Accepting Grammars and Systems

Henning Bordihn

Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg
Postfach 4120
D-39016 Magdeburg
Germany
email: bordihn@cs.uni-magdeburg.de

Henning Fernau

Lehrstuhl Informatik für
Ingenieure und Naturwissenschaftler
Universität Karlsruhe (TH)
Am Fasanengarten 5
D-76128 Karlsruhe
Germany
email: fernau@ira.uka.de

February 1, 1995

Abstract

We investigate several kinds of regulated rewriting (programmed, matrix, with regular control, ordered, and variants thereof) and of parallel rewriting mechanisms (Lindenmayer systems, uniformly limited Lindenmayer systems, limited Lindenmayer systems and scattered context grammars) as accepting devices, in contrast with the usual generating mode.

In some cases, accepting mode turns out to be just as powerful as generating mode, e.g., within the grammars of the Chomsky hierarchy, within random context, regular control, L systems, uniformly limited L systems, scattered context. Most of these equivalences can be proved using a metatheorem on so-called context condition grammars. In case of matrix grammars and programmed grammars without appearance checking, a straightforward construction leads to the desired equivalence result.

Interestingly, accepting devices are (strictly) more powerful than their generating counterparts in case of ordered grammars, programmed and matrix grammars with appearance checking (even programmed grammars with unconditional transfer), and 1lET0L systems. More precisely, if we admit erasing productions, we arrive at new characterizations of the recursivley enumerable languages, and if we do not admit them, we get new characterizations of the context-sensitive languages.

Moreover, we supplement [36, 5] in showing:

- The emptiness and membership problems are recursivley solvable for generating ordered grammars, even if we admit erasing productions.
- Uniformly limited propagating systems can be simulated by programmed grammars without erasing and without appearance checking, hence the emptiness and membership problems are recursively solvable for such systems.
- We briefly discuss the degree of nondeterminism and the degree of synchronization for devices with limited parallelism.

Contents

Note: Most of the content of this report is going to be published [3, 11, 10].

Keywords: Formal languages, parallel rewriting, accepting and generating rewriting systems.

C.R. Categories: F.4.2, F.4.3

Chapter 1

Introduction

As already SALOMAA pointed out in [29], any device that generates words (hence describing a formal language) can be interpreted as a system that accepts words. For the usually considered type-n grammars of the Chomsky hierarchy, it does not matter whether we consider these devices in generating or accepting mode, in the sense that the family of languages generable by type-n grammars equals the family of languages acceptable by type-n grammars, see [29, Theorem I.2.1].

For reasons unknown to us, other grammars have been studied only in the generating mode. Especially, this is true for grammars with regulated rewriting [5] and for grammars (systems) admitting some sort of parallelism. This is somewhat surprising since applications of formal languages (like compilers) use accepting, not generating devices. In this paper, we investigate different classes of accepting grammars and compare the such-obtained classes of formal languages with the well-known generating ones.

It might be that, having observed the triviality of the equivalence proof between generating and accepting mode within type-n grammars, one gained the intuition that generating and accepting devices are also equivalent in a trivial manner for other grammar types. In general, this is not the case, as we will see in the following. Nevertheless, we should keep the following catalogue of questions in mind, stemming from the above-sketched intuition:

- For which kind of grammar classes do we find a trivial equivalence between accepting and generating mode, i.e. when do we observe $x \Rightarrow y$ in generating mode of a grammar G iff $y \Rightarrow x$ in accepting mode of some (dual) grammar G'?

- When do we get equivalence between accepting and generating mode via a more complicated construction?

- Are there grammar classes for which accepting and generating mode yield different language classes? If yes, what is the relation between the language classes accepted or generated by the investigated device: inclusion, incomparability?

We will find representatives for all of the above-listed cases.

Conventions: $\subseteq$ denotes inclusion, $\subset$ denotes strict inclusion, $\#M$ is the number of elements in the set M. The empty word is denoted by λ. We consider two languages

L_1, L_2 to be equal iff $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$, and we simply write $L_1 = L_2$ in this case. We term two devices describing languages equivalent if the two described languages are equal.

A bit deviating from this definition, the emptiness problem for a language (more precisely: grammar) family is the next: Is there a Turing machine such that, given a grammar G from our family, decides whether there exists a word w with $w \in L(G)$ or not?

Let Sub(w), (Suf(w), Pref(w)) denote the set of subwords (suffixes, prefixes) of w. $d^l_x(L)$ and $d^r_x(L)$ denote the left and right derivatives of L, respectively.[1] The mirror image of a language L is denoted by Mi(L). The length of a word x is denoted by $|x|$. If $x \in V^*$, where V is some alphabet, and if $W \subseteq V$, then $|x|_W$ denotes the number of occurrences of letters from W in x. If W is a singleton set $\{a\}$, we simply write $|x|_a$ instead of $|x|_{\{a\}}$.

If X is some device, $L_{\text{gen}}(X)$ ($L_{\text{acc}}(X)$) denotes the language generated (accepted) by the device X. If X is some family of devices, $\mathcal{L}_{\text{gen}}(X)$ ($\mathcal{L}_{\text{acc}}(X)$) denotes the family of languages each of which can be generated (accepted) by some device in X.

We denote the classes of the Chomsky hierarchy by $\mathcal{L}$(FIN),[2] $\mathcal{L}_{\text{gen}}$(REG), $\mathcal{L}_{\text{gen}}$(CF), $\mathcal{L}_{\text{gen}}$(CS), $\mathcal{L}$(REC), $\mathcal{L}_{\text{gen}}$(RE).

For each device X considered in this paper, we will define separately what we mean by generating and accepting mode for X. Generally, the idea is the next:

- Instead of a start symbol, or generally a set of start words, we have a goal symbol, or generally a set of goal words. We use the notion 'axiom(s)' both in generating and in accepting case.

- Generally, we allow only productions of a special form in generative devices. Since they turn out to be the most interesting case, mainly context-free productions of the form $a \to w$, where a is some symbol and w is some (possibly empty) word, are considered. In accepting mode, we turn these restrictions 'around', coming to productions of the form $w \to a$ in the context-free case. Especially, accepting λ-productions are of the form $\lambda \to a$.

We call an accepting grammar G^d derived from a generating grammar G *dual* to G if G^d is obtained from G interpreting start words as goal words and productions of the form $v \to w$ as productions $w \to v$. Similarly, one can consider the dual H^d of an accepting grammar H. Obviously, $(G^d)^d = G$ and $(H^d)^d = H$.

- The most important thing about grammars is their dynamic interpretation via the yield relation $\Rightarrow$ (and its reflexive transitive closure $\overset{*}{\Rightarrow}$). In this paper, we introduce the corresponding yield relation (also denoted by $\Rightarrow$) of the accepting mode with textually the same words as in the generating case.

Instead of this formal approach, it would also be possible to introduce the accepting yield relation in order to mimic the generating one. At least, trying to do so might affirm the

[1] With our notations, we mostly follow [5].

[2] It makes no sense to differentiate between generating and accepting devices in the case of finite and recursive languages, since their concept is different.

intuition on side of the reader that generating and accepting words is something different, indeed.

Our approach has a further advantage which can be stated in the following **meta-observation**: In general, if $\mathcal{L}_{\mathrm{gen}}(X) \subseteq \mathcal{L}_{\mathrm{gen}}(Y)$, then $\mathcal{L}_{\mathrm{acc}}(X) \subseteq \mathcal{L}_{\mathrm{acc}}(Y)$. This follows from the fact that $\mathcal{L}_{\mathrm{gen}}(X) \subseteq \mathcal{L}_{\mathrm{gen}}(Y)$ is generally proved by simulating one derivation step $u \Rightarrow v$ in mechnism X via a number of derivation steps $u \overset{*}{\Rightarrow} v$ in mechanism Y, and due to our textual transfer of the notion of derivation to the accepting case, textually the same simulation proves $\mathcal{L}_{\mathrm{acc}}(X) \subseteq \mathcal{L}_{\mathrm{acc}}(Y)$.

Observe that the paper is organized in such a way that ordered and conditional grammars are treated before programmed and matrix grammars, since in our case the techniques developped in ordered grammars turn out to be basic to, e.g., programmed grammars.

Chapter 2

A Metatheorem, with Prerequisites

We already mentioned that, for type-n grammars, the descriptive power of the generating and the accepting mode coincide. More precisely, if G is a generating type-n grammar, then its dual G^d is an accepting type-n grammar such that $L_{\text{gen}}(G) = L_{\text{acc}}(G^d)$, and vice versa. In this case, we find $x \Rightarrow y$ in G iff $y \Rightarrow x$ in G^d.

For the sake of self-containment of this report, we first introduce type-n grammars formally.

2.1 The Chomsky Hierarchy

A *type-n grammar* is given by a construct $G = (V_N, V_T, P, S)$, where V_N, V_T, P, and $S \in V_N$ are the nonterminal alphabet, terminal alphabet,[1] finite set of productions, and axiom, respectively.

In a type-0 grammar G (or phrase structure grammar), productions are of the form $v \to w$ with $v, w \in V_G^*$. If G is generating, we additionally require $|v|_{V_N} > 0$. In this case, we call a production erasing if $w = \lambda$. If G is accepting, we dually require $|w|_{V_N} > 0$. Now, a production is erasing if $v = \lambda$. G is called λ-free iff G contains no erasing productions.

A type-1 grammar (or context-sensitive grammar) G is a type-0 grammar with productions $v \to w$ of a special form. In the generating case, $v = v_1Av_2$ and $w = v_1w'v_2$ with $v_1, v_2 \in V_G^*$, $w' \in V_G^* \setminus \{S\}$, and $A \in V_N$, where $w' = \lambda$ only if $A = S$. In the accepting case, $w = w_1Aw_2$ and $v = w_1v'w_2$ with $v_1, v_2 \in V_G^*$, $v' \in V_G^* \setminus \{S\}$, $A \in V_N$, where $v' = \lambda$ only if $A = S$.

A type-2 grammar (or context-free grammar) G is a type-0 grammar with productions $v \to w$ of a special form. In the generating case, $v = A$ and $w = w'$ with $A \in V_N$ and $w' \in V_G^*$. In the accepting case, $w = A$ and $v = v'$ with $A \in V_N$ and $v' \in V_G^*$.

A type-3 grammar (or regular grammar) G is a type-2 grammar with productions $v \to w$ of a special form. In the generating case, $v = A$ and $w = w'$ with $A \in V_N$, where $w' = \lambda$ only if $A = S$. Additionally, $w' \in \{A, \lambda\} \cdot V_T^*$. In the accepting case, $w = A$ and $v = v'$ with $A \in V_N$, where $v' = \lambda$ only if $A = S$. Additionally, $v' \in \{A, \lambda\} \cdot V_T^*$.

[1] Generally, we denote the total alphabet of G by $V_G = V_T \cup V_N$.

A phrase structure grammar $G = (V_N, V_T, P, S)$ induces a yield relation $x \Rightarrow y$ on V_G^* by $x \Rightarrow y$ iff there is a production $v \to w \in P$ such that $x = x_1 v x_2$ and $y = x_1 w x_2$ for some $x_1, x_2 \in V_G^*$. The reflexive transitive closure of $\Rightarrow$ is denoted by $\overset{*}{\Rightarrow}$.

The language generated by a generating phrase structure grammar $G = (V_N, V_T, P, S)$ is $L_{\text{gen}}(G) = \{w \in V_T^* \mid S \overset{*}{\Rightarrow} w\}$.The language accepted by an accepting phrase structure grammar G is $L_{\text{acc}}(G) = \{w \in V_T^* \mid w \overset{*}{\Rightarrow} S\}$.

As already mentioned, we denote the families of languages generated by type-0,..., type-3 grammars by $\mathcal{L}_{\text{gen}}(\text{RE})$, $\mathcal{L}_{\text{gen}}(\text{CS})$, $\mathcal{L}_{\text{gen}}(\text{CF})$, $\mathcal{L}_{\text{gen}}(\text{REG})$, and the families of languages accepted by type-0,..., type-3 grammars by $\mathcal{L}_{\text{acc}}(\text{RE})$, $\mathcal{L}_{\text{acc}}(\text{CS})$, $\mathcal{L}_{\text{acc}}(\text{CF})$, $\mathcal{L}_{\text{acc}}(\text{REG})$.

The following chain of strict inclusions called Chomsky hierarchy is well-known.

$$\mathcal{L}_{\text{gen}}(\text{REG}) \subset \mathcal{L}_{\text{gen}}(\text{CF}) \subset \mathcal{L}_{\text{gen}}(\text{CS}) \subset \mathcal{L}_{\text{gen}}(\text{RE})$$

For our purposes, the following lemma is very important, see [5, Theorem 0.2.2].

Lemma 2.1 (Kuroda normal form) For each context-sensitive language $L \subset V_T^+$, there is a grammar $G = (V_N, V_T, P, S)$ whose productions are of the following forms

$$A \to BC, \quad AB \to CD, \quad A \to a$$

where $A, B, C, D \in V_N$, $a \in V_T$ such that $L_{\text{gen}}(G) = L$.

2.2 cc Grammars

In this section, we introduce a type of grammar called 'grammar with context conditions' (cc grammar for short) for which the trivial relation between say generating grammars and their dual accepting counterparts is true, too.[2] Since cc grammars generalize phrase structure grammars, we show the well-known result once more.

A *grammar with context conditions (cc grammar)* is given by a construct $G = (V_N, V_T, P, \Omega)$, where V_N, V_T, P, and $\Omega \subset V_G^+$ are the nonterminal alphabet, terminal alphabet, set of production tables, and finite set of axioms, respectively (very similar to phrase structure grammar definitions). The set P is a finite set consisting of so-called tables $P_1, \ldots, P_t$. Each table consists of a finite number of productions p_{ij} of the form $(v_{ij} \to w_{ij}, g_{ij})$ (with $1 \le i \le t$ and $1 \le j \le |P_i|$), where $v_{ij}, w_{ij} \in V_G^{+3}$ and $g_{ij} \subseteq (V_G \cup \{\#\})^* \times \mathbb{N}$ (where $\# \notin V_G$ is a new limiting symbol). In addition, in generating mode, we may allow λ-productions of the form $v_{ij} \to \lambda$, and, dually, in accepting mode, λ-productions of the form $\lambda \to w_{ij}$ may occur. Sometimes, we call $v_{ij} \to w_{ij}$ the core production of the production p_{ij}. A table P_i is applied to a string x using the following steps:

1. x is partitioned into $x = x_1 v_1 x_2 v_2 \cdots x_n v_n x_{n+1}$. Define $x^\# := x_1 \# x_2 \# \cdots x_n \# x_{n+1}$.

[2] In this way, we generalize the so-called lrcc grammars introduced in [3].

[3] Observe that we do not exclude the further derivation of terminal symbols. Hence, we incorporate pure rewriting, too, and also the usual conventions in parallel rewriting.

2. Select n productions $p_{ij_1}, \ldots, p_{ij_n}$ from P_i (possibly with $p_{ij_r} = p_{ij_s}$) such that for all $1 \leq k \leq n$:
   - $v_k = v_{ij_k}$ and
   - $(x^{\#}, k) \in g_{ij_k}$.
3. Replace x using the selected productions yielding $y = x_1 w_{ij_1} x_2 w_{ij_2} \cdots x_n w_{ij_n} x_{n+1}$.

