (see, e.g., [5,7,12]), X^* denotes the free monoid generated by X or the set of all words over X. The empty word we denote by e. A (formal) language (over X) is a subset L of X^*, $L \subseteq X^*$. $\subset$ between sets denotes the strict inclusion.

For a word $w \in X^*$, $|w|$ denotes the length of w, and for $1 \leq i \leq |w|$, $w[i]$ is the letter at the i-th position of w. For $x \in X$ and $w \in X^*$, $|w|_x =_{Df} |\{i : w[i] = x\}|$ is the number of occurences of the letter x in the word w.

For a set M, $|M|$ denotes the cardinality of M, and $\mathcal{P}(M)$ denotes the set of all subsets of M. For a natural number k ($k \in \mathbb{N}$), w^k denotes the concatenation of k copies of the word w. w^* denotes the set $\{w^k : k \in \mathbb{N}\}$, w^+ denotes the set $\{w^k : k \in \mathbb{N} \setminus \{0\}\}$, and w^*q the set $\{p^kq : k \in \mathbb{N}\}$.

For a set L of words let $length(L) =_{Df} \{|w| : w \in L\}$. Two languages L_1 and L_2 are *length-equivalent*, $L_1 \sim_l L_2$, if $length(L_1) = length(L_2)$.

Definition 1. A *partial word* or a *punctured word* over X is an element from $X^*_\Diamond$, where $X_\Diamond =_{Df} X \cup \{\Diamond\}$. A *punctured language* is a subset of $X^*_\Diamond$. For a partial word $w \in X^*_\Diamond$, $Hol(w) =_{Df} \{i : 1 \leq i \leq |w| \wedge w[i] = \Diamond\}$ is the *set of holes* of w. For a natural number k, w is called *k-punctured* if $|Hol(w)| \leq k$. For a positive rational number $\delta < 1$, w is called *δ-punctured* if $\frac{|Hol(w)|}{|w|} \leq \delta$ (or $w = e$).

Definition 2. The partial word u is said to be *contained* in the partial word v, or v is a *restoration* of u, denoted by $u \subset v$, if $|u| = |v| \wedge \forall i(1 \leq i \leq |u| \wedge u[i] \neq \Diamond \rightarrow u[i] = v[i])$.

This notation is adopted from [2] where it is justified by the functional point of view for partial words. If $u \subset v$ and there are holes in u but not in v then we understand v as a possible extension or restoration of the partial word u, and u obtained by puncturing from v.

Definition 3. f is a *puncturing* over X if f is a function from X^* into $X^*_\Diamond$ such that $f(w) \subset w$ for each $w \in X^*$.

Definition 4. Let L_1 be a punctured language and L_2 be a (conventional) language (both over X). L_2 is a *restoration* of L_1, or L_1 is *extendable* to L_2, denoted by $L_1 \nearrow L_2$, if there exists a puncturing f such that $f(L_2) = L_1$.

Observation 1. *If* $L_1 \nearrow L_2$ *then* $L_1 \sim_l L_2$.

Now we define classes of punctured languages which are worth considering.

Definition 5. Let $\mathcal{L}$ be a class of languages, i.e., $\mathcal{L} \subseteq \mathcal{P}(X^*)$, k a natural number, and $\delta < 1$ a positive rational number. We define:

$\mathcal{L}_\Diamond =_{Df} \{L : L \subseteq X^*_\Diamond \wedge \exists L'(L' \in \mathcal{L} \wedge L \nearrow L')\}$,

$\mathcal{L}_{k-\Diamond} =_{Df} \{L : L \in \mathcal{L}_\Diamond \wedge \forall w(w \in L \rightarrow |Hol(w)| \leq k)\}$,

$\mathcal{L}_{\delta-\Diamond} =_{Df} \{L : L \in \mathcal{L}_\Diamond \wedge \forall w(w \in L \setminus \{e\} \rightarrow \frac{|Hol(w)|}{|w|} \leq \delta)\}$.

δ is also called the *puncturedness coefficient* for the languages in $\mathcal{L}_{\delta-\Diamond}$.

If $\mathcal{L}_\odot$ is one of the classes defined before then

$\mathcal{L}^+_\odot =_{Df} \{L : L \in \mathcal{L}_\odot \wedge \forall L'(L \nearrow L' \rightarrow L' \in \mathcal{L})\}$.

Remark. To avoid special exceptional cases, in the following $\mathcal{L}$ should be an infinite class containing infinite languages.

Observation 2. *For each* $\mathcal{L} \subseteq \mathcal{P}(X^*)$, $1 \leq k < k'$, *and* $0 \leq \delta < \delta' < 1$:

$$\mathcal{L} = \mathcal{L}_{0-\Diamond} \subseteq \mathcal{L}^+_{k-\Diamond} \subseteq \mathcal{L}_{k-\Diamond} \subset \mathcal{L}_{k'-\Diamond} \subset \bigcup_{l \in \mathbb{N}} \mathcal{L}_{l-\Diamond} \subset \mathcal{L}_\Diamond, \qquad \mathcal{L}^+_{k-\Diamond} \subseteq \mathcal{L}^+_{k'-\Diamond} \subseteq \mathcal{L}^+_\Diamond \subseteq \mathcal{L}_\Diamond,$$

$$\mathcal{L} \subseteq \mathcal{L}^+_{\delta-\Diamond} \subseteq \mathcal{L}_{\delta-\Diamond} \subset \mathcal{L}_{\delta'-\Diamond} \subset \mathcal{L}_\Diamond, \qquad \mathcal{L}_\Diamond \cap \mathcal{P}(X^*) = \mathcal{L}.$$

Proof. The strict inclusions are true because of differences between the maximal numbers of holes. To illustrate possible equalities look, for instance, at the classes $\mathcal{L} = \{L : L \subseteq a^*\} \in \mathcal{P}(\{a,b\}^*)$ and $\mathcal{L} = \{L : L \sim_l L'\}$ for some L', respectively. □

In general there exist languages L_1, L_2, L_3 such that $L_1 \in \mathcal{L}$, $L_2 \notin \mathcal{L}$, $L_3 \in \mathcal{L}_\odot$, $L_3 \nearrow L_1$, and $L_3 \nearrow L_2$. Then $L_3 \notin \mathcal{L}^+_\odot$. $\mathcal{L}^+_\odot$ is the class of all languages punctured in the sense of $\odot$ which are only extendable to languages from $\mathcal{L}$. On the other hand, $\mathcal{L}^\cap_\odot$ and $\mathcal{R}(\mathcal{L}_\odot)$ from the next definition are the classes of languages from $\mathcal{L}$ which are resistent with respect to $\odot$-puncturing and the full extent from $\mathcal{L}$ after $\odot$-puncturing and restoration, respectively. In the just mentioned case, $L_2 \in \mathcal{R}(\mathcal{L}_\odot)$ would follow.

Definition 6. For $\odot \in \{\Diamond, k-\Diamond, \delta-\Diamond\}$ we define:

$$\mathcal{L}^\cap_\odot =_{Df} \{L : L \in \mathcal{L} \wedge \forall L' \forall L'' (L' \in \mathcal{L}_\odot \wedge L' \nearrow L \wedge L' \nearrow L'' \rightarrow L'' \in \mathcal{L})\},$$

$$\mathcal{R}(\mathcal{L}_\odot) =_{Df} \{L : L \subseteq X^* \wedge \exists L' (L' \in \mathcal{L}_\odot \wedge L' \nearrow L)\}.$$

The classes $\mathcal{R}(\mathcal{L}_\odot)$ are called *restoration classes* of punctured languages.

Observation 3. *For each* $\mathcal{L} \subseteq \mathcal{P}(X^*)$, $1 \leq k < k'$, $0 < \delta < \delta' < 1$, *and* $\odot \in \{\Diamond, k-\Diamond, \delta-\Diamond\}$:

$$\mathcal{L}^\cap_\odot \subseteq \mathcal{L} = \mathcal{R}(\mathcal{L}^+_\odot) \subseteq \mathcal{R}(\mathcal{L}_{k-\Diamond}) \subseteq \mathcal{R}(\mathcal{L}_{k'-\Diamond}) \subseteq \mathcal{R}(\mathcal{L}_\Diamond) \quad \textit{and}$$

$$\mathcal{L} \subseteq \mathcal{R}(\mathcal{L}_{\delta-\Diamond}) \subseteq \mathcal{R}(\mathcal{L}_{\delta'-\Diamond}) \subseteq \mathcal{R}(\mathcal{L}_\Diamond).$$

In the following we want to restrict our investigations mainly to relationships between $\mathcal{L}$ and restoration classes $\mathcal{R}(\mathcal{L}_\odot)$.

Observation 4. $\{\{\Diamond^n : n \in \mathit{length}(L)\} : L \in \mathcal{L}\} \subseteq \mathcal{L}_\Diamond$ *and therefore*

$$\mathcal{R}(\mathcal{L}_\Diamond) = \{L : L \subseteq X^* \wedge \exists L' (L' \in \mathcal{L} \wedge L \sim_l L')\}.$$

Now let $L \in \mathcal{R}(\mathcal{L}_{k-\Diamond})$. Then there exist $L' \in \mathcal{L}$ and a set L'' of k-punctured words such that $L'' \nearrow L$ and $L'' \nearrow L'$, and there are two puncturings f and g such that $f(L) = g(L') = L''$. This means that for each $u \in L$ there exists $v \in L'$ such that $f(u) = g(v)$ is k-punctured, and for each $v \in L'$ there exists $u \in L$ with the same property. The words u and v have the same length and differ in at most k

positions. This gives cause for the following definition being based on the Hamming distance in coding theory [4].

