Technical Report

Nr.: FIN-11-2008

Colloquium on the Occasion of the 50th Birthday of Victor Mitrana Proceedings

Jürgen Dassow, Bianca Truthe (Herausgeber)

Arbeitsgruppe Automaten und Formale Sprachen

Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg

Impressum (§ 10 MDStV):

Herausgeber:
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Der Dekan

Verantwortlich für diese Ausgabe:
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Prof. Dr. Jürgen Dassow
Postfach 4120
39016 Magdeburg
E-Mail: dassow@ovgu.de

http://www.cs.uni-magdeburg.de/Preprints.html

Auflage: 90

Redaktionsschluss: 24.10.2008

Herstellung: Dezernat Allgemeine Angelegenheiten,
Sachgebiet Reproduktion

Bezug: Universitätsbibliothek/Hochschulschriften- und
Tauschstelle

Nr.: FIN-11-2008

Colloquium on the Occasion of the 50th Birthday of Victor Mitrana Proceedings

Jürgen Dassow, Bianca Truthe (Herausgeber)

Arbeitsgruppe Automaten und Formale Sprachen

Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg

Jürgen Dassow and Bianca Truthe (Editors)

Colloquium on the Occasion of the 50th Birthday of Victor Mitrana

Proceedings

Otto von Guericke University Magdeburg, Germany
June 27, 2008

Editors

Jürgen Dassow and **Bianca Truthe**

Otto von Guericke University Magdeburg
Faculty of Computer Science
Department of Knowledge and Language Engineering
Universitätsplatz 2
D-39106 Magdeburg
Germany

e-mail: {dassow,truthe}@iws.cs.uni-magdeburg.de

Preface

On the occasion of the 50th birthday of Victor Mitrana on June 26, 2008, the Research Group *Formal Languages and Automata* of the Otto von Guericke University Magdeburg has organized a colloquium at the Faculty of Computer Science. At this time, Victor Mitrana has been visiting our research group as a fellow of the Alexander von Humboldt Foundation. The scientific programme of the colloquium consisted of one invited lecture and three contributions. This volume contains the papers of all presented talks.

Victor Mitrana was born in Bucharest, Romania. He obtained the Master's Degree in Mathematics and Computer Science from the University of Bucharest in 1986 and the Doctoral Degree in 1993 with a thesis on Distributed Grammar Systems. In 2002, he was appointed professor at the University of Bucharest and, in 2003, he became a professor at the Rovira i Virgili University in Tarragona, Spain, thanks to the Ramón y Cajal Programme of the Spanish Government.

Victor Mitrana is a very active researcher in the field of formal languages, grammars and automata as well as in computational models inspired by biology. The results were published in more than 170 papers in international journals and conference proceedings as well as in several books.

We would like to mention a few milestones of Victor Mitrana's scientific success. He is one of the inventors of grammars systems. During the last two decades, more than 500 papers have been written on grammar systems. One of the first papers on this topic, however, was written by Victor Mitrana together with Adrian Atanasiu on Modular Grammars. He also invented hybrid grammar systems and has published several papers on systems of automata. Until then, only grammar systems had been studied but not their automata like counterparts.

This picture was taken by György Vaszil during the conference *Automata and Formal Languages* in Balatonfüred, Hungary, in May 2008.

A second field, we would like to mention, is duplication. This is one operation that can be applied to words where a copy of a word is inserted into that word. Victor Mitrana has investigated many different features of duplication. There are some papers on grammars which use duplication as an operation, sometimes only the duplication (duplication grammars). However, because the motivation came from biology, there are also some papers on evolutionary grammars where duplication is one of the operations and some other operations are used in addition. To both kinds of grammars, Victor Mitrana has contributed a lot. He has also investigated combinatorial properties of duplication and its relation to coding theory.

A third topic, he worked on, concerns evolutionary networks. Originally introduced by Erzsébet Csuhaj-Varjú and Arto Salomaa as networks of language processors, Victor Mitrana considered networks where the processors are of a certain type. In one direction, processors implement point mutations (insertion, deletion or substitution of a single letter), in another direction, the basic operation of the processors is splicing. Also, he has studied generating networks as well as accepting networks. Further, he investigated some complexity measures and was able to characterize some complexity classes by means of evolutionary networks. Additionally, he considered applications of such networks for solving NP-problems in polynomial time.

Victor Mitrana has contributed to other topics, too. They inclcude weighted automata and automata over groups, combinatorics of words, especially some types of sequences, contextual grammars, regulated grammars, the hairpin operation.

Victor Mitrana and our research group have been cooperating for many years. In the years 1995/96, he was a member of the group as a fellow of the Alexander von Humboldt Foundation of Germany. Also before and after this long term stay, he visited Magdeburg for shorter periods of time. There exist 25 joint papers with members of our reseach group – this number is still increasing.

As the invited speaker, György Vaszil gave a lecture on *Multiset languages and P Automata*. He did not only cooperate with Victor Mitrana scientifically, they also have in common, that both of them spent more than a year at our university as fellows of the Humboldt Foundation. The other talks were given in this order by Ralf Stiebe *On the Complexity of the Control Language in Tree Controlled Grammars*, Bianca Truthe *On Small Networks of Evolutionary Processors with Regular Filters*, and Jürgen Dassow on *Some Operations Preserving Primitivity of Words*. All these talks address topics that have also been studied by Victor Mitrana. Further, there exists cooperation with all four speakers.

Finally, we would like to thank all those who made this colloquium possible, especially the speakers for their contributions. A very special thank goes to Victor Mitrana for many years of kind and stimulating collaboration.

Dear Victor, we wish you all the best
for your future!

Jürgen Dassow and Bianca Truthe
Magdeburg, October 2008

INVITATION

On the occasion of the 50th birthday of

Prof. Dr. Victor Mitrana
University of Bucharest
at present
Fellow of the Alexander von Humboldt Foundation
Otto von Guericke University Magdeburg
Faculty of Computer Science
Department of Knowledge and Language Engineering

the hosting Working Group *Formal Languages and Automata* organizes a

COLLOQUIUM

on **Friday, 27 June 2008**, in **Building 29, Room 301**.

You are cordially welcome.

Programme:

9:15 Jürgen Dassow (Otto von Guericke University Magdeburg)
Welcome and Laudatio

9:25 György Vaszil (Hungarian Academy of Sciences, Budapest)
Multiset Languages and P Automata

10:05 Ralf Stiebe (Otto von Guericke University Magdeburg)
On the Complexity of the Control Language in Tree Controlled Grammars

10:25 Bianca Truthe (Otto von Guericke University Magdeburg)
On Small Networks of Evolutionary Processors with Regular Filters

10:45 Jürgen Dassow (Otto von Guericke University Magdeburg)
Primitivity Preserving Operations

11:00 Closing

Contents

J. Dassow, B. Truthe (Eds.): Colloquium on the Occasion of the 50th Birthday of Victor Mitrana.
Otto von Guericke Universität Magdeburg, Germany, June 27, 2008. Proceedings, pages 1 – 10.

Multiset Grammars, Multiset Automata, and Membrane Systems

GYÖRGY VASZIL

Computer and Automation Research Institute, Hungarian Academy of Sciences
Kende utca 13-17, H-1111 Budapest, Hungary

vaszil@sztaki.hu

Abstract: We review how different multiset processing devices, namely multiset grammars, multiset automata, membrane systems with symport/antiport, or P automata can be used to characterize multiset and string languages and also show how P automata can describe languages over infinite alphabets.

Keywords: Multiset processing devices, P automata, languages over infinite alphabets.

1. Introduction

Multiset languages, sets consisting of multisets, have been studied from several different points of view. In [7] a Chomsky-like hierarchy of multiset rewriting devices, so called multiset grammars, were presented for their characterization. In [2] multiset automata were introduced and a correspondence between the different types of multiset automata and grammars was established.

Another class of multiset processing devices called membrane systems are studied in the field of membrane computing. Membrane systems, or P systems were introduced in [11] as computing models inspired by the functioning of the living cell. Their main components are membrane structures consisting of membranes hierarchically embedded in the outermost skin membrane. Each membrane encloses a region containing a multiset of objects and possibly other membranes. Each region has an associated set of operators working on the objects contained by the region.

One of the most interesting variants of the model was introduced in [10] called P systems with symport/antiport. In these systems the modification of the objects present in the regions is not possible, they may only move through the membranes from one region to another. The movement is described by communication rules called symport/antiport rules associated to the regions. See the monograph [12] for a summary of notions and results of the area.

In the following we review how multiset grammars, multiset automata, and symport/antiport systems can be used to characterize multiset languages. We recall the results showing the equivalence of regular multiset grammars and multiset finite automata and that of monotone multiset grammars and multiset linear bounded automata. While regular multiset grammars and multiset finite automata characterize the Parikh sets of regular (string) languages, the class of multiset languages determined by monotone multiset grammars and multiset linear bounded automata are strictly included in the class of Parikh sets of monotone (that is, context-sensitive) string languages. We show, however, that this class can be characterized in terms of symport/antiport membrane systems, so called exponential-space symport/antiport acceptors. Finally, we review some basic results concerning P automata, an other variant of accepting symport/antiport P systems which also characterize the class of regular and context-sensitive languages, and moreover, can also be used for capturing the notion of languages over alphabets containing an infinite number of symbols.

2. Preliminaries

Let Σ be a set of symbols called alphabet, and let Σ^* be the set of all words over Σ, that is, the set of finite strings of symbols from Σ, and let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ where ε denotes the empty word. The set of finite subsets of Σ is denoted by 2^{Σ}.

Let U be a set of objects, and let $\mathbb{N}$ denote the set of non-negative integers. A multiset is a mapping $u : U \to \mathbb{N}$ which assigns to each object $a \in U$ its multiplicity $u(a)$ in u. The support of u is the set $supp(u) = \{a \mid u(a) \geq 1\}$. If $supp(u)$ is a finite set, then u is called a finite multiset. The set of all finite multisets over the set U is denoted by U°.

For two multisets u_1, u_2 over the same set of objects U, we have $u_1 \subseteq u_2$ if and only if $u_1(a) \leq u_2(a)$ for all $a \in U$; the union of the two multisets is defined as

$$(u_1 \cup u_2)(a) = u_1(a) + u_2(a),\ a \in U;$$

the difference is

$$(u_1 - u_2)(a) = u_1(a) - u_2(a) \text{ for } a \in U,$$

provided that $u_2 \subseteq u_1$.

A multiset u over the finite set of objects V can be represented as a string w over the alphabet V with $|w|_a = u(a)$ where $a \in V$ and $|w|_a$ denotes the number of occurrences of the symbol a in the string w, and with ε representing the empty multiset. Let $|w|$ denote the length of w, that is, the cardinality of the multiset represented by w. A multiset can also be represented as the Parikh vector of the corresponding string, thus, there is a natural, one-to-one correspondence between multiset languages and sets of vectors with integer coordinates.

3. Multiset Grammars and Multiset Automata

A *multiset grammar*, [7], is a construct $G = (N, T, S, P)$ where N, T are the disjoint alphabets of nonterminals and terminals, S is a multiset over $N \cup T$ and P is a finite set of multiset rewriting rules of the form $u \to v$ with $u, v \in (N \cup T)^\circ$ and $u(A) \geq 1$ for some $A \in N$. For two multisets α_1, α_2 over $(N \cup T)$, we write $\alpha_1 \Rightarrow \alpha_2$ if there exists $u \to v \in P$ such that $u \subseteq \alpha_1$ and $\alpha_2 = \alpha_1 - u \cup v$. We denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$. The language generated by G is defined as $L(G) = \{\alpha \in T^\circ \mid S \Rightarrow^* \alpha\}$.

Grammars as above are said to be monotone if $|u| \leq |v|$ for all rules $u \to v \in P$, context-free if $|u| = 1$ for all rules $u \to v \in P$, or regular if $|u| = 1$ and $v = aB$ or $v = a$ for some $a \in T$, $B \in N$.

We denote by $m\mathit{RE}, m\mathit{MON}, m\mathit{CF}, m\mathit{REG}$ the families of multiset languages generated by arbitrary, monotone, context-free, or regular multiset grammars, respectively. By *RE*, *MON*, *CF*, *REG*, we denote the families of recursively enumerable, context-sensitive, context-free, and regular languages, respectively, and by psX for a language family $X \in \{\mathit{RE}, \mathit{MON}, \mathit{CF}, \mathit{REG}\}$, we denote the families of Parikh vectors associated to the languages in X.

A multiset finite automaton, [2], consists of a finite control unit, an input store in which a multiset is placed, and a reading head which can detect whether or not a given symbol appears in the input. The automaton changes its state depending on the former state and the detection of a symbol in the input. If a symbol is detected, it is removed. If the input is eventually empty and the current state is an accepting state, the automaton accepts the initial multiset, otherwise it is rejected.

A multiset linear bounded automaton is a multiset finite automaton which can also "write" to the stored multiset. The multiset linear bounded automaton also changes its state based on the former state and the detection of a symbol in the stored multiset, but it can also add a symbol to the stored multiset.

Formally a *multiset finite automaton* is a structure $M = (Q, V, \delta, q_0, F)$ where Q is a finite set of states, V is the input alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times V \to 2^Q$ is the transition mapping.

A *multiset linear bounded automaton* is a construct $M = (Q, V, U, \delta, q_0, F)$ with Q, V, q_0, F as above, and the transition mapping $\delta : Q \times V \to 2^{Q \times (U \cup \{\varepsilon\})}$.

The *configuration* of a multiset automaton is a pair (q, u) where $q \in Q$ is the current state and $u \in V^\circ$ (or $u \in (V \cup U)^\circ$ in the case of multiset linear bounded automata) is the contents of the multiset store. We define the relation $(q, u) \vdash (s, v)$

- for multiset finite automata, if and only if there is an $a \in V$ such that $u(a) \geq 1$, $s \in \delta(q, a)$ and $v = u - a$, and
- for multiset linear bounded automata, if and only if there is a pair

$$(a, b) \in V \times (U \cup \{\varepsilon\})$$

such that $u(a) \geq 1$, $(s,b) \in \delta(q,a)$ and $v = u - a \cup b$.

The reflexive and transitive closure of $\vdash$ is denoted by $\vdash^*$. The *language* accepted by a multiset automaton M is defined as $L(M) = \{u \in V^\circ \mid (q_0, u) \vdash^* (q, \varepsilon), q \in F\}$. The classes of languages accepted by multiset finite automata and multiset linear bounded automata are denoted by $\mathcal{L}(MFA)$ and $\mathcal{L}(MLBA)$, respectively.

Since the proof of the equivalence of finite automata and regular string grammars with respect to their computational power can easily be transformed for the multiset case, and since the Parikh sets of regular and context-free languages coincide, we have the following.

Proposition 1 [2]. $\mathcal{L}(MFA) = mREG = mCF = psREG = psCF$.

A similar equivalence holds also in the case of linear bounded automata.

Proposition 2 [2]. $\mathcal{L}(MLBA) = mMON$.

The classes of $mMON$ and $psMON$ however, are different. From [7], we have that

$$mMON \subset psMON,$$

that is, that the language class defined by the Parikh sets of context-sensitive grammars strictly include the multiset languages generated by monotone multiset grammars. (The unary language $\{a^{2^n} \mid n \geq 1\}$, for example, is in $psMON - mMON$.) In the following we show how to characterize $psMON$ in terms of membrane systems.

4. Symport/antiport Acceptors and P Automata

A membrane system, or P system is a structure of hierarchically embedded membranes, each having a label and enclosing a region containing a multiset of objects and possibly other membranes. The out-most membrane which is unique and usually labeled with 1, is called the skin membrane. The membrane structure is denoted by a sequence of matching parentheses where the matching pairs have the same label as the membranes they represent.

The evolution of the contents of the regions of a P system is described by rules associated to the regions. Applying the rules synchronously in each region, the system performs a computation by passing from one configuration to another one. Several variants of the basic notion have been introduced and studied proving the power of the framework, see the monograph [12] for a summary of notions and results of the area. In the following we concentrate on communication rules called symport or antiport rules.

A symport rule is of the form (x, in) or $(x, out), x \in V^\circ$. If such a rule is present in a region i, then the objects of the multiset x can enter from the parent region or can leave

to the parent region, respectively. An antiport rule is of the form $(x, in; y, out), x, y \in V^\circ$, in this case, objects of x enter from the parent region and in the same step, objects of y leave to the parent region. All types of these rules might be equipped with a promoter or inhibitor multiset, denoted as $(x, in)|_Z, (x, out)|_Z$, or $(x, in; y, out)|_Z$, with $x, y \in V^\circ$, $Z \in \{z, \neg z \mid z \in V^\circ\}$, where if $Z = z$ then the rules can only be applied if region i contains the objects of multiset z, or if $Z = \neg z$, then region i must not contain any of the elements of z. (For more on symport/antiport see [10], for the use of promoters see [8].)

A *P system with symport/antiport* of degree $n \geq 1$ is a construct

$$\Pi = (V, \mu, E, w_1, \ldots, w_n, R_1, \ldots, R_n, F, in)$$

where

- V is an alphabet of objects,
- μ is a membrane structure of n membranes,
- $E \subseteq V$ is a set of objects (the ones which can be found in the environment in an arbitrary number of copies),
- $w_i \in V^\circ,\ 1 \leq i \leq n$, are the initial contents of the n regions,
- $R_i,\ 1 \leq i \leq n$, are the sets of symport/antiport rules associated to the regions,
- F is a set of final configurations, and
- $in \in \{0, 1, \ldots, n\}$ is the label of the input membrane, where if $i = 0$, the input is read from the environment.

The $n + 1$-tuple of finite multisets of objects present in finite number of copies in the environment and in the n regions of the P system Π describes a *configuration* of Π with

$$(\varepsilon, w_1, \ldots, w_n) \in (V^\circ)^{n+1}$$

being the initial configuration.

The *transition mapping* of a symport/antiport P system is a mapping

$$\delta : V^\circ \times (V^\circ)^{n+1} \to 2^{(V^\circ)^{n+1}}.$$

For two configurations $c = (u_0, u_1, \ldots, u_n)$, $c' = (u'_0, u'_1, \ldots, u'_n)$ and a multiset $u \in V^\circ$,

$$(u'_0, u'_1, \ldots, u'_n) \in \delta(u, (u_0, u_1, \ldots, u_n))$$

holds if there exists a maximal set of rules which, when applied in a parallel and synchronous manner in the regions, transfer the system from configuration (state) c to c' with input u, that is, while the multiset u enters the system from the environment.

We first consider the model called *exponential-space symport/antiport acceptor* introduced in [5]. Such a system is a symport/antiport system with

- a set of terminal objects $T \subseteq V$ containing a distinguished symbol \$,
- $in = 0$, which means that the input is read from the environment,
- rules of the following four types in the set R_1 corresponding to the skin region:
  1. $(u, in; v, out)$, $u, v \in (V - T)^\circ$, $|v| \geq |u|$,
  2. $(ua, in; v, out)$, $u, v \in (V - T)^\circ$, $|v| \geq |u|$, and $a \in T$,
  3. $(u, in; v, out)|_a$, $u, v \in (V - T)^\circ$, $a \in T$,
  4. for every $a \in T$ there is at least one rule of the form $(u, in; a, out)$,
- rules of the form $(u, in; v, out)$, $u, v \in (V - T)^\circ$, in the regions different from the skin region.