If we succeed going through these three steps, we write $x \Rightarrow y$ in this case. By $\overset{*}{\Rightarrow}$, we denote the reflexive and transitive closure of the relation $\Rightarrow$. The language generated by the cc grammar G is $L_{\text{gen}}(G) = \{w \in V_T^* \,|\, (\exists \omega \in \Omega)\omega \overset{*}{\Rightarrow} w\}$. The language accepted by the cc grammar G is $L_{\text{acc}}(G) = \{w \in V_T^* \,|\, (\exists \omega \in \Omega) w \overset{*}{\Rightarrow} \omega\}$.

Observe that cc grammars are a very broad framework, maybe comparable to selective substitution grammars [5].

Example 2.2 Every phrase structure grammar corresponds to an equivalent cc grammar G with one table and one one-letter-axiom letting $g = V_G^*\{\#\}V_G^* \times \{1\}$ for any production. Conversely, to every cc grammar $G = (V_N, V_T, \{P\}, \{\omega\})$ with one table with productions of the form $(v \to w, V_G^*\{\#\}V_G^* \times \{1\})$ with $|v|_{V_N} > 0$ in the generating or $|w|_{V_N} > 0$ in the accepting case, respectively, and one one-letter axiom $\omega \in V_N$, there corresponds an equivalent type-0 grammar (V_N, V_T, P', ω). This is also true if we restrict ourselves to so-called left derivations, where we let $g = V_T^*\{\#\}V_G^* \times \{1\}$ for any production. For context-sensitive grammars, it is possible to give another different characterization. Let $H = (V_N, V_T, P, S)$ be a generating context-sensitive grammar. For each production $y_1 A y_2 \to y_1 w y_2$, we put a production $(A \to w, g)$ into the table P' of the equivalent cc grammar $(V_N, V_T, \{P'\}, \{S\})$, where $(x_1 \# x_2, n) \in g$ iff $y_1 \in \text{Suf}(x_1)$ and $y_2 \in \text{Pref}(x_2)$ and $n = 1$. Dually, we may treat the case of accepting context-sensitive grammars.

In the following, we formulate the concept of dual grammar formally for cc grammars. If $G = (V_N, V_T, \{P_1, \ldots, P_t\}, \Omega)$ is a cc grammar, then $G^d = (V_N, V_T, \{P_1^d, \ldots, P_t^d\}, \Omega)$ is called *dual* to G iff $P_i^d = \{(w \to v, g) \,|\, (v \to w, g) \in P_i\}$. Obviously, $(G^d)^d = G$.

Our simple metatheorem proved analogously to [29, Theorem I.2.1] is the next.

Theorem 2.3 (i) $L_{\text{gen}}(G) = L_{\text{acc}}(G^d)$, for any generating cc grammar G.

(ii) $L_{\text{acc}}(G) = L_{\text{gen}}(G^d)$, for any accepting cc grammar G. □

Since we can interpret any type-n grammar as cc grammar G, and since we can reinterpret the dual G^d as type-n grammar, we obtain $\mathcal{L}_{\text{gen}}(X) = \mathcal{L}_{\text{acc}}(X)$ for $X \in \{\text{REG}, \text{CF}, \text{CS}, \text{RE}\}$ as a simple corollary. But we can handle other devices in this setting, too.

2.3 Random Context Grammars

A *random context grammar (RC grammar)* is a system $G = (V_N, V_T, P, S)$ where V_N, V_T, S are defined as in a usual Chomsky grammar, and P is a finite set of random context rules, that is, triples of the form $(\alpha \to \beta, Q, R)$ where $\alpha \to \beta$ is a rewriting rule over V_G and Q and R are subsets of V_N. For $x, y \in V_G^*$, we write $x \Rightarrow y$ iff $x = x_1 \alpha x_2$, $y = x_1 \beta x_2$ for some $x_1, x_2 \in V_G^*$, $(\alpha \to \beta, Q, R)$ is a triple in P, all symbols of Q appear in $x_1 x_2$, and no symbol of R appears in $x_1 x_2$. (Q is called the permitting context of $\alpha \to \beta$ and R is the forbidding context of this rule.)

The family of languages generated by RC grammars containing only context-free rules $\alpha \to \beta$ is denoted by $\mathcal{L}_{\text{gen}}(\text{RC}, \text{CF}, \text{ac})$. Considering only random context rules with empty permitting context, we are led to the language family $\mathcal{L}_{\text{gen}}(\text{fRC}, \text{CF})$. Symmetrically, allowing only rules with empty forbidding context, we come to the language family $\mathcal{L}_{\text{gen}}(\text{RC}, \text{CF})$.[4] If we do not permit λ-productions, we arrive at the language families $\mathcal{L}_{\text{gen}}(\text{RC}, \text{CF}{-}\lambda, \text{ac})$, $\mathcal{L}_{\text{gen}}(\text{fRC}, \text{CF}{-}\lambda)$, and $\mathcal{L}_{\text{gen}}(\text{RC}, \text{CF}{-}\lambda)$.

Similarly, the corresponding 'accepting classes' $\mathcal{L}_{\text{acc}}(\text{RC}, \text{CF}, \text{ac})$, $\mathcal{L}_{\text{acc}}(\text{fRC}, \text{CF})$, ... are defined.

Each RC grammar $G = (V_N, V_T, P, S)$ may be interpreted as a cc grammar $G' = (V_N, V_T, \{P'\}, \{S\})$ where, for each random context rule $(\alpha \to \beta, Q, R)$, we have one production $(\alpha \to \beta, g)$ in P' such that $(x_1 \# x_2, n) \in g$ iff, for all $v \in Q$, $v \in \text{Sub}(x_1) \cup \text{Sub}(x_2)$, and, for all $w \in R$, $w \notin \text{Sub}(x_1) \cup \text{Sub}(x_2)$ and $n = 1$.

Conversely, consider a cc grammar of the form $G = (V_N, V_T, \{P\}, \{S\})$ with $S \in V_N$ and productions $(\alpha \to \beta, g)$ such that $\alpha \to \beta$ is a context-free core production and there are subsets Q and R of V_N such that g may be characterized via $(x_1 \# x_2, n) \in g$ iff, for all $v \in Q$, $v \in \text{Sub}(x_1) \cup \text{Sub}(x_2)$, and, for all $w \in R$, $w \notin \text{Sub}(x_1) \cup \text{Sub}(x_2)$ and $n = 1$. To G, there corresponds an equivalent RC grammar G' defined in the obvious way. Hence, we immediately obtain:

Corollary 2.4 $\mathcal{L}_{\text{gen}}(X, Y) = \mathcal{L}_{\text{acc}}(X, Y)$, and $\mathcal{L}_{\text{gen}}(\text{RC}, Y, \text{ac}) = \mathcal{L}_{\text{acc}}(\text{RC}, Y, \text{ac})$, where $X \in \{\text{RC}, \text{fRC}\}$, $Y \in \{\text{CF}, \text{CF}{-}\lambda\}$. □

Other regulation mechanisms that may be treated similarly are:

- string random context grammars (also called semi-conditional grammars) [23, 5]
- random string context grammars (introduced below)
- a variant of conditional grammars introduced by Navrátil [20, 22]

We will refer to the metatheorem at various places. Its application is always the same. (1) Find a characterization of the grammars in question in terms of cc grammars. (2) Consider the duals of the cc grammars in the opposite mode (either accepting if we start

[4] In this case, no 'appearance checking' is possible. The letters 'ac' in $\mathcal{L}_{\text{gen}}(\text{RC}, \text{CF}, \text{ac})$ indicate possible appearance checks.

with a generating device or vice versa). (3) Re-interpret the dual cc grammars again in terms of the original mechanism.

The table mechanism is of course only needed in case of parallel systems.

Chapter 3

Ordered Grammars and Conditional Grammars

3.1 Ordered Grammars

An *ordered grammar* (cf. [13, 5]) is a quintuple $G = (V_N, V_T, P, S, \prec)$, where V_N, V_T, P, and $S \in V_N$ are the nonterminal alphabet, terminal alphabet, set of productions, and axiom, respectively. $\prec$ is a partial order on P.

Both in generating and in accepting mode, a production $w \to z$ is applicable to a string $x \in V^*$ if $x = x_1 w x_2$ for some $x_1, x_2 \in V_G^*$, and x contains no subword w' such that $w' \to z' \in P$ for some z' and $w' \to z' \succ w \to z$; the application of $w \to z$ to x yields $y = x_1 z x_2$. As usual, the yield relation is denoted by $\Rightarrow$, and its reflexive and transitive closure is denoted by $\overset{*}{\Rightarrow}$. The family of languages generated by ordered grammars with productions of type X is denoted by $\mathcal{L}_{\mathrm{gen}}(\mathrm{O}, X)$. The family of languages accepted by ordered grammars with productions of type X is denoted by $\mathcal{L}_{\mathrm{acc}}(\mathrm{O}, X)$.

In this section, as our main results, we are going to show that $\mathcal{L}_{\mathrm{acc}}(\mathrm{O,CF}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{RE})$ and $\mathcal{L}_{\mathrm{acc}}(\mathrm{O,CF}-\lambda) = \mathcal{L}_{\mathrm{gen}}(\mathrm{CS})$. These equalities imply $\mathcal{L}_{\mathrm{gen}}(\mathrm{O}, X) \subset \mathcal{L}_{\mathrm{acc}}(\mathrm{O}, X)$ for $X \in \{\mathrm{CF,CF}-\lambda\}$ (these are the only cases we consider in detail).

We first show a direct proof of $\mathcal{L}_{\mathrm{gen}}(\mathrm{O}, X) \subseteq \mathcal{L}_{\mathrm{acc}}(\mathrm{O}, X)$, because it is instructive. To this end, we need the following trivial lemma.[1]

Lemma 3.1 For any generating ordered grammar G, there exists an equivalent generating ordered grammar G' containing no rules $w \to z$, $w \to z'$ with $w \to z \succ w \to z'$. □

Proof. Leave the smaller rules out; they are never applicable. □

Theorem 3.2 For any ordered grammar G with context-free rules in generating mode, there is an equivalent ordered grammar G' with context-free rules in accepting mode.

[1] This lemma also corrects a small mistake in the proof of [5, Theorem 2.3.4]. See also the correct proof of [5, Theorem 6.2.3].

Proof. Without loss of generality, let us assume that $G = (V_N, V_T, P, S, \prec)$ does not contain rules $A \to w$, $A \to w'$ with $A \to w \succ A \to w'$. For $A \to w \in P$, let $X_{A \to w}$ be the set of nonterminals A' with $A' \to w' \succ A \to w$ for some w'.

We construct an accepting ordered grammar $G' = (V_N', V_T, P', S, \prec')$. To this end, let

$$V_N' = V_N \cup \{[w, A] \mid A \to w \in P\} \cup \{F\}$$

(the union being disjoint, F is an extra failure symbol). For any production $A \to w \in P$, we introduce the following production(s) and relation(s) in P' (hence describing P' and $\prec'$ completely).

- $w \to [w, A] \prec' [w, A] \to F$;
- $[w, A] \to A$ with $[w, A] \to A \prec' A' \to F$ for any $A' \in X_{A \to w}$;
- If $u \to V \in P'$ with $V \neq F$ and $u \neq [w, A]$, add the relation $u \to V \prec' [w, A] \to F$.

Firstly, we prove $L_{\text{gen}}(G) \subseteq L_{\text{acc}}(G')$. To this end, we show that, if $x \Rightarrow y$ in G, then $y \overset{*}{\Rightarrow} x$ in G'. From this, the assertion follows by simple induction. Assume $x \Rightarrow y$, i.e. there is a production $A \to w \in P$ and there are words x_1, x_2, such that $x = x_1 A x_2$, $y = x_1 w x_2$, and $A' \in X_{A \to w}$ implies $A' \notin \text{Sub}(x)$. Our construction provides a two-step-simulation of this generating step in the accepting grammar G'. In the first step, w is selected from y using $w \to [w, A]$, yielding $y' = x_1 [w, A] x_2$. In the second step, $[w, A]$ is replaced by A (it is tested by G' that $(X_{A \to w} \cup \{[v, B] \mid B \to v \in P, B \to v \neq A \to w\}) \cap \text{Sub}(y') = \emptyset$), yielding $x_1 A x_2 = x$.

Secondly, we prove $L_{\text{gen}}(G) \supseteq L_{\text{acc}}(G')$. To this end, we show that, if $y \Rightarrow y' \overset{*}{\Rightarrow} x$ in G' for $y, x \in V_G^*$, then $x \overset{*}{\Rightarrow} y$ in G. Obviously, we can assume without loss of generality, that there is no 'intermediate' $z \in V_G^*$, $z \notin \{x, y\}$ with $y' \overset{*}{\Rightarrow} z \overset{*}{\Rightarrow} x$ in the derivation under consideration. By definition, there is a production $v \to B \in P'$ and there are words y_1, y_2 such that $y = y_1 v y_2$ and $y' = y_1 B y_2$. By construction, either $B = F$ or $B = [v, A]$ for some $A \to v \in P$. Only the latter case may lead to some word in V_G^*. Namely, if $[u, C] \notin \text{Sub}(y_1 [v, A] y_2)$, $[u, C] \neq [v, A]$ and if $X_{A \to v} \cap \text{Sub}(y') = \emptyset$, we can apply $[v, A] \to A$, and this is the only derivation not introducing the failure symbol. These two derivation steps in the accepting grammar correspond to one application of $A \to v$ to x in the generating grammar G. □

As an example violating the given construction, consider $G = (\{A\}, \{a\}, P, A, \prec)$, where P contains the following productions and one order relation: $A \to aa \succ A \to a$. Hence, $L_{\text{gen}}(G) = \{aa\}$. The accepting grammar G' would contain the following productions and order relations: $aa \to [aa, A] \prec' [aa, A] \to F$; $[aa, A] \to A$; $a \to [a, A] \prec' [a, A] \to F$; $[a, A] \to A \prec' A \to F$; $aa \to [aa, A] \prec' [a, A] \to F$; $a \to [a, A] \prec' [aa, A] \to F$. Now, both $aa \Rightarrow [aa, A] \Rightarrow A$ and $a \Rightarrow [a, A] \Rightarrow A$ are possible derivations in G'. Hence, $L_{\text{acc}}(G') = \{a, aa\} \neq L_{\text{gen}}(G)$.

In this paper, we see that, in some cases, accepting versions of regulated grammars are as powerful as their generating counterparts. In ordered grammars, this is not the case. Intuitively, the reason behind this is the next: In generating ordered grammars, it can be only (negatively) tested whether the string to be rewritten contains single symbols, where in accepting mode, containment of whole strings can be tested. Especially, this can be used to check the left and right context of subwords to be derived next by applying some dummy rules. Similar devices have been introduced and investigated by Kelemen and Păun [23, 5] as 'semi-conditional grammars' or 'string random context grammars' for the generating case. Under some conditions, this string testing turns out to be more powerful than testing single symbols. It should be mentioned here that, in some sense, these 'string random context grammars' are not a straightforward generalization of random context grammars, since only one string is allowed as a (forbidding) test-string for each production, where testing finite sets of test-strings seems to be a more direct generalization. In some sense, this is exactly what we are doing when considering accepting ordered grammars, cf. the RSC grammars introduced below. So, the below results are not too surprising keeping the above remarks in mind. We also use similar simulation techniques as Păun in our case.