Definition 7. For two words u and v of the same length let

$$h(u,v) =_{Df} |\{i : 1 \leq i \leq |u| \wedge u[i] \neq v[i]\}|.$$

For a natural number k, two words u and v are called to be *k-similar*, denoted by $u \widetilde{_{k-sim}} v$, if $|u| = |v|$ and $h(u,v) \leq k$.
For a positive rational number $\delta < 1$, u and v are called to be *δ-similar*, $u \widetilde{_{\delta-sim}} v$, if
$|u| = |v|$ and $\frac{h(u,v)}{|u|} \leq \delta$.
Two languages $L_1, L_2 \subseteq X^*$ are called to be *k-similar*, denoted by $L_1 \widetilde{_{k-sim}} L_2$, if $\forall u \exists v(u \in L_1 \rightarrow v \in L_2 \wedge u \widetilde{_{k-sim}} v) \wedge \forall v \exists u(v \in L_2 \rightarrow u \in L_1 \wedge u \widetilde{_{k-sim}} v)$.
$L_1 \widetilde{_{\delta-sim}} L_2$ is defined appropriately for $0 < \delta < 1$.

Observation 5. $\mathcal{R}(\mathcal{L}_{k-\Diamond}) = \{L : L \subseteq X^* \wedge \exists L'(L' \in \mathcal{L} \wedge L \widetilde{_{k-sim}} L')\}$ *for* $k \in \mathbb{N}$.

$$\mathcal{R}(\mathcal{L}_{\delta-\Diamond}) = \{L : L \subseteq X^* \wedge \exists L'(L' \in \mathcal{L} \wedge L \widetilde{_{\delta-sim}} L')\} \text{ for } 0 < \delta < 1.$$

From now on we restrict ourselves to the classes of the Chomsky hierarchy. Their definitions and basic properties are well-known from the literature (see, e.g., [5,7,12]). We'll use the following abbreviations.

Definition 8. REG, CF, CS and RE denote the classes of all regular, context-free, context-sensitive and recursively enumerable languages (over X), respectively. We call them *Chomsky classes*. LIN denotes the class of all linear languages.

We close this section by an observation regarding to the classes $\mathcal{L}^+_\odot$ and $\mathcal{L}^\cap_\odot$.
By $\odot$-*similarity* we mean the relations $\sim_l$, $\widetilde{_{k-sim}}$, and $\widetilde{_{\delta-sim}}$, respectively, if $\odot$ is equal to $\Diamond$ or to $k-\Diamond$ or to $\delta-\Diamond$, respectively.

Observation 6.
For each $\mathcal{L} \subseteq \mathcal{P}(X^*)$, $k \geq 1$, $0 < \delta < 1$, *and* $\odot \in \{\Diamond, k-\Diamond, \delta-\Diamond\}$:

1) $\mathcal{L}^\cap_\odot = \mathcal{L} \subset \mathcal{L}^+_\odot = \mathcal{L}_\odot$ *if and only if* $\mathcal{L}$ *is closed under* $\odot$*-similarity.*

2) $\mathcal{L}^\cap_\odot \subset \mathcal{L} \subset \mathcal{L}^+_\odot \subset \mathcal{L}_\odot$ *if and only if* $\mathcal{L}$ *is not closed under* $\odot$*-similarity and there*
exist $L \in \mathcal{L}_\odot$ *which has each restoration in* $\mathcal{L}$.
This is true for each class from Definition 8.

3) $\mathcal{L}^\cap_\odot = \emptyset \wedge \mathcal{L} = \mathcal{L}^+_\odot \subset \mathcal{L}_\odot$ *if and only if each* $L \in \mathcal{L}_\odot \setminus \mathcal{L}$ *has a restoration outside from* $\mathcal{L}$.

3 Helpful facts

In this section we summarize some basic properties which we shall use in the following sections. Their proofs are well-known and can be found in the standard literature (e.g., [5,7,12]).

Lemma 1. $REG \subset LIN \subset CF \subset CS \subset RE$.

Definition 9. For a language L, $\mathcal{T}(L) =_{Df} \{a^{|p|} : p \in L\}$ is the *tally projection* of L. If $\mathcal{L}$ is a class of languages, then $\mathcal{T}(\mathcal{L}) =_{Df} \{\mathcal{T}(L) : L \in \mathcal{L}\}$.

Lemma 2. *Each of the Chomsky classes and also LIN are closed under tally projection.*

Lemma 3. $\mathcal{T}(REG) = \mathcal{T}(LIN) = \mathcal{T}(CF) \subset \mathcal{T}(CS) \subset \mathcal{T}(RE)$. *Especially, every context-free language over a one-letter alphabet is regular.*

Lemma 4. *For every regular language L there exists a natural number m such that every $w \in L$ with $|w| > m$ is of the form $w_1w_2w_3$ where: $|w_1w_2| < m$, $w_2 \neq e$, and $w_1w_2^iw_3 \in L$ for all $i \in \mathbb{N}$.*

Lemma 5. *For every context-free language L there exists a natural number m such that every $w \in L$ with $|w| > m$ is of the form $w_1w_2w_3w_4w_5$ where: $|w_2w_3w_4| < m$, $w_2w_4 \neq e$, and $w_1w_2^iw_3w_4^iw_5 \in L$ for all $i \in \mathbb{N}$.*

4 Restoration of k-punctured classes

Because puncturing maintains the inclusionship between languages and classes, it follows from Lemma 1 and Observation 2:

Theorem 1. *For $k \in \mathbb{N}$, $0 < \delta < 1$ and $\odot \in \{\Diamond, k-\Diamond, \delta-\Diamond\}$:*
$REG_\odot \subset LIN_\odot \subset CF_\odot \subset CS_\odot \subset RE_\odot$.

Corollary 1. $\mathcal{R}(REG_\odot) \subseteq \mathcal{R}(LIN_\odot) \subseteq \mathcal{R}(CF_\odot) \subseteq \mathcal{R}(CS_\odot) \subseteq \mathcal{R}(RE_\odot)$.

Obviously, any language class is contained in each of its restoration classes, see Observation 3. Now we consider whether this inclusion is strict. First, consider k-puncturing for $k \in \mathbb{N}$.

Theorem 2. *Let k and k' be natural numbers such that $0 \leq k' < k$. Then there exist $L \in \mathcal{R}(REG_{k-\Diamond}) \setminus \mathcal{R}(RE_{k'-\Diamond})$.*

Using Observation 5 this means that there exist languages which are k-similar to regular languages but not k'-similar to any recursively enumerable language if $k' < k$.

Proof. Let T be such a set which is not recursively enumerable and $T \subseteq a^*$, and define $L =_{Df} \{pa^{2k} : p \in T\} \cup \{pb^{2k} : p \in a^* \setminus T\}$. Then $L \widetilde{_{k-sim}}\, a^*a^kb^k$. Because of $a^*a^kb^k \in REG$ and Observation 5, $L \in \mathcal{R}(REG_{k-\Diamond})$. Assume $L \in \mathcal{R}(RE_{k'-\Diamond})$. Then $L \widetilde{_{k'-sim}}\, S$ for some $S \in RE$, and each $w \in S$ must be k'-similar to some word from L and therefore it has the form pu where $|u| = 2k$ and either u has more than k letters a or less than k letters a. Enumerating S and considering only words having a suffix of length $2k$ with more than k letters a and outputting the appropriate remaining prefixes whereby all possibly occuring letters b are converted to a enumerates T. This contradicts the nonenumerability of T. □

It follows with Corollary 1 and Observation 3 that the restoration classes of each of the Chomsky classes create a strict hierarchy with respect to the maximal number of holes per word:

Corollary 2. *For* $\mathcal{L} \in \{REG, LIN, CF, CS, RE\}$ *and* $0 \leq k' < k$: $\mathcal{R}(\mathcal{L}_{k'-\Diamond}) \subset \mathcal{R}(\mathcal{L}_{k-\Diamond})$.

It is worth to mention also some more concrete languages which demonstrate the strict inclusion:

$\{a^nb^na^{2k} : n \in \mathbb{N}\} \cup \{a^mb^nb^{2k} : m, n \in \mathbb{N} \wedge m \neq n\} \in \mathcal{R}(REG_{k-\Diamond}) \setminus \mathcal{R}(REG_{k'-\Diamond})$ can be shown using Lemma 4,

$\{a^nb^na^{n+2k} : n \in \mathbb{N}\} \cup \{a^lb^ma^nb^{2k} : \neg(l = m = n)\} \in \mathcal{R}(REG_{k-\Diamond}) \setminus \mathcal{R}(CF_{k'-\Diamond})$ can be shown using Lemma 5.

Next we show that each context-free language is a restoration of a punctured regular language with unbounded number of holes but not for bounded number of holes. The first part of this statement is a corollary from the following more general theorem.

Theorem 3. *Let* $\mathcal{L}$ *and* $\mathcal{L}'$ *be two language classes which are closed under tally projection. Then* $\mathcal{L}' \subseteq \mathcal{R}(\mathcal{L}_\Diamond)$ *if and only if* $\mathcal{T}(\mathcal{L}') \subseteq \mathcal{L}$.

Proof. First, assume $\mathcal{T}(\mathcal{L}') \subseteq \mathcal{L}$ and $L' \in \mathcal{L}'$. Then $L' \in \mathcal{R}(\mathcal{L}_\Diamond)$ because of $L' \sim_l \mathcal{T}(L')$, $\mathcal{T}(L') \in \mathcal{L}$, and by Observation 4.
Now, assume $\mathcal{T}(\mathcal{L}') \not\subseteq \mathcal{L}$ and let $L' \in \mathcal{T}(\mathcal{L}') \setminus \mathcal{L}$. Then $L' \in \mathcal{L}'$. If $L' \in \mathcal{R}(\mathcal{L}_\Diamond)$ then, by Observation 4, $L' \sim_l L$ for some $L \in \mathcal{L}$. Then also $\mathcal{T}(L) \in \mathcal{L}$, but $\mathcal{T}(L) = L' \notin \mathcal{L}$. □

Corollary 3. $CF \subset \mathcal{R}(REG_\Diamond)$.

Proof. The inclusion follows immediately by Theorem 3 and Lemmas 2 and 3. The strict inclusion follows with Observation 3 and Lemma 1 from Theorem 2. □

Theorem 4. *$CF \not\subseteq \mathcal{R}(REG_{k-\Diamond})$ for any fixed $k \in \mathbb{N}$, even more :*
$LIN \not\subseteq \bigcup_{k=0}^{\infty} \mathcal{R}(REG_{k-\Diamond})$.