We can consider the multiset languages, the set of multisets accepted by an exponential-space symport/antiport acceptor Π as

$$L_m(\Pi) = \bigcup m_T(u_1) \cup m_T(u_2) \cup \ldots \cup m_T(u_t)$$

where $c_0, c_1, \ldots, c_t$ is a sequence of configurations with $c_{i+1} \in \delta(u_{i+1}, c_i)$ and $\$ \notin u_i$ for all $0 \leq i \leq t-1$, $\$ \in u_t$, and where c_0 is the initial configuration, $c_t \in F$, and $m_T(u) \in T^\circ$ is the multiset of terminal objects contained by the multiset $u \in V^\circ$, that is, $m_T(u) \subseteq u$ and $u - m_T(u) \in (V - T)^\circ$.

We can also associate strings to the accepted multisets. A string $a_1 \ldots a_n\$$ with $a_i \in T - \{\$\}$, $1 \leq i \leq n$, is accepted if the terminal symbols are brought into the system from the environment in the required order (by rules of type 2) and after reading the end marker \$, the computation halts.

$$L_{str}(\Pi) = \bigcup str_T(u_1) \cdot str_T(u_2) \cdot \ldots \cdot str_T(\bar{u}_t)$$

where $c_0, c_1, \ldots, c_t$ is a sequence of configurations with $\delta(u_{i+1}, c_i) = c_{i+1}$ and $\$ \notin u_i$ for all $0 \leq i \leq t-1$, $\bar{u}_t = u_t - \$$, and where c_0 is the initial configuration, $c_t \in F$, and $str_T(u) \in T^*$ is the set of terminal strings corresponding to the multiset $m_T(u) \in T^\circ$ of terminal symbols from u.

Let us call an exponential-space symport/antiport acceptor *restricted* if it only uses rules of type 1. and 2. in the skin region.

The classes of multiset and string languages accepted by arbitrary and by restricted symport/antiport acceptors are denoted by $\mathcal{L}_m(ESAA)$, $\mathcal{L}_{str}(ESAA)$, $\mathcal{L}_m(rESAA)$, and $\mathcal{L}_{str}(rESAA)$, respectively.

Restricted exponential-space symport/antiport acceptors characterize regular languages.

Theorem 3 [5].

1. $\mathcal{L}_m(rESAA) = mREG = psREG = mCF = psCF$, *and*
2. $\mathcal{L}_{str}(rESAA) = REG$.

Moreover, the unrestricted variants characterize the class of context-sensitive languages in the string case which means that considering the accepted multiset languages, we obtain a characterization of the Parikh sets of languages generated by monotone grammars.

Theorem 4 [5].

1. $mMON \subset \mathcal{L}_m(ESAA) = psMON$, *and*
2. $\mathcal{L}_{str}(ESAA) = MON$.

Context-sensitive languages can also be characterized by an other device called P automaton which was proposed in [3].

P automata are accepting P systems which combine characteristics of classical automata and distributed natural systems being in interaction with their environment. The behavior of a P automaton is described by its accepted language which is obtained by a mapping from the set of accepted sequences of multisets of objects which enter the system from the environment.

A *P automaton* is a symport/antiport system with the following properties.

- $in = 0$, which means that the input is read from the environment,
- F defines the (not necessarily halting) final configurations, as the n-tuple $F = (F_1, \ldots, F_n)$ where $F_i \subseteq V^\circ$, $1 \leq i \leq n$, are either finite sets of multisets over V, or $F_i = V^\circ$

A configuration $c = (v_0, v_1, \ldots, v_n)$ is said to be final, denoted as $c \in F = (F_1, \ldots, F_n)$, if $v_i \in F_i$, $1 \leq i \leq n$.

Let also $f : V^\circ \to T^*$ be a mapping which maps nonempty multisets in V° to nonempty words over the alphabet T and $f(u) = \varepsilon$ if and only if u is the empty multiset.

A language $L \subseteq T^*$ is accepted by the P automaton Π if it is

$$L(\Pi, f) = \{f(u_1) \cdot f(u_2) \cdot \ldots \cdot f(u_t) \in T^* \mid \text{there is } c_t \in F \text{ and a sequence } c_i \text{ with } \delta(u_{i+1}, c_i) = c_{i+1} \text{ for all } 0 \leq i \leq t-1\},$$

where c_0 is the initial configuration, δ is the transition mapping of Π.

Since the mapping f only maps the empty multiset to ε, that is, since all nonempty input multisets are taken into account when the string of the accepted language is formed, P

automata satisfy the requirement that they should not make any distinction between terminal and nonterminal objects, that they should not completely discard any of the multisets imported in any of the steps of the computation from the accepted language.

Of course, the mapping f should be in some sense simple if we would like to make sure that the computing power of the P automaton lies in the symport/antiport system and not in f itself. For now, let us fix the alphabet as $T = V$ and the mapping as $f_1(u) = a$ for $u = a^k,\ k \geq 1$, with $f_1(\emptyset) = \varepsilon$.

Theorem 5 [1].

1. *For any context-sensitive language L, a P automaton Π can be constructed with object alphabet V, such that $L = L(\Pi, f_1)$ for a mapping f_1 defined as above.*
2. *For any P automaton Π with object alphabet V and mapping $f : V^\circ \to T^*$ for some alphabet T, such that f is linear-space computable, the language $L(\Pi, f) \subseteq T^*$ is context-sensitive.*

We might also consider variations of P automata which restrict the forms of the rules. The notion of *P finite automaton* was defined in [4] as a P automaton where

- the object alphabet $V \cup \{a\}$ contains a distinguished symbol a,
- the set R_1 corresponding to the skin region contains rules of the form $(x, in; y, out)|_Z$ with $x \in \{a\}^\circ$, $y \in (V \cup \{a\})^\circ$, $Z \in \{z, \neg z\},\ z \in V^\circ$, and
- if $i \neq 1$, the set R_i contains rules of the form $(x, in; y, out)|_Z$ with $Z \in \{z, \neg z\}$ and $x, y, z \in V^\circ$.

As we can see, P finite automata can only input multisets of the form a^k, containing several copies of the distinguished symbol a. Therefore, it is appropriate if we define the mapping of the input multisets to the alphabet $T = \{a_1, a_2, \ldots\}$ as $f_2 : \{a\}^\circ \to T^*$ with $f_2(a^k) = a_k,\ k \geq 1$, and $f_2(\emptyset) = \varepsilon$ for the empty multiset.

As it is proved in [4] the rule restrictions introduced in the model of P finite automata also characterize the class of regular languages.

Theorem 6 [4]. *A language L is regular if and only if there is a P finite automaton Π with object alphabet $V \cup \{a\}$, such that $L = L(\Pi, f_2)$ for a mapping f_2 defined as above.*

5. Unconventional Aspects of P Automata

In this section, we would like to propose a topic which is based on one of the unconventional aspects of membrane systems, that is, to use symport/antiport systems for the description of languages over infinite alphabets. The idea comes very naturally if we

recall that that the language accepted by these systems corresponds to the sequence of multisets entering during a successful computation, and notice that the number of possible multisets which make up this sequence, that is, the number of possible symbols which make up the accepted string is not fixed in advance, but it can be arbitrary high.

If we think in terms of P automata, the set of finite multisets over V, that is, the domain of the mapping f is infinite, so its range could also easily be defined to be infinite. This idea is explored in the case of P finite automata in [4], where the mapping producing the terminal words is defined as $f : \{a\}^\circ \to T^*$ for an infinite alphabet $T = \{a_1, a_2, \ldots\}$ as $f(a^i) = a_i$ for any $i \geq 1$.

Since P finite automata over finite alphabets accept exactly the class of regular languages, the resulting infinite alphabet language class can be considered as the extension of the class of regular languages to infinite alphabets, and this class behaves in several respects differently from infinite alphabet language classes defined using other ideas, such as, for example, the machine model called finite memory automata from [6], or the infinite alphabet regular expressions introduced in [9]. Given an infinite alphabet $\Sigma = \{a_1, a_2, \ldots\}$, P finite automata are able to describe, for example, the language $\{a_{2i} \mid i \geq 1\}$ which can be described by infinite alphabet regular expressions but cannot be accepted by finite memory automata, and also the language $\{a_i a_i \mid i \geq 1\}$ which is accepted by finite memory automata but cannot be captured by infinite alphabet regular expressions.

References

[1] E. CSUHAJ-VARJÚ, O. H. IBARRA, and GY. VASZIL, On the computational complexity of P automata. *Natural Computing* **5** (2006), 109–126.

[2] E. CSUHAJ-VARJÚ, C. MARTÍN-VIDE, and V. MITRANA, Multiset Automata. In: C. CALUDE, GH. PĂUN, G. ROZENBERG, and A. SALOMAA (eds.), *Proceedings of the Workshop on Multiset Processing: Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View, Curtea de Arges, August 21-25, 2000*. Lecture Notes in Computer Science **2235**, Springer-Verlag, Berlin, 2001, 69–83.

[3] E. CSUHAJ-VARJÚ and GY. VASZIL, P automata, or purely communicating accepting P systems. In: *Membrane Computing. International Workshop WMC-CdeA, Curtea de Arges, Romania, August 19–23, 2002. Revised Papers*. Lecture Notes in Computer Science **2597**, Springer-Verlag, Berlin, 2003, 219–233.

[4] J. DASSOW and GY. VASZIL, P finite automata and regular languages over countably infinite alphabets. In: *Membrane Computing. 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 2006. Revised, Selected, and Invited Papers*. Lecture Notes in Computer Science **4361**, Springer-Verlag, Berlin, 2006, 352–366.

[5] O. H. IBARRA and GH. PĂUN, Characterizations of context-sensitive language classes and other language classes in terms of symport/antiport P systems. *Theoretical Computer Science* **358** (2006), 88–103.

[6] M. KAMINSKY and N. FRANCEZ, Finite memory automata. *Theoretical Computer Science* **134** (1994), 329–363.

[7] M. KUDLEK, C. MARTÍN-VIDE, and GH. PĂUN, Toward a Formal Macroset Theory. In: C. CALUDE, GH. PĂUN, G. ROZENBERG, and A. SALOMAA (eds.), *Proceedings of the Workshop on Multiset Processing: Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View, Curtea de Arges, August 21–25, 2000*. Lecture Notes in Computer Science **2235**, Springer-Verlag, 2001, 123–134.

[8] C. MARTÍN-VIDE, A. PĂUN, and GH. PĂUN, On the power of P systems with symport rules. *Journal of Universal Computer Science* **8** (2002), 317–331.

[9] F. OTTO, Classes of regular and context-free languages over countably infinite alphabets. *Discrete Applied Mathematics* **12** (1985), 41–56.

[10] A. PĂUN and GH. PĂUN, The power of communication: P systems with symport/antiport. *New Generation Computing* **20** (2002), 295–305.

[11] GH. PĂUN, Computing with membranes. *Journal of Computer and Systems Sciences* **61** (2000), 108–143.

[12] GH. PĂUN, *Membrane Computing. An Introduction*. Springer-Verlag, 2002.

J. Dassow, B. Truthe (Eds.): Colloquium on the Occasion of the 50th Birthday of Victor Mitrana.
Otto von Guericke Universität Magdeburg, Germany, June 27, 2008. Proceedings, pages 11 – 28.

Some Operations Preserving Primitivity of Words

JÜRGEN DASSOW

Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg
PSF 4120, D-39016 Magdeburg Germany

dassow@iws.cs.uni-magdeburg.de

GEMA M. MARTÍN, FRANCISCO J. VICO

Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
Severo Ochoa, 4, Parque Tecnológico de Andalucía,
E-29590 Campanillas – Málaga, Spain

{gema,fjv}@geb.uma.es

Abstract: We investigate some operations where essentially, from a given word w, the word ww' is constructed where w' is a modified copy of w or a modified mirror image of w. We study whether ww' is a primitive word provided that w is primitive. For instance, we determine all cases with an edit distance of w and w' at most 2 such that the primitivity of w implies the primitivity of ww'. The operations are chosen in such a way that in the case of a two-letter alphabet, all primitive words of length at most 11 can be obtained from single letters.

Keywords: Primitive words, primitivity preserving operations.

1. Introduction

A word w over an alphabet V is said to be a primitive word if and only if there is no word $u \in \Sigma^+$ with $w = u^n$ for some natural number $n > 1$. The set of all primitive words over V is denoted by Q_V. There are a lot of papers on relations of Q_V to other language families as the families of the Chomsky hierarchy (e. g. in [4] and [16], it has been shown that Q_V is neither a deterministic nor an unambiguous context-free language, in [10] relations to regular languages are given), Marcus contextual grammars (see [6]), to (poly-)slender languages (see [5]) and some languages and language families related to codes (see e. g. [17]). Moreover, there are papers on combinatorial properties of primitive words and of the sets Q_V; we refer to [2], [1], [8].

However, there is only a small number of results concerning the closure of Q_V under operations. There are some papers where it was investigated whether the application

of homomorphisms to primitive words leads to primitive words in all cases or leads to primitive words with a finite number of exceptions or to non-primitive words in all cases; we refer to [12], [13], [14], [9]. Substitutions form another operation which was investigated with respect to preservation of primitivity. There were substitutions of very short subwords in the focus, especially point mutations (deletions, insertions and substitutions of one letter) were studied. We refer to [15] for details. A further study in this direction concerns insertions (see [11]).

Obviously, there is a large variety of operations from which one can expect that Q_V is closed under them (since the portion of primitive words is very high). In this paper we consider some operations where essentially, from a given word w, the word ww' is constructed where w' is a modified copy of w or a modified mirror image of w. The modifications are of such a form that the edit distance of w and w' is very small or very large (i. e., it is very near to the length of w).

We have two reasons for this investigation. The first one is of combinatorial nature. Obviously, ww is not primitive for all w. We are interested in conditions for changes of the second copy w to w' such that ww' is primitive for all w. Especially, how many changes or deletions or insertions of letters are necessary and how many such operations are possible. For example, we shall determine all possible transformation where the edit distance of w and w' is at most two and primitivity is preserved.

The second reason comes from the theory of dynamical systems. In the paper [7] a dynamical system based on regular languages has been proposed. The regular languages are essentially described by primitive words. Since in dynamical systems one needs mutations in order to develop the system, one is interested in devices which describe primitive words and allow mutations. Here the use of operations which preserve primitivity is of interest. Then a primitive word can be given as a sequence of operations; and a mutation is the replacement of one operation by another one or a deletion or insertion of an operation in the sequence. This ensures primitivity of the word obtained from the mutated sequence of operations. Obviously, it is not necessary to generate all primitive words, however, the set of generated primitive words should contain a good approximation of any primitive word where the quality of approximations is determined by the dynamic system (especially its fitness function). We have chosen the operations under which Q_V is closed in such a way that, if the underlying alphabet V consists of two letters, then by the operations we can generate all primitive words of length ≤ 11 (as can be shown by computer calculations) and a sufficient large amount of primitive words of the length up to twenty.

Thus this paper can also be considered as a continuation of the investigations of devices generating only primitive words (see e. g. [3]).

The paper is organized as follows. In Section 2, we present and recall some notations and some results on primitive words which are used in the sequel. In Section 3, we introduce some operations where we first construct ww and perform then some small modifications of the second copy yielding ww'. We prove that all operations where the edit distance of w and w' is 1 preserve primitivity. An analogous result is shown for the

edit distance 2 if at least one change of a letter is used. In Section 4, we consider analogous operations as in Section 2, but start from ww^R and modify w^R. In Section 5 we consider ww' where w' is obtained from w or w^R by a drastic change, i. e., the Hamming distance of w' and w or w^R is almost the length of w. Moreover, we give some further operations where the length is almost doubled and primitivity is preserved.

2. Some Notation and Facts

For a given alphabet V, we denote by V^* and V^+ the set of all and all non-empty words over V, respectively. The empty word is designated by λ. Given a word $w \in V^*$ and $x \in V$, we denote its length by $|w|$ and the number of occurrences of x in w by $\#_x(w)$. For a word $w = x_1x_2 \dots x_n \in V^+$ with $x_i \in V$ for $1 \leq i \leq n$, we define the mirror image w^R by $w^R = x_nx_{n-1} \dots x_1$. Given two words $w = x_1x_2 \dots x_n \in V^+$ and $w' = y_1y_2 \dots y_n \in V^+$ with $x_i, y_i \in V$ for $1 \leq i \leq n$, the Hamming distance $d(w, w')$ is defined by

$$d(w, w') = \#(\{i \mid x_i \neq y_i\})$$

and the edit distance $ed(w, w')$ of w and w' is the minimal number of changes, deletions and insertions of letters in order to transform w into w'.
Throughout the paper we assume that V has at least two elements.

A word $w \in V^+$ is said to be a primitive word if and only if there is no word $u \in V^+$ such that $w = u^n$ for some natural number $n > 1$. By Q_V we denote the set of all primitive words over V. If V is understood from the context we omit the index V and write simply Q.

Lemma 1. *For any words $v, v' \in V^*$, $vv' \in Q$ if and only if $v'v \in Q$.*

Proof. Let us prove one implication; the other one is analogous.
Let $vv' \in Q$. Let us suppose $v'v \notin Q$, that is, there exists $u \in Q$ with $|u| < |v'v|$ and $n > 1$ such that $v'v = u^n$. Therefore $v' = u^kp$, $v = qu^{n-k-1}$ and $u = pq$ for some words $p, q \in V^*$ and some $k < n$. That implies

$$vv' = qu^{n-k-1}u^kp = qu^{n-1}p = q(pq)^{n-1}p = (qp)^n \notin Q.$$

Thus we have a contradiction to our supposition which proves $v'v \in Q$. □

The following statement holds trivially.

Lemma 2. *If $w \in Q$, then also $w^R \in Q$.* □

Lemmas 1 and 2 can be interpreted as follows: If we apply a cyclic shift or the mirror image to a primitive word, then we obtain a primitive word, again. Thus cyclic shifts and reversal are operations which preserve primitivity.

For the following two lemmas, the reader is referred to [17] for the proof.

Lemma 3. *For two non-empty words u and v, $uv = vu$ if and only if there is a word z such that $u = z^n$ and $v = z^m$ for some natural numbers n and m.* □

Lemma 4. *In a free monoid V^*, the equation $a^m b^n = c^p$, where $a, b, c \in V^*$ and $m, n, p \geq 2$, has only trivial solutions, where a, b and c are powers of some word in V^*.* □

Lemma 5. *For any $x \in V$, $y \in V$ and $z \in V^*$, if $xz = zy$, then $x = y$.*

Proof. If $z = \lambda$, then $x = y$ immediately. If $z = a_1 a_2 \dots a_n$ with $a_i \in V$ for $1 \leq i \leq n$, then $x = a_1, a_1 = a_2, a_2 = a_3, \dots a_{n-1} = a_n, a_n = y$ and consequently $x = y$. □

In the sequel we shall use the following notation. If $w = w_1 w_2 \dots w_r = z_1 z_2 \dots z_s$ for some words $w_1, \dots w_r, z_1, \dots, z_s \in V^*$ such that $|w_1 w_2 \dots w_i| = |z_1 z_2 \dots z_j|$ for some i and j, we write

$$w_1 w_2 \dots w_i | w_{i+1} w_{i+2} \dots w_r = z_1 z_2 \dots z_j | z_{j+1} z_{j+2} \dots z_s,$$

i. e., by the symbol | we mark a certain position in the word. Mostly, | will mark the middle of a word of even length, or it will be put after the m-th letter if the word has odd length $2m - 1$.