We are now going to show our more general results. We first need two lemmas.

Lemma 3.3 $\mathcal{L}_{\mathrm{acc}}(\mathrm{O,CF}{-}\lambda)$ and $\mathcal{L}_{\mathrm{acc}}(\mathrm{O,CF})$ are closed under (finite) union.

Proof. In the case of two given grammars, make the two sets of nonterminals disjoint, add an additional symbol S (the new goal symbol) and two productions $S_1 \to S$, $S_2 \to S$ (S_1, S_2 being the goal symbols of the original grammars). □

By Theorem 3.2 and [31, Теорема 4.2], we immediately get the next assertion.

Lemma 3.4 $\mathcal{L}_{\mathrm{gen}}(\mathrm{CF}) \subseteq \mathcal{L}_{\mathrm{acc}}(\mathrm{O,CF}{-}\lambda)$. □

Theorem 3.5 $\mathcal{L}_{\mathrm{acc}}(\mathrm{O,CF}{-}\lambda) = \mathcal{L}_{\mathrm{gen}}(\mathrm{CS})$.

Proof. The inclusion $\subseteq$ is easily seen by an lba-construction as follows. Given an ordered grammar G with context-free λ-free productions, a linear bounded automaton LBA accepting the same language would do the following step-by-step simulation: Firstly, LBA selects a production $w \to A$ whose application is going to be simulated; secondly, LBA checks whether w is contained in the current string x written on a working tape of the form $x\#\cdots\#$, and whether, for any $w' \to A' \succ w \to A$, w' is not contained in x; thirdly, if both tests succeed, LBA picks (nondeterministically) an occurrence of w in x and replaces it by $A\#^{|w|-1}$; finally, LBA moves all $\#$-symbols to the right of the working tape, hence regaining some tape of the form $y\#\cdots\#$, $y \in V_G^*$.

The inclusion $\supseteq$ is more involved. Let $L \in \mathcal{L}_{\mathrm{gen}}(\mathrm{CS})$, $L \subseteq V^+$. Obviously,

$$L = \bigcup_{a,b\in V} \{a\}d_a^l(d_b^r(L))\{b\} \cup (L \cap (V \cup V^2)).$$

This identity proves that, by our lemmas, it is sufficient for the proof of the present assertion to show that $\{a\}M\{b\} \in \mathcal{L}_{\mathrm{acc}}(\mathrm{O,CF}{-}\lambda)$ for $M \in \mathcal{L}_{\mathrm{gen}}(\mathrm{CS})$, $\lambda \notin M$.[2] Let $G =$

[2] Our proof parallels [5, page 87]. See also [23].

(V_N, V_T, P, S) be a context-sensitive grammar without λ-productions in Kuroda normal form generating M. Let us assume a unique label r being attached to any rule of the form $XU \to YZ$ (the set of lables is denoted by Lab). We construct an ordered grammar $G' = (V_N', V_T', P', S', \prec)$ with context-free λ-free rules accepting $\{a\}M\{b\}$ as follows. Let

$$V_N' = V_N \cup \{A, B, S', F\} \cup \{(A,r), [A,r], (Y,r), [Z,r] \,|\, r : XU \to YZ \in P\}$$

(the unions being disjoint), $V_T' = V_T \cup \{a, b\}$.[3] P' contains the following rules:

1. At first, we give some simple starting and terminating rules as well as the direct simulation of the context-free rules.
   (a) $a \to A \prec \{y \to F \,|\, y \in V_{G'} \setminus V_T'\}$;
   (b) $b \to B \prec \{y \to F \,|\, y \in V_{G'} \setminus (V_T' \cup \{A\})\}$;
   (c) $x \to X \prec \{(A,r) \to F, [A,r] \to F \,|\, r \in Lab\}$ for context-free rules $X \to x \in P$;
   (d) $ASB \to S'$;
2. For any 'real' context-sensitive production of the form $r : XU \to YZ \in P$, we introduce the following rules, simulating an application of r:
   (a) $A \to [A,r] \prec \{[A,s] \to F, (A,s) \to F \,|\, s \in Lab\}$;
   (b) $Y \to [Y,r] \prec \{A \to F, (A,s) \to F, [A,s'] \to F, [T,s] \to F, (T,s) \to F \,|\, s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N\}$;
   (c) $Z \to (Z,r) \prec \{A \to F, (A,s) \to F, [A,s'] \to F, [T,s'] \to F, (T,s) \to F \,|\, s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N\}$;
   (d) $[A,r] \to (A,r) \prec \{A \to F, (A,s) \to F, [A,s'] \to F,\ [T,s'] \to F,\ (T,s') \to F,\ z(Z,r) \to F,\ [Y,r]y \to F \,|\, s \in Lab,\ s' \in Lab \setminus \{r\},\ T \in V_N,\ z \in V_{G'} \setminus \{[Y,r]\},\ y \in V_{G'} \setminus \{(Z,r)\}\}$; (the left and right context checks cause the necessity to have left and right markers)
   (e) $(A,r) \to A \prec (Z,r) \to U \prec [Y,r] \to X \prec \{A \to F,\ [A,s] \to F,\ (A,s') \to F,\ [T,s'] \to F,\ (T,s') \to F \,|\, s \in Lab,\ s' \in Lab \setminus \{r\},\ T \in V_N\}$;

At first, the endmarker symbols a and b are replaced by A and B, respectively. The presence of the symbol A indicates that a simulation of a production of G may take place (or that the final rule $ASB \to S'$ may be applied). The case of context-free productions is again trivial. In the case of 'real' context-sensitive productions, the rules given under number 2 must be applied subsequently. The application of $A \to [A,r]$ fixes the production to be simulated. Afterwards, one occurrence of Y and one occurrence of Z is selected, replacing it by $[Y,r]$ and (Z,r), respectively. Note that the greater rules prevent that more than one Y or more than one Z is replaced. The end of the selection process is signalled by $[A,r] \to (A,r)$. Moreover, this production checks (via the greater rules) whether two

[3] Note that the case $a = b$ is possible.

adjacent symbols YZ have been replaced or not (this prevents shortcuts in the derivation process, too). This crucial check is done by testing whether the symbol to the right of $[Y, r]$ is different from (Z, r) (only negative checks are possible in ordered grammars); this test is equivalent to checking whether (Z, r) is to the right of $[Y, r]$, as long as it is guaranteed that $[Y, r]$ is not the right tail of the string we are going to transform. This is the reason for introducing the right marker B.[4] A similar argument is valid for the check of the left context of (Z, r). Note that we need both tests in order to prevent shortcuts, only replacing either Y by $[Y, r]$ or Z by (Z, r). Note further that this is the only part of the simulation where we really need to check for strings instead of single symbols. Finally, the rules from the last part in number 2 serve for the actual simulation of an application of $r : XU \to YZ$ in G. □

Employing [31, Теорема 4.2], we immediately get the next assertion.

Corollary 3.6 $\mathcal{L}_{\text{gen}}(\text{CF}) \subset \mathcal{L}_{\text{gen}}(\text{O,CF}-\lambda) \subset \mathcal{L}_{\text{acc}}(\text{O,CF}-\lambda)$. □

Corollary 3.7 $\mathcal{L}_{\text{gen}}(\text{O,CF}) \subseteq \mathcal{L}_{\text{acc}}(\text{O,CF}) = \mathcal{L}_{\text{gen}}(\text{RE})$.

Proof. We have to show $\mathcal{L}_{\text{gen}}(\text{RE}) \subseteq \mathcal{L}_{\text{acc}}(\text{O,CF})$. By [29, Theorem I.9.10], any recursively enumerable language $L \subseteq \Sigma^*$ is the homomorphic image of a context-sensitive language $L' \subseteq V^*$. Hence, $L = h(L')$ for a morphism $h : V^* \to \Sigma^*$. Without loss of generality, we may assume $\Sigma \cap V = \emptyset$. Now, let $V'' = \{A'' \mid A \in V\}$ a set of new symbols. Then, we can use the same construction as in the above theorem, only adding the following rules:

- $w \to A'' \prec \{B \to F \mid B \in V\}$ for $h(A) = w$; [5]
- $A'' \to A$ for $A \in V$;
- $c \to F$ for $c \in \Sigma \cup V''$ which are greater than any of the rules given by the construction of the above theorem. □

The last corollary from our theorem is the next normal form result:

Corollary 3.8 For any accepting ordered grammar (with or without λ-rules), there exists an equivalent one containing no productions $w \to A \prec w' \to A'$ with $w \in \text{Sub}(w')$. □

We are now going to show that the inclusion in Corollary 3.7 is strict. Note that the following propositions also solve some problems marked as open in [5, pages 146,147].

Lemma 3.9 For $\mathcal{L}_{\text{gen}}(\text{O,CF}-\lambda)$ and $\mathcal{L}_{\text{gen}}(\text{O,CF})$, the emptiness problem is recursively solvable.

[4] Using the Penttonen normal form [24], we would not need the right marker B.

[5] Note that the case $w = \lambda$ is possible.

Proof. In [31, Следствие 2 on page 98], the solvability of the emptiness problem for ordered grammar without λ-productions is stated. If $G = (V_N, V_T, P, S, \prec)$ is an ordered grammar containing λ-productions, we can introduce a new terminal letter Λ and replace each λ-production $A \to \lambda$ by $A \to \Lambda$, yielding an ordered grammar G' without λ-productions, such that $L_{\text{gen}}(G) = h(L_{\text{gen}}(G'))$, where the morphism h is given by $b \mapsto b$ for $b \in V_T$, and $\Lambda \mapsto \lambda$. Obviously, $L_{\text{gen}}(G) = \emptyset$ iff $L_{\text{gen}}(G') = \emptyset$, and the latter problem is solvable by [31]. □

Corollary 3.10 $\mathcal{L}_{\text{gen}}(\text{O,CF}) \subset \mathcal{L}(\text{REC})$

Proof. By the above lemma, we know the solvability of the emptiness problem for $\mathcal{L}_{\text{gen}}(\text{O,CF})$. By a construction similar to Теорема 4.5 in [31], we know that $\mathcal{L}_{\text{gen}}(\text{O,CF})$ is effectively closed under intersection with regular languages. Applying the same idea as in the proof of [18, Theorem 4], it can be shown that the membership problem is solvable for ordered grammars. Since any recursively enumerable language is effectively the morphic image of a context-sensitive language, by [18, Theorem 3(b)], there is a language in $\mathcal{L}(\text{CS}) \setminus \mathcal{L}_{\text{gen}}(\text{O,CF})$. □

In the same way, we can show the existence of a language in $\mathcal{L}_{\text{gen}}(\text{M,CF}-\lambda,\text{ac}) \setminus \mathcal{L}_{\text{gen}}(\text{O,CF})$. From [18], we may conclude further that $\mathcal{L}_{\text{gen}}(\text{O,CF})$ is closed neither under intersection nor under complementation, since this class embraces all context-free languages. Note that these results on ordered grammars supplement the ones listed on page 147 in [5].

Obviously, we get as an easy corollary:

Corollary 3.11 $\mathcal{L}_{\text{gen}}(\text{O,CF}) \subset \mathcal{L}_{\text{acc}}(\text{O,CF})$ □

As already noted by Friš, ordered grammars are a special case of 'grammars with T_3 restrictions', or 'conditional grammars of type $(2(-\lambda), 3)$', as called by Păun in [22].

3.2 Conditional Grammars

Following Păun, we define:[6]

A *conditional grammar of type* (i,j), or (i,j)*-grammar* for short, $i \in \{0,1,2,2-\lambda,3\}$, $j \in \{0,1,2,3\}$, is a pair (G,ρ), where $G = (V_N, V_T, P, S)$ is a type-i grammar and ρ is a mapping of P into the family of type-j languages over V_G. Both in generating and in accepting mode, for $x, y \in V_G^*$, we write $x \Rightarrow y$ iff $x = x_1vx_2$, $y = x_1wx_2$, $v \to w \in P$, and $x \in \rho(v \to w)$. The language generated by (G,ρ) is denoted by $L_{\text{gen}}(G,\rho)$. The family of languages generated by conditional grammars of type (i,j) is written $\mathcal{L}_{\text{gen}}(\text{K},i,j)$. The language accepted by (G,ρ) is denoted by $L_{\text{acc}}(G,\rho)$. The family of languages accepted by conditional grammars of type (i,j) is written $\mathcal{L}_{\text{acc}}(\text{K},i,j)$.

[6] Conditional grammars as defined in [5] only comprise conditional grammars of type $(i,3)$.

If $G = (V_N, V_T, P, S, \prec)$ is a generating ordered grammar with context-free rules, the conditional grammar $((V_N, V_T, P, S), \rho)$ of type $(2, 3)$ (which is λ-free iff G is λ-free) generates $L(G)$, if ρ is defined as follows: $\rho(A \to w) = V_G^* \setminus V_G^*\{A' \mid A \to w \prec A' \to w'\}V_G^*$ [13, p. 421]. Similarly, if $G = (V_N, V_T, P, S, \prec)$ is an accepting ordered grammar, the conditional grammar $((V_N, V_T, P, S), \rho)$ of type $(2, 3)$ accepts $L(G)$, defining ρ as follows: $\rho(w \to A) = V_G^* \setminus V_G^*\{w' \mid w \to A \prec w' \to A'\}V_G^*$.

In generating mode, the generalization from ordered grammars with context-free λ-free rules towards $(2 - \lambda, 3)$-grammars enhances the power of generation [29, Theorem II.3.7]. This is not true any more in the accepting case, as we will see in the following. The key to show this is the equivalence of generating and accepting mode for (i, j)-grammars.

Theorem 3.12 $\mathcal{L}_{\text{gen}}(\mathrm{K}, i, j) = \mathcal{L}_{\text{acc}}(\mathrm{K}, i, j)$ for any i, j.

Proof. We want to apply our metatheorem. To any (i, j)-grammar $G = (V_N, V_T, P, S, \rho)$, there corresponds a cc grammar $G' = (V_N, V_T, \{P'\}, \{S\})$ with $v \to w \in P$ iff $(v \to w, \mathrm{gsm}_{v \to \#}(\rho(v \to w)) \times \{1\}) \in P'$, where the gsm-mapping $\mathrm{gsm}_{v \to \#}$ replaces exactly one occurrence of v by $\#$, and keeps the other symbols unchanged. Since type-j languages are closed under non-erasing gsm-mappings, $\mathrm{gsm}_{v \to \#}(\rho(v \to w))$ is a type-j language over $V_G \cup \{\#\}$ with exactly one occurrence of $\#$.

On the other hand, if $G' = (V_N, V_T, \{P'\}, \{S\})$ is a cc grammar with $S \in V_N$ and productions $(v \to w, L(v \to w) \times \{1\}) \in P'$ (where $v \to w$ is a type-i production), then the (i, j)-grammar $G = (V_N, V_T, P, S, \rho)$ with $v \to w \in P$ iff $(v \to w, L(v \to w) \times \{1\}) \in P'$, and $\rho(v \to w) = \mathrm{gsm}_{v \to \#}^{-1}(L(v \to w)) \cap V_G^*$, is equivalent to G' and is indeed of type (i, j), since type-j languages are closed under inverse gsm-mappings which means that $\rho(v \to w)$ is a type-j language over V_G.

As sketched above, the claim is now a direct consequence of our metatheorem.

For clarity, we give a direct proof of the statement, too.