Proof. We consider $L' =_{Df} \{a^n b^n : n \in \mathbb{N}\} \in LIN$ and show that $L' \notin \mathcal{R}(REG_{k-\Diamond})$ for fixed $k \geq 0$. Let us assume the opposite. Then, by Observation 5, there exists $L \in REG$ with $L \widetilde{_{k-sim}} L'$.
For each $w \in L$ there exits $w' \in L'$ with $w \widetilde{_{k-sim}} w'$ and therefore $|w|_a \leq |w|_b + 2k$. For L there exists m according to Lemma 4. Let $w \in L$ such that $|w| > 2 \cdot (k+1) \cdot m$. By Lemma 4, w has the form $w_1 w_2 w_3$ where $|w_1 w_2| < m < \frac{|w|}{2(k+1)}$, $w_2 \neq e$, and $w_1 w_2^i w_3 \in L$ for each i.
Case 1). b occures in w_2. Then with $i = k+1$ we get a contradiction because of $|w_1 w_2^{k+1}| < (k+1) \cdot m < \frac{|w|}{2} \leq \frac{|w_1 w_2^{k+1} w_3|}{2}$ and therefore $|w_1 w_2^{k+1}|_b \leq k$ (there cannot be more than k letters b in the first half of a word which is k-similar to a word from L') but on the other hand $|w_2^{k+1}|_b \geq k+1$ because of b in w_2.
Case 2). $w_2 \in a^+$. Because of $w_1 w_2^i w_3 \in L$ it follows that
$|w_1 w_2^i w_3|_a = |w_1 w_3|_a + i \cdot |w_2| \leq |w_1 w_2^i w_3|_b + 2k = |w_1 w_3|_b + 2k$. For $i > 2k + |w_1 w_3|$ this yields to a contradiction which proves the theorem. □

Corollary 4. $\bigcup_{k=0}^{\infty} \mathcal{R}(REG_{k-\Diamond}) = \mathcal{R}(\bigcup_{k=0}^{\infty} REG_{k-\Diamond}) \subset \mathcal{R}(REG_{\Diamond})$.

In contrast to Corollary 3 appropriate results are not true higher up in the Chomsky hierarchy. This immediately follows from Theorem 3 by Lemma 3 (Remark that $\mathcal{T}(\mathcal{L}') \subseteq \mathcal{L}$ is equivalent to $\mathcal{T}(\mathcal{L}') \subseteq \mathcal{T}(\mathcal{L})$, if $\mathcal{L}$ is closed under tally projection.).

Corollary 5. $CS \not\subseteq \mathcal{R}(CF_{\Diamond})$, $\quad RE \not\subseteq \mathcal{R}(CS_{\Diamond})$.

5 Restoration of δ-punctured classes

Now we consider punctured languages where the number of holes per word is not bounded by a constant but the ratio of the number of holes per word to the length of the word is bounded. Besides our observations in Section 2, we stated first simple results for such classes in Theorem 1 and Corollary 1. An analog result to Theorem 2 we can only prove if $\delta' < \frac{\delta}{2}$. For $\delta' \geq \frac{\delta}{2}$ we get Theorem 6 below.

Theorem 5. *Let δ and δ' be rational numbers such that $0 < \delta < 1$ and $0 \leq \delta' < \frac{\delta}{2}$. Then there exist $L \in \mathcal{R}(REG_{\delta-\Diamond}) \setminus \mathcal{R}(RE_{\delta'-\Diamond})$.*

Proof. Let $\delta = \frac{r}{s}$ for natural numbers $r < s$, and let T be such a set which is not recursively enumerable and $T \subseteq a^+$. We define:
$L =_{Df} \{a^{ns} : a^n \in T\} \cup \{a^{n \cdot (s-r)} b^{nr} : a^n \notin T\}$. Then $L \widetilde{_{\delta-sim}} \{a^{ns} : n \in \mathbb{N}\}$ and therefore $L \in \mathcal{R}(REG_{\delta-\Diamond})$. Assume $L \widetilde{_{\delta'-sim}} S$ for some $S \in RE$. The suffix of length nr of any word $w \in S$ with $|w| = ns$ either has at most $\delta' \cdot |w| < \frac{\delta}{2}|w| = \frac{nr}{2}$ letters b or more than $\frac{nr}{2}$ letters b. Enumerating all words from S with a length ns for $n \in \mathbb{N}$

and less than the half letters b in their suffix of length nr, yields to an enumeration of T. □

Theorem 6. *Let δ and δ' be rational numbers such that $0 < \frac{\delta}{2} \leq \delta' < \delta < 1$ and $\delta' < \frac{1}{2}$. Then there exist $L \in \mathcal{R}(LIN_{\delta-\Diamond}) \setminus \mathcal{R}(RE_{\delta'-\Diamond})$.*

Proof. Let T and $\delta = \frac{r}{s}$ be the same as in the former proof, and let t be the minimum of $\{s-r, r\}$. We define:
$L =_{Df} \{a^{n\cdot(s-t)}b^{nt}a^{nr}b^{n\cdot(s-r)} : a^n \in T\} \cup \{b^{nr}a^{n\cdot(s-r)}b^{n\cdot(s-t)}a^{nt} : a^n \notin T\}$. Then $L \widetilde{\delta-sim} \{a^{ns}b^{ns} : n \in \mathbb{N}\}$ and therefore $L \in \mathcal{R}(LIN_{\delta-\Diamond})$. Assume $L \widetilde{\delta'-sim} S$ for some $S \in RE$. Then for each $w \in S$ with a length $2ns$, either $h(w, a^{n(s-t)}b^{nt}a^{nr}b^{n(s-r)}) < \min\{2nr, ns\}$ (if $a^n \in T$) or $h(w, b^{nr}a^{n(s-r)}b^{n(s-t)}a^{nt}) < \min\{2nr, ns\}$ (if $a^n \notin T$). (Both of them isn't possible because of $h(a^{n(s-t)}b^{nt}a^{nr}b^{n(s-r)}, b^{nr}a^{n(s-r)}b^{n(s-t)}a^{nt}) = \min\{4nr, 2ns\}$.) This yields to an enumeration of T. □

Corollary 6. *For $\mathcal{L} \in \{LIN, CF, CS, RE\}$ and $0 \leq \delta' < \delta < 1$ and $\delta' < \frac{1}{2}$ $\mathcal{R}(\mathcal{L}_{\delta'-\Diamond}) \subset \mathcal{R}(\mathcal{L}_{\delta-\Diamond})$, and further $\mathcal{R}(REG_{\delta'-\Diamond}) \subset \mathcal{R}(REG_{\delta-\Diamond})$ if $\delta' < \frac{\delta}{2}$.*

On the analogy of Theorem 4 we show

Theorem 7. $LIN \not\subseteq \bigcup_{0\leq\delta<\frac{1}{2}} \mathcal{R}(REG_{\delta-\Diamond})$.

Proof. We consider $L' =_{Df} \{a^nb^n : n \in \mathbb{N}\} \in LIN$ and assume $L' \widetilde{\delta-sim} L$ for some fixed δ with $0 \leq \delta < \frac{1}{2}$ and for some $L \in REG$. Again by Lemma 4, every sufficiently long $w \in L$ has the form $w = w_1w_2w_3$ where $w_2 \neq e$ and $z_i \in L$ for each i, if we define $z_i =_{Df} w_1w_2^iw_3$. Because of $L' \sim_l L$, w_2 has an even length $l \geq 2$, and $z_i \widetilde{\delta-sim} z'_i$ for a uniquely determined $z'_i = a^{n_i}b^{n_i} \in L'$. Choose i such that $|w_1| < n_i$ and $|w_3| < n_i$. This means, the centre of the word z_i is within w_2^i. Then for each $j \in \mathbb{N}$, the centre of z_{i+2j} is by $j\cdot l = j\cdot|w_2|$ positions to the right from the centre of z_i. The word left from the centre of z_{i+2j} must be similar to a^{n_i+jl}, and the word right from this centre must be similar to b^{n_i+jl}. Therefore $h(z_{i+2j}, z'_{i+2j}) = h(z_i, z'_i) + j\cdot|w_2|_b + j\cdot|w_2|_a = h(z_i, z'_i) + j\cdot|w_2| = h(z_i, z'_i) + jl$. We have $|z_{i+2j}| = 2n_i + 2jl$, and therefore $\lim_{j\to\infty} \frac{h(z_{i+2j}, z'_{i+2j})}{|z_{i+2j}|} = \frac{1}{2} > \delta$. This contradicts to $L' \widetilde{\delta-sim} L$. □

Now, Theorem 4 and Corollary 4 appear as consequences from Theorem 7.

Whether $LIN \not\subseteq \mathcal{R}(REG_{\delta-\Diamond})$ is true or not for $\delta \geq \frac{1}{2}$ remains open. We assume that for $\frac{1}{2} \leq \delta < 1$, $LIN \subseteq \mathcal{R}(REG_{\delta-\Diamond})$ but $CF \not\subseteq \mathcal{R}(REG_{\delta-\Diamond})$.

One level higher, $CS \not\subseteq \bigcup_{0\leq\delta<1} \mathcal{R}(REG_{\delta-\Diamond})$ is true because of Corollary 5.

Restricting to slender context-free languages the situation again becomes clear.

A language L is called to be *slender* if there is a natural number k such that

$$|\{w : w \in L \wedge |w| = n\}| \leq k \quad \text{for every } n \geq 0.$$

These languages are interesting from the theoretical point of view (see, e.g., [8-11]), but they also have important applications in cryptography [1].
If $\mathcal{L}$ is a class of languages, then $SL_{\mathcal{L}}$ denotes the *class of slender languages in* $\mathcal{L}$ [10].

Theorem 8. $SL_{LIN} = SL_{CF} \subset \mathcal{R}(REG_{\frac{1}{2}-\Diamond})$.