3. Operations with an Almost Duplication

Obviously, the word ww obtained from w by a duplication leads from any word w to a non-primitive word. In order to obtain primitive words from a primitive word w one has to perform some changes in the second occurrence of w, i. e., one has to consider words of the form ww' where w' differs only slightly from w. In most cases the edit distance of w and w' will be at most 2, and thus ww' can be considered as an almost duplication of w. We start with the case where we only change some letters to obtain w' from w.

Theorem 6.

(i) *Let w be a primitive word of some length n and w' an arbitrary word of length n such that the Hamming distance $d(w, w')$ is a power of 2, then ww' is primitive, too.*

(ii) *If d is not a power of 2, then there are a primitive word w and a word w' with $d(w, w') = d$ such that ww' is not a primitive word.*

Proof. (i) Obviously, $|ww'|$ is even. Let us suppose $ww' \notin Q$, that is, there exists $p \in \mathbb{N}$ and $v \in V^+$ of length at least 2 such that $ww' = v^p$.
If $p = 2$, then $ww' = v^2$. Since $|w| = |w'|$, we get $w = w' = v$ and thus $d(w, w') = 0$ which contradicts the assumption on the Hamming distance of w and w'.
If p is even, and $p > 2$, we have $\frac{p}{2} \geq 2$ and $v^{\frac{p}{2}} = w \notin Q$, which is a contradiction.

If p is odd, i. e., $p = 2m+1$ for some $m \geq 1$, then $|v|$ is even (since otherwise $|v^n|p = |ww'|$ would be odd). Thus there are words v' and v'' of length $\frac{|v|}{2}$ such that $v = v'v''$. Then we get $w = v^m v' = (v'v'')^m v'$ and $w' = v''v^m = v''(v'v'')^m$. The Hamming distance is $d(w,w') = (2m+1)d(v',v'')$. Since $2m+1$ is an odd number, $d(w,w')$ is not a power of 2 in contrast to our supposition.

(ii) Let d be not a power of 2. Then there is an odd number $q > 1$ and a number p such that $d = qp$. Let $q = 2m+1$ for some $m \geq 1$. We now set

$$v' = 10^p, \quad v'' = 11^p, \quad w = (v'v'')^m v', \quad \text{and} \quad w' = (v''v')^m v''.$$

Obviously, the word w is primitive, $d(w,w') = (2m+1)d(v',v'') = (2m+1)p = qp = d$ and $ww' = (v'v'')^{2m+1} \notin Q$. □

By part (ii) of the preceding theorem, if w is a primitive word and $d(w,w')$ is not a power of 2, in general, ww' is not a primitive word. However, if we require that the changes occur in special positions it is possible to obtain preservation of primitivity. As an example we give the following operation.

Definition 7. *For any odd natural numbers $n \geq 3$, any alphabet V, and any mapping $h : V \to V$ with $h(a) \neq a$ for all $a \in V$, we define the operation $O_{n,h} : V^n \to V^{2n}$ by*

$$O_{n,h}(x_1x_2\ldots x_n) = x_1x_2\ldots x_nh(x_1)x_2\ldots x_{i-1}h(x_i)x_{i+1}\ldots x_{n-1}h(x_n)$$

where $i = \frac{n+1}{2}$.

Theorem 8. *For any odd natural number $n \geq 5$, any primitive word q of length n, and any mapping $h : V \to V$ with $h(a) \neq a$ for all $a \in V$, $O_{n,h}(q)$ is a primitive word.*

Proof. Let $w = x_1x_2\ldots x_n$ with $x_j \in V$ for $1 \leq j \leq n$ and $i = \frac{n+1}{2}$. Then

$$O_{n,h}(x_1x_2\ldots x_n) = x_1x_2\ldots x_nh(x_1)x_2x_3\ldots x_{i-1}h(x_i)x_{i+1}x_{i+2}\ldots x_{n-1}h(x_n)$$

has an even length.

Let us suppose that $O_{n,h}(w) \notin Q$, that is, there exist a $p \geq 2$ and $v \in Q$ such that $O_{n,h} = v^p$.
If p is even and $p > 2$, then $v^{\frac{p}{2}} = w$ and $\frac{p}{2} \geq 2$, which contradicts $w \in Q$. If $p = 2$, then $x_1x_2\ldots x_nh(x_1)x_2\ldots x_{n-1}h(x_n) = v^2$, that is,

$$v = x_1x_2\ldots x_{n-1}x_n = h(x_1)x_2x_3\ldots x_{i-1}h(x_i)x_{i+1}x_{i+2}\ldots x_{n-1}h(x_n).$$

Thus $x_i = h(x_i)$, which is a contradiction.
Thus p is odd, say $p = 2m+1$ for some $m \geq 1$. As above there are words v, v_1 and v_2 such that $v = v_1v_2$ and $|v_1| = |v_2|$ and

$$x_1\ldots x_{n-1}x_n|h(x_1)x_2\ldots x_{i-1}h(x_i)x_{i+1}\ldots x_{n-1}h(x_n) = (v_1v_2)^m v_1|v_2(v_1v_2)^m.$$

Since v_1 starts with x_1 (first occurrence) and ends with x_n (last occurrence in the first part), $v_1 = x_1 v_1' x_n$ and analogously, $v_2 = h(x_1) v_2' h(x_n)$. Therefore we have that $O_{n,h}(w)$ has the form

$$(x_1 v_1' x_n h(x_1) v_2' h(x_n))^m x_1 v_1' x_n | h(x_1) v_2' h(x_n) (x_1 v_1' x_n h(x_1) v_2' h(x_n))^m.$$

Since the letters x_i and x_n do not occur in the first occurrence of v, by the definition of $O_{n,h}$, the last letter of the first occurrence of v_1 (in the first part of the word) and last letter of the the first occurrence of v_2 in the second part coincide, i. e., $x_n = h(x_n)$ which is a contradiction. □

We now discuss some operations where the edit distance of w to w' is at most 2 and at least one deletion or one insertion is performed to obtain w'; more precisely, we consider

(a) the deletion of an arbitrary letter,
(b) the deletion of an arbitrary letter and the change of an arbitrary remaining letter,
(c) the insertion of an arbitrary letter,
(d) the insertion of an arbitrary letter and the change of an arbitrary letter of w.

We now give the formal definition of these operations.

Definition 9. *For any natural numbers n, i, j, i' with $1 \leq i \leq n$, $0 \leq i' \leq n$, $1 \leq j \leq n$ and $i \neq j$, letters $x, y, z \in V$ with $x \neq y$, and a word $w = x_1 x_2 \dots x_n$, $x_i \in V$, of length n, we define the following operations*

$$D_{n,i},\ D_{n,i,j,x,y} : V^n \to V^{2n-1} \text{ and } I_{n,i',z},\ I_{n,i',z,j,x,y} : V^n \to V^{2n+1}$$

by

$$D_{n,i}(x_1 x_2 \dots x_n) = x_1 x_2 \dots x_n x_1 x_2 \dots x_{i-1} x_{i+1} x_{i+2} \dots x_n,$$

$$D_{n,i,j,x,y}(x_1 \dots x_n) = \begin{cases} x_1 \dots x_n x_1 \dots x_{i-1} x_{i+1} \dots x_{j-1} y x_{j+1} \dots x_n, & x_j = x, i < j, \\ x_1 \dots x_n x_1 \dots x_{j-1} y x_{j+1} \dots x_{i-1} x_{i+1} \dots x_n, & x_j = x, i > j, \\ \textit{undefined}, & \textit{otherwise}, \end{cases}$$

$$I_{n,i',z}(x_1 x_2 \dots x_n) = x_1 x_2 \dots x_n x_1 x_2 \dots x_{i'} z x_{i+1} x_{i+2} \dots x_n,$$

$$I_{n,i',z,j,x,y}(x_1 \dots x_n) = \begin{cases} x_1 \dots x_n x_1 \dots x_{i'} z x_{i'+1} \dots x_{j-1} y x_{j+1} \dots x_n, & x_j = x, i' < j, \\ x_1 \dots x_n x_1 \dots x_{j-1} y x_{j+1} \dots x_{i'} z x_{i'+1} \dots x_n, & x_j = x, i' > j, \\ \textit{undefined}, & \textit{otherwise}. \end{cases}$$

Theorem 10. *If $n \geq 2$, $1 \leq i \leq n$, and q is a primitive word of length n, then $D_{n,i}(q) \in Q$ also holds.*

Proof. Let us assume $i = 1$. Let $q = xw \in Q$, where $x \in V$ and $w \in V^+$.
Then $D_{n,i}(q) = xww$. Obviously, $|xww|$ is odd.

Let us suppose $xww \notin Q$, that is, there exists an odd number $p \in \mathbb{N}$, i. e., $p = 2m-1$ for some $m \geq 2$, and $v \in V^+$ such that $xww = v^p$ (without loss of generality, we can assume that $v \in Q$).

As in the preceding proof, there are words $v' \in V^*$ and $v'' \in V^+$ such that $v = xv'v''$

$$xw|w = (xv'v'')^{m-1}xv'|v''(xv'v'')^{m-1}.$$

Then $w = (v'v''x)^{m-1}v' = (v''xv')^{m-1}v''$. Since $|(v'v''x)^{m-1}| = |(v''xv')^{m-1}|$, we have $v' = v'' = z$.
Moreover, $xw|w = (xzz)^{m-1}xz|z(xzz)^{m-1}$. Thus $w = (zzx)^{m-1}z = (zxz)^{m-1}z$ which first implies $(zxz)^{m-1} = (xzz)^{m-1}$, then $zxz = xzz$ and finally $xz = zx$. By Lemma 3, z is a power of x. Therefore $q = xw = (xzz)^{m-1}xz$ is a power of x which contradicts $q \in Q$. This contradiction proves $xww \in Q$.

Let us consider $i \geq 2$. Let $q = wxw' \in Q$ with $|w| = i-1$. By Lemma 1, we have $xw'w \in Q$. Hence, by the first part of this proof $D_{n,1}(xw'w) = xw'ww'w \in Q$, which implies $D_{n,i}(q) = wxw'ww' \in Q$ by Lemma 1. □

Theorem 11. *If $w \in V^+$ such that $D_{n,i,j,x,y}(w)$ is defined, then $D_{n,i,j,x,y}(w) \in Q$ holds.*

Proof. We first discuss $D_{n,n,j,x,y}$. Let $w = x_1x_2\dots x_n$. Then

$$D_{n,n,j,x,y}(w) = x_1x_2\dots x_{j-1}xx_{j+1}x_{j+2}\dots x_nx_1x_2\dots x_{j-1}yx_{j+1}x_{j+2}\dots x_{n-1}.$$

Let us assume that $D_{n,n,j,x,y}(w) \notin Q$. Then there is a word $v \in V^+$ such that

$$D_{n,n,j,x,y}(w) = v^p$$

for some $p \geq 2$. Since $D_{n,n,j,x,y}(w)$ has odd length, p and the length of v are odd numbers. Let $p = 2m+1$ for some $m \geq 1$. Thus there are words $v_1 \in V^+$ and $v_2 \in V^+$ such that $v = x_1v_1v_2$, $k-1 = |v_1| = |v_2|$ and

$$x_1x_2\dots x_{j-1}xx_{j+1}x_{j+2}\dots x_n|x_1x_2\dots x_{j-1}yx_{j+1}x_{j+2}\dots x_{n-1} = v^mx_1v_1|v_2v^m.$$

Then $|v| = 2k-1$. We set $s = 2k-1$. We distinguish some cases.

Case 1. Let $1 \leq j \leq k-1$. Then by definition of $D_{n,n,j,x,y}$,

$$x_1v_1 = x_1x_2\dots x_{j-1}xx_{j+1}\dots x_{k-1}x_k = z_1xz_2x_k$$

and

$$v_2 = x_1x_2\dots x_{j-1}yx_{j+1}\dots x_{k-1} = z_1yz_2.$$

Thus, we get,

$$v = z_1xz_2x_kz_1yz_2.$$

If $m \geq 2$, the first part of the word is

$$z_1xz_2x_kz_1yz_2z_1xz_2x_kz_1yz_2v^{m-2}z_1xz_2x_k \tag{1}$$

and that of the second part is

$$z_1yz_2z_1xz_2x_kz_1yz_2z_1xz_2x_kz_1yz_2v^{m-2} \tag{2}$$

and these two words differ in the $(|z_1xz_2x_kz_1yz_2z_1|+1)$-st letter, which contradicts the definition of $D_{n,n,j,x,y}$. If $m = 1$, we get a contradiction by the same arguments.

Case 2. Let $j = k$. Then the k-th letter in the second part is y. On the other hand, it is x_1 since there starts the word v. Thus $x_1 = y$. This gives

$$x_1v_1 = x_1x_2\dots x_{k-1}x_k = yzx,\ v_2 = x_1x_2\dots x_{k-1} = yz \text{ and } v = yzxyz$$

with $z = x_2x_3\dots x_{k-1}$. Then the first and second part are

$$yzxyzyzxyzv^{m-2}yzx \text{ and } yzyzxyzyzxyzv^{m-2},$$

respectively. We obtain $zx = yz$ by looking on the words starting in the position $|z|+3$. Thus by Lemma 5, $x = y$ in contrast to the definition of $D_{n,n,j,x,y}$.

Case 3. Let $k+1 \leq j \leq 2k-1$. Then $v = x_1v_1v_2'xv_2''$. Moreover, $|v_2'| = j-k-1$. Furthermore, y stands in the j-th position of $v_2'xv_2''x_1v_1$, i. e., $x_1v_1 = x_1v_1'yv_1''$ with $|v_1'| = j-k-1$. Therefore $v = x_1v_1'yv_1''v_2'xv_1''$ and $|v_1'| = |v_2'|$ and $|v_1''| = |v_2''|$. Then we get for the second part

$$x_1v_1'yv_1''v_2'yv_2''x_1v_1'yv_1''v_2'xv_2''x_{2s-1}x_{2s}\dots x_n$$

by the definition of $D_{n,n,j,x,y}$ and from the form

$$v_2'xv_2''x_1v_1'yv_1''v_2'xv_2''v^{m-1}$$

given by our assumption. Considering the words starting in the position $(|x_1v_1'yv_1''|+1)$ and in the position $(|x_1v_1'yv_1''v_2'y|+1)$, we see that $v_1' = v_2' = z$ and $v_1'' = v_2'' = z'$. Looking on the subwords starting in the first position and in the position $|v_1'|+2$, we get $x_1z = zx$ and $yz' = xx_1$. By Lemma 5, $x_1 = x$ and $y = x_1$, which contradicts $x \neq y$.

Case 4. Let $j = hs+q$ for some $h \geq 1$ and $1 \leq q \leq k-1$. Then $x_j = x$ is the q-th letter of v. Thus $v = v_1'xv_1''v_2$ with $|v_1'| = q-1$.
We now compute the position of y in v. Since the second part starts with v_2 of length $k-1$ and $hs+q = k-1+(h-1)s+s+q-(k-1) = k_1+(h-1)s+k+q$, y is the $(k+q)$-th letter of v. Therefore $v = v_1'xv_1''v_2'yv_2''$ with $|v_1'| = |v_2'|$. Moreover, $|v_1''| = |v_2''|+1$. Now we get easily the same situation as in Case 1; thus we get (1) and (2) and a difference in the $(|z_1|+1)$-st position.

Case 5. Let $j = hs + k$ for some $h \geq 1$. Then x is the k-th letter of v. We compute the position of y in v. Since the second part starts with v_2 of length $k-1$ and

$$hs + k = k - 1 + hs + k - (k-1),$$

y is the first letter of v. Therefore we get $v = yzxyz$ as in Case 2, which leads to a contradiction.

Case 6. Let $j = hs + q$ for some $h \geq 1$ and $k+1 \leq q \leq 2k-1$. Then $x_j = x$ is the q-th letter of v. Thus $v = x_1 v_1 v_2' x v_2''$ with $|x_1 v_1 v_2'| = q - 1 \geq k$. Furthermore, $|v_2''| = 2k - 1 - q$. We now compute the position of y in v. Since the second part starts with v_2 of length $k-1$ and $hs + q = k - 1 + hs + q - (k-1)$, y is the $(q-k+1)$-st letter of v. Therefore

$$v = x_1 v_1' y v_1'' v_2' x v_2'' \text{ with } |x_1 v_1'| = q - k.$$

Therefore $|v_1''| = k - (q - k + 1) = 2k - 1 - q$. Hence $|v_1''| = |v_2''|$ and consequently also $|v_1'| = |v_2'|$. Therefore we have exactly the situation of Case 3, which leads to contradiction.

Let us now consider $i = 1$, i. e., the operation $D_{n,1,j,x,y}$. By the first part of this proof

$$D_{n,n,n-j+1,x,y}(w^R) = x_n x_{n-1} \dots x_1 x_n x_{n-1} \dots x_{j+1} y x_{j-1} x_{j-2} \dots x_2 \in Q,$$

by Lemma 2,

$$x_2 x_3 \dots x_{j-1} y x_{j+1} x_{j+2} \dots x_n x_1 x_2 \dots x_n \in Q,$$

and by Lemma 1

$$x_1 x_2 \dots x_n x_2 x_3 \dots x_{j-1} y x_{j+1} x_{j+2} \dots x_n = D_{n,1,j,x,y}(w) \in Q.$$

We now consider the case $j < i$. We set

$$\overline{w} = x_{i+1} x_{i+2} \dots x_n x_1 x_2 \dots x_i.$$

Moreover, let $x_j = x$. By the first part of this proof we get

$$D_{n,n,n-i+j,x,y}(\overline{w}) = x_{i+1} \dots x_n x_1 \dots x_i x_{i+1} \dots x_n x_1 \dots x_{j-1} y x_{j+1} \dots x_{i-1} \in Q.$$

Hence, by Lemma 1

$$x_1 \dots x_i x_{i+1} \dots x_n x_1 \dots x_{j-1} y x_{j+1} \dots x_{i-1} x_{i+1} \dots x_n = D_{n,i,j,x,y}(w) \in Q.$$

If $i < j$ we can prove that $D_{n,i,j,x,y}(w) \in Q$ analogously to the case $j < i$ using $D_{n,1,j,x,y}$ instead of $D_{n,n,j,x,y}$. □

Theorem 12. *If q is a primitive word of length n, $0 \leq i \leq n$ and $z \in V$, then $I_{n,i,z}(q) \in Q$.*

Proof. Let w be a primitive word of length n and $a \in V$. Then $I_{n,n,a}(w) = wwa$. Let us assume that $I_{n,n,a}(w) \notin Q$. By Lemma 1, $aww \notin Q$. Now we conclude as in the proof of Theorem 10 (Case $i = 1$) that $w = (zza)^{m-1}az$ and z is a power of a, which yields that w is a power of a in contrast to the primitivity of w.
In order to prove the closure of $I_{n,i,z}$ for $1 \leq i \leq n-1$ we use Lemma 1, again. □

Theorem 13. *If $q \in Q$ and $I_{n,i,z,j,x,y}(q)$ is defined, then $I_{n,i,z,j,x,y}(q) \in Q$.*

Proof. Let $w = x_1x_2 \dots x_{j-1}xx_{j+1}x_{j+2} \dots x_n$. Then

$$I_{n,n,a,j,x,y} = x_1x_2 \dots x_nx_1x_2 \dots x_{j-1}yx_{j+1}x_{j+2} \dots x_na.$$

If we assume that $I_{n,n,a,j,x,y}$ is not in Q, then

$$x_1 \dots x_{j-1}yx_{j+1} \dots x_nax_1 \dots x_n = D_{n+1,n+1,j,y,x}(x_1 \dots x_{j-1}yx_{j+1} \dots x_na) \notin Q,$$

which is a contradiction to Theorem 11. The general case can be obtained using Lemmas 1 and 2. □

Let a word ww' be given with $ed(w, w') = 1$. Then w' is obtained by a change (i. e., $d(w, w') = 1 = 2^0$), either by a deletion or by an insertion. By the Theorems 6, 10 and 12, ww' is in Q provided that $w \in Q$. If $ed(w, w') = 2$ we have again $ww' \in Q$ if two changes, or a deletion and a change, or a change and an insertion are performed (by Theorems 6, 11 and 13). In the remaining cases, in general, primitivity is not preserved. Performing two deletions we can get a non-primitive word, as can be seen from $w = 110^p1$ which results in 110^p1110^p1 and gives $110^p110^p = (110^p)^2 \notin Q$ if we delete the first and last letters of the second copy (note that the statement holds for any length $n \geq 4$ since it holds for any $p \geq 1$). The same holds for two insertions; e. g. the duplication 10^p10^p of $w = 10^p \in Q$ yields $10^p110^p1 = (10^p1)^2$ by inserting a 1 before and after the second copy of 10^p. Furthermore, if we cancel the first letter and insert a 1 before the last 0 in the duplication 110110 of $110 \in Q$, we get $110110 = (110)^2 \notin Q$, again.
Therefore we have a complete picture for the case that the edit distance is at most 2.