At first, let us show $\mathcal{L}_{\text{gen}}(\mathrm{K}, i, j) \subseteq \mathcal{L}_{\text{acc}}(\mathrm{K}, i, j)$. Let $G = (V_N, V_T, P, S)$ and ρ be such that (G, ρ) is an (i, j)-grammar in generating mode. Without loss of generality, we may assume that, for any $v \to w \in P$, $\rho(v \to w) \subseteq V_G^*\{v\}V_G^*$. We define an equivalent accepting (i, j)-grammar (G', ρ') with $G' = (V_N', V_T, P', S)$ as follows: $V_N' = V_N \cup \{[w, v] \mid v \to w \in P\}$; for any $v \to w \in P$, P' contains the following productions (which are the only ones in P'), and ρ' is defined as follows:

- $\rho'(w \to [w, v]) = V_G^*$;[7]
- if $y_1 v y_2 \in \rho(v \to w)$, then $y_1[w, v]y_2 \in \rho'([w, v] \to v)$.

Any derivation step of G is simulated by two steps of G'. At first, G' selects a w to be replaced, and then the actual simulation takes place, accompanied by the appropriate checks according to the original grammar (G, ρ).

The language $\rho'([w, v] \to v)$ can be also described in the following manner. Let E_v be some generalized sequential machine replacing exactly one occurrence of v as substring of

[7] Note that words containing some $[w', v']$ are not in V_G^*.

$x \in \rho(v \to w)$ by $[w,v]$. Then, $E_v(\rho(v \to w)) = \rho'([w,v] \to v)$. Since each of the type-j languages is an AFL, ρ' maps onto type-j languages (see [29, page 135]).

The other implication $\mathcal{L}_{\text{acc}}(\mathrm{K},i,j) \subseteq \mathcal{L}_{\text{gen}}(\mathrm{K},i,j)$ is shown by a similar two-step-simulation. Let $G = (V_N, V_T, P, S)$ and ρ be such that (G,ρ) is an (i,j)-grammar in accepting mode. We define an equivalent generating (i,j)-grammar (G',ρ') with $G' = (V_N', V_T, P', S)$ as follows: $V_N' = V_N \cup \{[v,w] \mid w \to v \in P\}$; for any $w \to v \in P$, P' contains the following productions, and ρ' is defined as follows:

- $\rho'(v \to [v,w]) = V_G^*$;
- if $y_1 w y_2 \in \rho(w \to v)$, then $y_1[v,w]y_2 \in \rho'([v,w] \to w)$.

Similarly, if $\rho(w \to v)$ is of type j, then $\rho'([v,w] \to w)$ is of type j, too. □

Especially, we find:

Corollary 3.13 (i) $\mathcal{L}_{\text{gen}}(\text{O,CF}-\lambda) \subset \mathcal{L}_{\text{gen}}(\mathrm{K},2-\lambda,3) = \mathcal{L}_{\text{gen}}(\text{CS})$

(ii) $\mathcal{L}_{\text{acc}}(\text{O,CF}-\lambda) = \mathcal{L}_{\text{acc}}(\mathrm{K},2-\lambda,3) = \mathcal{L}_{\text{gen}}(\mathrm{K},2-\lambda,3) = \mathcal{L}_{\text{gen}}(\text{CS})$

(iii) $\mathcal{L}_{\text{gen}}(\text{O,CF}) \subset \mathcal{L}(\text{REC}) \subset \mathcal{L}_{\text{gen}}(\mathrm{K},2,3) = \mathcal{L}_{\text{gen}}(\text{RE})$

(iv) $\mathcal{L}_{\text{acc}}(\text{O,CF}) = \mathcal{L}_{\text{acc}}(\mathrm{K},2,3) = \mathcal{L}_{\text{gen}}(\mathrm{K},2,3) = \mathcal{L}_{\text{gen}}(\text{RE})$ □

Moreover, we state the following observation: Analyzing the transformation from accepting ordered grammars with context-free rules to $(2,3)$-grammars, we see that we do not need the full power of regular languages in order to accept all context-sensitive (and even enumerable when permitting λ-productions) languages. We can restrict ourselves to local languages (sometimes also called locally testable languages, see [28]). We do not know whether we loose generative power when imposing a similar restriction upon the generating mode.

Finally, we want to turn to an appropriate natural generalization of random context grammars which we call random string context grammars (not to be confused with string random context grammars [5]).

3.3 Random String Context Grammars

A *random string context grammar* is a system $G = (V_N, V_T, P, S)$, where V_N, V_T, S have their usual meaning, and P is a finite set of random string context rules, i.e. triples of the form $(\alpha \to \beta, Q, R)$, where $\alpha \to \beta$ is a rewriting rule, and Q, R are finite subsets of V_G^*. We write $x \Rightarrow y$ iff $x = x_1 \alpha x_2$, $y = x_1 \beta x_2$ for some $x_1, x_2 \in V_G^*$, $(\alpha \to \beta, Q, R) \in P$, $Q \subseteq \text{Sub}(x_1) \cup \text{Sub}(x_2)$, $R \cap (\text{Sub}(x_1) \cup \text{Sub}(x_2)) = \emptyset$.

$\mathcal{L}_{\text{gen}}(\text{RSC,CF}[-\lambda],\text{ac})$ ($\mathcal{L}_{\text{acc}}(\text{RSC,CF}[-\lambda],\text{ac})$) denotes the family of languages generated (accepted) by random string context grammars with [λ-free] context-free productions. If any permitting context Q is empty, we put an f (indicating we only consider forbidding context) in front of RSC. If any forbidding context R is empty, we call the corresponding language families $\mathcal{L}_{\text{gen}}(\text{RSC,CF}[-\lambda])$ ($\mathcal{L}_{\text{acc}}(\text{RSC,CF}[-\lambda])$).

Applying our metatheorem 2.3, we obtain:

Corollary 3.14 $\mathcal{L}_{\text{gen}}(X,Y) = \mathcal{L}_{\text{acc}}(X,Y)$, and $\mathcal{L}_{\text{gen}}(\text{RSC},Y,\text{ac}) = \mathcal{L}_{\text{acc}}(\text{RSC},Y,\text{ac})$, where $X \in \{\text{RSC},\text{fRSC}\}$, $Y \in \{\text{CF},\text{CF}-\lambda\}$. □

Note that random string context grammars with context-free rules are a restriction of $(i,3)$-grammars. For any production $(A \to w, Q, R)$ of a given RSC grammar, introduce a production $A \to w$ into the simulating $(i,3)$-grammar attached with the language

$$\rho(A \to w) = (\bigcap_{B\in Q} V_G^*\{B\}V_G^*) \setminus (\bigcup_{C\in R} V_G^*\{C\}V_G^*).$$

In some sense, RSC grammars are more closely related to 2-conditional grammars of type $(i,3)$ as introduced by Navrátil[8], see [22, page 185], where the context left and right to the letter/word to be replaced is considered separately.

It is known that context-free fRC grammars are equivalent to generating ordered grammars regarding their descriptional power [5, Theorem 2.3.4]. We are going to prove a similar characterization of context-free fRSC grammars via accepting ordered grammars.

Theorem 3.15 (i) $\mathcal{L}_{\text{gen}}(\text{fRSC,CF}-\lambda) = \mathcal{L}_{\text{gen}}(\text{CS}) = \mathcal{L}_{\text{acc}}(\text{O,CF}-\lambda)$;

(ii) $\mathcal{L}_{\text{gen}}(\text{fRSC,CF}) = \mathcal{L}_{\text{gen}}(\text{RE}) = \mathcal{L}_{\text{acc}}(\text{O,CF})$.

Proof. In random context grammars, it is rather easy to show the equivalence of fRC grammars and generating ordered grammars. Of course, by our above theorems, any fRSC grammar can be simulated by an accepting ordered grammar, first constructing an equivalent linear bounded automaton or an equivalent Turing machine.

On the other hand, a construction analogous to [5, Theorem 2.3.4] showing the above-claimed equivalences of fRSC grammars and accepting ordered grammars is rather involved, since in an ordered grammar a production $w \to A$ might be blocked by a greater rule $w' \to A'$, with, e.g., $wx = yw'$ for some x, y with $|y| < |w|$. In fRSC grammars, we can only check the context of w in the string to be rewritten, disregarding possible overlaps.

We already noticed that, from the construction given in the proof of Theorem 3.5, it is possible to get a normal form representation for ordered grammars. Unfortunately, in the construction given there, overlap occurs in point (2d) when testing the left context. We may overcome this difficulty by introducing additional symbols of the form $[B, r]$ into the nonterminal alphabet of the simulating accepting ordered grammar.[9] These symbols should be introduced by a rule $B \to [B, r]$ between (2a) and (2b) and appropriately handled by the other (test) productions. (2d) is replaced by

[8] Navrátil called these grammars 'context-free grammars with (regular) conditions' in [20]. Observe that our metatheorem applies also to these grammars.

[9] This modification cannot be done when starting with a grammar in Penttonen normal form.

(2d1) $[A,r] \to (A,r) \prec \{A \to F, B \to F, (A,s) \to F, [A,s'] \to F,\ [T,s'] \to F,\ [B,s'] \to F,$ $(T,s') \to F,\ [Y,r]y \to F \mid s \in Lab,\ s' \in Lab \setminus \{r\},\ T \in V_N,\ y \in V_{G'} \setminus \{(Z,r)\}\};$

(2d2) $[B,r] \to B \prec \{B \to F,\ z(Z,r) \to F,\ [T,s'] \to F,\ [B,s'] \to F,\ \mid\ s' \in Lab \setminus \{r\},$ $T \in V_N,\ z \in V_{G'} \setminus \{[Y,r]\}\}$

This overlapfree normal form enables us to apply the standard procedure from [5, Theorem 2.3.4], delivering the claimed result.

The case including λ-rules is treated as before. □

Combining this with our previous results, we see that fRSC grammars are more powerful than fRC grammars. More precisely, we find:

Corollary 3.16 (i) $\mathcal{L}_{\text{gen}}(\text{fRC,CF}-\lambda) = \mathcal{L}_{\text{acc}}(\text{fRC,CF}-\lambda) \subset \mathcal{L}_{\text{gen}}(\text{fRSC,CF}-\lambda) = \mathcal{L}_{\text{acc}}(\text{fRSC,CF}-\lambda) = \mathcal{L}_{\text{gen}}(\text{CS});$

(ii) $\mathcal{L}_{\text{gen}}(\text{fRC,CF}) = \mathcal{L}_{\text{acc}}(\text{fRC,CF}) \subset \mathcal{L}_{\text{gen}}(\text{fRSC,CF}) = \mathcal{L}_{\text{acc}}(\text{fRSC,CF}) = \mathcal{L}_{\text{gen}}(\text{RE}).$ □

Furthermore, for RC grammars, it is known that the restriction to only forbidding (or to only permitting) context restricts the descriptive power of the mechanism. For RSC grammars, such a restriction cannot be observed. A similar remark is valid as regards appearance checking.

We conclude this section comparing RC and RSC grammars with respect to the restriction to only forbidding (or to only permitting) context.

Theorem 3.17 (i)
- $\mathcal{L}_{\text{gen}}(\text{fRC,CF}-\lambda) \subset \mathcal{L}_{\text{gen}}(\text{RC,CF}-\lambda,\text{ac}) \subset \mathcal{L}_{\text{gen}}(\text{CS});$
- $\mathcal{L}_{\text{gen}}(\text{RC,CF}-\lambda) \subset \mathcal{L}_{\text{gen}}(\text{RC,CF}-\lambda,\text{ac}) \subset \mathcal{L}_{\text{gen}}(\text{CS});$
- $\mathcal{L}_{\text{gen}}(\text{fRC,CF}) \subset \mathcal{L}(\text{REC}) \subset \mathcal{L}_{\text{gen}}(\text{RC,CF,ac}) = \mathcal{L}_{\text{gen}}(\text{RE});$
- $\mathcal{L}_{\text{gen}}(\text{RC,CF}) \subset \mathcal{L}(\text{REC}) \subset \mathcal{L}_{\text{gen}}(\text{RC,CF,ac}) = \mathcal{L}_{\text{gen}}(\text{RE});$

(ii)
- $\mathcal{L}_{\text{gen}}(\text{fRSC,CF}-\lambda) = \mathcal{L}_{\text{gen}}(\text{RSC,CF}-\lambda,\text{ac}) = \mathcal{L}_{\text{gen}}(\text{CS});$
- $\mathcal{L}_{\text{gen}}(\text{RSC,CF}-\lambda) \subseteq \mathcal{L}_{\text{gen}}(\text{RSC,CF}-\lambda,\text{ac}) = \mathcal{L}_{\text{gen}}(\text{CS});$
- $\mathcal{L}_{\text{gen}}(\text{fRSC,CF}) = \mathcal{L}_{\text{gen}}(\text{RSC,CF,ac}) = \mathcal{L}_{\text{gen}}(\text{RE});$
- $\mathcal{L}_{\text{gen}}(\text{RSC,CF}) \subseteq \mathcal{L}_{\text{gen}}(\text{RSC,CF,ac}) = \mathcal{L}_{\text{gen}}(\text{RE});$

Proof. $\mathcal{L}_{\text{gen}}(\text{RSC,CF}-\lambda,\text{ac}) \subseteq \mathcal{L}_{\text{gen}}(\text{CS})$ and $\mathcal{L}_{\text{gen}}(\text{RSC,CF,ac}) \subseteq \mathcal{L}_{\text{gen}}(\text{RE})$ can be shown by standard constructions.

$\mathcal{L}_{\text{gen}}(\text{RSC,CF}-\lambda,\text{ac}) \subseteq \mathcal{L}_{\text{gen}}(\text{CS})$ is seen via a standard construction of a simulating linear bounded automaton. Similarly, or by appealing to Church's thesis, $\mathcal{L}_{\text{gen}}(\text{RSC,CF,ac}) \subseteq \mathcal{L}_{\text{gen}}(\text{RE})$ can be shown.

For the strictness of the inclusion $\mathcal{L}_{\mathrm{gen}}(\mathrm{fRC,CF}{-}\lambda) \subset \mathcal{L}_{\mathrm{gen}}(\mathrm{RC,CF}{-}\lambda, \mathrm{ac})$ see [31] combined with [5, Theorem 1.2.4]. The strictness of the inclusions $\mathcal{L}_{\mathrm{gen}}(\mathrm{RC,CF}) \subset \mathcal{L}(\mathrm{REC}) \subset \mathcal{L}_{\mathrm{gen}}(\mathrm{RC,CF,ac})$ follows combining [5, Theorem 1.2.4] and [18, Corollary 5]. □

The only thing left open is whether the inclusions $\mathcal{L}_{\mathrm{gen}}(\mathrm{CS}) \supseteq \mathcal{L}_{\mathrm{gen}}(\mathrm{RSC,CF}{-}\lambda)$ and $\mathcal{L}_{\mathrm{gen}}(\mathrm{RE}) \supseteq \mathcal{L}_{\mathrm{gen}}(\mathrm{RSC,CF})$ are strict or not. basically because we did not see any way to prevent multiple applications of the same rule. This situation contrasts a bit with the one encountered with string random context grammars with left derivation (another possible variant of random context grammars), whereas the forbidding context restriction still poses problems, where the permitting context restriction is fairly easily solved, see [5, page 89]. Another idea to prove the strictness of the inclusions $\mathcal{L}_{\mathrm{gen}}(\mathrm{RSC,CF}[-\lambda]) \subseteq \mathcal{L}_{\mathrm{gen}}(\mathrm{RSC,CF}[-\lambda], \mathrm{ac})$ would be to use [18], but we were not able to show this either.