Proof. Ilie [8] and Raz [11] have shown that a context-free language is slender if and only if it is a *union of paired loops*. That means, for some $m \geq 1$ there exist words u_i, v_i, w_i, x_i, y_i for $i = 1, \ldots, m$ such that $L = \bigcup_{i=1}^{m} \{u_i v_i^n w_i x_i^n y_i : n \geq 0\}$. As a consequence of this result, $SL_{LIN} = SL_{CF}$ [8]. Also, slenderness of a context-free language is decidable, and a given slender context-free language can be effectively written as a union of paired loops [9]. Now let $L \in SL_{CF}$ be a union of paired loops, i.e., $L = \bigcup_{i=1}^{m} L_i$, $L_i = \{u_i v_i^n w_i x_i^n y_i : n \geq 0\}$ for $i = 1, \ldots, m$. It is easy to see that each paired loop L_i is $\frac{1}{2}$-similar to a regular language, and so is L: Let $L_i = \{p_n : n \in \mathbb{N}\}$, where $p_n =_{Df} u_i v_i^n w_i x_i^n y_i$. For each $\nu \in X(= \{a, b\})$, $|p_n|_\nu = |u_i w_i y_i|_\nu + n \cdot |v_i x_i|_\nu$. Thus there exists $n_0 \in \mathbb{N}$ such that either for each $n \geq n_0$, $|p_n|_a \geq |p_n|_b$, or for each $n \geq n_0$, $|p_n|_b > |p_n|_a$. Let $L_i' =_{Df} \{p_n : n < n_0\} \cup \{a^{|p_n|} : n \geq n_0\}$ if $|p_n|_a \geq |p_n|_b$ for $n \geq n_0$, and $L_i' =_{Df} \{p_n : n < n_0\} \cup \{b^{|p_n|} : n \geq n_0\}$ if $|p_n|_b > |p_n|_a$ for $n \geq n_0$. Then $L_i \underset{\frac{1}{2}-sim}{\sim} L_i'$ and $L_i' \in REG$. The strict inclusion is true because of there exist non-slender languages in $\mathcal{R}(REG_{\frac{1}{2}-\Diamond})$. □

Again, in contrast to Theorem 8 we have by Theorem 3 (classes of slender languages are closed under tally projection):

Corollary 7. $SL_{CS} \not\subseteq \mathcal{R}(CF_\Diamond)$, $\quad SL_{RE} \not\subseteq \mathcal{R}(CS_\Diamond)$.

6 Conclusion

Starting from processes in molecular biology we came to the notions of punctured languages and their restoration. Apart from elementary relationships between various different classes of punctured languages and their extents we restricted our investigations mainly to relationships between language classes $\mathcal{L}$ and their restoration classes after puncturing. A restoration class $\mathcal{R}(\mathcal{L}_\odot)$ may be interpreted in two different ways: as the class of all languages which may be restored from languages from $\mathcal{L}$ after $\odot$-puncturing or as the class of all languages which are similar to languages from $\mathcal{L}$ in a $\odot$-adequate sense. We have seen that, for each class $\mathcal{L}$ from the Chomsky hierarchy, the classes of languages which are similar to languages from $\mathcal{L}$ create a strict hierarchy with respect to the similarities determined by the number of differences between words and also with respect to the similarities determined by the ratio of the number of differences per word to the length of the word. For regular

languages, the latter is true if the gap between the similarities is great enough. Further we have seen that there exist linear languages which are not k-similar to any regular language for any $k \in \mathbb{N}$ and which are not δ-similar to any regular language for any $\delta < \frac{1}{2}$. Whether such languages exist for $\frac{1}{2} \leq \delta < 1$ remains open. If they exist then they must be non-slender.

Acknowledgment. I am grateful to Peter Leupold for our discussions which aroused my interest in partial words and for supplying me some basic material, and to Sándor Horváth who found a mistake in my former proof of Theorem 6.

References

[1] M.Andrasiu, J.Dassow, G.Păun, A.Salomaa, *Language-theoretic problems arising from Richelieu cryptosystems*, Theoretical Computer Science 116 (1993), 339–357.

[2] J.Berstel, L.Boasson, *Partial words and a theorem of Fine and Wilf*, Theoretical Computer Science 218 (1999), 135–141.

[3] F.Blanchet-Sadri, D.K.Luhmann, *Conjugacy on partial words*, Theoretical Computer Science 289 (2002), 297–312.

[4] R.W.Hamming, *Error detecting and error correcting codes*, Bell System Techn. Journ. 29 (1950), 147–160.

[5] M.A.Harrison, *Introduction to formal language theory*, Addison-Wesley, Reading (Mass.), 1978.

[6] T.Head, G.Păun, D.Pixton, *Language theory and molecular genetics*, in G.Rozenberg, A.Salomaa (Eds.), *Handbook of formal languages, Vol. 2*, Springer-Verlag, Berlin-Heidelberg, 1997, 295–360.

[7] J.E.Hopcroft, J.D.Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading (Mass.), 1979.

[8] L.Ilie, *On a conjecture about slender context-free languages*, Theoretical Computer Science 132 (1994), 427–434.

[9] L.Ilie, *On lengths of words in context-free languages*, Theoretical Computer Science 242 (2000), 327–359.

[10] G.Păun, A.Salomaa, *Thin and slender languages*, Discrete Appl. Math. 61 (1995), 257–270.

[11] D.Raz, *Length considerations in context-free languages*, Theoretical Computer Science 183 (1997), 21–32.

[12] G.Rozenberg, A.Salomaa (Eds.), *Handbook of formal languages, Vol. 1*, Springer-Verlag, Berlin-Heidelberg, 1997, Chapter 1 and Chapter 2.

A Normal Form for Regular Expressions

Benedek Nagy

Department of Computer Science, Institute of Informatics, University of Debrecen,
Debrecen, Hungary
Research Group on Mathematical Linguistics, Rovira i Virgili University,
Tarragona, Spain
nbenedek@inf.unideb.hu

Abstract. The normal forms play important roles in many branches of computer science. In this paper, we present a normal form for regular expressions. We analyze a subclass of the regular languages, namely the union-free regular languages. These languages can given by regular expressions without the operation union. In a union-free language the words look like each other, each word contains the shortest word of the language in scattered way. We show that each regular language can be a finite union of union-free languages. This decomposition is not unique, but some of them contain the minimum number of union-free languages. Therefore the union-complexity of a regular language can be defined. Using this decomposition one can write the regular expressions to normal form.

Keywords: normal form, union-free languages, regular expressions, regular languages

1 Introduction

The normal forms of expressions are useful and widely used, for example in logic. In the normal form we have an ordering of the operations of the language. Using tree form of the expression we can say that all quantors are in a higher level than all of the Boolean operators (prefix form for first order logical expressions). Analogically we can use disjunctive (conjunctive) normal forms for Boolean expressions in which all the disjunctions (conjunctions) are higher level than the conjunctions (disjunctions), and the negations can be only the nodes which are leaves but one.

In formal language theory the normal forms are usually used for special forms of grammars (such as Chomsky-, Greinbach-, Pentonnen-, Révész- etc. normal forms). In this paper we detail the regular languages and their description by not grammars, but regular expressions. The regular languages are the most common, well-known and well-applicable languages. They are the simplest languages in the Chomsky-hierarchy. They can be described by regular expressions. In this paper we will consider a special subclass of the regular languages. A regular language can contain words which are completely different. This can happen if the regular expression of the language contains the operation union (in regular expression we use $+$). For example $a + b^*$. The operation union ($+$) is very powerful, when we allow infinite sums the expressions can describe all the type 0 (i.e. the whole recursive enumerable class) languages in Chomsky-hierarchy. We will investigate the languages which can be described regular expression without $+$. The words of a language of this union-free family have the same "shape". In [2] the algebraic properties of union-free languages

were examined, in this paper we use another approach. We use these, union-free regular languages to decomposition of regular languages. In this paper, based on this decomposition, we will give a normal form for regular expressions, in which all of the unions are higher level then the concatenations and the Kleene-stars.

The structure of the paper is as follows. In Section 2 we describe some normal forms used in logic. After this, in the next section we define and analyze the properties of union-free languages. In Section 4 we show how the regular languages can be decomposed into finite union of union-free languages, the result of this decomposition is a kind of normal form. Some properties of this normal form will also be analyzed. In the last section we summarize our results and we show some interesting open questions.

2 Normal forms in logic

As we mentioned before the normal forms of expressions are very important in both of theoretical and practical computer science. In this section we recall normal forms of logic. We show the normal forms (prenex, disjunctive (conjunctive) normal forms) and the way to get them. The tree representation of expressions are also widely known and used. They show the structure of the expressions.

The Boolean variables and their negation are literals. A logical formula is called an elementary conjunction (clause), if it is a conjunction of literals. The disjunction of elementary conjunctions is a disjunctive normal form. Similarly, an elementary disjunction is a disjunction of literals. A conjunction of elementary disjunctions is a conjunctive normal form.

Well-known that for each Boolean formula there is an equivalent formula in conjunctive normal form and in disjunctive normal form, too. Moreover these normal forms are not unique, i.e. usually there are more formulae in disjunctive (conjunctive) normal form for a given one.

There are more ways to get these normal forms starting with a formula. One way is to use the truth-table of the formula, and based on these values one can construct a formula which is equivalent to the original one. The second way is to use logical equivalences. Replacing a part (a subformula) of a formula by an equivalent part the result is equivalent to the original. These equivalences can be found in any textbook on mathematical logic (see for instance [1]). Some examples: $A \supset B \equiv \neg A \vee B$, $\neg(A \vee B) \equiv \neg A \wedge \neg B$, $\neg\neg A \equiv A$ etc.

For first order logic, the so-called prenex form is used as normal form. A formula is in prenex form if all the quantors are in the beginning of the formula and their effect go to the whole formula. Each formula has an equivalent formula in prenex form. There is an algorithm to get equivalent prenex form for a given formula using logical equivalences. These equivalences move the quantors to higher level of the expression-tree. For instance, $\neg\exists x A(x) \equiv \forall x \neg A(x)$, $\neg\forall x A(x) \equiv \exists x \neg A(x)$. Using them the negation and the quantor change their place in the tree (moreover the quantor changes to the opposite quantor). At $A \wedge \exists x B(x) \equiv \exists x(A \wedge B(x))$ the quantor moves higher than the conjunction etc.