4. Concatenation of an Almost Mirror Image

In this section, again, we consider words of the form ww'. However, instead of an almost copy w' of w we choose w' in such a way that the Hamming/edit distance of w' and the mirror image w^R is small.

We start with the remark that, in general, for a primitive word w, the word ww^R is not a primitive word. For example, if we concatenate 100110 and its mirror image, we obtain $100110011001 = (1001)^3 \notin Q$. Moreover, if we delete one letter in w^R, the obtained operation is not primitivity preserving as can be seen from the following counterexample.

Let $w = 01001$. Since $w^R = 10010$, $ww^R = 0100110010$. If we delete the first letter of w^R, then we obtain $010010010 = (010)^3 \notin Q$.

We define formally three operations which are analogous to some with a small Hamming distance $d(w, w')$ considered in the preceding section.

Definition 14. *For any natural numbers n, i, j with $1 \leq i \leq n$ and $2 \leq j \leq n$, all letters $x, y \in V$ with $x \neq y$, and a word $w = x_1x_2 \dots x_n$, $x_i \in V$, of length n, we define the following operations*

$$M_{n,i,x,y} : V^n \to V^{2n}, \text{ and } M'_{n,j,x,y} : V^n \to V^{2n-1}$$

by

$$M_{n,i,x,y}(x_1x_2 \dots x_n) = \begin{cases} x_1x_2 \dots x_nx_nx_{n-1} \dots x_{i+1}yx_{i-1}x_{j-2} \dots x_1, & x_i = x, \\ \textit{undefined}, & \textit{otherwise}, \end{cases}$$

$$M'_{n,j,x,y}(x_1x_2 \dots x_n) = \begin{cases} x_1x_2 \dots x_nx_nx_{n-1} \dots x_{j+1}yx_{j-1}x_{j-2} \dots x_2, & x_j = x, \\ \textit{undefined}, & \textit{otherwise}. \end{cases}$$

For all odd natural numbers n, all mappings $h : V \to V$ with $h(a) \neq a$ for all $a \in V$, and all words $w = x_1x_2 \dots x_n$, $x_i \in V$, of length n, we define the operation $O'_{n,h} : V^n \to V^{2n}$ by

$$O'_{n,h}(x_1x_2 \dots x_n) = x_1x_2 \dots x_nh(x_n)x_{n-1} \dots x_{i+1}h(x_i)x_{i-1}x_{i-2} \dots x_2h(x_1)$$

where $i = \frac{n+1}{2}$.

Theorem 15. *If $w \in Q$ such that $M_{n,i,x,y}(w)$ is defined, then $M_{n,i,x,y}(w) \in Q$ also holds.*

Proof. Let $w = x_1x_2 \dots x_n$. Then

$$w' = M_{n,i,x,y}(w) = x_1x_2 \dots x_{i-1}xx_{i+1}x_{i+2}x_nx_nx_{n-1} \dots x_{i+1}yx_{i-1}x_{i-2} \dots x_1.$$

Let $u_1 = x_1 \dots x_{i-1}$ and $u_2 = x_{i+1} \dots x_n$. Then

$$w = u_1xu_2 \text{ and } w' = u_1xu_2u_2^Ryu_1^R.$$

Let us assume that $w' \notin Q$. Then $w' = v^p$ for some $p \geq 2$ and some word $v \in V^+$.

If p is even and $p > 2$, then $v^{\frac{p}{2}} = w$ and $\frac{p}{2} \geq 2$, which contradicts $w \in Q$. If $p = 2$, then

$$v = u_1xu_2 = u_2^Ryu_1^R. \tag{3}$$

We now count the number of occurrences of x and get

$$\#_x(u_1xu_2) = \#_x(u_1) + 1 + \#_x(u_2)$$

and

$$\#_x(u_2^R y u_1^R) = \#_x(u_2^R) + \#_x(u_1^R) = \#_x(u_2) + \#_x(u_1).$$

Thus

$$\#_x(u_1 x u_2) \neq \#_x(u_2^R y u_1^R)$$

which contradicts (3).

If p is odd, say $p = 2m+1$ for some $m \geq 1$, then $w' = v^m v_1 v_2 v^m$ where $v = v_1 v_2$ and $|v_1| = |v_2|$. If $i > |v|$, then by the construction of w' we get $w' = vzv^R$ with $z = v^{m-1} v_1 v_2 v^{m-1}$ and by our assumption ($w' = v^{2m+1}$) we have $w' = vzv$. Therefore $v = v^R$.

Now let $i \leq |v|$. Then v_1 and v_2 and v satisfy the following conditions:

- $v_2 = v_1^R$ (by construction),
- $v_2^R = ((v_1)^R)^R = v_1$,
- $v^R = (v_1 v_2)^R = v_2^R v_1^R = v_1 v_2 = v$.

Hence in both cases we have $v = v^R$.

Now assume that x occurs in the j-th factor v where $1 \leq j \leq m$ (or equivalently, $(j-1)|v| < i \leq j|v|$), i. e., for this factor v we have $v = v_3 x v_4$. Then

$$w' = v^{j-1} v_3 x v_4 v^{m-j} v_1 v_2 v^{m-j} v_4^R y v_3^R v^{j-1}$$

by definition of $M_{n,i,x,y}$, and

$$w' = v^{j-1} v_3 x v_4 v^{m-j} v_1 v_2 v^{m-j} v_3 x v_4 v^{j-1}$$

by assumption. Therefore

$$v_3 x v_4 = v_4^R y v_3^R$$

Now we can construct a contradiction as above by counting the number of occurrences of x. Let x occur in v_1, i. e., $v_1 = v_5 x v_6$. Then $v_2 = v_6^R y v_5^R$. Thus

$$v = v_1 v_2 = v_5 x v_6 v_6^R y v_5^R$$

Then

$$v^R = v_5 y v_6 v_6^R x v_5^R \neq v$$

in contradiction to $v = v^R$. □

Theorem 16. *If $w \in Q$ such that $M'_{n,j,x,y}(w)$ is defined, then $M'_{n,i,x,y}(w) \in Q$ also holds.*

Proof. Let $w = x_1x_2\dots x_n$. Then

$$M'_{n,j,x,y}(w) = x_1x_2\dots x_nx_nx_{n-1}\dots x_{j+1}yx_{j-1}x_{j-2}\dots x_2.$$

Obviously, $|M_{n,j,x,y}(w)| = 2n+1$, i. e., the length of $M_{n,j,x,y}(w)$ is odd.
If $M'_{n,j,x,y}(w)$ is not a primitive word, then $M_{n,j,x,y}(w) = v^p$ for some primitive word v of odd length and some odd number p with $p \geq 3$, say $p = 2m+1$ with $m \geq 1$. As in the preceding proofs we get $v = v_1x_nv_2$ with

$$M'_{n,j,x,y}(w) = v^mv_1x_n|v_2v^m = (v_1x_nv_2)^mv_1x_n|v_2(v_1x_nv_2)^m$$

and $|v_1| = |v_2|$. Let $|v_1| = q$, i. e., $|v| = 2q+1$.

Let $2 \leq j \leq 2q+1$. Then considering the $(m+1)$-st factor v of $M'_{n,j,x,y}(w)$, we obtain $v = v_1x_n|v_2 = x_1x_2\dots x_qx_n|x_nx_q\dots x_2$. Let $z = x_2x_3\dots x_qx_n$. Then $v = x_1zz^R$. On the other hand, for $2 \leq j \leq 2q+1$, by definition of $M'_{n,j,x,y}(w) = M'_{n,j,x,y}(x_1zz^Rv^{2m})$, $M'_{n,j,x,y}(w)$ does not end with $(zz^R)^R = zz^R$. Thus we have a contradiction to the fact that $M_{n,j,x,y}(w)$ ends with v and therefore with zz^R.

Let $j = 2q+2$. Then the $(2q+2)$-nd letter of w is x. Moreover, the $(2q+2)$-nd letter of w is the first letter of the second factor v of $M'_{n,j,x,y}(w)$ which is x_1. Hence $x = x_1$. On the other hand, by the definition of $M'_{n,j,x,y}(w)$, counting from the end, y is the $(2q+1)$-st letter of $M'_{n,j,x,y}(w)$, which means that y is the first letter of the last factor v of $M_{n,j,x,y}(w)$. Thus $y = x_1$. Hence we get $x = y$ in contradiction to the definition of $M'_{n,j,x,y}$.

Let $2q+3 \leq j \leq n$. Then we can derive a contradiction by analogous argument (in the case that $m(2q+1) < j \leq n$, we get $v = v_1x_nv_2 = x_1zz^R$ by considering the first factor v_1 and the last factor v_2 in $M'_{n,j,x,y}(w)$). □

Finally in this section, we give a result which is the counterpart of Theorem 8. We omit the proof which can be given in analogy to the proof of Theorem 8.

Theorem 17. *For any odd natural number $n \geq 5$, any primitive word q of length n, and any mapping $h : V \to V$ with $h(a) \neq a$ for all $a \in V$, $O'_{n,h}(q)$ is a primitive word.* □

5. Further Operations with an Almost Duplication of Length

First in this section, we discuss the situation where w' in ww' is obtained from w or w^R by large changes.

If we change all letters in the second part, primitivity is not preserved in general. For instance, if we take the primitive word $w = 100110$, then by changing all letters of w we

obtain $100110011001 = (1001)^3 \notin Q$; and starting with the primitive word $w = 10010110$ and changing all letters of w^R we get $1001011010010110 = w^2 \notin Q$.

Theorem 18. *Let w and w' be two words of length n such that $n - d(w, w')$ is a power of 2, then ww' is a primitive word.*

Proof. The proof can be given in a way analogous to the proof of Theorem 6. □

The following definition and result are analogies to $D_{n,n}$ and Theorem 10.

Definition 19. *For any natural numbers n, any natural number i with $1 \leq i \leq n$, and any mapping $h : V \to V$ with $h(a) \neq a$ and $h(h(a)) = a$ for all $a \in V$, we define the operation $D_{n,h} : V^n \to V^{2n-1}$ by*

$$D_{n,h}(x_1x_2 \dots x_n) = x_1x_2 \dots x_nh(x_1x_2 \dots \dots x_{n-1}).$$

Theorem 20. *For any natural numbers n, any natural number i with $1 \leq i \leq n$, any mapping $h : V \to V$ with $h(a) \neq a$ and $h(h(a)) = a$ for all $a \in V$, and any $w \in Q$, $D_{n,h}(w) \in Q$ also holds.*

Proof. Let $w = x_1x_2 \dots x_n$ with $x_j \in V$ for $1 \leq j \leq n$. Then

$$D_{n,h}(x_1x_2 \dots x_n) = x_1x_2 \dots x_nh(x_1 \dots x_{n-1})$$

has an odd length.

Let us suppose that $D_{n,h}(w) \notin Q$, that is, there exist a $p \geq 2$ and $v \in Q$ such that $D_{n,n,h} = v^p$.
Thus p is odd, say $p = 2m + 1$ for some $m \geq 1$. As above there are words v, v_1 and v_2 such that $v = v_1x_nv_2$ and

$$x_1x_2 \dots x_n|h(x_1 \dots x_{n-1}) = (v_1x_nv_2)^mv_1x_n|v_2(v_1x_nv_2)^m.$$

Since $|(v_1x_nv_2)^mv_1| = |v_2(v_1x_nv_2)^m|$, $|v_1| = |v_2|$.
Furthermore $v_2 = h(v_1)$ by definition of $D_{n,h}$. Therefore we get

$$x_1x_2 \dots x_n|h(x_1 \dots x_{n-1}) = (v_1x_nh(v_1))^mv_1x_n|h(v_1)(v_1x_nh(v_1))^m.$$

Thus $(h(v_1)h(x_n)v_1)^mh(v_1) = h(v_1)(v_1x_nh(v_1))^m$, that is,

$$(h(v_1)h(x_n)v_1)^mh(v_1) = (h(v_1)v_1x_n)^mh(v_1).$$

Hence $h(x_n)v_1 = v_1x_n$. Therefore, by Lemma 5, $h(x_n) = x_n$ in contrast to the supposition concerning h. □

By Theorem 18, from a word $w \in Q$ we obtain a primitive word ww' where w' is constructed from w by changing all letters except one letter. This result does not hold for the mirror image, i. e., if one concatenates w with its mirror image and changes all letters of the mirror image besides one letter, in general, one does not obtain a primitive word. For example, if $w = 11100 \in Q$ and $i = 3$, then we obtain $1110011100 = (11100)^2 \notin Q$. However, if we restrict to special positions, then the corresponding statement is true, as shown by the following two theorems.

Definition 21. *For any natural numbers n and i with $1 \leq i \leq n$ and any mapping $h : V \to V$ with $h(a) \neq a$ for all $a \in V$, we define the operations*

$$M_{n,1,h},\ M_{n,n,h} : V^n \to V^{2n}$$

by

$$M_{n,1,h}(x_1x_2\dots x_n) = x_1x_2\dots x_nx_nh(x_{n-1}x_{n-2}\dots x_1),$$
$$M_{n,n,h}(x_1x_2\dots x_n) = x_1x_2\dots x_nh(x_nx_{n-1}\dots x_2)x_1.$$

Theorem 22. *For any $n \geq 2$, any mapping $h : V \to V$ with $h(a) \neq a$ for all $a \in V$ and any $w \in Q$, $M_{n,1,h}(w) \in Q$ also holds.*

Proof. Let $w = x_1x_2\dots x_n$, where $x_i \in V$. Then

$$M_{n,1,h}(w) = x_1x_2\dots x_{n-1}x_nx_nh(x_{n-1}x_{n-2}\dots x_1)$$

has an even length.

Let us suppose that $M_{n,1,h}(w) \notin Q$, that is, there exists a $p \in \mathbb{N}$ and $v \in Q$ such that $x_1x_2\dots x_{n-1}x_nx_nh(x_{n-1}x_{n-2}\dots x_1) = v^p$.

If p is even and $p > 2$, then $v^{\frac{p}{2}} = w$ and $\frac{p}{2} \geq 2$, which contradicts $w \in Q$. If $p = 2$, then $x_1x_2\dots x_{n-1}x_nx_nh(x_{n-1}x_{n-2}\dots x_1) = v^2$, that is,

$$v = x_1x_2\dots x_{n-1}x_n = x_nh(x_{n-1}x_{n-2}\dots x_1).$$

Then $x_n = x_1$ and $x_n = h(x_1)$, which is a contradiction.

If p is odd, then $p = 2m+1$ for some $m \geq 1$ and $v = x_1v'x_nv''$ with $v', v'' \in V^*$, which can be shown as in the proof of Theorem 11. Since

$$x_1\dots x_{n-1}x_n|x_nh(x_{n-1}x_{n-2}\dots x_1) = v^mx_1v'|x_nv''v^m,\ |v'| = |v''|.$$

We distinguish the cases $v' \neq \lambda \neq v''$ and $v' = \lambda = v''$.

Supposing $v' \neq \lambda \neq v''$ and $v' = y_1\dots y_r$ and $v'' = z_1\dots z_r$. Then

$$\begin{aligned} &x_1\dots x_{n-1}x_n|x_nh(x_{n-1}x_{n-2}\dots x_1) \\ &\quad = (x_1y_1\dots y_rx_nz_1\dots z_r)^mx_1y_1\dots y_r|x_nz_1\dots z_r(x_1y_1\dots y_rx_nz_1\dots z_r)^m \end{aligned}$$

and $y_r = x_n$. Since $h(x_1y_1y_2\dots y_r) = z_rz_{r-1}\dots z_1x_n$ by construction, $h(y_r) = x_n$, which contradicts $y_r = x_n$

Supposing $v' = \lambda = v''$, we get

$$x_1\dots x_{n-1}x_n|x_nh(x_{n-1}x_{n-2}\dots x_1) = (x_1x_n)^mx_1|x_n(x_1x_n)^m,$$

which implies $x_n = x_1$ and $x_n = h(x_1)$, so it is a contradiction.

Therefore $Q_{n,1,h}(w) \in Q$. □

Theorem 23. *For any $n \geq 2$, any mapping $h : V \to V$ with $h(a) \neq a$ for all $a \in V$ and any $w \in Q$, $M_{n,n,h}(w) \in Q$ also holds.*

Proof. Let $w = x_1x_2 \dots x_n$. Let us assume that $M_{n,n,h}(w) \notin Q$. Then there is a word $v \in V^+$and a natural number $p \geq 2$ such that $M_{n,n,h}(w) = v^p$.

If $p = 2$, then $v = x_1x_2 \dots x_n = h(x_nx_{n-1} \dots x_2)x_1$. Hence $x_1 = h(x_n)$ and $x_n = x_1$, which is a contradiction. If $p > 2$ and even, then $w = v^{\frac{n}{2}} \in Q$ in contrast to our supposition.

If p is odd, i. e., $p = 2m + 1$ for some $m \geq 1$, then there are words v_1 and v_2 with $v = v_1v_2$, $|v_1| = |v_2|$ and

$$x_1x_2 \dots x_n|h(x_nx_{n-1} \dots h(x_2)x_1 = v^mv_1|v_2v^m.$$

Let $k = |v_1|$. Then

$$v_1 = x_1x_2 \dots x_k \quad \text{and} \quad v_2 = h(x_kx_{k-1} \dots x_2)x_1$$

by definition of $M_{n,n,h}$. Thus $x_{2k+1} = x_1$ and $h(x_{2k+1}) = x_1$ in contrast to the required property of h that $h(a) \neq a$ for all $a \in V$. □

We now define an operation where we duplicate the word, but the copy is shifted some positions to the left. Hence, on one hand, no change is done in the copy, but on the other hand, the position of the letters are changed essentially. An analogous operation is performed where we shift an almost completely changed version of the word.