Chapter 4

Matrix Grammars, Programmed Grammars and Grammars with Regular Control

4.1 Matrix Grammars

A *matrix grammar* ([1, 5]) is a quintuple $G = (V_N, V_T, M, S, F)$, where V_N, V_T, and S are defined as in Chomsky grammars (the alphabet of nonterminals, the terminal alphabet, and the axiom), M is a finite set of matrices each of which is a finite sequence $m : (\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \ldots, \alpha_n \to \beta_n)$, $n \geq 1$, of 'usual' rewriting rules over V_G, and F is a finite set of occurrences of such rules in M. For some words x and y in V_G^* and a matrix $m : (\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \ldots, \alpha_n \to \beta_n)$ in M, we write $x \underset{m}{\Longrightarrow} y$ (or simply $x \Rightarrow y$ if there is no danger of confusion) iff there are strings $x_0, x_1, \ldots, x_n$ such that $x_0 = x$, $x_n = y$, and for $1 \leq i \leq n$, either

$$x_{i-1} = z_{i-1}\alpha_i z'_{i-1},\ x_i = z_{i-1}\beta_i z'_{i-1} \text{ for some } z_{i-1},\ z'_{i-1} \in V_G^*$$

or $x_{i-1} = x_i$, the rule $\alpha_i \to \beta_i$ is not applicable to x_{i-1}, and this occurrence of $\alpha_i \to \beta_i$ appears in F. One says that the rules whose occurrences appear in F are used in *appearance checking mode*, and that a matrix grammar is defined with (without) appearance checking if $F \neq \emptyset$ ($F = \emptyset$). The language generated (accepted) by G is defined as $L_{\text{gen}}(G) = \{w \in V_T^* \mid S \overset{*}{\Rightarrow} w\}$ ($L_{\text{acc}}(G) = \{w \in V_T^* \mid w \overset{*}{\Rightarrow} S\}$).

The family of languages generated (accepted) by matrix grammars with appearance checking and with context-free rules $\alpha \to \beta$ in their matrices shall be denoted by $\mathcal{L}_{\text{gen}}(\text{M}, \text{CF}, \text{ac})$ ($\mathcal{L}_{\text{acc}}(\text{M,CF, ac})$), and for the family of languages generated (accepted) by matrix grammars without appearance checking and with context-free rules we shall use the notation $\mathcal{L}_{\text{gen}}(\text{M,CF})$ ($\mathcal{L}_{\text{acc}}(\text{M,CF})$). If λ-rules are forbidden in the matrices we arrive at the families $\mathcal{L}_{\text{gen}}(\text{M,CF}\text{–}\lambda, \text{ac})$ ($\mathcal{L}_{\text{acc}}(\text{M,CF}\text{–}\lambda, \text{ac})$) or $\mathcal{L}_{\text{gen}}(\text{M,CF}\text{–}\lambda)$ ($\mathcal{L}_{\text{acc}}(\text{M,CF}\text{–}\lambda)$), respectively.

Clearly, the concept of the matrix grammar $G^d = (V_N, V_T, M^d, S, F^d)$ dual to a matrix grammar $G = (V_N, V_T, M, S, F)$ should be determined as follows:

- if a matrix $m : (\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \ldots, \alpha_n \to \beta_n)$ appears in M, then $m' : (\beta_n \to \alpha_n, \ldots, \beta_1 \to \alpha_1)$ is contained in M^d,
- there are no further matrices in M^d,
- the occurrence of some rule $\beta_i \to \alpha_i$ in some matrix $m' : (\beta_n \to \alpha_n, \ldots, \beta_1 \to \alpha_1)$ is in F^d iff the corresponding occurrence of this rule (i.e. $\alpha_i \to \beta_i$ in the corresponding matrix m from which m' is constructed as above) is in F.

By a proof analogous to the proof of [29, Theorem I.2.1], we find the following theorem.

Theorem 4.1 (i) $L_{\text{gen}}(G) = L_{\text{acc}}(G^d)$, for any generating matrix grammar G without appearance checking.

(ii) $L_{\text{acc}}(G) = L_{\text{gen}}(G^d)$, for any accepting matrix grammar G without appearance checking. □

Even in case of matrix grammars, we might have argued with our metatheorem. We show this construction in the following in case of matrix grammars with erasing core productions. Contrary to the so-far applications of the metatheorem, this construction does not carry over to the case of pure matrix grammars.

Let $m_1, \ldots, m_t$ be the matrices of $G = (V_N, V_T, M, S)$, and

$$(\alpha_{i1} \to \beta_{i1}, \ldots, \alpha_{ik_i} \to \beta_{ik_i})$$

be the production sequence of matrix m_i. We introduce a unique label p_{ij} ($1 \leq i \leq t$, $1 \leq j \leq k_i$). In the equivalent cc grammar $G' = (V'_N, V_T, P, \{S'\})$ with $V'_N = V_N \cup \{p_{ij} \mid 1 \leq i \leq t, 1 \leq j \leq k_i\} \cup \{S'\}$ (the unions being disjoint), we have a start table containing $\{S' \to Sp_{i1}, \{\#\} \times \{1\} \mid 1 \leq i \leq t\}$, a termination table $\{(p_{i1} \to \lambda, V_T^*\{\#\} \times \{1\}) \mid 1 \leq i \leq t\}$, and, for each p_{ij}, $j < k_i$, a table $\{(\alpha_{ij} \to \beta_{ij}, V_G^*\{\#\}V_G^*\{\#\} \times \{1\}), (p_{ij} \to p_{i,j+1}, V_G^*\{\#\}V_G^*\{\#\} \times \{2\})\}$, and, for each p_{ik_i}, a table $\{(\alpha_{ik_i} \to \beta_{ik_i}, V_G^*\{\#\}V_G^*\{\#\} \times \{1\})\} \cup \{(p_{ik_i} \to p_{s1}, V_G^*\{\#\}V_G^*\{\#\} \times \{2\}) \mid 1 \leq s \leq t\}$.

On the other hand, a cc grammar of the given form can be readily transformed into an equivalent matrix grammar.

Observe that nearly the same construction is valid for matrix grammars with leftmost restrictions of type *left-3*, see [5], showing that also in that case (without appearance checking) generating and accepting grammars are equally powerful.

A similar but more awkward construction is possible in the non-erasing case.

Nevertheless, in this case we think that the direct proof given through the concept of dual grammars is more lucid than the deviation via cc grammars.

We would like to add a caveat regarding transformations into cc grammars in this place. Observe that we could also include appearance checking features in cc grammars which are not reversible, e.g., via a table $\{(p_{ij} \to p_{i,j+1}, (V_G^* \setminus (V_G^*\{\alpha_{ij}\}V_G^*))\{\#\} \times \{1\})\}$. Taking the dual grammar, we would exclude β_{ij} instead of α_{ij}, which gives some feeling why appearance checking really behaves different, as it is shown below.

Considering matrix grammars with appearance checking, we find the assertion of Theorem 4.1 to be violated by the following example: let $G = (\{A\}, \{a\}, \{(A \to aa, A \to a)\}, A, F)$ be a generating matrix grammar, where F contains both $A \to aa$ and $A \to a$, then we have $L_{\mathrm{gen}}(G) = \{aa\}$ but $L_{\mathrm{acc}}(G^d) = \{a, aa\}$.

This difficulty cannot be overcome by some normal form result similar to Lemma 3.1. This will be seen by Corollary 4.12 and can be illustrated by the next example.

Example 4.2 Let $G = (\{S, S', X, Y\}, \{a, b\}, M, S', F)$ be the generating matrix grammar with appearance checking, where M contains the matrices

$$\begin{aligned}
m_0 &: \ (S' \to S)\,,\\
m_1 &: \ (S' \to X)\,,\\
m_2 &: \ (X \to SX)\,,\\
m_3 &: \ (X \to b)\,,\\
m_4 &: \ (X \to Y, S \to a, S \to aS)\,,
\end{aligned}$$

and F consists of all rules from matrix m_4. Then $L_{\mathrm{gen}}(G) = \{a, b\} \cup \{a^{2n-1}b \mid n \geq 1\}$. This can be seen as follows. Starting with m_0, we get the uniquely determined derivation $S' \underset{m_0}{\Longrightarrow} S \underset{m_4}{\Longrightarrow} a$. If we first apply the matrix m_1, then either we can terminate to b by m_3 or we can derive SX by m_2. A string S^nX can be rewritten by m_2 yielding $S^{n+1}X$ or by m_3 deriving S^nb. The application of m_4 to S^nX can be disregarded since it introduces Y which is a trap symbol. From S^nb the derivation must be continued by m_4, and we get

$$S^nb \underset{m_4}{\Longrightarrow} x_1b \underset{m_4}{\Longrightarrow} x_2b \underset{m_4}{\Longrightarrow} \ldots \underset{m_4}{\Longrightarrow} x_nb\,,$$

where $|x_i|_S = n - i$ and $|x_i|_a = 2i$, for $1 \leq i \leq n-1$, and $x_n = a^{2n-1}$.

On the other hand, by G^d the following derivation is possible: $a^2 \underset{m_4'}{\Longrightarrow} aS \underset{m_4'}{\Longrightarrow} S \underset{m_0'}{\Longrightarrow} S'$. Hence, $a^2 \in L_{\mathrm{acc}}(G^d)$ and $a^2 \notin L_{\mathrm{gen}}(G)$.

4.2 Programmed Grammars

A *programmed grammar* ([25, 5]) is a construct $G = (V_N, V_T, P, S)$, where V_N, V_T, and S are defined as in Chomsky grammars, again, and P is a finite set of productions of the form $(r : \alpha \to \beta,\ \sigma(r),\ \phi(r))$, where $r : \alpha \to \beta$ is a rewriting rule labelled by r and $\sigma(r)$ and $\phi(r)$ are two sets of labels of such core rules in P. By $\mathrm{Lab}(P)$, we denote the set of all labels of the productions appearing in P.

A sequence of words over V_G^*, $y_0, y_1, \ldots, y_n$, is referred to as a derivation in G iff, for $1 \leq i \leq n$, there are productions $(r_i : \alpha_i \to \beta_i,\ \sigma(r_i),\ \phi(r_i)) \in P$ such that

$$y_{i-1} = z_{i-1}\alpha_i z'_{i-1},\ y_i = z_{i-1}\beta_i z'_{i-1},\ \text{and, if } 1 \leq i < n,\ r_{i+1} \in \sigma(r_i)$$

or

$$\alpha_i \notin \mathrm{Sub}(y_{i-1}),\ y_{i-1} = y_i,\ \text{and, if } 1 \leq i < n,\ r_{i+1} \in \phi(r_i).$$

Note that – in terms of the concept of matrix grammars – in the latter case, the derivation step is done in appearance checking mode. The set $\sigma(r_1)$ is called success field and the set $\phi(r_1)$ failure field of r_1. The language generated by G is defined as

$$L_{\mathrm{gen}}(G) = \{w \in V_T^* \,|\, \text{there is a derivation } S = y_0, y_1, \ldots, y_n = w\}$$

if G is a generating programmed grammar and

$$L_{\mathrm{acc}}(G) = \{w \in V_T^* \,|\, \text{there is a derivation } w = y_0, y_1, \ldots, y_n = S\}$$

if G is an accepting one.

The family of languages generated (accepted) by programmed grammars of the form $G = (V_N, V_T, P, S)$ containing only context-free core rules is denoted by $\mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ac})$. When no appearance checking features are involved, i.e. $\phi(r) = \emptyset$ for each rule in P, we are led to the families $\mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF})$ ($\mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF})$). The special variant of a programmed grammar, where the success field and the failure field coincide for each rule in the set P of productions, is said to be a programmed grammar with *unconditional transfer* . According to the notation which has been introduced up to here, we shall denote the class of languages generated (accepted) by programmed grammars with context-free productions and with unconditional transfer by $\mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ut})$ ($\mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF,ut})$). If erasing rules are forbidden, we replace the component CF by CF–λ in that notation.

Let $G = (V_N, V_T, P, S)$ be a programmed grammar (in generating or in accepting mode). Then the dual grammar G^d is the quadruple (V_N, V_T, P^d, S), where P^d is constructed as follows: with a production $(r : \alpha \rightarrow \beta, \sigma(r), \phi(r))$ we associate the production $(r^d : \beta \rightarrow \alpha,\ \sigma^d(r),\ \phi^d(r))$ with $\sigma^d(r) = \{s \in \mathrm{Lab}(P) \,|\, r \in \sigma(s)\}$ and $\phi^d(r) = \{s \in \mathrm{Lab}(P) \,|\, r \in \phi(s)\}$. Then we have $(G^d)^d = G$, and G^d is of the same type as G.

Again, similar to [29, Theorem I.2.1], we find the next theorem. Also in this case, we might have applied our metatheorem directly, but we did not do this, since as in matrix grammars the required construction is much more complicated than the natural concept of a dual grammar.

Theorem 4.3 (i) $L_{\mathrm{gen}}(G) = L_{\mathrm{acc}}(G^d)$, for any generating programmed grammar G without appearance checking.[1]

(ii) $L_{\mathrm{acc}}(G) = L_{\mathrm{gen}}(G^d)$, for any accepting programmed grammar G without appearance checking. □

That this statement does not hold for programmed grammars in general can be illustrated by analogous arguments as given for the matrix case (e.g., see Example 4.2).

[1] Note that a programmed grammar with unconditional transfer is not a programmed grammar without appearance checking.

4.3 Grammars With Regular Control

Let $G' = (V_N, V_T, P, S)$ be a type-n grammar with labelled rules, i.e. $P = \{r_1 : \alpha_1 \to \beta_1,\, r_2 : \alpha_2 \to \beta_2,\, \ldots,\, r_n : \alpha_n \to \beta_n\}$, let $\mathrm{Lab}(P)$ be the set of all labels $\{r_1, r_2, \ldots, r_n\}$, $F \subseteq \mathrm{Lab}(P)$, and let R be a regular language over the alphabet $\mathrm{Lab}(P)$. Then $G = (V_N, V_T, P, S, R, F)$ is refered to be a (type-n) *grammar with regular control* (cf. [15, 5, 29]). The language generated by G consists of all words w for which there is a word $r_{i_1} r_{i_2} \cdots r_{i_k}$ in R and there are strings x_0, x_1, …, x_k such that $x_0 = S$ and $x_k = w$ if G' is generating, and $x_0 = w$, $x_k = S$ if G' is accepting, respectively, and for $1 \leq j \leq k$, either

$$x_{j-1} = z_{j-1}\alpha_{i_j} z'_{j-1},\ x_j = z_{j-1}\beta_{i_j} z'_{j-1} \text{ for some } z_{j-1},\ z'_{j-1} \in V_G^*\,,$$

or $x_{j-1} = x_j$, the rule with label r_{i_j} is not applicable to x_{j-1}, and $r_{i_j} \in F$. The rules with a label in F are used in appearance checking mode, since, during a derivation according to a word in R, they can be passed over if not applicable. Clearly, if $F = \emptyset$ then G is said to be without appearance checking.