These normal forms are widely used and one can understand how they works and why they are important. In the tree-form of the expressions the normal forms are special trees. For example, in the tree of a prenex formula all quantors are in higher level than the other operators and literals. After this sideview we are going back to formal language theory, and show a normal form of regular expressions.

3 The union-free languages

In this section we recall the definitions of regular expression and regular languages [3, 4]. We define the union-free languages as well. We start this section with the basic definitions.

3.1 Basic definitions

In the next definition we use the well-known regular operators, such as union, concatenation and Kleene-star ($+, \cdot, *$ respectively).

Definition 1. *The finite expressions are regular expressions using the letters of the alphabet and symbols $+, \cdot, *$ in the following way.*
The letters of the alphabet (with the empty word (signed by λ) and the empty set (empty language)) are regular expressions.
If r, q are regular expressions then $r + q, r \cdot q$ and r^* are regular expressions as well. Note, that the brackets can be used in regular expressions to show the order of the operations $(+, \cdot, *)$. If it is obvious, then we omit the sign of the operator concatenation $(\cdot)$, as usual.

We call a language regular if there is a regular expression which describes it.

We call a regular expression union-free (regular) expression, if only the operators $\cdot, *$ are used in it. Consequently a language, which can be defined by a union-free expression is a union-free (regular) language.

Note, that another important and well-examined class of the regular languages the finite languages. Each of them contains only finite number of words. They can be described by the (strongly) star-free regular expressions, in which only the concatenation and the union are the allowed operations and the Kleene-star is not used. In this paper we will use the star-freeness in this strong sense (in the literature other (set-theoretical) operations such as intersection and complement is allowed to use at the (extended) star-free expressions).

Each regular expression can be written in a tree form, in which exactly the leaves are the terminal symbols of the language (λ is also allowed), and at the other nodes are the operations.

Note, that the operation Kleene-plus is used sometimes. It is an abbreviation: $r^+ = r \cdot r^*$.

Example 1. Let $V = \{a_1, a_2, \ldots, a_n\}$ be a finite alphabet. The language $V^* = (a_1 + a_2 + \ldots + a_n)^*$ is union-free ($V^* = (a_1^* a_2^* \ldots a_n^*)^*$). The language V^+ is union-free if and only if V is singular (and the language V is union free only in this case also).

The previous example shows, that the Kleene-star and Kleene-plus have different properties.

Example 2. Let $V = \{a, b, c\}$. The language containing the words $bab, baba, babc^*, ...$ given by the regular expression $bab(a + c^*)^*$ is union-free. The union-free expression $bab(a^*c^*)^*$ describes it as well. In Figure 1 the tree forms are presented for both regular expressions.

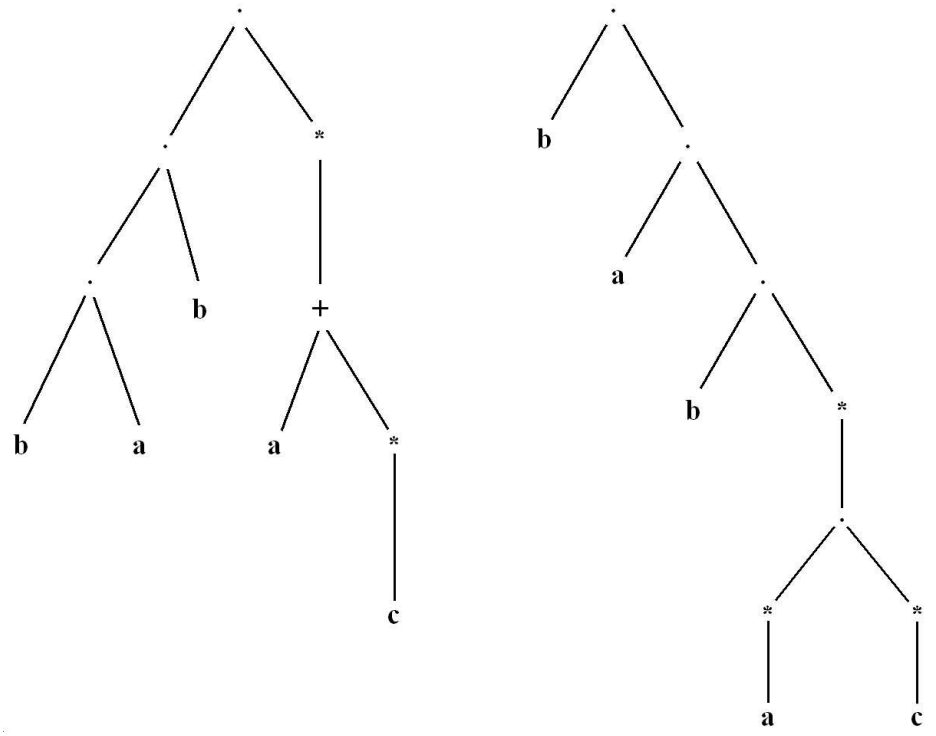

Fig. 1. Examples for regular expressions in tree form

As Figure 1 shows one can express the regular expressions by trees.

3.2 Some properties of union-free languages

Now we detail some properties of the languages defined above.

Lemma 1. *There are infinitely many non-comparable union-free languages.*

Proof. All the languages containing exactly 1 word are union-free languages. There are infinitely many of them. □

Lemma 2. *A union-free language is infinite if and only if there is no star-free regular expression to describe it.*

Proof. Without $*$ the languages contains only 1 word. □

Corollary 1. *A union-free language is either infinite or contains only one word.*

Lemma 3. *Let L be an infinite union-free language. There are infinitely many sequences of union-free languages starting with L, in which each language is a proper subset of the previous one.*

Proof. First we construct an infinite union-free language L_1, which is strictly included in L. According to lemma 2 there is a Kleene-star in the union-free regular expression of L. Let us change this star operation to Kleene-plus, i.e. substitute the part $(r)^*$ of the original expression by $(r)(r)^*$. Clearly, by this modification we get a description of a new infinite union-free language, which is a proper subset of L. And now, the procedure can be continued for the language L_1, which is also an infinite union-free one. Let $L_0 = L$. It is evident that for any infinite subsequence of $L = L_0, L_1, ..., L_i, ...$ starting with L hold the conditions of the lemma. Therefore their number is infinite. □

Corollary 2. *There is no smallest infinite union-free language. (For each infinite union-free language L there is a union-free language which is a proper subset of L.)*

Now, we describe some other interesting properties of the union-free languages.

We have the following useful lemma, which can be useful to decide if a language cannot be union-free.

Lemma 4. *The shortest word of a union-free language L is unique.*

Proof. Trivial, it is the word obtained from the regular expression substituting every part r^* (where r is a regular expression such that r^* is a part of the expression) by λ. □

One of the simple similarity facts of the words of a union-free language is the following.

Proposition 1. *In a union-free language each word contains the shortest word of the language in scattered way.*

Proof. It is obvious. □

Let L be a union-free language. Note, that the $\lambda \in L$ (i.e. the shortest word of the language is the empty word) if and only if every terminal is under a Kleene-star in the tree of the regular expression.

Now, we are in the position to claim the theorem about the closure properties of union-free languages.

Theorem 1. *The union-free language-family is closed under the following operations: concatenation, Kleene-star, substitution by union-free expressions.*

It does not closed under the following operations: union (of course), complement, intersection, substitution by regular language.

Proof. The cases of concatenation and Kleene-star: trivial by using regular expression of the definition of union-free languages.

The substitution by union-free expression is also trivial.

Union: consider the following two languages: $\{a\}, \{b\}$.

Complement: assume the language $\{a^*\}$ over the alphabet $\{a, b, c\}$. Then the complement is $V^*bV^* + V^*cV^*$, which has two shortest words, namely b and c. (Or for binary alphabet, the complementer of the language defined by $(aa)^*$ has the following two shortest words: a, b.)

Intersection: V^*aV^*, V^*bV^* (we cannot do without union, because the letter a and b can be in two kind of order in the words) The intersection language has two shortest words: ab, ba.

Consequence: it is not closed under substitution by regular expression. □

As a consequence of the previous theorem we have:

Corollary 3. *The union-free language-family is closed under Kleene +, and for any fixed natural number n it closed under the n-th power.*

4 Decomposition of regular languages into union-free languages

In this part we show a normal form of regular expressions using union-free expressions.

For the sake of simplicity assume that, the given regular expression is fully bracketed (i.e. its tree is a binary tree, which means that all union and concatenation has exactly two components, while the Kleene-stars have only one).

The following equivalences of the regular expressions will be useful to make the decomposition into union-free languages.

(Considering all the possibilities, in which situation a union can be, we get the following equivalences.)

Proposition 2. *The following equivalence holds. (See Figure 2 as well.)*

$$(1) \qquad (x + y)^* \text{ can be written in the form } (x^*y^*)^*$$

Where x and y are arbitrary regular expressions.

Using this fact a union-sign (+) can be deleted from the expression using Kleene-star operators, when the union was under a Kleene-star in the expression-tree.

Proposition 3. *The following equivalences can be used in regular expressions to move union to higher level than the concatenation.*

(2) $(x + y)z$ *can be written in the form* $(xz + yz)$,

(3) $x(z + v)$ *can be written in the form* $(xz + xv)$,

(4) $(x + y)(z + v)$ *can be written in the form* $(xz + xv) + (yz + yv)$.

(Where x, y, z and v are arbitrary regular expressions.)