Definition 24. *For any natural numbers n and i with $1 \leq i \leq n - 1$ and any mapping $h : V \to V$ with $h(a) \neq a$ for all $a \in V$, we define the operation $S_{n,i} : V^n \to V^{2n}$ by*

$$S_{n,i}(x_1x_2 \dots x_n) = x_1x_2 \dots x_ix_1x_2 \dots x_nx_{i+1}x_{i+2} \dots x_n.$$

Theorem 25. *For any natural numbers $n \geq 2$ and i with $1 \leq i \leq n - 1$ and any word $q \in Q$ of length n, $S_{n,i}(q) \in Q$ also holds.*

Proof. Let $q = ww' \in Q$ with $w = x_1x_2 \dots x_{i-1}$ and $w' = x_ix_{i+1} \dots x_n$, where $x_j \in V$ for $1 \leq j \leq n$. Then $S_{n,i}(q) = www'w'$.

Assume $www'w' \notin Q$, that is, there exist a number $p \in \mathbb{N}$, $p > 2$ and a word $v \in Q$ such as $www'w' = v^p$, that is, $w^2(w')^2 = v^p$. It is known, by Lemma 4, $w = u^k, w' = u^l, v = u^m$. Since $ww' \in Q$ and $ww' = u^{k+l}$, we have a contradiction.

Therefore $www'w' \in Q$. □

We mention that an analogous statement does not hold, if one uses the mirror image instead of a copy. The following example shows that then primitivity is not preserved. Let $w = 01$ and $i = 1$; using the mirror image and shifting it by one position to the left we get $0101 \notin Q$.

Finally in the following theorem we present some operations which, together with the above operations, allow the generation of all primitive words of length ≤ 11 (as can be shown by computer calculations) and of a considerable amount of primitive words of length up to twenty.

Theorem 26. *Let $w \in Q$ be a primitive word of length $n \geq 2$ and $x \in V$ and $y \in V$ two different letters of V.*

(i) *Then wx^n and wx^{n-1} and wxy^{n-2} are in Q, too.*

(ii) *If n is even, then $w(xy)^{(n-2)/2}x$ and $w(xy)^{(n-2)/2}y$ are primitive words, too.*

Proof. We omit the easy proofs for (i).

(ii) We only prove the statement for $w(xy)^{(n-2)/2}x$; the other proof can be given analogously.

Assume that $w(xy)^{(n-2)/2}x \notin Q$. Then there is a word $v \in V^+$ such that

$$w(xy)^{(n-2)/2}x = v^p$$

for some $p \geq 2$. Since $w(xy)^{(n-2)/2}x$ has odd length, p and the length of v are odd numbers. Let $p = 2m+1$ for some $m \geq 1$. Thus there are $v_1, v_2 \in V^+$ such that

$$v = v_1v_2,\ |v_1| = |v_2| + 1 \text{ and } w|(xy)^{(n-2)/2}x = v^m v_1 | v_2 v^m.$$

By $w(xy)^{(n-2)/2}x = v^{2m+1}$, we have $v = (xy)^k x$ for some $k \geq 1$, and then $v_1 = (xy)^r$, $v_2 = (xy)^{r-1}x$ and

$$w|(xy)^{(n-2)/2}x = ((xy)^k x)^m (xy)^r | (xy)^{r-1}x(xy)^k x)^m.$$

Since the $(n+2(r-1)+2)$-nd letters in both representations differ, we have a contradiction. □

References

[1] J.-P. ALLOUCHE and J. SHALLIT, *Automatic Sequences. Theory, Applications, Generalizations*. Cambridge University Press, Cambridge, 2003.

[2] D. CALLAN, The value of a primitive word. *Math. Monthly* **107** (2000), 88–89.

[3] P. DÖMÖSI, D. HAUSCHIDT, G. HORVATH, and M. KUDLEK, Some results on small context-free grammars generating primitive words. *Publ. Math. Debrecen* **54** (1999), 667–686.

[4] P. DÖMÖSI, S. HORVATH, and M. ITO, Formal languages and primitive words. *Publ. Math. Debrecen* **42** (1993), 315–321.

[5] P. DÖMÖSI, S. HORVATH, M. ITO, and M. KATSURA, Some results on primitive words, palindroms and polyslender languages. *Publ. Math. Debrecen* **65** (2004), 13–28.

[6] P. DÖMÖSI, M. ITO, and S. MARCUS, Marcus contextual languages consisting of primitive words. *Discrete Mathematics* **308** (2008), 4877–4881.

[7] J. DASSOW, G. M. MARTÍN, and F. J. VICO, Dynamical systems based on regular sets. Submitted.

[8] T. HARJU and D. NOWITZKI, Counting bordered and primitive words of a fixed weight. *Theor. Comp. Sci.* **340** (2005), 273–279.

[9] H. K. HSIAO, Y. T. YEH, and S. S. YU, Square-free-preserving and primitive-preserving homomorphisms. *Acta Math. Hungar.* **101** (2003), 113–130.

[10] M. ITO, M. KATSURA, H. J. SHYR, and S. S. YU, Automata accepting primitive words. *Semigroup Forum* **37** (1988), 45–52.

[11] L. KARI and G. THIERRIN, Word insertions and primitivity. *Utilitas Mathematica* **53** (1998), 49–61.

[12] V. MITRANA, Primitive morphisms. *Inform. Proc. Lett.* **64** (1997), 277–281.

[13] V. MITRANA, Some remarks on morphisms and primitivity. *Bull. EATCS* **62** (1997), 213–216.

[14] GH. PĂUN and G. THIERRIN, Morphisms and primitivity. *Bull. EATCS* **61** (1997), 85–88.

[15] GH. PĂUN, N. SANTEAN, G. THIERRIN, and SH. YU, On the robustness of primitive words. *Discrete Appl. Math.* **117** (2002), 239–252.

[16] H. PETERSEN, On ambiguity of primitive words. In: *Symp. Theor. Aspects Comp. Sci.* '94 Lecture Notes in Computer Science **775**, Springer-Verlag, Berlin, 1994, 679-690.

[17] H. SHYR, Free monoids and languages. Hon Min Book Co., Taichung, 1991.

J. Dassow, B. Truthe (Eds.): Colloquium on the Occasion of the 50th Birthday of Victor Mitrana.
Otto von Guericke Universität Magdeburg, Germany, June 27, 2008. Proceedings, pages 29 – 36.

On the Complexity of the Control Language in Tree Controlled Grammars

RALF STIEBE

Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
PSF 4120, D-39016 Magdeburg, Germany

stiebe@iws.cs.uni-magdeburg.de

Abstract: It is shown that any context-sensitive language can be generated by a tree controlled grammar whose control language is accepted by a deterministic finite automaton with at most 5 states.

Keywords: Tree controlled grammar, regular control, state complexity.

1. Introduction

Regulated rewriting is a common means to increase the generative power of context-free grammars. A context-free core grammar is combined with a mechanism which controls the derivation process. For a thorough introduction to regulated rewriting, see [3, 2]. In tree controlled grammars, introduced by Čulik and Maurer [1], the structure of the derivation trees is restricted as all words belonging to a level of the derivation tree have to be in a given regular language. It was shown by Păun [5] that tree controlled grammars with λ-free context-free core grammars are equivalent to context-sensitive grammars.

Recently, Dassow and Truthe [4] have investigated the generative power of tree controlled grammars when restricting the control language to subfamilies of the regular languages. In particular, they discussed the generative power with respect to the state complexity of the control language. The problem whether the hierarchy defined by the state complexity collapses was left open. We will settle this problem by showing that a state complexity of 5 for the control language is sufficient to obtain the full generative power of tree controlled grammars.

2. Definitions and Basic Notations

We assume that the reader is familiar with the basic concepts of formal language theory. The families of regular, context-free, context-sensitive, recursively enumerable languages are denoted by *REG*, *CF*, *CS*, *RE*, respectively.

For a derivation tree $\mathcal{T}$ of height k for a derivation in a context-free grammar and a number $0 \leq j \leq k$, the *word of level* j is given by the nodes of depth j read from left to right.

Definition 1. *A* tree controlled grammar *is a tuple*

$$G = (N, T, P, S, R),$$

where $G' = (N, T, P, S)$ *is a context-free grammar and* $R \subseteq (N \cup T)^*$ *is a regular language.*

The language $L(G)$ *consists of all words* $w \in T^*$ *generated by* G' *with a derivation tree whose words of all levels (except the last one) are in* R*.*

Let $\mathcal{L}$ be a subfamily of *REG*. By $\mathcal{TC}_\lambda(\mathcal{L})$ ($\mathcal{TC}(\mathcal{L})$, respectively) we denote the family of languages generated by tree controlled grammars with control languages from $\mathcal{L}$ and arbitrary context-free core grammars (λ-free context-free core grammars, respectively). It is known that $\mathcal{TC}_\lambda(\mathit{REG}) = \mathit{RE}$ and $\mathcal{TC}(\mathit{REG}) = \mathit{CS}$ [5].

For $n \geq 1$, let REG_n be the family of languages that are accepted by deterministic finite automata with at most n states. Dassow and Truthe studied the families $\mathcal{TC}(\mathit{REG}_n)$ and obtained the following results, where *E0L* and *ET0L* denote the families generated by E0L and ET0L systems.

Theorem 2 [4].

1. $\mathcal{TC}(\mathit{REG}_1) \subseteq \mathcal{TC}(\mathit{REG}_2) \subseteq \mathcal{TC}(\mathit{REG}_3) \subseteq \cdots \subseteq \mathcal{TC}(\mathit{REG}) = \mathit{CS}$.
2. $\mathit{E0L} = \mathcal{TC}(\mathit{REG}_1) \subset \mathcal{TC}(\mathit{REG}_2)$.
3. $\mathit{ET0L} \subset \mathcal{TC}(\mathit{REG}_4)$.

The principal proof technique for our result will be the simulation of *queue automata* by tree controlled grammars. Intuitively, a queue automaton consists of a finite control with a queue as storage. In a step of the automaton, the first symbol of the queue is removed and a sequence of symbols is appended to the end of the queue.

Definition 3. *A* queue automaton *is a tuple* $\mathcal{A} = (Z, \Sigma, \Gamma, \delta, z_0, F)$*, where* Z *is the finite set of states,* Σ *is the input alphabet,* Γ *is the tape alphabet with* $\Sigma \subseteq \Gamma$*,* $\delta \subset Z \times \Gamma \times Z \times \Gamma^+$ *is the finite transition relation,* $z_0 \in Z$ *is the initial state,* $F \subseteq Z$ *is the set of accepting states.*

The automaton $\mathcal{A}$ *is called a* linearly bounded *queue automaton if* $\delta \subseteq Z \times \Gamma \times Z \times \Gamma$.

A configuration of $\mathcal{A}$ *is given by a pair* (z, w)*, where* $z \in Z$*,* $w \in \Gamma^+$*. The successor relation* $\vdash$ *on the set of configurations is defined as* $(z, aw) \vdash (z', wx)$ *iff* $(z, a, z', x) \in \delta$.

The language accepted by $\mathcal{A}$*,* $L(\mathcal{A})$*, is defined as*

$$L(\mathcal{A}) = \{w \in \Sigma^+ : (z_0, w) \vdash^* (z_f, y), \textit{ for some } z_f \in F, y \in \Gamma^*\}.$$

It is well-known that the language family accepted by queue automata is equal to the family of recursively enumerable languages. The proof is usually performed by constructing an equivalent queue automaton from a given Turing machine, and vice versa. These constructions preserve linear boundedness. Hence, the language family accepted by linearly bounded queue automata equals the family of context-sensitive languages. Moreover, the following technical result on queue automata can be easily shown analogously to similar results for Turing machines (we leave the proof to the reader).

Lemma 4. *Any recursively enumerable (context-sensitive) language can be accepted by a (linearly bounded) queue automaton $\mathcal{A} = (Z, \Sigma, \Gamma, \delta, z_0, \{q\})$, such that*

- *all reachable accepting configurations of $\mathcal{A}$ have the form $(z_f, \square^n), n \geq 1$, for a special symbol $\square \in \Gamma \setminus \Sigma$ (called the blank symbol);*
- *$\delta \cap \{q\} \times \Gamma \times Z \times \Gamma^+ = \emptyset$ (the accepting state q has no successor);*
- *$\delta \cap Z \times \Gamma \times \{z_0\} \times \Gamma^+ = \emptyset$ (the initial state z_0 has no predecessor).*

3. The Result

The idea of the construction is to rewind the accepting computation of a linearly bounded queue automaton by means of a tree controlled grammar. We will first give a simple construction where the size of the deterministic finite automaton for the control language depends on the size of the tape alphabet of the queue automaton. Later, this construction will be refined to limit the number of states by 5.

For a Cartesian product $X_1 \times X_2 \times \cdots \times X_n$, let $\mathrm{pr}_i : X_1 \times X_2 \times \cdots \times X_n \to X_i$ denote the projection on the i-th component, i. e., the mapping

$$\mathrm{pr}_i : X_1 \times X_2 \times X_n \to X_i$$

with

$$\mathrm{pr}_i(x_1, x_2, \ldots, x_n) = x_i.$$

Lemma 5. *For any linearly bounded queue automaton $\mathcal{A}$, there is a tree controlled grammar G such that $L(G) = L(\mathcal{A})$.*

Proof. Let $\mathcal{A} = (Z, \Sigma, \Gamma, \delta, z_0, \{q\})$ be in the normal form as in Lemma 4 with the blank symbol $\square$.

The tree controlled grammar G is obtained as $G = (N, \Sigma, P, S, R)$, where

$$
\begin{aligned}
N &= N_1 \cup N_2, \\
&\quad N_1 = \Gamma \times \Gamma, \\
&\quad N_2 = \Gamma \times \Gamma \times Z, \\
P &= \{p_1\} \cup P_2 \cup P_3 \cup P_4, \\
&\quad p_1 = (\square, \square, q) \to (\square, \square, q)(\square, \square), \\
&\quad P_2 = \{(a, x) \to (y, a) : a, x, y \in \Gamma\}, \\
&\quad P_3 = \{(b, x, z') \to (y, a, z) : x, y \in \Gamma, (z, a, z', b) \in \delta\}, \\
&\quad P_4 = \{(a, b) \to b, (a, b, z_0) \to b : a, b \in \Sigma\}, \\
S &= (\square, \square, q), \\
R &= \{A_1 A_2 \cdots A_n : n \geq 1, A_1 \in N_2, A_i \in N_1 \text{ for } 2 \leq i \leq n, \\
&\qquad \text{pr}_1(A_1) = \text{pr}_2(A_n), \text{pr}_1(A_i) = \text{pr}_2(A_{i-1}) \text{ for } 2 \leq i \leq n\}.
\end{aligned}
$$

A word in R can be seen as the encoding of a configuration of $\mathcal{A}$. More specifically, a configuration $(z, a_1 a_2 a_3 \cdots a_{n-1} a_n)$ is encoded by

$$(a_n, a_1, z)(a_1, a_2)(a_2, a_3) \ldots (a_{n-1}, a_n) \in R.$$

We now consider the tree of a successful derivation in G in detail. As noted above, all level words (except the last one) are encodings of configurations of $\mathcal{A}$. On the root level we find the word $(\square, \square, q)$, i. e., the encoding of the accepting configuration of length 1. Now suppose that some level contains a word $(\square, \square, q)(\square, \square)^{j-1}$, encoding the accepting configuration of length j. If the first symbol is replaced using rule p_1, the next level must have the form $(\square, \square, q)(\square, \square)(x_1, \square) \cdots (x_{j-1}, \square)$, as the remaining symbols are replaced using rules from P_2. The control language requires that $x_i = \square$, $1 \leq i \leq j-1$, and thus the next level word is $(\square, \square, q)(\square, \square)^j$, encoding the accepting configuration of length $j+1$.

Next, consider a level encoding a non-initial configuration $(z', a_1 a_2 \cdots a_n)$ where $z' \neq z_0$, i. e., with the word $(a_n, a_1, z')(a_1, a_2)(a_2, a_3) \cdots (a_{n-1}, a_n)$. The first symbol has to be rewritten using a rule from P_3, the remaining symbols are rewritten using P_2, giving a word of the form $(x_n, a_0, z)(x_1, a_1)(x_2, a_2) \cdots (x_{n-1}, a_{n-1})$, where $(z, a_0, z', a_n) \in \delta$. In view of the control language R, $x_i = a_{i-1}$ has to hold, for $1 \leq i \leq n$. Hence, the next level word describes a configuration $(z, a_0 a_1 a_2 \cdots a_{n-1})$ with $(z, a_0, z', a_n) \in \delta$, i. e., a predecessor configuration. On the other hand, for any predecessor configuration, the encoding word can be obtained at the next level by choosing for the replacement of the first symbol that rule from P_3 which corresponds to the appropriate transition and for the replacement of the other symbols the appropriate rules from P_2.

Finally, consider a level encoding a configuration $(z_0, a_1 a_2 \cdots a_n)$, i. e., with the word $(a_n, a_1, z_0)(a_1, a_2)(a_2, a_3) \cdots (a_{n-1}, a_n)$. The only possibility to rewrite the first symbol is to use the rule $(a_n, a_1, z_0) \to a_1$ if $a_n, a_1 \in \Sigma$. Hence the next level of the derivation tree is the final. The remaining symbols have to be rewritten using rules of P_4, implying

that $a_1, a_2, \ldots, a_n \in \Sigma$ and giving the word $a_1 a_2 \cdots a_n$ as the next level and as the yield of the derivation.

Consequently, a terminal word is generated by G iff it is accepted by $\mathcal{A}$. □

A deterministic finite automaton accepting the control language R in the above proof requires $|\Gamma|^2 + 2$ states, as it must store the second component of the current symbol for comparison with the next symbol and the first component of the first symbol for comparison with the last symbol; moreover two separate initial and failure states are needed. To construct a tree controlled grammar with a control language with a fixed number of states, we modify the grammar as follows. The symbols of the queue automaton are encoded by a bit vector of length $k = \lceil \log_2 |\Gamma| \rceil$. A reverse computation step of $\mathcal{A}$ is simulated in k derivation levels of the tree controlled grammar. In each sub-step, one bit is passed from a symbol to its right neighbour. The details of the construction will be given in the proof of the following theorem.

Theorem 6. $\mathcal{TC}(REG_5) = CS$.

Proof. Let $\mathcal{A} = (Z, \Sigma, \Gamma, \delta, z_0, \{q\})$ be a linearly bounded queue automaton as in the proof of Lemma 5 with the blank symbol $\square \in \Gamma$. Let $k = \lceil \log_2 |\Gamma| \rceil$ and let $\varphi : \Gamma \to \{0,1\}^k$ be an encoding of Γ with $\varphi(\square) = (0, 0, \ldots, 0)$.