By $\mathcal{L}_{\mathrm{gen}}(\mathrm{rC,CF,ac})$, the family of languages generated by context-free grammars with regular control is denoted. In the same manner as, e.g., for matrix grammars, the classes $\mathcal{L}_{\mathrm{gen}}(\mathrm{rC,CF}-\lambda,\mathrm{ac})$, $\mathcal{L}_{\mathrm{gen}}(\mathrm{rC,CF})$, $\mathcal{L}_{\mathrm{gen}}(\mathrm{rC,CF}-\lambda)$, $\mathcal{L}_{\mathrm{acc}}(\mathrm{rC,CF,ac})$, $\mathcal{L}_{\mathrm{acc}}(\mathrm{rC,CF}-\lambda,\mathrm{ac})$, $\mathcal{L}_{\mathrm{acc}}(\mathrm{rC,CF})$, and $\mathcal{L}_{\mathrm{acc}}(\mathrm{rC,CF}-\lambda)$ are introduced.

Given a grammar $G = (V_N, V_T, P, S, R, F)$ with regular control, its dual is determined as $G^d = (V_N, V_T, P^d, S, \mathrm{Mi}(R), F)$, where $P^d = \{r_i : \beta_i \to \alpha_i \,|\, r_i : \alpha_i \to \beta_i \in P\}$. Indeed, G^d is a grammar with regular control, since the family of regular languages is closed under mirror image[2]. Once more, by analogous arguments as in the proof of Theorem I.2.1 in [29], we find a trivial equivalence between generating and accepting mode for grammars with regular control if appearance checking is forbidden.

Theorem 4.4 (i) $L_{\mathrm{gen}}(G) = L_{\mathrm{acc}}(G^d)$, for any generating grammar G with regular control that is defined without appearance checking.

(ii) $L_{\mathrm{acc}}(G) = L_{\mathrm{gen}}(G^d)$, for any accepting grammar G with regular control and without appearance checking. □

We summarize our previous results in the following.

Corollary 4.5 $\mathcal{L}_{\mathrm{gen}}(X,Y) = \mathcal{L}_{\mathrm{acc}}(X,Y)$, where $X \in \{\mathrm{M,P,rC}\}$, $Y \in \{\mathrm{CF,CF}-\lambda\}$. □

Even in this case our metatheorem is directly applicable. Let G be a grammar with regular control, $G = (V_N, V_T, P, S, R, F)$, where $F = \emptyset$. Since R is a regular language, there is a finite automaton[3] A which recognizes R, $A = (\mathrm{Lab}(P), Z, z_0, Q, \delta)$, where $\mathrm{Lab}(P)$ is the input alphabet, $Z = \{z_0, z_1, \ldots, z_k\}$ is the set of states, z_0 is the initial state, $Q \subseteq Z$

[2] By this argument, the concept of the dual to a grammar with control, where the control language is taken from any language family which is closed under mirror image, can be given in this way.

[3] For details concerning the notion of finite automaton, confer to, e.g., [29] or [5].

is the set of final states, and δ is the transition function of A. We construct an equivalent cc grammar $G' = (V_N', V_T, P', \{S'\})$ with $V_N' = V_N \cup \{[r,z] \mid r \in \mathrm{Lab}(P),\ z \in Z\} \cup \{S'\}$ (the unions being disjoint), P' containing a start table $\{(S' \to S[r,z_0],\ \{\#\} \times \{1\}) \mid r \in \mathrm{Lab}(P)\}$, a terminating table $\{([r,q] \to \lambda,\ V_T^*\{\#\} \times \{1\}) \mid r \in \mathrm{Lab}(P),\ q \in Q\}$, and for each production $(r : \alpha \to \beta) \in P$, a table $\{(\alpha \to \beta,\ V_G^*\{\#\}V_G^*\{\#\} \times \{1\})\} \cup \{([r,z] \to [r',z'],\ V_G^*\{\#\}V_G^*\{\#\} \times \{2\}) \mid r' \in \mathrm{Lab}(P),\ z' \in \delta(z,r)\}$.

Conversely, a cc grammar of the given form can be readily transformed into an equivalent grammar with regular control.

4.4 Appearance Checking Features

Theorem 4.6 For $Y \in \{\mathrm{CF}\text{–}\lambda, \mathrm{CF}\}$, we have

(i) $\mathcal{L}_{\mathrm{acc}}(\mathrm{M}, Y) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P}, Y) = \mathcal{L}_{\mathrm{acc}}(\mathrm{rC}, Y)$,

(ii) $\mathcal{L}_{\mathrm{acc}}(\mathrm{M}, Y, \mathrm{ac}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P}, Y, \mathrm{ac}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{rC}, Y, \mathrm{ac})$.

Proof. (i) Because of Corollary 4.5 it is sufficient to refer to the following. In [5, Theorem 1.2.2] the equivalence of the classes $\mathcal{L}_{\mathrm{gen}}(\mathrm{M}, Y)$ and $\mathcal{L}_{\mathrm{gen}}(\mathrm{P}, Y)$, and in [29, Theorem V.6.1 and Theorem V.6.6] the equivalence of the language families $\mathcal{L}_{\mathrm{gen}}(\mathrm{M}, Y)$ and $\mathcal{L}_{\mathrm{gen}}(\mathrm{rC}, Y)$ has been proved.

(ii) (a) $\mathcal{L}_{\mathrm{acc}}(\mathrm{M}, Y, \mathrm{ac}) \subseteq \mathcal{L}_{\mathrm{acc}}(\mathrm{P}, Y, \mathrm{ac})$.

Let $G = (V_N, V_T, M, S, F)$ be an accepting matrix grammar (with appearance checking), $M = \{m_1, m_2, \ldots, m_r\}$, $m_i : (r_{i1}, r_{i2}, \ldots, r_{ik_i})$, $r_{ij} : v_{ij} \to A_{ij}$, $1 \le i \le r$, $1 \le j \le k_i$. We set

$$V_T' = \{a' \mid a \in V_T\} \quad \text{and} \quad \overline{V_N} = V_N \cup V_T' \cup \{\overline{S}\},$$

(the unions being disjoint), and we define a homomorphism h by

$$h(a) = a' \text{ for } a \in V_T, \quad h(A) = A \text{ for } A \in V_N .$$

Now, we construct the accepting programmed grammar $\overline{G} = (\overline{V_N}, V_T, P, \overline{S})$, where P contains the following productions:

$$\begin{array}{ll}
(1) & ([a] : a \to a',\ \{[b] \mid b \in V_T\},\ \{[1,1],[2,1],\ldots,[r,1]\}), \\
(2) & ([i,j] : h(v_{ij}) \to A_{ij},\ \{[i,j+1]\},\ \delta(i,j)) \\
 & \qquad\qquad \text{if } r_{ij} : v_{ij} \to A_{ij},\ 1 \le i \le r,\ 1 \le j < k_i , \\
(3) & ([i,k_i] : h(v_{ik_i}) \to A_{ik_i},\ \{[1,1],[2,1],\ldots,[r,1],*\},\ \delta(i,k_i)) \\
 & \qquad\qquad \text{if } r_{ik_i} : v_{ik_i} \to A_{ik_i},\ 1 \le i \le r , \\
(4) & (* : S \to \overline{S},\ \{*\},\ \emptyset),
\end{array}$$

where, for $1 \le i \le r$, $1 \le j \le k_i$,

$$\delta(i,j) = \begin{cases} \emptyset & r_{ij} \notin \mathrm{F} \\ \sigma([i,j]) & r_{ij} \in \mathrm{F} \end{cases} .$$

Obviously, any derivation by $\overline{G}$ has to start by application of a production of type (1) and thus, all occurrences of terminal symbols have to be replaced by their primed versions. Now, the productions of type (2) and (3) are used in order to simulate the application of matrices in M. The goal symbol can be derived if and only if the sentential form S has been obtained by application of a production of type (3). Hence, $L(\overline{G}) = L(G)$.

Clearly, if G is defined without erasing rules, $\overline{G}$ is an accepting programmed grammar without erasing rules, too.

(ii) (b) $\mathcal{L}_{\text{acc}}(\mathrm{P}, Y, \mathrm{ac}) \subseteq \mathcal{L}_{\text{acc}}(\mathrm{rC}, Y, \mathrm{ac})$.

We omit this proof since the construction in the proof of [29, Theorem V.6.1] for the corresponding generating devices can be used here.

(ii) (c) $\mathcal{L}_{\text{acc}}(\mathrm{rC}, Y, \mathrm{ac}) \subseteq \mathcal{L}_{\text{acc}}(\mathrm{M}, Y, \mathrm{ac})$.

Let $G = (V_N, V_T, P, S, R)$ be an accepting context-free grammar with regular control, $V_N = \{A_1, A_2, \ldots, A_n\}$, and $P = \{r_1 : v_1 \to A_{i_1},\, r_2 : v_2 \to A_{i_2},\, \ldots,\, r_m : v_m \to A_{i_m}\}$, and let A be the finite automaton which recognizes R, $A = (\mathrm{Lab}(P), Z, z_0, Q, \delta)$, where $\mathrm{Lab}(P)$ is the input alphabet, $Z = \{z_0, z_1, \ldots, z_k\}$ is the set of states, z_0 is the initial state, Q is the set of final states, and δ is the transition function of A. We construct an accepting matrix grammar $G' = (V_N', V_T, M, S', F')$, where

$$V_N' = V_N \cup \{[A, z] \mid A \in V_N,\ z \in Z\} \cup \{E\}\,,$$

(the unions being disjoint), and M containing the following matrices:

$$\begin{array}{ll}
(1) & ([A_1, z_0] \to E,\ \ldots,\ [A_n, z_0] \to E,\\
 & \ [A_1, z_1] \to E,\ \ldots,\ [A_n, z_1] \to E,\\
 & \qquad\qquad \vdots\\
 & \ [A_1, z_k] \to E,\ \ldots,\ [A_n, z_k] \to E,\ v_j \to [A_{i_j}, \delta(z_0, r_j)]) \text{ for } 1 \le j \le m,\\
(2) & ([A, z] \to A,\ v_j \to [A_{i_j}, \delta(z, r_j)]) \text{ for each } A \in V_N,\ z \in Z,\ 1 \le j \le m\,,\\
(3) & ([S, q] \to S') \text{ for each } q \in Q\,.
\end{array}$$

Let F' contain all productions with the letter E on their right-hand sides as well as all productions $v_j \to A_{i_j}$ in the matrices of type (2) whose label r_j is an element of F. Any derivation must start with a matrix of type (1) which simulates the application of one rule $r_j : v_j \to A_{i_j} \in P$ with $|v_j|_{V_N} = 0$ introducing the nonterminal $[A_{i_j}, z]$ such that z is the state that is reached by A after reading r_j as first input symbol. Since a further application of a matrix of type (1) introduces the trap symbol, there is exactly one nonterminal of the form $[A, z]$, $A \in V_N, z \in Z$, in every sentential form which is not a terminal word and yields the goal symbol. The task of a matrix of type (2) is twice. At first it simulates the application of a rule in P, and secondly a transition of the automaton A is calculated in the following manner: if we have simulated a derivation according to $r_1 r_2 \cdots r_{i_l} \in (\mathrm{Lab}(P))^*$ then the second component of the nonterminal of the form $[A, z]$, $A \in V_N, z \in Z$, in the derived sentential form equals the state that is reached by A after reading the input $r_{i_1} r_{i_2} \cdots r_{i_l}$. Thus, the goal symbol S' can be reached (by an application of the matrix (3)) iff the iterated application of matrices of type (2) simulates a derivation according to G

which yields S. Hence, $L_{\rm acc}(G') = L_{\rm acc}(G)$. Clearly, if G is λ-free then G' is λ-free, too. □

If appearance checking features are permitted in the accepting case, like in ordered grammars we can test both the containment and the noncontainment of a string. Indeed, we are led to results corresponding to the statements of Theorem 3.5 and Corollary 3.7 for accepting ordered grammars. First let us show the following.

Theorem 4.7 $\mathcal{L}_{\rm acc}({\rm O}, X) \subseteq \mathcal{L}_{\rm acc}({\rm P}, X, {\rm ut})$, where $X \in \{{\rm CF}, {\rm CF} - \lambda\}$.

Proof. The proof of [31, Теорема 4.3] of our statement for the generating case is transferable to the accepting case, cf. our meta-observation. □

Corollary 4.8 (i) For any (generating) context-sensitive grammar, there is an equivalent accepting nonerasing programmed grammar with unconditional transfer.

(ii) For any (generating) type-0 grammar, there is an equivalent accepting programmed grammar with unconditional transfer.

Proof. The assertion follows from Theorem 4.7, if we take into consideration the results presented in Theorem 3.5 and Corollary 3.7. □

Theorem 4.9 (i) $\mathcal{L}_{\rm acc}({\rm P}, {\rm CF}, {\rm ac}) \subseteq \mathcal{L}_{\rm acc}({\rm RE})$,

(ii) $\mathcal{L}_{\rm acc}({\rm P}, {\rm CF}\text{–}\lambda, {\rm ac}) \subseteq \mathcal{L}_{\rm acc}({\rm CS})$.

Proof. (i) Clearly, this statement follows from the Church's thesis, but it can also be proved directly. Let $G = (V_N, V_T, P, S)$ be an accepting programmed grammar with appearance checking. First let us mention that, without loss of generality, we can assume that $\phi(r) = \emptyset$ if $(r : v \to A, \sigma(r), \phi(r))$ is a production with $v = \lambda$ (a rule $\lambda \to A$ is applicable to any string over V_G).

We consider an accepting type-0 grammar $G' = (V_N', V_T, P', S')$ constructed as follows[4]. We set

$$V_N' = V_N \cup \{B, C, S'\} \cup \bigcup_{r \in {\rm Lab}(P)} \{A_r, B_r, C_r, D_r\},$$

where the unions shall be disjoint. P' contains the following rules:

1. Here we introduce some starting and ending rules.

$$\begin{array}{ll} (1) & a \to Ba \quad \text{for } a \in V_T \\ (2) & a \to aC \quad \text{for } a \in V_T \\ (3) & C \to A_rC \quad \text{for } r \in {\rm Lab}(P) \\ (4) & BSA_rC \to S' \quad \text{for } r \in {\rm Lab}(P) \end{array}$$

[4] Our construction is based on the proving idea of [5, Lemma 1.2.4].

(Because of the last rule, a derivation is accepting if (and – by the following rules – only if) each of the following conditions is satisfied:

- B has been derived exactly once as left marker,
- C has been derived exactly once as right marker,
- there is exactly one application of a rule of type (3) during the derivation.)

2. For any production $(r : v \to A, \sigma(r), \phi(r))$ in P, $v \neq \lambda$, we introduce the following rules in P'.