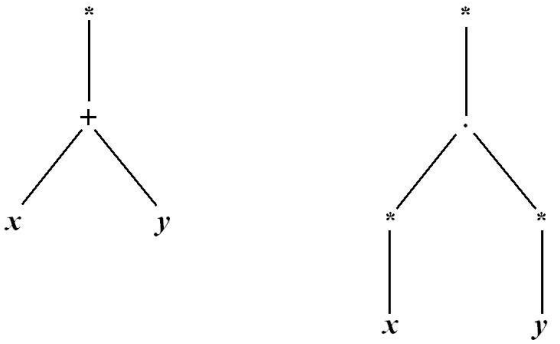

Fig. 2. A possible rewriting of a regular expressions to a union-free form

Using the above equivalences the union can be substituted or moved upward in the tree of the regular expression.

The following algorithm gives us the result.
If there are no other operation in a higher level than the union we are finished the construction. If there is a union which is immediately under a Kleene-star than using (1) we can erase it from that part of the regular expression. If there is a union which is under a concatenation then we can use one of the equivalences (2)-(4) according to the places and numbers of unions. (2) is useful if in the left component of the concatenation is a union, (3) is used in the case of the right side is a union. Finally, (4) is useful if in both sides of the concatenation are the operation union used.
This algorithm can translate arbitrary regular expression to a special form defined below.

In the following definition we use the regular expressions, whose tree has union only at its root (allowing non-binary unions).

Definition 2. *A regular expression is in normal form if it is finite union of union-free expressions.*

Now, using the definition of normal form of regular expressions, and using the above construction we get the following theorem.

Theorem 2. *For each regular expression there is an equivalent one, which is in normal form.*

Proof. Trivial, by the previous Propositions and algorithm. □

A decomposition $L = \bigcup_{i=1}^{n} L_i$ is proper, if there are no languages L_j such that $L_j \subseteq \bigcup_{i=1}^{j-1} L_i \cup \bigcup_{i=j+1}^{n} L_i$. A normal form is called proper normal form if it means a proper decomposition.

Now, we define the minimal decomposition, and the union-complexity of the languages.

Definition 3. *$L = \bigcup_{i=1}^{n} L_i$ is a minimal decomposition of the language L if each L_i is a union-free language and there is no $m < n$ such that $L = \bigcup_{i=1}^{m} L_i$, where each L_i is union-free.*

n is called the union-complexity of language L.

Note that every minimal decomposition is a proper decomposition.

Also note that the union-complexity of a language is 1 iff it is union-free, and finite iff it is regular. Each recursively enumerable language, which is non-regular has infinite union-complexity. For all finite languages the union-complexity is exactly the cardinality of the language.

Lemma 5. *If the number of the union-free languages in a minimal decomposition n for an infinite language L, then there are some decompositions for each $m > n$.*

Proof. Adding a singular language to the union which is subset of L one can increase the number of union up to infinity. □

Now we are dealing with special minimal decompositions. The minimal decomposition of a regular language using maximal union-free languages (there is no L_i' such that $L_i' \supset L_i$, and replacing L_i by L_i' the union language L is the same as before) is not unique.

An example is as follows. Using the binary alphabet $\{a, b\}$, let the analyzed regular language contain all words that do not contain bb.

Clearly, this language has union complexity 2. Moreover there are two minimal decompositions using maximal union-free languages (with regular expressions they are):

$$((ba)^*a^*)^* + ((ba)^*a^*)^*b((ab)^*a^*)^*,$$

and

$$((ab)^*a^*)^* + ((ba)^*a^*)^*b((ab)^*a^*)^*.$$

Lemma 6. *For using minimal decompositions we need all the infinite number of union-free languages.*

Proof. Let L be a union-free language. Since the union-complexity of L is 1, to describe L, we need exactly one language, but this is L. □

According to corollary 2 there is no finite subset of union-free languages which enough to describe all infinite union-free languages.

Let $\mathbf{L}_n$ be the family of languages which can be written as union of n union-free languages.

Theorem 3. *The families $\boldsymbol{L}_n$ and $\boldsymbol{L}_m$ in the following relation:*

$$\boldsymbol{L}_n \supset \boldsymbol{L}_m \text{ iff } n > m.$$

Proof. The inclusion is trivial from the previous lemmas, for strict inclusion consider the finite languages contains exactly n words. □

Using the equivalence of proposition 2 the union can be removed under a Kleene-star operation. Moreover we have the following theorem about the relation between regular languages defined by regular expressions and the union-free ones.

Theorem 4. *Let r be a regular expression. If all operations union are under some Kleene-star operations in the tree form of r, then r defines a union-free regular language, and therefore the union-complexity of this language is 1.*

Proof. Using the equivalences (2-4) among regular expressions one can move the operations union above the concatenations in the tree of the regular expression. Using these equivalences the operations union move up to the level immediately below a Kleene-star. With the equivalence (1) the union can be removed from that level. □

As a special consequence of the previous facts that for each regular language L the language L^* is a union free regular one.

5 Conclusions, further remarks

There are some very useful normal forms in computer science, and specially in formal language theory as well. A grammar is in normal form when its rules have only some special forms. In this paper we investigated a normal form for the regular expressions.

In regular expression the concatenation, Kleene-star and union are used. Without star the finite languages can be described.

In this paper we analyzed another subclass of the regular languages. The properties of the union-free regular languages were described, these languages are defined by union-free regular expressions. (Regular expressions without concatenation are useless.)

We have the following open questions. Is there any possible restriction for the used union-free languages to obtain a unique minimal decomposition. (As we have shown the minimal decomposition of a regular language using maximal union-free languages is not unique.)

Is there any useful (not too complex) algorithm to calculate the union-complexity or a proper decomposition of a given regular language? (We have a lower bound as the number of the shortest words of the language.)

What is the relation between the well-examined star-complexity, (generalized) star-height and the union-complexity of a language? It is an interesting problem, to analyze what language class can be described by the union-free expressions allowing intersection or complement. (In literature the definition of star-free languages usually allow to use these set-theoretical operations among languages.) It is easy to show that allowing both of them, via De Morgan law one get the whole regular class.

6 Acknowledgements

The author wishes to thank the referees for their valuable comments.

This research is supported by a grant from the International Visegrad Fund.

References

1. John Lane Bell and Moshé Machover: *A course in mathematical logic*, North-Holland Publishing Company, Amsterdam, 1977.
2. Siniša Crvenković, Igor Dolinka and Zoltán Ésik, *On equations for union-free regular languages*, Inform. and Comput. **164** (2001), no. 1, 152–172.
3. J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Publishing Company, Reading MA, 1979.
4. Sheng Yu, *Regular languages*, In: *Handbook of formal languages* (eds: G. Rozenberg, A. Salomaa, 3 volumes), Springer-Verlag, Berlin, 1997.

Self–Reproduction by Self–Assembly and Fission*

Jiří Wiedermann

Institute of Computer Science, Academy of Sciences of the Czech republic,
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz

Abstract. We introduce so–called biomata which represent a novel approach to the construction of self–reproducing automata within the automata theory. The design of our automata has been motivated by the ideas of cellular biology on the origin of life. Unlike the von Neumann's model our model replicates by fission and need not give much attention to the exact guiding of its own assemblage; rather, this process relies on self–assembly abilities of the respective parts produced by the biomaton from input objects not possessing such quality. The model represents an interesting fusion of computational and self–organizational processes. We believe that by capturing the basic aspects of the assumed origin of real life our modelling leads to a conceptually simpler and hence more plausible scenario of natural self–reproduction than the previous attempts did.

1 Introduction

In late 1940s, when John von Neumann started his quest for a logical, rather than material, basis of biological self–reproduction, he first proposed a mechanistic model. It consisted of a "robot" operating in a sea of its own spare parts. The robot had some elementary functions for moving around, identifying and collecting the required parts and assembling them together and possessed a tape with instructions for building a copy of itself by making use of these elementary functions. After constructing a replica of itself, the robot finally copied its instruction tape and inserted it into the replicated robot which could then start the same activities. By this design, it is generally agreed that von Neumann discovered the basic principles for the process of self–reproduction. Namely, there had to be a program, instruction sequence to be used in two different ways: (1) to be interpreted as instructions for constructing an offspring, and (2) to be copied passively, without being interpreted. Quite understandingly von Neumann was not able to construct a working model of his mechanistic self–reproducing automaton which would represent a convincing proof of soundness of his design idea. However, in 1953, following Stanislaw Ulam's vision of cellular automata, he invented a cellular automaton implementation of his mechanistic model. His cellular "robot" made use of a cellular automaton with 29 states per cell and consisted of approximatively 200 000 cells [7]. By this the topic of self–reproduction entered the field of the automata theory and it seems that until now nobody has questioned the uniqueness of von Neumann's scenario of self–reproduction in this field. There seems to be no formal computational model of self–reproduction based on a different scenario than the one mentioned above.

In this paper we suggest a different scenario of self–reproduction. Our model reflects the ideas of theoretical biology on the origin of life. According to these

* This research was partially supported by grant No. 1ET100300419 within the National Research Program "Information Society".

ideas life emerges in the form of protocells from the union of two fundamentally different kinds of replicating systems: an information genom and a membrane in which it resides (cf. [6]). Thus, replication lies in the heart of both systems. Roughly speaking, a genom controls its own replication and production and properties of parts for building the membrane which itself is a spontaneously replicating entity. The membrane shields the genom from the environment. This is enough to give rise to a self-replicating system. Addition of randomness into the genom replicating process leads to darwinian evolution of the whole system, and to so–called minimal life systems, but this question is outside the scope of the present paper (cf. [8] for an attempt to model minimal life).

In our setting, the basic functional aspects of protocells are modelled by so–called biomata. A biomaton consists of a so–called Turing computational field and its encapsulating membrane. By the activity of the Turing field certain input objects that permeate the membrane from outside are transformed into new objects with self–assembly properties; the input objects alone do not possess such properties. The membrane "grows" by incorporating new objects into the already existing membrane. All computations within the Turing field are controlled by a special object representing a finite state program playing the role of a genom. The genom itself can become a subject of the processing in the Turing field and indeed, inside the membrane its copy can be produced from suitable input objects. Providing that at that time the encompassing membrane doubles its volume, by definition it will split by fission into two parts each containing a copy of the original genom: the biomaton will reproduce itself. Even from the above rough sketch of biomata it is obvious that they combine computational processes with self–organizational ones. In our model the fission represents a non–computational process whose action and result are merely postulated. In real life a membrane fission is a consequence of the physical laws acting upon the membrane. Neither the self–assembly processes nor the membrane fission are under the computational control of the genom. In a sense, the respective non–computational mechanisms evoke the idea of an oracle that similarly as in the case of Turing machines is used for achieving the effects that principally cannot be obtained in a pure computational way by a device itself.