The tree controlled grammar G is obtained as $G = (N, \Sigma, P, S, R)$, where

$$
\begin{aligned}
N &= N_1 \cup N_2, \\
&\quad N_1 = \{0,1\}^{k+1}, \\
&\quad N_2 = \{0,1\}^{k+1} \times Z \times \{1,2\ldots,k\}, \\
P &= \{p_1\} \cup P_2 \cup P_3 \cup P_4 \cup P_5, \\
&\quad p_1 = (0^{k+1}, q, 1) \to (0^{k+1}, q, 1) 0^{k+1}, \\
&\quad P_2 = \{(a_1, \ldots, a_k, a_{k+1}) \to (y, a_1, \ldots, a_k) : a_1, \ldots, a_{k+1}, y \in \{0,1\}\}, \\
&\quad P_3 = \{(a_1, \ldots, a_k, a_{k+1}, z, i) \to (y, a_1, \ldots, a_k, z, i+1) : \\
&\qquad\qquad a_1, \ldots, a_{k+1}, y \in \{0,1\}, z \in Z, 1 \le i < k\}, \\
&\quad P_4 = \{(\varphi(b), x, z', k) \to (y, \varphi(a), z, 1) : x, y \in \{0,1\}, (z, a, z', b) \in \delta\}, \\
&\quad P_5 = \{(y, \varphi(b)) \to b, (y, \varphi(b), z_0, 1) \to b : b \in \Sigma, y \in \{0,1\}\}, \\
S &= (0^{k+1}, q, 1), \\
R &= \{A_1 A_2 \cdots A_n : n \ge 1, A_1 \in N_2, A_i \in N_1 \text{ for } 2 \le i \le n, \\
&\qquad\qquad \mathrm{pr}_1(A_1) = \mathrm{pr}_{k+1}(A_n), \mathrm{pr}_1(A_i) = \mathrm{pr}_{k+1}(A_{i-1}) \text{ for } 2 \le i \le n\}.
\end{aligned}
$$

We set $N_{2,i} = \{0,1\}^{k+1} \times Z \times \{i\}$, for $1 \le i \le k$. A word from R encodes a configuration of the queue automaton as follows. A configuration $(z, a_1 a_2 a_3 \cdots a_{n-1} a_n)$ is encoded by

$$(\mathrm{pr}_k(\varphi(a_n)), \varphi(a_1), z, 1)(\mathrm{pr}_k(\varphi(a_1)), \varphi(a_2))(\mathrm{pr}_k(\varphi(a_2)), \varphi(a_3)) \ldots (\mathrm{pr}_k(\varphi(a_{n-1})), \varphi(a_n)).$$

Similar to the proof of Lemma 5, we will now discuss the successful derivation trees in G. On the root level, we find the word $S = (0^{k+1}, q, 1)$, which encodes the accepting configuration of length 1. If the word of some level encodes the accepting configuration of length j and rule p_1 is applied to the first symbol, then the next level encodes the accepting configuration of length $j+1$.

Now consider a level word $\alpha_1 = A_1 A_2 \cdots A_n$ encoding a configuration. The symbols have the forms

$$\begin{aligned} A_1 &= (a_{n,k}, a_{1,1}, a_{1,2} \ldots, a_{1,k}, z', 1), \\ A_i &= (a_{i-1,k}, a_{i,1}, a_{i,2}, \ldots, a_{i,k}), \text{ for } 2 \leq i \leq n. \end{aligned}$$

As α_1 encodes a configuration of the queue automaton, $(a_{i,1}, a_{i,2}, \ldots, a_{i,k}) = \varphi(x_i)$ has to hold for appropriate $x_i \in \Gamma$, $1 \leq i \leq n$. The symbol A_1 has to be rewritten using a rule from P_3 (with the exception of $z' = z_0$, discussed below), which implies that the remaining symbols are replaced using rules of P_2. In view of R, the next level has to be labelled $\alpha_2 = A_1^2 A_2^2 \cdots A_n^2$ with

$$\begin{aligned} A_1^2 &= (a_{n,k-1}, a_{n,k}, a_{1,1}, a_{1,2} \ldots, a_{1,k-1}, z', 2), \\ A_i^2 &= (a_{i-1,k-1}, a_{i-1,k}, a_{i,1}, a_{i,2}, \ldots, a_{i,k-1}), \text{ for } 2 \leq i \leq n. \end{aligned}$$

By analogous arguments for words in $N_{2,j} N_1^*$, $2 \leq j < k$, we obtain after $k-1$ levels the word $\alpha_k = A_1^k A_2^k \cdots A_n^k \in R$ with

$$\begin{aligned} A_1^k &= (a_{n,1}, a_{n,2}, \ldots, a_{n,k}, a_{1,1}, z', k), \\ A_i^k &= (a_{i-1,1}, a_{i-1,2}, \ldots, a_{i-1,k}, a_{i,1}), \text{ for } 2 \leq i \leq n. \end{aligned}$$

On the next level, A_1^k is replaced using a rule from P_4 and the remaining symbols using a rule from P_2. One obtains a word $\alpha_{k+1} = A_1^{k+1} A_2^{k+1} \cdots A_n^{k+1} \in R$ with

$$\begin{aligned} A_1^{k+1} &= (a_{n-1,k}, b_{1,1}, \ldots, b_{1,k}, z, 1), \\ A_2^{k+1} &= (b_{1,k}, a_{1,1}, \ldots, a_{1,k}), \\ A_i^{k+1} &= (a_{i-2,k}, a_{i-1,1}, a_{i-1,2}, \ldots, a_{i-1,k}) = A_{i-1}, \text{ for } 3 \leq i \leq n, \end{aligned}$$

where $(b_{1,1}, \ldots, b_{1,k}) = \varphi(x_0)$, $(z, x_0, z', x_n) \in \delta$. Hence, the configuration encoded by α_{k+1} is a predecessor of that encoded by α_1. On the other hand, the encoding of any predecessor configuration can be reached by choosing the appropriate rules.

Finally, if and only if a level word describes an initial configuration of $\mathcal{A}$, the input word can be reached as terminal word on the next level by using the rules of P_5.

The control language R can be accepted by a deterministic finite automaton with six states. However, note that the rules of G imply that any derivable sentential form over N is a word from $N_2 N_1^*$. Instead of R, we can use any regular language R' such

that $R' \cap N_2 N_1^* = R$. Such a language is the one accepted by the deterministic finite automaton

$$\mathcal{M} = (\{z_{00}, z_{01}, z_{10}, z_{11}, \mathit{fail}\}, N \cup \Sigma, f, z_{00}, \{z_{00}, z_{11}\})$$

with the transition function f defined as

$$\begin{aligned} f(z_{ab}, (b, a_1, \ldots, a_k)) &= z_{aa_k}, \text{ for } a, b, a_1, \ldots, a_k \in \{0,1\}; \\ f(z_{ab}, (a_0, a_1, \ldots, a_k, z, i)) &= z_{a_0 a_k}, \text{ for } a, b, a_0, a_1, \ldots, a_k \in \{0,1\}, z \in Z, i \in \{1, \ldots, k\}; \\ f(z, A) &= \mathit{fail}, \text{ in all other cases.} \end{aligned}$$

Obviously, $\mathcal{M}$ accepts only words over N. When receiving an input from $N_2 N_1^*$, $\mathcal{M}$ works as follows. A state of the form z_{ab} is meant to store two bits: the first bit of the symbol from N_2 is a, while the last bit of the currently read symbol is b. If the first bit of the next symbol is unequal to the stored one, the input is rejected. Finally, $\mathcal{M}$ accepts iff it reaches a state z_{aa}, thus if additionally the last bit of the last symbol is equal to the first of the first one. □

Corollary 7. $\mathcal{TC}_\lambda(\mathit{REG}_5) = \mathit{RE}$.

Proof. Note that any recursively enumerable language L' can be expressed as $h(L)$ for appropriate homomorphism h and context-sensitive language L. In the construction of the tree controlled grammar for L as in the proof of Theorem 6, one has just to change the rules in P_5 by replacing on the right-hand sides the letters from Σ by their images under h. □

References

[1] K. ČULIK and H. A. MAURER, Tree controlled grammars. *Computing* **19** (1977), 129–139.

[2] J. DASSOW, GH. PĂUN, and A. SALOMAA, Grammars with controlled derivations. In: G. ROZENBERG and A. SALOMAA (eds.), *Handbook of Formal Languages, Volume II*, Springer-Verlag, Berlin, 1997, 101–154.

[3] J. DASSOW and GH. PĂUN, *Regulated Rewriting in Formal Language Theory*, EATCS Monographs in Theoretical Computer Science 18, Springer-Verlag, Berlin, 1989.

[4] J. DASSOW and B. TRUTHE, On Two Hierarchies of Subregularly Tree Controlled Languages. In: C. CÂMPEANU and G. PIGHIZZINI (eds.), *10th International Workshop on Descriptional Complexity of Formal Systems, DCFS 2008, Charlottetown, Prince Edward Island, Canada, July 16–18, 2008, Proceedings*. University of Prince Edward Island, 2008, 145–156.

[5] GH. PĂUN, On the generative capacity of conditional grammars. *Information and Control* **43** (1979), 178–186.

J. Dassow, B. Truthe (Eds.): Colloquium on the Occasion of the 50th Birthday of Victor Mitrana.
Otto von Guericke Universität Magdeburg, Germany, June 27, 2008. Proceedings, pages 37 – 52.

On Small Accepting Networks of Evolutionary Processors with Regular Filters

BIANCA TRUTHE

Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
PSF 4120, D-39016 Magdeburg, Germany

truthe@iws.cs.uni-magdeburg.de

Abstract: We show that every context-sensitive language can be accepted by an accepting network of non-increasing evolutionary processors with one substitution processor and one output node whose communication is controlled by regular languages. Every recursively enumerable language can be accepted by a network with three evolutionary processors: one substitution processor, one insertion processor and one output node. Also with insertion and deletion processors only (without substitution nodes), all recursively enumerable languages can be accepted. Then one insertion node, one deletion node and one output node are sufficient.

Keywords: Accepting networks of evolutionary processors, regular filters, size complexity.

1. Introduction

Motivated by some models of massively parallel computer architectures (see [10, 9]) networks of language processors have been introduced in [6] by ERZSÉBET CSUHAJ-VARJÚ and ARTO SALOMAA. Such a network can be considered as a graph where the nodes are sets of productions and at any moment of time a language is associated with a node.

Inspired by biological processes, JUAN CASTELLANOS, CARLOS MARTÍN-VIDE, VICTOR MITRANA and JOSÉ M. SEMPERE introduced in [4] a special type of networks of language processors which are called networks with evolutionary processors because the allowed productions model the point mutation known from biology. The sets of productions have to be substitutions of one letter by another letter or insertions of letters or deletion of letters; the nodes are then called substitution node or insertion node or deletion node, respectively. Results on networks of evolutionary processors can be found e. g. in [4, 5, 3, 2].

Accepting networks of evolutionary processors with regular filters were first investigated by JÜRGEN DASSOW and VICTOR MITRANA in [7]. Especially, they have shown that every network of non-increasing processors accepts a context-sensitive language.

In [8] and [1], we investigated the generative capacity of networks with evolutionary processors where only two types of nodes are allowed. Especially, we proved two results:

- Networks with substitution nodes and insertion nodes (but without deletion nodes) generate all context-sensitive languages and one substitution node and one insertion node are sufficient.
- Networks with insertion nodes and deletion nodes (but without substitution nodes) generate all recursively enumerable languages and one insertion node and one deletion node are sufficient.

In the present paper, we show the dual case:

- Every context-sensitive language can be accepted by an accepting network of evolutionary processors with regular filters and with one substitution node, one deletion node and one output node.
- Every recursively enumerable language can be accepted by an accepting network of evolutionary processors with regular filters and with one insertion node, one deletion node and one output node.

Further, we show that every recursively enumerable language can be accepted by a network with one substitution processor, one insertion processor and one output node.

Whereas networks consisting of substitution processors only cannot generate other languages than finite ones, accepting pure substitution networks can accept infinite languages. The reason is that generating networks start with a finite set (and substitution nodes cannot increase the length of the words) while accepting networks can get infinitely many input words. We show that all context-sensitive languages can be accepted by networks of substitution processors and that one substitution processor is sufficient (apart from an output node).

2. Definitions

We assume that the reader is familiar with the basic concepts of formal language theory (see e. g. [13]). We here only recall some notations used in the paper.

By V^* we denote the set of all words (strings) over V (including the empty word λ). The length of a word w is denoted by $|w|$.

In the proofs we shall often add new letters of an alphabet U to a given alphabet V. In all these situations, we assume that $V \cap U = \emptyset$.

A phrase structure grammar is specified as a quadruple $G = (N, T, P, S)$ where N is a set of non-terminals, T is a set of terminals, P is a finite set of productions which are written as $\alpha \to \beta$ with $\alpha \in (N \cup T)^* \setminus T^*$ and $\beta \in (N \cup T)^*$, and $S \in N$ is the axiom.

The grammar G is called monotone, if $|\alpha| \leq |\beta|$ holds for every rule $\alpha \to \beta$ of P where the exception $S \to \lambda$ is permitted if S does not occur on a right hand side of a rule. A monotone grammar is in Kuroda normal form if all its productions have one of the following forms:

$$AB \to CD,\ A \to CD,\ A \to x, \text{ where } A,B,C,D \in N,\ x \in N \cup T;$$

again, $S \to \lambda$ is permitted if S does not occur on a right hand side of a rule.

We call a production $\alpha \to \beta$ a

- substitution if $|\alpha| = |\beta| = 1$,
- deletion if $|\alpha| = 1$ and $\beta = \lambda$.

We regard insertion as a counterpart of deletion. We write $\lambda \to a$, where a is a letter. The application of an insertion $\lambda \to a$ derives from a word w any word $w_1 a w_2$ with $w = w_1 w_2$ for some (possibly empty) words w_1 and w_2.

We now introduce the basic concept of this paper, the accepting networks of evolutionary processors.

Definition 1.

(i) *An accepting network of evolutionary processors of size n is a tuple*

$$\mathcal{N}^{(n)} = (U, V, N_1, N_2, \ldots, N_n, E, j, O)$$

where

- *U and V are finite alphabets (the input and network alphabet, resp.), $U \subseteq V$,*
- *for $1 \leq i \leq n$, $N_i = (M_i, I_i, O_i)$ where*
  - *M_i is a set of evolution rules of a certain type, $M_i \subseteq \{a \to b \mid a,b \in V\}$ or $M_i \subseteq \{a \to \lambda \mid a \in V\}$ or $M_i \subseteq \{\lambda \to b \mid b \in V\}$,*
  - *I_i and O_i are regular sets over V,*
- *E is a subset of $\{1,2,\ldots,n\} \times \{1,2,\ldots,n\}$,*
- *j is a natural number such that $1 \leq j \leq n$, and*
- *O is a subset of $\{1,2,\ldots,n\}$.*

(ii) *A configuration C of $\mathcal{N}^{(n)}$ is an n-tuple $C = (C(1), C(2), \ldots, C(n))$ if $C(i)$ is a subset of V^* for $1 \leq i \leq n$.*

(iii) *Let $C = (C(1), C(2), \ldots, C(n))$ and $C' = (C'(1), C'(2), \ldots, C'(n))$ be two configurations of $\mathcal{N}^{(n)}$. We say that C derives C' in one*

- *evolution step (written as $C \Longrightarrow C'$) if, for $1 \leq i \leq n$, $C'(i)$ consists of all words $w \in C(i)$ to which no rule of M_i is applicable and of all words w for which there are a word $v \in C(i)$ and a rule $p \in M_i$ such that $v \Longrightarrow_p w$ holds,*

– *communication step (written as $C \vdash C'$) if, for $1 \leq i \leq n$,*

$$C'(i) = (C(i) \setminus O_i) \cup \bigcup_{(k,i) \in E} C(k) \cap O_k \cap I_i.$$

The computation of a network $\mathcal{N}^{(n)}$ on an input word $w \in U^*$ is a sequence of configurations $C_t^w = (C_t^w(1), C_t^w(2), \ldots, C_t^w(n))$, $t \geq 0$, such that

– *$C_0^w = (C_0^w(1), C_0^w(2), \ldots, C_0^w(n))$ where $C_0^w(j) = \{w\}$ and $C_0^w(i) = \emptyset$ for $1 \leq i \neq j \leq n$,*

– *for any $t \geq 0$, C_{2t}^w derives C_{2t+1}^w in one evolution step: $C_{2t}^w \Longrightarrow C_{2t+1}^w$,*

– *for any $t \geq 0$, C_{2t+1}^w derives C_{2t+2}^w in one communication step: $C_{2t+1}^w \vdash C_{2t+2}^w$.*

(iv) *The languages $L_{\mathrm{w}}(\mathcal{N})$ weakly accepted by $\mathcal{N}$ and $L_{\mathrm{s}}(\mathcal{N})$ strongly accepted by $\mathcal{N}$ are defined as*

$$L_{\mathrm{w}}(\mathcal{N}) = \{\, w \in U^* \mid \exists t \geq 0 \, \exists o \in O : C_t^w(o) \neq \emptyset \,\},$$
$$L_{\mathrm{s}}(\mathcal{N}) = \{\, w \in U^* \mid \exists t \geq 0 \, \forall o \in O : C_t^w(o) \neq \emptyset \,\},$$

where $C_t^w = (C_t^w(1), C_t^w(2), \ldots, C_t^w(n))$, $t \geq 0$ is the computation of $\mathcal{N}$ on w.

Intuitively a network with evolutionary processors is a graph consisting of some, say n, nodes $N_1, N_2, \ldots, N_n$ (called processors) and the set of edges given by E such that there is a directed edge from N_k to N_i if and only if $(k,i) \in E$. The node N_j is called the input node; every node N_o with $o \in O$ is called an output node. Any processor N_i consists of a set of evolution rules M_i, an input filter I_i and an output filter O_i. We say that N_i is a substitution node or a deletion node or an insertion node if $M_i \subseteq \{a \to b \mid a, b \in V\}$ or $M_i \subseteq \{a \to \lambda \mid a \in V\}$ or $M_i \subseteq \{\lambda \to b \mid b \in V\}$, respectively. The input filter I_i and the output filter O_i control the words which are allowed to enter and to leave the node, respectively. With any node N_i and any time moment $t \geq 0$ we associate a set $C_t(i)$ of words (the words contained in the node at time t). Initially, the input node N_j contains an input word w; all other nodes do not contain words. In an evolution step, we derive from $C_t(i)$ all words applying rules from the set M_i. In a communication step, any processor N_i sends out all words $C_t(i) \cap O_i$ (which pass the output filter) to all processors to which a directed edge exists (only the words from $C_t(i) \setminus O_i$ remain in the set associated with N_i) and, moreover, it receives from any processor N_k such that there is an edge from N_k to N_i all words sent by N_k and passing the input filter I_i of N_i, i. e., the processor N_i gets in addition all words of $(C_t(k) \cap O_k) \cap I_i$. We start with an evolution step and then communication steps and evolution steps are alternately performed. The language accepted consists of all words w such that if w is given as an input word in the node N_j then, at some moment t, $t \geq 0$, one output node contains a word (weak acceptance) or all output nodes contain a word (strong acceptance).

3. Networks of Non-Increasing Nodes

Deletion and substitution nodes do not increase the length of the words. Such nodes are also called non-increasing. In a network with only non-increasing nodes, the length of every word in every node at any step in the computation is bounded by the length of the input word.

In this section, we show that every context-sensitive language can be accepted by an accepting network of evolutionary processors with deletion and substitution nodes but no insertion nodes. Especially, one substitution node (which is the input node), one deletion node and one output node are sufficient.

This is the inverse situation to that one considered in [8], where we have shown that networks with insertion nodes and substitution nodes (but without deletion nodes) generate all context-sensitive languages and one insertion node and one substitution node are sufficient.