(5) $wA_r \to w'A_r z$ for $w, w' \in (V_N \cup V_T)^*$, $z \in V_N \cup V_T$, $|w| = |v|$, $v \neq w = w'z$
(6) $vA_r \to vB_r$
(7) $zB_r \to B_r z$ for $z \in V_N \cup V_T$
(8) $vB_r \to AC_r$
(9) $C_r z \to zC_r$ for $z \in V_N \cup V_T$
(10) $C_r C \to A_s C$ for $s \in \sigma(r)$
(11) $BwA_r \to BwD_r$ for $w \in (V_N \cup V_T)^*$, $|w| \leq |v| - 1$
(12) $D_r z \to zD_r$ for $z \in V_N \cup V_T$
(13) $D_r C \to A_s C$ for $s \in \phi(r)$

The nonterminal A_r checks whether the rule with label r is applicable. In the affirmative case it introduces B_r which can be replaced only by simulating the core rule of the production that is labeled by r. At this, C_r is introduced which moves to the right and derives A_s such that s is in the success field of r. If the production with label r is not applicable D_r is introduced which also moves to the right and derives A_s with $s \in \phi(r)$.
3. For any production $(r : \lambda \to A, \sigma(r), \emptyset) \in P$, we introduce a rule $A_r \to B_r$, and we add the rules (7) – (10) constructed above. (Clearly, the rule (8) is of the form $B_r \to AC_r$, now.)

Obviously, the sentential forms contain at most one letter A_r, B_r, C_r, or D_r for some $r \in \mathrm{Lab}(P)$. Hence, $L(G') = L(G)$, and the proof is complete for the type-0 case.

(ii) Now, let G be length-increasing. By the proof of Theorem I.2.1 in [29], the construction of G' in (i) implies that there is an equivalent generating type-0 grammar to G with

$$WS(x, G') \leq |x| + 3$$

for any $x \in L(G')$. From the workspace theorem [29, Theorem III.10.1] we obtain $L(G') \in \mathcal{L}_{\mathrm{gen}}(\mathrm{CS})$. In conclusion, $L(G) \in \mathcal{L}_{\mathrm{acc}}(\mathrm{CS})$, and the proof is finished. □

Because of the fact that any programmed grammar with unconditional transfer is an arbitrary programmed grammar (with appearance checking) of the same type, we arrive the expected corollary.

Corollary 4.10 (i) $\mathcal{L}_{\text{acc}}(\text{M}, \text{CF}, \text{ac}) = \mathcal{L}_{\text{acc}}(\text{P}, \text{CF}, \text{ac}) = \mathcal{L}_{\text{acc}}(\text{P}, \text{CF}, \text{ut}) = = \mathcal{L}_{\text{acc}}(\text{rC}, \text{CF}, \text{ac}) = \mathcal{L}_{\text{acc}}(\text{RE})$,

(ii) $\mathcal{L}_{\text{acc}}(\text{M,CF–}\lambda, \text{ac}) = \mathcal{L}_{\text{acc}}(\text{P,CF–}\lambda, \text{ac}) = \mathcal{L}_{\text{acc}}(\text{P,CF–}\lambda, \text{ut}) = = \mathcal{L}_{\text{acc}}(\text{rC,CF–}\lambda, \text{ac}) = \mathcal{L}_{\text{acc}}(\text{CS})$. □

In conclusion, we find a hierarchical result for programmed grammars in accepting mode which is still open in the generating case.

Corollary 4.11 For $Y \in \{\text{CF,CF–}\lambda\}$, we have

$\mathcal{L}_{\text{acc}}(\text{P}, Y) \subset \mathcal{L}_{\text{acc}}(\text{P}, Y, \text{ut})$ □

The main result of this section is given in the next corollary.

Corollary 4.12 Let $X, Y \in \{\text{M}, \text{P}, \text{rC}\}$. Then, we have

(i) $\mathcal{L}_{\text{gen}}(X, \text{CF}, \text{ac}) = \mathcal{L}_{\text{acc}}(Y, \text{CF}, \text{ac})$,

(ii) $\mathcal{L}_{\text{gen}}(X, \text{CF–}\lambda, \text{ac}) \subset \mathcal{L}_{\text{acc}}(Y, \text{CF–}\lambda, \text{ac})$.

Proof. By [5, Theorem 1.2.4 and Theorem 1.2.5], $\mathcal{L}_{\text{gen}}(\text{M,CF–}\lambda, \text{ac}) \subset \mathcal{L}_{\text{gen}}(\text{CS})$ and $\mathcal{L}_{\text{gen}}(\text{M}, \text{CF}, \text{ac}) = \mathcal{L}_{\text{gen}}(\text{RE})$ hold. By [29, Theorem V.6.1 and Theorem V.6.6], we have $\mathcal{L}_{\text{gen}}(\text{M}, X, \text{ac}) = \mathcal{L}_{\text{gen}}(\text{rC}, X, \text{ac})$, $X \in \{\text{CF,CF–}\lambda\}$. Since the family of generated type-n languages and the family of accepted type-n languages coincide for $n \in \{0, 1\}$, the statement is a direct consequence of Corollary 4.10. □

Note that we could have proved, e.g., $\mathcal{L}_{\text{acc}}(\text{CS}) \subseteq \mathcal{L}_{\text{acc}}(\text{M, CF–}\lambda, \text{ac})$ also directly, using an argument similar to the one given for ordered grammars. To conclude this section, we enclose such a proof below.

Let $L \in \mathcal{L}_{\text{gen}}(\text{CS})$, $L \subseteq V^+$. Obviously,

$$L = \bigcup_{a,b \in V} \{a\} d^l_a(d^r_b(L))\{b\} \cup (L \cap (V \cup V^2)).$$

This identity proves that, since accepting matrix languages are easily seen to be closed under finite union, it is sufficient for the proof of the present assertion to show that $\{a\}M\{b\} \in \mathcal{L}_{\text{acc}}(\text{M,CF}-\lambda\text{,ac})$ for $M \in \mathcal{L}_{\text{gen}}(\text{CS})$, $\lambda \notin M$.

We consider a generating context-sensitive grammar $G = (V_N, V_T, P, S)$ without λ-productions in Kuroda normal form generating M. We construct an accepting matrix grammar $\overline{G} = (\overline{V_N}, V_T \cup \{a, b\}, \overline{M}, \overline{S}, \overline{F})$ as follows. Let

$$\overline{V_N} = V_N \cup \{A, B, \overline{S}, E\} \cup \{Y' \mid Y \in V_N\}$$

(the unions beeing disjoint), $(\overline{V_N} \cup V_T) \setminus \{Y' \mid Y \in V_N\} = \{C_1, C_2, \ldots, C_n\}$, let $\overline{M}$ contain the following matrices:

$$\begin{array}{ll}
(a) & (ASB \to \overline{S}), \\
(b) & (A \to E,\ B \to E,\ a \to A,\ b \to B), \\
(c) & (A \to A,\ B \to B,\ x \to X) \quad \text{for all context-free rules } X \to x \in P, \\
(d) & (A \to A,\ B \to B,\ Y \to Y',\ Z \to Z', \\
 & Y'C_1 \to E,\ Y'C_2 \to E,\ \ldots,\ Y'C_n \to E, \\
 & C_1Z' \to E,\ C_2Z' \to E,\ \ldots,\ C_nZ' \to E, \\
 & Y' \to X,\ Z' \to U) \quad \text{for } XU \to YZ \in P,
\end{array}$$

and let $\overline{F}$ contain exactly all rules with the symbol E on their right-hand sides. One can easily prove that $L_{\text{acc}}(\overline{G}) = \{a\}M\{b\}$.

4.5 Comparison Results

Finally, we want to compare our results obtained so far in the accepting case with the corresponding known results in the generating case, cf. [5, Theorem 1.2.4] as regards matrix (or one of the equivalent regulation mechanisms) and random context grammars.

Theorem 4.13

- $\mathcal{L}_{\text{gen}}(\text{RC,CF}) = \mathcal{L}_{\text{acc}}(\text{RC,CF}) \subseteq \mathcal{L}_{\text{gen}}(\text{M,CF}) = \mathcal{L}_{\text{acc}}(\text{M,CF}) \subset \mathcal{L}(\text{REC})$.
- $\mathcal{L}_{\text{gen}}(\text{RC,CF} - \lambda) = \mathcal{L}_{\text{acc}}(\text{RC,CF} - \lambda) \subseteq \mathcal{L}_{\text{gen}}(\text{M,CF} - \lambda) = \mathcal{L}_{\text{acc}}(\text{M,CF} - \lambda) \subset \mathcal{L}(\text{CS})$.
- $\mathcal{L}_{\text{gen}}(\text{RC,CF,ac}) = \mathcal{L}_{\text{acc}}(\text{RC,CF,ac}) = \mathcal{L}_{\text{gen}}(\text{M,CF,ac}) = \mathcal{L}_{\text{acc}}(\text{M,CF,ac}) = \mathcal{L}(\text{RE})$.
- $\mathcal{L}_{\text{gen}}(\text{RC,CF}-\lambda, \text{ac}) = \mathcal{L}_{\text{acc}}(\text{RC,CF}-\lambda, \text{ac}) = \mathcal{L}_{\text{gen}}(\text{M,CF}-\lambda, \text{ac}) \subset \mathcal{L}_{\text{acc}}(\text{M,CF}-\lambda, \text{ac}) = \mathcal{L}(\text{CS})$. □

Now, we turn our attention to devices who are parallel in nature.

Chapter 5

Lindenmayer Systems

An *ET0L system* is a quadruple $G = (V, V', \{P_1, \ldots, P_r\}, \omega)$, where V' is a non-empty subset of the alphabet V, $\omega \in V^+$, and each so-called table P_i is a finite subset of $V \times V^*$ which satisfies the condition that, for each $a \in V$, there is a word $w_a \in V^*$ such that $(a, w_a) \in P_i$ (again, the elements of P_i are written as $a \to w_a$ in the generating case), such that each P_i defines a finite substitution $\sigma_i : V^* \to 2^{V^*}$. In the generating case, $x \Rightarrow y$ iff $y \in \sigma_i(x)$ for some i, and $L_{\text{gen}}(G) = \{v \in V'^* \,|\, \omega \overset{*}{\Rightarrow} v\}$. If no table P_i contains a λ-rule, i.e. $P_i \subseteq V \times V^+$ for each i, the system is called propagating; in this case, we add the letter P to the notation of the system. If for any table P_i, $a \to w \in P_i$ and $a \to w' \in P_i$ imply $w = w'$, the system is called deterministic; in this case, we add the letter D to the notation of the system.

The basic properties of the derivation in L systems (in contrast to the sequential rewriting mechanisms considered above) may be summarized as follows.

- Any symbol [subword] in the word under consideration has to be rewritten in one derivation step (in parallel).
- In the derivation, terminal symbols may be replaced.

Coming to accepting L systems, one necessary change in the formulation of the rewriting rule is that we must replace subwords instead of symbols, as indicated in square brackets above. But, this ad hoc approach is not sound for several reasons.

- In contrast to the generating process (where the now discussed item is trivial), we are forced to consider the partition of the word to be derived into non-overlapping [!] subwords separately as the first step inside a derivation step. Where in L systems, it is still easy to rule out segmentations which leave out some subwords underived, and are, hence, not partitions observed in the derivation process of L systems, we get into trouble within systems which are only partially parallel. We discuss this item in detail at the appropriate places.

- λ-productions cannot meaningfully be interpreted within the above intuition: Replace every λ occurring as some subword ... This problem is circumvented if we interpret the set of productions defining some inverse substitution.

- The intuition of the 'deterministic'-restriction in L systems is twofold:

  - in a static interpretation, we say an L system is deterministic if the left-hand sides of the productions are distinct in each table, or formally: each table P_i satisfies the condition: if $a \to w \in P_i$ and $a \to w' \in P_i$, then $w = w'$;
  - in a dynamic interpretation, a substitution σ_i is deterministic iff, for any $w \in V^*$, there is at most (and hence, exactly one) $v \in \sigma_i(w)$.

  In the generating mode, these two intuitions lead to the same notion of determinism, which is not the case in accepting mode. We will discuss this item in connection with pure grammars and systems. Here, we restrict ourselves to the static interpretation. In the accepting mode, this would mean that we require the left-hand sides of the productions of one table to be different.

We now formally introduce accepting L systems: Each table P_i is a finite subset of $V^* \times V$ which satisfies the condition[1] that, for each $a \in V$, there is a word $w_a \in V^*$ such that $(w_a, a) \in P_i$ (the elements of P_i are written as $w_a \to a$ again), such that each P_i defines a finite substitution $\sigma_i : V^* \to 2^{V^*}$, $\sigma_i(a) = \{w \mid w \to a \in P_i\}$. In the accepting case, $y \Rightarrow x$ iff $x \in \sigma_i^{-1}(y)$ for some i, and $L_{\text{acc}}(G) = \{v \in V'^* \mid v \overset{*}{\Rightarrow} \omega\}$. If no table P_i contains a λ-rule, i.e. $P_i \subseteq V^+ \times V$ for each i, we call the system propagating; in this case, we add the letter P to the notation of the system. If for any table P_i, $w \to a \in P_i$ and $w \to a' \in P_i$ imply $a = a'$, we call the system deterministic; in this case, we add the letter D to the notation of the system.

If $G = (V, V', \{P_1, \dots, P_r\}, \omega)$ is an ET0L system, then the number r of tables is denoted by $\text{Syn}(G)$. For $L \in \mathcal{L}_{\text{gen}}(\text{ET0L})$,

$$\text{Syn}_{\text{gen}}(L, \text{ET0L}) = \min\{\text{Syn}(G) \mid G \text{ is a generating ET0L system, and } L_{\text{gen}}(G) = L\}.$$

Analogously, for $L \in \mathcal{L}_{\text{acc}}(\text{ET0L})$,

$$\text{Syn}_{\text{acc}}(L, \text{ET0L}) = \min\{\text{Syn}(G) \mid G \text{ is an accepting ET0L system, and } L_{\text{acc}}(G) = L\}.$$

If $G = (V, V', \{P_1, \dots, P_r\}, \omega)$ is a generating ET0L system, for $1 \leq j \leq r$ and $a \in V$, we define $\text{Det}(G, P_j, a) = \#\{a \to y \mid a \to y \in P_j\}$. The degree of nondeterminism of

[1] This so-called 'completeness condition' for L systems cannot be transferred to the accepting case requiring that for any possible left-hand side, there is at least one rule in P_i, since the specification of P_i should remain finite. One might wish to abolish the completeness condition totally, and, at least in the generating case of systems with extensions [21], this modification does not alter the descriptive power. But we do not investigate this item in the accepting case in this paper.

the system G is defined as $\mathrm{Det}(G) = \max\{\mathrm{Det}(G, P_j, a) \,|\, 1 \leq j \leq r, a \in V\}$. For $L \in \mathcal{L}_{\mathrm{gen}}(\mathrm{ET0L})$,

$$\mathrm{Det}_{\mathrm{gen}}(L, \mathrm{ET0L}) = \min\{\mathrm{Det}(G) \,|\, G \text{ is a generating ET0L system, and } L_{\mathrm{gen}}(G) = L\}.$$

Analogously, if $G = (V, V', \{P_1, \ldots, P_r\}, \omega)$ is an accepting ET0L system, for $1 \leq j \leq r$ and $y \in V^*$, we define $\mathrm{Det}(G, P_j, y) = \#\{y \to a \,|\, y \to a \in P_j\}$. The degree of nondeterminism of the system G is defined as $\mathrm{Det}(G) = \max\{\mathrm{Det}(G, P_j, y) \,|\, 1 \leq j \leq r, y \in V^*\}$. For $L \in \mathcal{L}_{\mathrm{acc}}(\mathrm{ET0L})$,

$$\mathrm{Det}_{\mathrm{acc}}(L, \mathrm{ET0L}) = \min\{\mathrm{Det}(G) \,|\, G \text{ is an accepting ET0L system, and } L_{\mathrm{acc}}(G) = L\}.$$

As regards the comparison of the descriptive power, L systems are quite trivial, since any application of a finite substitution in generating mode can be simulated by an application of the inverse of a finite substitution in accepting mode and vice versa. More specifically, if P is a table of a generating ET0L system, we call $P^d = \{v \to w \,|\, w \to v \in P\}$ dual to P. If $G = (V, V', \{P_1, \ldots, P_r\}, \omega)$ is a generating ET0L system, its dual $G^d = (V, V', \{P_1^d, \ldots, P_r^d\}, \omega)$ is an accepting ET0L system with $L_{\mathrm{acc}}(G^d) = L_{\mathrm{gen}}(G)$. Similarly, one can define the dual of an accepting ET0L system with an analogous property.