We believe that the main contribution of our paper lies in pointing to a new research direction within the automata theory that aims towards the design and exploitation of a formal model of self–replicating automata which are not based on classical computational mechanisms as the von Neumann's models were. While for biologists this model represents an abstraction of a protocell, within automata theory it formalizes an alternative scenario of self–replication that is closer to reality than the previous models. The mathematical theory of self–assembly is an emerging research field (cf. [1]) and our framework opens new avenues for its further development.

The paper consists of 4 sections. The first section contains the introduction. The biomaton itself is described in Section 2 in two subsections. The first subsection introduces the so–called multitransducer that represents the computational analog of a genom and in fact defines the Turing computational field. The second subsection explains how the multitransducer must be designed in order to produce its encap-

sulating membrane, how its genom gets replicated, and finally how the fission of the membrane is achieved. A multitransducer operating in this way gives rise to a biomaton. Section 3 discusses the previous achievements from the viewpoint of the automata theory with regard to cellular automata, P–systems, evolution theory and artificial life. The closing fourth section contains the summary of the achievements.

The paper describes the research in progress and so far neither the model nor the formalism and the terminology are definitive; similarly, the results and their interpretation only start to emerge. Some preliminary ideas related to the concept of a multitransducer in the context of minimal life have been presented in a workshop on membrane computing [8].

2 The Biomaton

Interactive finite multitransducer First, we will concentrate on the information processing aspects of our model. For that purpose we will make use of modified finite automata. We will use them in the mode of transducers (or as Mealy automata) — i.e., as automata processing multisets of the finite input strings of symbols and producing similar strings of output symbols. Moreover, we will consider a so–called *multitransducer* which is a multiset of transducers of finitely many types that all work asynchronously. Even though the description of a multitransducer, that is, of all types of automata together, is finite, the cardinality of their multiset can be arbitrarily large. This cardinality varies with time and depends on the number of strings that are available for processing at each time — see the description of the multitransducer's activity in the sequel. Thus, from the computational viewpoint a multitransducer is a highly parallel information processing device.

Now we will give a formal definition of a multitransducer for the simplest case when each automaton reads its inputs via a single input port. Then we will describe the way the machine works.

Definition 1. *An interactive asynchronous finite multitransducer with single–input ports is the six–tuple* $T = (I, O, S, B, F, \delta)$*, where*

- I *and* O *are finite alphabets of symbols,* I *is the input and* O *is the output alphabet;*
- S *is a finite alphabet of states;*
- $B \subseteq S$ *is the subset of initially active states;*
- $F \subseteq S$ *is the subset of final states;*
- δ *is the transition function of form* $I \times S \to S \times O \times S \times \{0, 1\} \times \mathbf{N}$ *which for each* $v \in I$ *read (and "consumed") at the input port of some automaton and each state* $s \in S$ *assigns a new state* $r \in S$*, sends* $w \in O$ *at the input port, and sets the activation value of state* $q \in S$ *to either 0 or 1; here 0 denotes non–initial (passive) and 1 initial (active) state; this is formally written as* $\delta(v, s) = (r, w, q, 0, t)$ *or* $\delta(v, s) = (r, w, q, 1, t)$*, respectively;* t *is the speed parameter saying that it takes* $t \in \mathbf{N}$ *time units to realize the transition at hand.*

In the multitransducer each type of the Mealy automaton is described by its own transition function of form as given in Definition 1. We assume that the multitransducer finds itself in an environment consisting of a multiset of strings. Also this

multiset is not given beforehand, it can change over time, that is, the multiplicity of the same strings in it can vary. A multitransducer operates by systematically and repeatedly transforming strings into other strings. The input strings are read sequentially, symbol by symbol by automata via their input ports and the output strings are produced in a similar way at their output ports. The automata work in an asynchronous manner. We assume that each automaton has its own clock. For simplicity we also suppose that in all automata the duration of one unit of time is the same, however, the clocks are not synchronized. Since there is no notion of global time it is not possible to define a "configuration of the system" at a given time. By allowing more than one input port each automaton can be designed so as to be able to process several strings in parallel, similarly as classical multihead automata. In order that the operation of a multitransducer can work smoothly we assume that each automaton reads the strings in a selective way, that is, it has a specific ability to find this string in the environment for the processing that is programmed for, as long as such a string exists in the environment. This property can also be seen as a property of the environment — it is as though the environment attempted to process each string by each multitransducer's automaton, and as long as such a pair (string, automaton–able–to–process–this–string) exists, then processing takes place. Henceforth, the environment has a potential for realization of highly parallel computations. Should there be two automata able to process a given string, one of them is selected randomly. After being processed, a string "disappears", being transformed into a corresponding output string. The environment in which a multitransducer operates in the way described above is called *Turing computational field*. The apparently strange behavior of the Turing computational field is motivated by the idea of modelling the chemical reactions by such a field. Namely, certain chemical reactions take place if there are corresponding reactants (i.e., inputs) available and only if there is a corresponding catalyzer (i.e., a corresponding automaton with an activated initial state) ready. Note that in a similar way also the computations within the membrane systems are defined (cf. [4],[5]).

A multitransducer differs from the set of standard Mealy automata in two aspects. First, the set of initial states of all automata is not fixed, i.e., it is not given once for all at the beginning of the computation. Rather, depending on the course of computation this set can change with time: some states can loose their property of being initial states, others can obtain this property. The instructions for activation/deactivation of initial states are included in the transition function of the multitransducer. The states that are at the moment initial states will be also called *active states.* The initial activation of states is given by set B as a part of the multitransducer definition. Depending on the inputs read in subsequent steps and on their order, (recall that automata work asynchronously) the activity of states can change. The dynamic activation of its states enables the multitransducer to switch "off" or "on" certain automata and to control the interactive processing in this way. The automata whose initial state has been deactivated cease to be a part of the Turing computational field and remain so unless their initial state gets reactivated by an other automaton. The intended use of the initial state activating mechanism is to model the gene switching in real cells. The second point of the departure of a

multitransducer from the definition of classical automata is the possibility of controlling the processing speed of individual transitions. In order to be able to change the speed of transitions we assume that with each transition, a so–called *speed parameter* (a natural number), is associated that defines the speed taken by the realization of that transition. This possibility can be used in tuning the synchronization among various automata[1].

With respect to the process of self–reproduction it seems natural to claim that a multitransducer cannot transform non–empty strings into empty strings and vice versa, that is, a multitransducer can neither generate something from nothing nor nothing from anything. Note that, syntactically, in the transition function representation, there is no visible "boundary" between the automata of which the multitransducer consists — from the description of its activity it is clear that once the processing of a string gets started by a transition containing an active state on its left–hand side the processing will be prolonged by any transition that applies to the new state and the symbol read at that very moment. In this way, the processing goes on via a chain of admissible transitions until the string gets "consumed" and a final state is reached. If the initial state of the automaton at hand is still active, than a new processing can be launched. In what follows instead the term "string" we will often use the term "object" to denote either a symbol or a string of symbols. Depending on the context we will consider an object either as data it represents or as a physical object possibly having certain self–assembly properties which will be used to our advantage.

Encapsulating the multitransducer Since the final goal of our efforts is modelling of self–reproduction we will have to "engage" a multitransducer in its own replication. The resulting device will be called a biomaton. To this end, following the ideas from cellular biology, we will let the multitransducer replicate its "genom" and build its own "body" — a *membrane* endowed by a self-replicating property. The purpose of the membrane will be

- to protect the multitransducer's control mechanisms from the unwanted influence of the environment;
- to restrict the range of multitransducer's computational influence (i.e., the Turing computational field) to a certain finite domain;
- to let selectively pass some input objects into the membrane;
- to enable the biomaton's development (especially its growth and multiplication).

The membrane is constructed of so–called *tiles* which are special objects produced by specific automata in the Turing computational field encapsulated by the membrane. The membrane allows translocation of certain input objects from the outside environment; tiles are produced from such input objects. Tiles have a special shape and special properties. Their shape is such that they are able to form a three–dimensional spherical structure — a membrane which prevents the encapsulated objects to escape and allows certain input objects to enter. The tiles possess self–assembly property meaning that any random cluster of tiles that are sufficiently close

[1] The speed parameter can be avoided at the expense of allowing epsilon transitions in the formal definition of a multitransducer.

to each other will spontaneously self–assemble into a membrane and, moreover, if there should be further tiles in the vicinity of such a membrane they will get spontaneously incorporated into it. In this way a membrane can grow. When roughly doubling its volume, a membrane tears in two approximatively equal parts that both spontaneously again organize into membranes. The pace of membrane growth depends on the supply of tiles which are generated by automata from elements that are not endowed by self–assembly property.

The activities of a multiset of automata within the Turing computational field are controlled by a "program" that takes the form of a rewritten tape which finds itself inside the membrane. This tape contains the description of the multitransducer's transition function in a linear form. This description consists of a series of segments each of which corresponds to one transition of form $I \times S \to S \times O \times S \times \{0,1\} \times \mathbf{N}$. Of course, such a program resembles a genom residing inside a biological cell. Similarly as a genom, it consists of a series of instructions for production of various objects that can be further used for membrane construction or for constructing the genom's copy. All this happens via activation or deactivation of the respective automata. From the viewpoint of a multitransducer, a program is an object as any other objects and therefore the program tape can also become a subject of an automaton's processing within that multitransducer. An important automaton in that respect is a copying automaton whose task is to produce a copy of the program tape. Such an automaton has two inputs — one by which it reads the current program tape and the other by which it accepts "stuff" (objects) from which a tape's copy is to be constructed. Of course, the copying process does not destroy the original tape. Note that it is (also) here where our scenario of self–reproduction deviates from the classical von Neumann's ideas. Namely, in our case in the multitransducer's description there is no need to give a "recipe" how to build the "body" — a membrane, when and how to split it, how to see that there is a single copy of the program tape in each newly emerging membrane, etc. While this is technically possible (as shown by von Neumann), in our case the self–assembly processes, their proper triggering and timing by a multitransducer, and a postulation of a non–computational (albeit in reality natural) operation of membrane splitting take care about this kind of self–reproduction activities that von Neumann had to program laboriously.