Theorem 2. *For any context-sensitive language L, there is an accepting network $\mathcal{N}$ of evolutionary processors with exactly one substitution node, one deletion node and one output node without rules that weakly and strongly accepts the language L:*

$$L = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N}).$$

Proof. Let L be a context-sensitive language and $G = (N,T,P,S)$ be a grammar in Kuroda normal form with $L(G) = L$. Let $R_1, R_2, \ldots, R_7$ be the following sets:

$$\begin{aligned}
R_1 &= \{\, x \to x_{p,0},\ x_{p,0} \to A \mid A \to x \in P,\ A \in N,\ x \in N \cup T \,\},\\
R_2 &= \{\, C \to C_{p,1} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\},\\
R_3 &= \{\, D \to D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\},\\
R_4 &= \{\, C_{p,1} \to C_{p,3} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\},\\
R_5 &= \{\, D_{p,2} \to D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\},\\
R_6 &= \{\, C_{p,3} \to A \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\},\\
R_7 &= \{\, D_{p,4} \to B \mid p = AB \to CD \in P,\ A,B,C,D \in N \,\}.
\end{aligned}$$

We construct a network of evolutionary processors

$$\mathcal{N} = (T, V, (M_1, V^*, O_1), (M_2, I_2, V^*), (\emptyset, I_3, \emptyset), \{\,(1,2),(2,1),(1,3),(2,3)\,\}, 1, \{3\})$$

with

$$\begin{aligned}
V &= N \cup T \cup \bigcup_{p=A\to x} \{x_{p,0}\} \cup \bigcup_{\substack{p=A\to CD\\ p=AB\to CD}} \{C_{p,1}, D_{p,2}, C_{p,3}, D_{p,4}\},\\
M_1 &= R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6 \cup R_7,\\
O_1 &= \{S, \lambda\} \cup V^* \setminus ((N \cup T)^* \bar{O} (N \cup T)^*),
\end{aligned}$$

where

$$\begin{aligned}\bar{O} = &\{\, x_{p,0} \mid p = A \to x \in P,\ A \in N,\ x \in N \cup T \,\} \\ &\cup \{\, C_{p,1} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\} \\ &\cup \{\, C_{p,1}D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\} \\ &\cup \{\, C_{p,3}D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\} \\ &\cup \{\, C_{p,3}D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\} \\ &\cup \{\, AD_{p,4} \mid p = AB \to CD \in P,\ A,B,C,D \in N \,\} \\ &\cup \{\, \lambda \,\},\end{aligned}$$

and

$$\begin{aligned}M_2 &= \{\, D_{p,4} \to \lambda \mid p = A \to CD \in P,\ A,B,C,D \in N \,\}, \\ I_2 &= (N \cup T)^* \{\, AD_{p,4} \mid p = A \to CD \in P,\ A,B,C,D \in N \,\} (N \cup T)^*, \\ I_3 &= \begin{cases} \{S, \lambda\} & \text{if } \lambda \in L, \\ \{S\} & \text{otherwise.} \end{cases}\end{aligned}$$

The network $\mathcal{N}$ has only one output node. Therefore, there is no difference between weak and strong acceptance, and we write $L(\mathcal{N})$ for the language accepted by the network $\mathcal{N}$.

First, we prove that every word $w \in L(G)$ is accepted by the network $\mathcal{N}$.

If $\lambda \in L(G)$ then $(\{\lambda\}, \emptyset, \emptyset) \Longrightarrow (\{\lambda\}, \emptyset, \emptyset) \vdash (\emptyset, \emptyset, \{\lambda\})$ and λ is accepted by the network $\mathcal{N}$.

Any derivation $w \Longrightarrow^* v$ with $v \neq \lambda$ of the grammar G can be simulated by the network $\mathcal{N}$ in reverse direction (by a reduction $v \Longrightarrow^* w$). We show that the application of a rule of the grammar G can be simulated by the network $\mathcal{N}$ in reverse direction. A direct reduction $v \Longrightarrow w$ always starts in the first node, the first step is an evolution step, and ends in the first node after a communication step (so the next step would be an evolution step again). Then $v \in C_{2t}(1)$ and $w \in C_{2(t+k)}(1)$ for two numbers $t \geq 0$ and $k > 0$. In the sequel, A, B, C, D are non-terminals, x is a non-terminal or terminal symbol and $w_1 w_2 \in (N \cup T)^*$.

Case 1. Application of a rule $p = A \to x \in P$ to a word $w_1 A w_2$.

This application leads in the grammar G to the word $w_1 x w_2$. We assume that the word $w_1 x w_2$ is in the first node at some moment before an evolution step ($w_1 x w_2 \in C_{2t}(1)$). We apply the rule $x \to x_{p,0} \in R_1$ and obtain the word $w_1 x_{p,0} w_2$ which cannot pass the output filter, so it remains in the first node. Then we apply the rule $x_{p,0} \to A \in R_1$ and obtain $w_1 A w_2 \in (N \cup T)^*$. This word also does not pass the output filter, so it remains in the first component: $w_1 A w_2 \in C_{2(t+2)}(1)$. Hence, the application of a rule $p = A \to x \in P$ to a word $w_1 A w_2$ can be simulated reversely in two evolution steps (the two corresponding communication steps have no effect).

Case 2. Application of a rule $p = AB \to CD \in P$ to a word $w_1 AB w_2$.

We assume $w_1 CD w_2 \in C_{2t}(1)$. This word is changed to $w_1 C_{p,1} D w_2$ (by an appropriate rule of R_2) which cannot pass the output filter, so it remains in the first node. It

is then changed to $w_1C_{p,1}D_{p,2}w_2$ (by R_3), and further, without leaving the first node, changed to $w_1C_{p,3}D_{p,2}w_2$ (by R_4), to $w_1C_{p,3}D_{p,4}w_2$ (by R_5), to $w_1AD_{p,4}w_2$ (by R_6) and finally to w_1ABw_2 (by R_7). This word is not communicated in the next step, since it cannot pass the output filter and we have $w_1ABw_2 \in C_{2(t+6)}(1)$. Hence, the application of a rule $p = AB \to CD \in P$ to a word w_1ABw_2 can be simulated reversely in six evolution steps (the six corresponding communication steps have no effect).

Case 3. Application of a rule $p = A \to CD \in P$ to a word w_1Aw_2.

We assume $w_1CDw_2 \in C_{2t}(1)$. As in the case before, this word is changed to the word $w_1C_{p,1}Dw_2$ and further, without leaving the first node, changed to $w_1C_{p,1}D_{p,2}w_2$ (by a rule of R_3), to $w_1C_{p,3}D_{p,2}w_2$ (by R_4), to $w_1C_{p,3}D_{p,4}w_2$ (by R_5), and to $w_1AD_{p,4}w_2$ (by R_6). This word passes the output filter of the node and the input put filter of the second node. So we have $w_1AD_{p,4}w_2 \in C_{2(t+5)}(2)$. The second node changes the word to w_1Aw_2. In the next communication step, this word moves to the first node and we obtain $w_1Aw_2 \in C_{2(t+6)}(1)$. Hence the application of a rule $p = A \to CD \in P$ to a word w_1Aw_2 can be simulated reversely in six evolution steps and two effective communication steps (the other four have no effect).

For any derivation $S \Longrightarrow^* w$ in the grammar G to a terminal word $w \in T^+$, there is a computation $C_0, C_1, \ldots, C_{2t+1}$ with $C_0 = (\{w\}, \emptyset, \emptyset)$ and $C_{2t+1} = (C_{2t+1}(1), C_{2t+1}(2), \emptyset)$ with $S \in C_{2t+1}(1) \cup C_{2t+1}(2)$ (the final reduction to S is achieved by some rule $x \to S$ in the first node or – if the first direct derivation is $S \Longrightarrow_p AB$ – by the rule $B_{p,4} \to \lambda$ in the second node). From both nodes, S reaches the third node in the next communication step. Hence, $S \in C_{2(t+1)}(3)$.

Thus, we have the inclusion $L(G) \subseteq L(\mathcal{N})$. We now show $L(\mathcal{N}) \subseteq L(G)$.

Let us first consider the case that the computation starts with λ in the first node. No rule can be applied. The word leaves the node (because it passes the output filter O_1). It enters the third node if λ belongs to $L(G)$, otherwise the word is lost and there is no word in the network any more. Hence, if λ is not in $L(G)$ the third node never obtains a word. Thus, $\lambda \in L(\mathcal{N})$ if and only if $\lambda \in L(G)$.

We now show that if a word $w \in T^+$ is accepted by the network then, at some time, the symbol S enters the third node and then a derivation $S \Longrightarrow^* w$ in G has been simulated reversely by reducing w to S in the network.

The computation starts in the first node with a word $w \in T^+$. The word can only be accepted if it can be reduced to S or λ (only these can enter the third node). No word can be reduced to λ, because if a word enters the deletion node, then it contains at least two letters A and $D_{p,4}$ for a rule $p = A \to CD$ and only $D_{p,4}$ can be deleted before the word moves back to the first node. Hence, if a word is accepted then it can be reduced to S (after the last reduction step, it moves from the first or second node to the output node).

In the sequel, A, B, C, D denote non-terminals, x denotes a non-terminal or terminal symbol and $w_1w_2 \in (N \cup T)^*$.

We now consider a word $w = w_1xw_2$ in the first node after an even number of computation steps (the next step is an evolution step). If we can apply a rule $x \to x_{p,0} \in R_1$

to w then we obtain the word $w_1 x_{p,0} w_2$ which does not leave the node (because it does not pass the output filter). If now another rule than $x_{p,0} \to A$ is applied, then the word passes the output filter and leaves the network. If $x_{p,0} \to A$ is applied, then we obtain the word $w_1 A w_2$ which does not leave the node and the derivation $w_1 A w_2 \Longrightarrow w_1 x w_2$ is possible in G (due to the construction of the set R_1).

Let us now consider a word $w = w_1 C D w_2$ in the first node after an even number of computation steps (the next step is an evolution step). If a rule of R_1 is applied then we have the situation described for the word $w = w_1 x w_2$. If we can apply another rule, then we have one of the following cases:

Case 1. Application of a rule $D \to D_{p,2} \in R_3$.
This leads to the word $w_1 C D_{p,2} w_2$ in the first node, which is then sent out. Since the other nodes do not accept it, the word is lost.

Case 2. Application of a rule $C \to C_{p,1} \in R_2$.
This leads to the word $w_1 C_{p,1} D w_2$ in the first node which is kept in the node. If the next evolution step does not yield the word $w_1 C_{p,1} D_{p,2} w_2$, then the word disappears in the next communication step. (Here is the reason why the rules in R_1 make the 'detour' via intermediate symbols. If there would be direct rules $x \to A$, we could apply them here and had more cases to discuss.) Let us assume, we obtain $w_1 C_{p,1} D_{p,2} w_2$, then this word is kept in the first node. The next evolution step yields $w_1 C_{p,3} D_{p,2} w_2$ or the word is lost. Also this word remains in the node. The fourth evolution step leads to $w_1 C_{p,3} D_{p,4} w_2$ or the word is lost. This word remains in the node, too. The fifth evolution step leads to $w_1 A D_{p,4} w_2$ or the word is lost. There are two possibilities for the rule p that belongs to $D_{p,4}$.

Case 2.1. $p = A \to CD$. In this case, the word $w_1 A D_{p,4} w_2$ is sent out and caught by the second node. The second node deletes the symbol $D_{p,4}$. In the next communication step, the word $w_1 A w_2$ is sent back to the first node. The described six evolution steps (together with the corresponding communication steps) represent the inverse of the derivation $w_1 A w_2 \Longrightarrow w_1 C D w_2$ in G.

Case 2.2. $p = AB \to CD$. In this case, the word $w_1 A D_{p,4} w_2$ remains in the first node. The next evolution step yields $w_1 A B w_2$ or a word that is lost, because applying any rule of $R_1 \cup R_2 \cup R_3$ (rules of R_5, R_6, R_7 and other rules of R_4 are not applicable) leads to a word which passes the output filter but no input filter. The word $w_1 A B w_2$ remains in the first node. The described six evolution steps (the communication steps have no effect) represent the inverse of the derivation $w_1 A B w_2 \Longrightarrow w_1 C D w_2$ in G.

Hence, in this case, the derivation $w_1 A w_2 \Longrightarrow w_1 C D w_2$ or $w_1 A B w_2 \Longrightarrow w_1 C D w_2$ (which in G is obtained by the initially chosen rule p) is simulated reversely.

Other rules are not applicable to the word w.

By the case distinction above, we have shown that, for every reduction $w \Longrightarrow v$ in the network $\mathcal{N}$, the derivation $v \Longrightarrow w$ is possible in the grammar G. Hence, if a word w can be reduced to S, then the derivation $S \Longrightarrow^* w$ exists in G. Together, we obtain that if a

word w is accepted by the network $\mathcal{N}$ then it is generated by the grammar G.

With the first part of the proof, we obtain $L(G) = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N}) = L$. □

In generating networks, only substitution nodes yield not more than finite languages. But accepting networks can accept infinitely many input words. Surprisingly, even every context-sensitive language can be accepted by a network with only one substitution node (and one output node without rules). The trick is that the substitution processor can simulate the deletion by marking symbols as deleted. The filters then 'ignore' the deletion markers.

Theorem 3. *For any context-sensitive language L, there is an accepting network $\mathcal{S}$ of evolutionary processors with exactly one substitution node and one output node without rules that weakly and strongly accepts the language L:*

$$L = L_{\mathrm{w}}(\mathcal{S}) = L_{\mathrm{s}}(\mathcal{S}).$$

Proof. Let L be a context-sensitive language and $G = (N, T, P, S)$ be a grammar in Kuroda normal form with $L(G) = L$. The network $\mathcal{S}$ is constructed similarly to the network $\mathcal{N}$ in the proof of Theorem 2.

Let $R_1, R_2, \ldots, R_7$ be the sets used for $\mathcal{N}$ and let R_8 be the additional set

$$R_8 = \{\, D_{p,4} \to \textvisiblespace \mid p = A \to CD \in P,\ A, B, C, D \in N \,\}.$$

We construct a network of evolutionary processors

$$\mathcal{S} = (T, V^{\mathcal{S}}, (M_1^{\mathcal{S}}, \emptyset, O_1^{\mathcal{S}}), (\emptyset, I_2^{\mathcal{S}}, \emptyset), \{\,(1,2)\,\}, 1, \{2\})$$

with

$$\begin{aligned}
V^{\mathcal{S}} &= N \cup T \cup \{\textvisiblespace\} \cup \bigcup_{p=A\to x} \{x_{p,0}\} \cup \bigcup_{\substack{p=A\to CD \\ p=AB\to CD}} \{C_{p,1}, D_{p,2}, C_{p,3}, D_{p,4}\}, \\
M_1^{\mathcal{S}} &= R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6 \cup R_7 \cup R_8, \\
O_1^{\mathcal{S}} &= \{\textvisiblespace\}^*\{S\}\{\textvisiblespace\}^* \cup \{\lambda\} \cup V^* \setminus ((N \cup T \cup \{\textvisiblespace\})^* \bar{O}' (N \cup T \cup \{\textvisiblespace\})^*),
\end{aligned}$$

where

$$\begin{aligned}
\bar{O}^{\mathcal{S}} = {} & \{\, x_{p,0} \mid p = A \to x \in P,\ A \in N,\ x \in N \cup T \,\} \\
& \cup \{\, C_{p,1} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P \,\} \\
& \cup \{\, C_{p,1} \varepsilon D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \varepsilon \in \{\textvisiblespace\}^* \,\} \\
& \cup \{\, C_{p,3} \varepsilon D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \varepsilon \in \{\textvisiblespace\}^* \,\} \\
& \cup \{\, C_{p,3} \varepsilon D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \varepsilon \in \{\textvisiblespace\}^* \,\} \\
& \cup \{\, A \varepsilon D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \varepsilon \in \{\textvisiblespace\}^* \,\} \\
& \cup \{\, \lambda \,\},
\end{aligned}$$

and

$$I_2^{\mathcal{S}} = \begin{cases} \{␣\}^*\{S\}\{␣\}^* \cup \{\lambda\} & \text{if } \lambda \in L, \\ \{␣\}^*\{S\}\{␣\}^* & \text{otherwise.} \end{cases}$$

The network $\mathcal{S}$ has only one output node. Therefore, there is no difference between weak and strong acceptance, and we write $L(\mathcal{S})$ for the language accepted by the network $\mathcal{S}$.

The network $\mathcal{S}$ behaves almost in the same way as the network $\mathcal{N}$ in the proof of Theorem 2 does. The main difference is that symbols are not deleted but marked by the special symbol ␣. This can be done by the single substitution processor as well (using rules of R_8). When simulating a derivation $w \Longrightarrow v$ in G, we have to take into account that the corresponding word in the substitution node may contain gaps in form of several occurrences of the special symbol ␣. This is realized by the new formulation of the set $\bar{O}^{\mathcal{S}}$ of such subwords that are forbidden to leave the node.

An input word $w \in T^+$ can be reduced to a word $s \in \{␣\}^*\{S\}\{␣\}^*$ if and only if w is generated by the grammar G. Then and only then, s moves to the output node. If the input word is λ, then it is not modified in the first node but sent out. The second node receives the word if $\lambda \in L$. If $\lambda \notin L$ then the output node does not receive anything.

Hence, we have proved $L(G) = L_{\mathrm{w}}(\mathcal{S}) = L_{\mathrm{s}}(\mathcal{S}) = L$. □

This number of processors is optimal since the input node and output node have to be different (otherwise every input word would be accepted).

4. Networks of Non-Deleting Nodes

The main difference between context-sensitive and non-context-sensitive grammars is that, in arbitrary phrase structure grammars, erasing rules (λ-rules) are allowed. In order to simulate a λ-rule in reverse direction, we introduce an insertion node.

Theorem 4. *For any recursively enumerable language L, there is an accepting network $\mathcal{N}$ of evolutionary processors with exactly one substitution node, one insertion node and one output node without rules that weakly and strongly accepts the language L:*

$$L = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N}).$$

Proof. Let L be a recursively enumerable language and $G = (N, T, P, S)$ be a grammar in Kuroda normal form with $L(G) = L$.

The idea of the proof is to extend the network $\mathcal{S}$ constructed in the proof of Theorem 3 by an inserting processor who is responsible for the reverse simulation of λ-rules, see Figure 1.

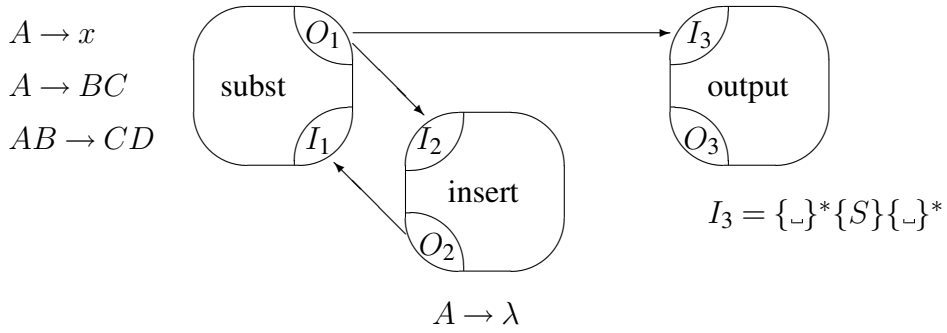

Figure 1: A network for simulating a grammar in Kuroda normal form

For a formal definition, we have to implement that the insertion node gets the opportunity to do something (the substitution processor must indicate that a word can pass to the insertion node).