The idea of an embodied transducer emerging above is captured in the following "descriptive" definition of a biomaton:

Definition 2. *A* biomaton *is a self–reproducing multitransducer which works in the following way:*

- *the activity of the multitransducer's Turing computational field is controlled by the multitransducer's transition function which is represented as a special object — called tape;*
- *this tape resides inside a membrane with a certain initial volume which has been constructed by self–organization from tiles that are produced by the Turing computational field from specific input objects permeating freely the membrane from outside; the respective input objects do not possess self–assembly properties; the membrane steadily increases its volume by incorporating new tiles;*

- *along with the growth of the membrane the process of tape copying is in progress; the copy of the tape is also built by the Turing computational field from input objects permeating the membrane;*
- *the growth and copying processes are synchronized so that at the time when the copying process ends the initial volume of the membrane doubles; at that time the membrane splits into two membranes, each retaining one copy of the tape.*

Note that after the fission each of the pair of newly emerging biomata has the original size of their parent biomaton. Thus, under a sufficient supply of input objects the same process can be repeated *ad infinitum*.

Even from the above informal description one can see that for its self–replication a biomaton needs a hierarchy of objects with various properties. We start with simple input objects possessing no self–assembly abilities. However, these objects must be such that a multitransducer can generate out of them other objects already possessing self–assembly properties (tiles). These self–assembly objects further self–organize into complex structures (membranes) that by definition are endowed with still other emergent properties not possessed by their parts (e.g., membrane splitting).

In order to show that the definition of a biomaton is sound one has to prove that an entity satisfying it does exist. In a sense this is a problem similar to that von Neumann faced after describing the idea of his mechanistic self–replicating robot without actually constructing it. In our case, a proof of the last claim concerning the existence of a biomaton would require a formal design of a concrete multitransducer with properties according to Definition 2. While we believe that in principle this is possible, for the time being we feel that we do not have a sufficiently developed formalism for capturing all the necessary spacial, temporal and functional aspects of self–organizational processes needed for our purposes. Initial attempts in this direction can be seen in the emerging mathematical theory of self–assembly (cf. [1]). In [9] a so–called globular universe (a kind of cellular automaton) has been described in which the existence of self–replicating structures resembling the tape from Definition 2 is constructively shown. Moreover, these structures are shown to posses an evolutionary potential, i.e., they can evolve so as to realize any given finite control mechanism. Nevertheless, for the time being we have to refer to an "indirect" evidence pointing to the existence of biomata. Namely, self–assembly and splitting of a sufficiently large membrane which are basic assumptions postulated in our model are justified by the existence of similar phenomena in reality, at the level of real bacteria. There, the physical laws work "as needed" for a bacterium to operate correctly. Both the self–assembly property and the physical laws acting, e.g. in the case of a membrane splitting or input objects permeating the membrane, are "present" all the time without a need to be invoked; what is done in a bacterium is harnessing these essentially non-computational phenomena for the purpose of life. All these non-computational phenomena are captured by our model at the level of assumptions. This evidence from real life is supported by efforts in cellular biology for synthesizing life from scratch (cf. [3]).

3 Discussion

Let us compare "our" scenario of self–reproduction with that of von Neumann (as briefly sketched at the beginning of this paper). Obviously, the basic principles are the same: in both cases there is a program that is both actively interpreted and passively copied. But there is a difference in both approaches concerning the replication: while von Neumann builds a copy separately, outside the original body, right from the scratch, taking care over all details of the body building, in our approach a copy emerges by splitting the original body without taking care over the details of such a process. In our setting there is never a phase of a "half finished" automaton that is not yet functional. To our mind, our approach reflects the self–reproduction on the level of the simplest cells while von Neumann's approach (via cellular automata) corresponds more to multicellular organisms. In order that a cellular automaton should reproduce itself it needs a supply of finished fully functional cells that need to be only activated. In our case, the transformation from a "non–living" to an "animated" entity is more gradual: we start with input objects having no self–assembly properties, produce out of them objects with such properties, and finally let them self–organize. It seems that such a process needs less sophisticated central computational control and leads to a better parallelism exploitation. That is perhaps why it has been favored by evolution.

The idea of biomata brings new impetuses into the automata theory since it introduces a new computational model mixing data processing with object construction while utilizing non–standard computational resources. This leads to new classes of computational problems which wait to be formulated, formalized and solved. A characterization of the processing power of multitransducers (or biomata) is open. Undoubtedly, any progress along these lines must be matched by an analogous progress in the theory of self–assembly.

In the context of computational models it is of interest to discuss the relation of biomata to the membrane systems (cf. [5]). Although originating from the same source of ideas (viz. cell biology), we see the main differences between the two systems both in their different purposes for which they were designed and in their different architecture. These points are summarized bellow:

- in standard membrane systems the membrane is a part of the model that is not produced by a model; in the case of biomata, the membrane is a product of input processing;
- for their activity the membrane systems make use of a hierarchy of distributed computations; the biomata make use essentially of three different cooperating resources:
  - distributed computational power controlled centrally via biomaton's tape and state switching;
  - distributed self–assembly processes governed by local assembly rules;
  - non–computational phenomena modelling the effect of physical laws;
- the computations of membrane systems are governed by rules, whereas the biomata are controlled by finite state machines (of course, the computational power of both mechanisms is the same);

– the "program" of biomata is both actively interpreted for controlling the biomaton's activities and passively copied for the self–reproduction purposes; without modifying their functionality, this cannot be mirrored by the membrane systems;
– the primary aim in the design of membrane automata has been their computational universality, as indicated e.g. in [4]; in the case of biomata, the aim has been to achieve their self–reproduction ability;
– an evolutionary aspect can easily be introduced into a framework of biomata; in fact for such a purpose it is enough to admit errors in the copying process of genetic information. By the very construction of biomata, the "genotype" of the system is closely related to its "phenotype" and thus the system as a whole can become a subject of darwinian evolution. The membrane systems cannot be straightforwardly adapted for such a modelling.

We believe that our model is of interest also in the context of cellular and evolutionary biology, exactly for reasons mentioned in the last item of the previous paragraph: it enables a further insight into the mechanisms of adaptive evolution and perhaps will also enable computational experiments along these lines.

To some extent, perhaps the biomata can also contribute to the eternal question on the relationship between living and non–living matter (cf. [2]). Namely, in biomata we start with the input objects not endowed by self–assembly property, in the next step we construct ("compute"?) objects already possessing such a property, and we end up with a "living matter", to some extent. All this happens in an abstract medium, within a mathematical model; a possible candidate for such a model has been proposed in [9]. In this context, biomata are a typical instance of artificial life. To what extent our models correspond to reality remains to be seen.

The last remark concerns the relation between computing and constructing. For our approach it has been of a prime importance that the objects can be seen both as data for information processing (e.g. the multitransducer's tape, the "genom", has been seen as a set of instructions to be interpreted by the Turing's computational field), and real physical objects obeying physical (or chemical) laws (e.g. the tape can be copied, from the input objects self–assembly objects can be produced, the tiles organize themselves spontaneously into a membrane, an oversized membrane ruptures and splits). Perhaps we are witnessing the dawn of a new field of computing, a "constructive computing", with self–assembly being its harbinger.

4 Conclusion

We devised a biomaton — a novel model of a self–reproducing machine which is driven by a finite state program. From the surrounding input objects a biomaton constructs its "spare" parts endowed by self–assembly properties. Consequently, these parts organize themselves spontaneously into the biomaton's "body" that takes the form of a membrane. Eventually, the biomaton produces a copy of its program and splits its membrane into two equal parts, each containing one copy of the original control program. When compared with the standard von Neumann's model of self–reproduction our scenario leads to a new model of self–reproduction

that captures this process at the level of a single cell rather than at the level of multicellular organisms as von Neumann's cellular automaton model in fact does. In the automata theory, in the related context of cellular and evolutionary biology, and in artificial life our model seems to have a great potential for its further development and investigations. The new model presents a case of constructive computing, in which the physical properties of data representations are equally important as the computational properties of the data themselves.

References

1. Adleman, L.: Toward a mathematical theory of self-assembly. Tech. Rep. 00-722, Dept. of Computer Science, University of Southern California, 2000.
2. Brooks, R.: The relationship between matter and life. Nature, Vol. 409, 18 January 2001, pp. 409–411
3. Hanczyc, M. M., Fukijawa, S. M., Szostak, J. W.: Experimental Models of Primitive Cellular Compartments: Encapsulation, Growth, and Division. Science. 302 (2003), pp. 618-622.
4. Paun, G., Rozenberg, G.: A Guide to Membrane Computing. Theoretical Computer Science 287 (2002), pp. 73-100.
5. Paun, G.: Membrane Computing. An Introduction, Springer, 2002
6. Szostak, J.W., Bartel, D.P., Luisi, P.L.: Synthesizing Life. Nature 409 (2001) 389-390.
7. von Neumann, J.: Theory of Selfreproducing Automata. A. Burks (Ed.), University of Illinois Press, Urbana and London, 1966
8. Wiedermann, J.: Coupling computational and non–computational processes: minimal artificial life. Pre–proceedings of the Fifth Workshop on Membrane Computing (WMC5), G. Mauri, Gh. Paun, C. Zandroni (Eds.), Dept. of Comp. Sci., University of Milan — Bicocca, Italy, June 16–16, 2004, 444 p.
9. Wiedermann, J.: Self–reproducing self–assembling evolutionary automata. Manuscript, September 2004