We construct a network of evolutionary processors

$$\mathcal{N} = (T, V^{\mathcal{N}}, (M_1^{\mathcal{N}}, I_1^{\mathcal{N}}, O_1^{\mathcal{N}}), (\emptyset, I_2^{\mathcal{N}}, \emptyset), (M_3^{\mathcal{N}}, I_3^{\mathcal{N}}, O_3^{\mathcal{N}}), \{(1,2),(1,3),(3,1)\}, 1, \{2\})$$

with

$$\begin{aligned}
V^{\mathcal{N}} &= V^{\mathcal{S}} \cup \{ x' \mid x \in N \cup T \}, \\
M_1^{\mathcal{N}} &= M_1^{\mathcal{S}} \cup \{ x \to x' \mid x \in N \cup T \} \cup \{ x' \to x \mid x \in N \cup T \}, \\
I_1^{\mathcal{N}} &= I_3^{\mathcal{N}} = O_3^{\mathcal{N}} = (N \cup T \cup \{\lrcorner\})^* \{ x' \mid x \in N \cup T \} (N \cup T \cup \{\lrcorner\})^*, \\
O_1^{\mathcal{N}} &= O_1^{\mathcal{S}}, \\
I_2^{\mathcal{N}} &= I_2^{\mathcal{S}}, \\
M_3^{\mathcal{N}} &= \{ \lambda \to A \mid A \to \lambda \in P \},
\end{aligned}$$

where $V^{\mathcal{S}}$, $M_1^{\mathcal{S}}$, $O_1^{\mathcal{S}}$, and $I_2^{\mathcal{S}}$ are defined as in the proof of Theorem 3.

Between two simulation phases, the substitution node can mark a symbol such that the word can leave the node and enter the insertion node. This processor inserts a nonterminal that belongs to a λ-rule of the grammar G and returns the word to the substitution node. This processor then has to unmark the primed symbol. If marking or unmarking is not performed in the correct moment, the word will be lost. Due to the definition of the filters, we can connect all nodes with each other (to obtain a complete graph) without changing the behaviour of the network. □

5. Networks without Substitution Processors

In [1], we have shown that every recursively enumerable language can be generated by a network of one inserting processor and one deleting processor. Similar to the proof of this statement, we can prove the following result.

Theorem 5. *For any recursively enumerable language L, there is an accepting network $\mathcal{N}$ of evolutionary processors with exactly one insertion node, one deletion node and one output node without rules that weakly and strongly accepts the language L:*

$$L = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N}).$$

Proof. Let L be a recursively enumerable language and $G = (N, T, P, S)$ be a grammar in Kuroda normal form with $L(G) = L$.

We define the sets of partial prefixes and partial suffixes of a word u by

$$\begin{aligned} \mathit{PPref}(u) &= \{x \mid u = xy,\ |y| \geq 1\}, \\ \mathit{PSuf}(u) &= \{y \mid u = xy,\ |x| \geq 1\}, \end{aligned}$$

respectively.

Let $V = N \cup T$ and $V_{\text{␣}} = V \cup \{\text{␣}\}$. We define a homomorphism $h : V^* \to V_{\text{␣}}^*$ by $h(a) = a$ for $a \in T$ and $h(A) = A\text{␣}$ for $A \in N$ and set

$$W = \{\, h(w) \mid w \in V^* \,\}.$$

We construct the following network

$$\mathcal{N} = (T, X, (M_1, I_1, O_1), (M_2, I_2, O_2), (\emptyset, \{S\text{␣}\}, \emptyset), E, 1, \{3\})$$

of evolutionary processors with

$$\begin{aligned} X &= V_{\text{␣}} \cup \bigcup_{p \in P} \{p_1, p_2, p_3, p_4\}, \\ M_1 &= \{\, \lambda \to \text{␣} \,\} \cup \{\, \lambda \to p_i \mid p \in P, 1 \leq i \leq 4 \,\} \cup \{\, \lambda \to A \mid A \in N \,\}, \\ I_1 &= W \setminus \{\, S\text{␣} \,\}, \\ O_1 &= X^* \setminus (W R_{1,1} W), \\ M_2 &= \{\, p_i \to \lambda \mid p \in P, 1 \leq i \leq 4 \,\} \cup \{\, x \to \lambda \mid x \in V_{\text{␣}} \,\}, \\ I_2 &= W R_{1,2} W, \\ O_2 &= X^* \setminus (W R_{2,2} W), \\ E &= \{(1,2), (2,1), (2,3)\} \end{aligned}$$

where

$$\begin{aligned} R_{1,1} = \bigcup_{p = u \to v \in P} & (\{p_1 h(v), p_1 h(v) p_2, p_1 p_3 h(v) p_2, p_1 p_3 h(v) p_2 p_4\} \\ & \cup \{p_1 p_3\} \mathit{PSuf}(h(u)) \{h(v) p_2 p_4\}) \\ R_{1,2} = \{\, & p_1 p_3 h(uv) p_2 p_4 \mid p = u \to v \in P \,\}, \\ R_{2,2} = \bigcup_{p = u \to v \in P} & (\{p_1 p_3 h(u)\} \mathit{PPref}(h(v)) \{p_2 p_4\} \cup \{p_3 h(u) p_2 p_4, p_3 h(u) p_4, h(u) p_4\}). \end{aligned}$$

The reverse simulation of the application of a rule $p = a_1 \dots a_s \to b_1 \dots b_t$ to a sentential form $\alpha a_1 \cdots a_s \beta$ with $x = h(\alpha)$ and $y = h(\beta)$ has the following form.

In the insertion node, we have

$$\begin{aligned}
xh(b_1)\dots h(b_t)y &\Longrightarrow xp_1h(b_1)\dots h(b_t)y \\
&\Longrightarrow xp_1h(b_1)\dots h(b_t)p_2y \\
&\Longrightarrow xp_1p_3h(b_1)\dots h(b_t)p_2y \\
&\Longrightarrow xp_1p_3h(b_1)\dots h(b_t)p_2p_4y \\
&\Longrightarrow xp_1p_3{\sqcup}h(b_1)\dots h(b_t)p_2p_4y \\
&\Longrightarrow xp_1p_3a_s{\sqcup}h(b_1)\dots h(b_t)p_2p_4y \\
&\Longrightarrow^* xp_1p_3a_2{\sqcup}\dots a_s{\sqcup}(b_1)\dots h(b_t)p_2p_4y \\
&\Longrightarrow xp_1p_3{\sqcup}a_2{\sqcup}\dots a_s{\sqcup}h(b_1)\dots h(b_t)p_2p_4y \\
&\Longrightarrow xp_1p_3a_1{\sqcup}\dots a_s{\sqcup}h(b_1)\dots h(b_t)p_2p_4y.
\end{aligned}$$

This word leaves the insertion node and enters the deletion node. There, the evolution continues to

$$\begin{aligned}
xp_1p_3a_1{\sqcup}\dots a_s{\sqcup}h(b_1)\dots h(b_t)p_2p_4y &\Longrightarrow^{|h(b_t)|} xp_1p_3a_1{\sqcup}\dots a_s{\sqcup}h(b_1)\dots h(b_{t-1})p_2p_4y \\
&\Longrightarrow^* xp_1p_3a_1{\sqcup}\dots a_s{\sqcup}h(b_1)p_2p_4y \\
&\Longrightarrow^{|h(b_1)|} xp_1p_3a_1{\sqcup}\dots a_s{\sqcup}p_2p_4y \\
&\Longrightarrow xp_3a_1{\sqcup}\dots a_s{\sqcup}p_2p_4y \\
&\Longrightarrow xp_3a_1{\sqcup}\dots a_s{\sqcup}p_4y \\
&\Longrightarrow xa_1{\sqcup}\dots a_s{\sqcup}p_4y \\
&\Longrightarrow xa_1{\sqcup}\dots a_s{\sqcup}y.
\end{aligned}$$

This word leaves the node and enters the output node if it is $S_{\sqcup}$ (corresponding to the axiom of the grammar G) or it enters the insertion node for the next simulation phase. Only those words remain in the network that are obtained in the sequence described above; all other words get lost. □

Every regular language R is accepted by a network of two nodes where none of the nodes contains any rules. The input node keeps all words not belonging to R; all words belonging to R move to the output node which accepts all arriving words. Hence, the network accepts exactly the language R. Such networks are optimal with respect to the number of processors, because input node and output node have to be different. Otherwise the network would accept every input word.

Corollary 6. *Every regular language can be accepted by a pure deleting or pure inserting network with two nodes. This number of processors is optimal.*

Pure deleting networks can accept non-context-free languages.

Lemma 7. *There is a network of deletion nodes only that accepts the language*

$$L = \{\, a^n b^n c^n \mid n \geq 1 \,\}.$$

Proof. Let $T = \{a, b, c\}$ and $O = \{a\}^+\{b\}^+\{c\}^+$. We construct the following network

$$\mathcal{N} = (T, T, N_{\text{in}}, N_a, N_b, N_c, N_{\text{out}}, E, 1, \{5\})$$

of deletion processors with the nodes $N_{\text{in}} = (\emptyset, \emptyset, O)$, $N_x = (\{\, x \to \lambda \,\}, T^*, O)$ for $x \in T$ and $N_{\text{out}} = (\emptyset, \{\, abc \,\}, \emptyset)$, and the set $E\,\{\,(1,5),(1,2),(2,3),(3,4),(4,2),(4,5)\,\}$ of edges. The network is illustrated in Figure 2.

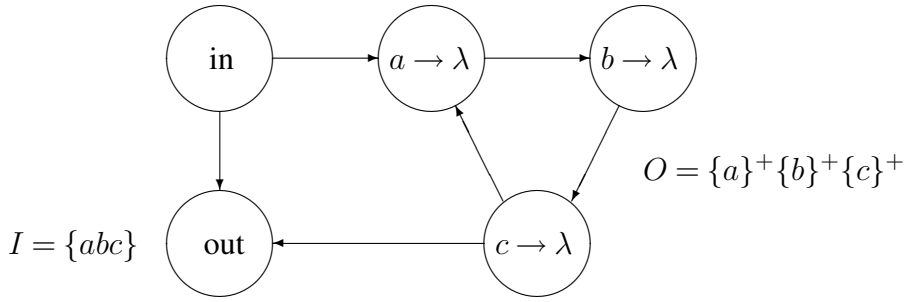

Figure 2: A deleting network for accepting the language $\{\, a^n b^n c^n \mid n \geq 1 \,\}$

Every word which has not the form $a^p b^q c^r$ with $p \geq 1$, $q \geq 1$, $r \geq 1$, remains in the first node for ever. The word abc will be sent directly to the output node and is accepted. A word $a^p b^q c^r$ with $p \geq 1$, $q \geq 1$, $r \geq 1$, and $pqr > 1$ is sent into the 'cycle' where one letter of a, b and c is deleted. The word moves on to the next node only if there is one letter of each kind left. If the word is stuck in a node, then it was abc when it entered the node which deletes a or p, q and r were not equal. If the word leaves the node which deletes c, then it has the form $a^{p-1} b^{q-1} c^{r-1}$. If this word is abc it goes to the output node and the input word is accepted. Otherwise it starts same cycle. Hence, a word is accepted if and only if it belongs to the language L. □

In this manner, networks can be constructed for similarly structured languages. They are accepted after a cyclic deletion process.

Also pure inserting networks can accept non-context-free languages.

Lemma 8. *There is a network of insertion nodes only that accepts the language*

$$L = \{\, a^n b^n c^n \mid n \geq 1 \,\}.$$

Proof. Let $T = \{a, b, c\}$, $V = T \cup \{\, a', b', c' \,\}$, and

$$O = \{aa'\}^*\{a\}^+\{bb'\}^*\{b\}^+\{cc'\}^*\{c\}^+.$$

We construct the following network

$$\mathcal{N} = (V, T, N_{\text{in}}, N_a, N_b, N_c, N_{\text{out}}, E, 1, \{5\})$$

of insertion processors with the nodes $N_{\text{in}} = (\emptyset, \emptyset, O)$, $N_x = (\{\lambda \to x'\}, V^*, O)$ for $x \in T$ and $N_{\text{out}} = (\emptyset, \{aa'\}^+\{bb'\}^+\{cc'\}^+ \cup \{abc\}, \emptyset)$, and the set

$$E\{(1,5),(1,2),(2,3),(3,4),(4,2),(4,5)\}$$

of edges. The network is illustrated in Figure 3.

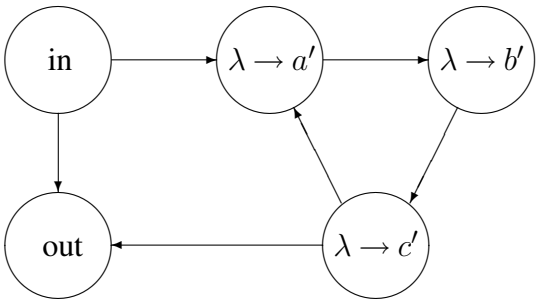

Figure 3: An inserting network for accepting the language $\{a^n b^n c^n \mid n \geq 1\}$

Every word which has not the form $a^p b^q c^r$ with $p \geq 1$, $q \geq 1$, $r \geq 1$, remains in the first node for ever. The word abc will be sent directly to the output node and is accepted. A word $a^p b^q c^r$ with $p \geq 1$, $q \geq 1$, $r \geq 1$, and $pqr > 1$ is sent into the 'cycle' where one letter of a, b and c is marked (by inserting a primed version after the first non-marked letter). The word moves on to the next node only if there is one unmarked letter of each kind left. If the word is stuck in a node, then a primed letter was inserted at a wrong position. If the word leaves the node which marks c, then it has the form $aa'a^{p-1}bb'b^{q-1}cc'c^{r-1}$. This word runs in the cycle until it stucks or it is transformed into the word $(aa')^p(bb')^q(cc')^r$. In the latter case, we have $p = q = r$, the word moves to the output node and, hence, the input word is accepted. Hence, a word is accepted if and only if it belongs to the language L. □

It remains as a task to characterize the family of languages that are accepted by pure deleting networks (which have only deleting processors) and pure inserting networks.

References

[1] A. ALHAZOV, J. DASSOW, C. MARTÍN-VIDE, YU. ROGOZHIN, and B. TRUTHE, On Networks of Evolutionary Processors with Nodes of Two Types. Submitted.

[2] A. ALHAZOV, C. MARTÍN-VIDE, and YU. ROGOZHIN, On the number of nodes in universal networks of evolutionary processors. *Acta Inf.* **43** (2006), 331–339.

[3] J. CASTELLANOS, P. LEUPOLD, and V. MITRANA, On the size complexity of hybrid networks of evolutionary processors. *Theor. Comput. Sci.* **330** (2005), 205–220.

[4] J. CASTELLANOS, C. MARTÍN-VIDE, V. MITRANA, and J. M. SEMPERE, Solving NP-complete problems with networks of evolutionary processors. In: *Proc. IWANN*, Lecture Notes in Computer Science **2084**, Springer-Verlag, Berlin, 2001, 621–628.

[5] J. CASTELLANOS, C. MARTÍN-VIDE, V. MITRANA, and J. M. SEMPERE, Networks of evolutionary processors. *Acta Informatica* **38** (2003), 517–529.

[6] E. CSUHAJ-VARJÚ and A. SALOMAA, Networks of parallel language processors. In: GH. PĂUN and A. SALOMAA (eds.), *New Trends in Formal Language Theory*. Lecture Notes in Computer Science **1218**, Springer-Verlag, Berlin, 1997, 299–318.

[7] J. DASSOW and V. MITRANA, Accepting Networks of Non-Increasing Evolutionary Processors. In: I. PETRE and G. ROZENBERG (eds.), *Proceedings of NCGT 2008. Workshop on Natural Computing and Graph Transformations. September 8, 2008, Leicester, UK*. University of Leicester, 2008, 29–41.

[8] J. DASSOW and B. TRUTHE, On the Power of Networks of Evolutionary Processors. In: J. DURAND-LOSE and M. MARGENSTERN (eds.), *Machines, Computations and Universality, MCU 2007, Orléans, France, September 10–13, 2007, Proceedings*. Lecture Notes in Computer Science **4664**, Springer, 2007, 158–169.

[9] S. E. FAHLMANN, G. E. HINTON, and T. J. SEIJNOWSKI, Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines. In: *Proc. AAAI National Conf. on AI*, William Kaufman, Los Altos, 1983, 109–113.

[10] W. D. HILLIS, *The Connection Machine*. MIT Press, Cambridge, 1985.

[11] F. MANEA and V. MITRANA, All NP-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size. *Information Processing Letters* **103** (2007) 3, 112–118.

[12] M. MARGENSTERN, V. MITRANA, and M. J. PÉREZ-JIMÉNEZ, Accepting Hybrid Networks of Evolutionary Processors. In: C. FERRETTI, G. MAURI, and C. ZANDRON (eds.), *10th International Workshop on DNA Computing*. Lecture Notes in Computer Science **3384**, Springer, 2005, 235–246.

[13] G. ROZENBERG and A. SALOMAA, *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

About the Authors

Jürgen Dassow is professor at the *Otto-von-Guericke-Universität Magdeburg* in Germany and the leader of the Research Group on Automata and Formal Languages. He is well known for his work on regulated rewriting and on cooperating distributed grammar systems. His widely ranged research covers many areas of automata and language theory like Lindenmayer Systems, controlled derivations, grammar systems, biologically motivated formal systems, grammatical approaches to picture generation, and descriptional complexity.

Gema Maria Martín Reyes studied Mathematics and Computer Science at the *Universidad de Málaga* in Spain. At present, she is a Ph.D. student in the Research Group on Biomimetics at the University of Málaga under the supervision of Francisco J. Vico. Her research interest focuses on the dynamics of complexity under evolutionary conditions from a formal point of view and connections between plausible dynamics of complexity in formal systems and conditions in which complexity could have evolved on Earth.

Ralf Stiebe studied Mathematics at the *Otto-von-Guericke-Universität Magdeburg* in Germany. He achieved the doctoral degree with a thesis on Edge and Valence Grammars. Since then, he is Research Assistent in the Group on Automata and Formal Languages. His interest moved to Siromoney Matrix Languages which he studied very intensively. Further, he worked on different grammars and automata like extended finite automata, weighted grammars and automata, blind counter automata, and tree controlled grammars.

Bianca Truthe studied Computer Science at the *Otto-von-Guericke-Universität Magdeburg* and is now Research Assistent in the Group of Jürgen Dassow. She achieved the doctoral degree with a thesis on Chain Code Picture Languages. Supported by the Humboldt Foundation of Germany, she spend 15 months in the Research Group of Mathematical Linguistics at the Rovira i Virgili Universitat Tarragona in Spain. Her interests are in parallel communicating grammar systems, tree controlled languages, and networks of evolutionary processors.

György Vaszil is a senior research fellow of the Hungarian Academy of Sciences in the Theoretical Computer Science Research Group at the Computer and Automation Research Institute in Budapest. He stayed in the Research Group of Jürgen Dassow in Magdeburg for one year with a scholarship of the Alexander von Humboldt Foundation of Germany. His main interest is in nature-motivated computational models, bio-moleculare computing, membrane systems, P automata, and grammar systems.

Francisco J. Vico is currently associate professor at the School of Computer Science of the *Universidad de Málaga* in Spain. After many years of research on brain function modeling, his interests are now in artificial life and life's complexity. He is the leader of the Research Group on Biomimetics whose members work on computer modeling and simulation of the emergence of living matter, and the evolution of complex forms and behaviors. He published papers on neural networks as well as on e-Training in medicine and healthcare.