Theorem 5.1 $\mathcal{L}_{\mathrm{gen}}(\mathrm{E[P][T]0L}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{E[P][T]0L})$ □

This result is equally easily seen using cc grammars: For any ET0L-table t, we introduce in the simulating cc grammar a table $\overline{t}$ consisting exactly of the productions $(v \to w, \{\#\}^* \times \mathbb{N})$ whenever $v \to w \in t$.

The notation (consistently used throughout this paper) should mean that you may omit the same letters occurring in brackets on both sides of the equation.

Obviously, our theorem allows also to carry over results on the synchronization degree from generating systems [26] to accepting ones.

In the following, we consider (syntactically) deterministic systems. It is clear that the trivial equivalence of the above theorem does not hold for these systems. In fact, we will prove inequivalence of generating and accepting mode in the following. First of all, we define some sort of normal form.

Let G be some ET0L system (be it generating or accepting). G is called *symmetrically deterministic* or ESDT0L system iff for any table P and any two productions $v_1 \to w_1, v_2 \to w_2 \in P$, $v_1 = v_2$ is equivalent to $w_1 = w_2$.

Lemma 5.2 $\mathcal{L}_{\mathrm{gen}}(\mathrm{EDT0L}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{ESDT0L})$

Proof. Since the inclusion '$\supseteq$' of the claim is trivial, the other direction '$\subseteq$' remains to be shown. Let $G = (V, V', \{P_1, \ldots, P_r\}, \omega)$ be a generating deterministic ET0L system with $V = \{a_1, \ldots, a_t\}$, $V' = \{a_{s+1}, \ldots, a_t\}$. Let $F, \Lambda_1, \ldots, \Lambda_t$ be new symbols. We define a new total alphabet $\overline{V} = V \cup \{F, \Lambda_1, \ldots, \Lambda_t\}$.

For each table P in G, define

$$\mathrm{conflict}(P) = \{a_i \to x \in P \,|\, (\exists j \neq i)(a_j \to x \in P)\} \quad \text{and}$$

$$\text{no-conflict}(P) = P \setminus \text{conflict}(P)$$

Define

$$\begin{aligned} SD(P) &= \text{no-conflict}(P) \cup \{a_i \to \Lambda_i x \,|\, a_i \to x \in \text{conflict}(P)\} \\ &\cup \; \{\Lambda_i \to F^{i+1} \,|\, a_i \in V\} \cup \{F \to F\}, \quad \text{and} \\ T_j &= \{\Lambda_j \to \lambda\} \cup \{b \to b \,|\, b \in \overline{V} \setminus \{\Lambda_j\}\}. \end{aligned}$$

Consider the symmetrically deterministic system

$$\overline{G} = (\overline{V}, V', \{SD(P_1), \ldots, SD(P_r), T_1, \ldots, T_t\}, \omega).$$

It is easily seen that $L_{\mathrm{gen}}(G) = L_{\mathrm{gen}}(\overline{G})$. □

Lemma 5.3 $\mathcal{L}_{\mathrm{acc}}(\mathrm{ET0L}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{EDT0L})$

Proof. We have to show the inclusion $\subseteq$. The proof is very much alike the previous one. Let $G = (V, V', \{P_1, \ldots, P_r\}, \omega)$ be an accepting ET0L system.

For each table P in G, define

$$\text{conflict}(P) = \{x \to y \in P \,|\, (\exists z \neq y)(x \to z \in P)\} \quad \text{and}$$

$$\text{no-conflict}(P) = P \setminus \text{conflict}(P)$$

We introduce new symbols $\Lambda_{x \to y}$ if there is a table P such that $x \to y \in \text{conflict}(P)$. We assume a suitable enumeration of these symbols, i.e. $\{\Lambda_{x \to y} \,|\, (\exists 1 \leq j \leq r)(x \to y \in \text{conflict}(P_j))\} = \{\Lambda_i \,|\, 1 \leq i \leq I\}$.

We define a new total alphabet $\overline{V} = V \cup \{F, \Lambda_1, \ldots, \Lambda_I\}$.

Define

$$\begin{aligned} DET(P) &= \text{no-conflict}(P) \cup \{\Lambda_{x \to y} x \to y \,|\, x \to y \in \text{conflict}(P)\} \\ &\cup \; \{F^{i+1} \to \Lambda_i \,|\, 1 \leq i \leq I\} \cup \{F \to F\}, \quad \text{and} \\ T_j &= \{\lambda \to \Lambda_j\} \cup \{b \to b \,|\, b \in \overline{V} \setminus \{\Lambda_j\}\}. \end{aligned}$$

Consider the deterministic system

$$\overline{G} = (\overline{V}, V', \{DET(P_1), \ldots, DET(P_r), T_1, \ldots, T_I\}, \omega).$$

It is easily seen that $L_{\mathrm{acc}}(G) = L_{\mathrm{acc}}(\overline{G})$. □

In other words, the degree of nondeterminism $\mathrm{Det}_{\mathrm{acc}}(L, \mathrm{ET0L}) = 1$ for any ET0L language L. This contrasts the situation found within generating ET0L systems, since there is an ET0L language L with $\mathrm{Det}_{\mathrm{gen}}(L, \mathrm{ET0L}) = 2$. Since the dual of say an accepting ESDT0L system is again an ESDT0L system generating the same language, we may state the following corollary.

Corollary 5.4 $\mathcal{L}_{\mathrm{acc}}(\mathrm{ESDT0L}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{ESDT0L}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{EDT0L}) \subset$
$\subset \mathcal{L}_{\mathrm{acc}}(\mathrm{EDT0L}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{ET0L}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{ET0L})$ □

We do not know what happens if we restrict our attention to propagating systems.

Chapter 6

Uniformly Limited Systems

Before turning to accepting version of uniformly limited (ul) systems, we will improve some known results in the generating case.

A *k-uniformly-limited ET0L system* (abbreviated as $k$ulET0L system) is a quintuple $G = (V, V', \{P_1, \dots, P_r\}, \omega, k)$ such that $k \in \mathbb{N}$ and $(V, V', \{P_1, \dots, P_r\}, \omega)$ is an ET0L system.[1] The notions of propagating and pure systems are inherited from L systems. Basically, the idea of one derivation step in $k$ul systems is the next: Choose one table P_i and k symbols in the string x to be derived and replace each of this selected symbols a according to some rule $a \to w \in P_i$ yielding a new string y. One seemingly technical problem remains: What should one do with strings x shorter than k? We consider two different modes:

- In Wätjen's mode, each symbol in a word shorter than k is replaced. The family of languages generable in such a manner is denoted by $\mathcal{L}_{\text{gen}}(k\text{ulET0L})$.
- In Salomaa's mode [30] or in the exact mode, words shorter than k do not yield further words except from the first derivation step, where, starting from the start symbol S (instead of the axiom ω), any word w with $S \to w \in P_i$ is obtainable. The family of languages generable in such a manner is denoted by $\mathcal{L}_{\text{gen}}(k\text{ulET0L,ex})$.

Finally, we define both for the generating and the accepting case

$$\mathcal{L}(\text{ulE[P]T0L[,ex]}) = \bigcup_{k \geq 1} \mathcal{L}(k\text{ulE[P]T0L[,ex]}).$$

Wätjen and Unruh [36] called a generating $k$ulET0L system $G = (\Sigma, \Delta, \{P_1, \dots, P_r\}, S, k)$ pseudo-synchronized iff for every $a \in \Delta$ and $w \in \Sigma^*$, if w is derivable from a after a positive number of non-parallel context-free derivation steps (considering G as a E0S system, i.e. a context-free grammar with rewriting rules for terminal symbols), then $w \notin \Delta^*$.

[1] We inherited our denotations for ET0L systems from above, hence we deviate from the usual denotations in the literature of $k$ulET0L systems [36].

We prefer the following formulation of pseudo-synchronization in the generating case: we call a generating (exact) $k$ulET0L system $G = (\Sigma, \Delta, \{P_1, \dots, P_r\}, S, k)$ with failure symbol F pseudo-synchronized iff for every $a \in \Delta$ and $w \in \Sigma^*$, if $S \overset{*}{\Rightarrow} a \Rightarrow u \Rightarrow w$ with $|w|_F = 0$ implies $|u|_{\Sigma\setminus\Delta} \geq k$.

Theorem 6.1 Let $k \in \mathbb{N}$. For every $k$ulET0L system $G = (\Sigma, \Delta, \{P_1, \dots, P_r\}, \omega, k)$ (either in Wätjen's or in Salomaa's mode), there exists an equivalent pseudo-synchronized $k$ulET0L system G' (either in Wätjen's or in Salomaa's mode) containing the same number of tables. If G is propagating, G' is propagating, too.

Proof. Define $G' = (\Sigma', \Delta, \{P'_1, \dots, P'_r\}, S, k)$ by $\Sigma' = \Sigma \cup \Delta' \cup \{S, F\}$ (disjoint union) where $\Delta' = \{a' \mid a \in \Delta\}$. By $x \mapsto x$ for $x \in \Sigma$ and $a' \mapsto a$ for $a \in \Delta$, we define a morphism $g : (\Sigma \cup \Delta')^* \to \Sigma^*$.

Let Ω denote the set of sentential forms of length $< 2lk$ (where l is the length of the longest right-hand side of a production in G) derivable in G.[2] If the Ω obtained in this way is empty, we set $\Omega = \{\omega\}$. Now let $\Omega' = \{x \mid \exists w \in \Omega(g(x) = w \wedge (w \in \Delta^* \vee |w|_{\Sigma\setminus\Delta} \geq k))\}$.

Finally, we set $P'_i = \{y \to x \mid y \in \Sigma \setminus \Delta \wedge y \to w \in P_i \wedge g(x) = w\} \cup \{a' \to x \mid a \in \Delta \wedge a \to w \in P_i \wedge g(x) = w\} \cup \{a \to F \mid a \in \Delta\} \cup \{S \to w \mid w \in \Omega'\} \cup \{F \to F\}$. □

As already said in the section on Lindenmayer systems, we consider determinism as a purely syntactical concept. This means, a $k$ulET0L system G (be it generating or accepting) is called deterministic if G is deterministic, taken as an ordinary Lindenmayer system.

The construction given in the proof of the preceding theorem unfortunately destroys determinism for two reasons.

1. If Ω' contains more than one word, there is more than one production with left-hand side S in each table.

2. Since we include all productions of the form $y \to x$ (or $a' \to x$) if there is a production of the form $y \to w$ (or $a \to w$) such that $g(x) = w$, and since g is not injective, we introduce nondeterminism.

The first item can be overcome introducing a separate start table $\text{init}_w = \{S \to w\} \cup \{y \to F | y \neq S\}$ for each $w \in \Omega'$ instead of including the start productions into the simulating tables.

The second item is solved similarly: since, for any table P_i, there is only a finite number of possible combinations

$$\{y \to x \mid y \in \Sigma \setminus \Delta \wedge y \to w \in P_i \wedge g(x) = w\} \cup \{a' \to x \mid a \in \Delta \wedge a \to w \in P_i \wedge g(x) = w\},$$

we may introduce one table $P_{i,j}$ for any such combination labelled j, $1 \leq j \leq j_i$, such that $G' = (\Sigma', \Delta, \{\text{init}_w \mid w \in \Omega'\} \cup \{P_{1,1}, \dots, P_{1,j_1}, \dots, P_{r,1}, \dots, P_{r,j_r}\}, S, k)$ is a deterministic pseudo-synchronized system simulating G if G is deterministic.

The following fact is now clear. Compare with [36, page 296f.].

[2] We do not care about effectivity issues in this place.

Corollary 6.2 For every $k \in \mathbb{N}$, we have
$\mathcal{L}_{\text{gen}}(k\text{ulE[P][D][T]0L}) = \mathcal{L}_{\text{gen}}(k\text{ulE[P][D][T]0L,ex})$ □

We want to improve [36, Theorem 4.1], where the relation

$$\mathcal{L}_{\text{gen}}(k\text{ulEPT0L}) \subseteq \mathcal{L}_{\text{gen}}(\text{P,CF}-\lambda\text{,ac})$$

was shown.

Theorem 6.3 For every $k \in \mathbb{N}$, we have

(i) $\mathcal{L}_{\text{gen}}(k\text{ulEPT0L,ex}) \subseteq \mathcal{L}_{\text{gen}}(\text{P,CF}-\lambda)$

(ii) $\mathcal{L}_{\text{gen}}(k\text{ulET0L,ex}) \subseteq \mathcal{L}_{\text{gen}}(\text{P,CF})$

Proof. Our construction is very similar to [36, Theorem 4.1], hence we give the construction without proof.

For any exact $k$ulET0L system $G = (V, V', \{P_1, \ldots, P_r\}, a_n, k)$, $k \in \mathbb{N}$, we construct an equivalent context-free programmed grammar G' such that G' is λ-free iff G is a propagating system. Let $V = \{a_1, \ldots, a_n\}$, $V' = \{a_1, \ldots, a_m\}$ with $m < n$. Let $W = \{A_1, \ldots, A_n\}$ be a new alphabet, and $g : W^* \to V^*$ be the bijective morphism given by $g(A_i) = a_i$ for $i = 1, \ldots, n$.

We define a context-free programmed grammar $G' = (V_N, V_T, P, S)$ where

$$V_N = W \cup \{A_{i,j} \mid 1 \le i \le n, 1 \le j \le r\} \cup \{S\}$$

and $V_T = V'$ with new symbols $A_{i,j}$ and S.

The following productions are contained in P:

$$(\text{init}_w : S \to g^{-1}(w), \{\text{term}_\mu \mid 1 \le \mu \le m\} \cup \{f_{\nu,\rho,1} \mid 1 \le \nu \le n, 1 \le \rho \le r\}, \emptyset)$$

where $a_n \to w \in P_i$ for some P_i;

$$(\text{term}_\mu : A_\mu \to a_\mu, \{\text{term}_1, \ldots, \text{term}_\mu\}, \emptyset)$$

for $1 \le \mu \le m$;

$$(f_{i,j,\kappa} : A_i \to A_{i,j}, \{f_{1,j,\kappa+1}, \ldots, f_{n,j,\kappa+1}\}, \emptyset)$$

for $1 \le i \le n$, $1 \le j \le r$, $1 \le \kappa < k$;

$$(f_{i,j,k} : A_i \to A_{i,j}, \{P_{\nu,j,1,v} \mid 1 \le \nu \le n, a_\nu \to v \in P_j\}, \emptyset)$$

for $1 \le i \le n$, $1 \le j \le r$;

$$(P_{i,j,\kappa,w} : A_{i,j} \to g^{-1}(w), \{P_{\nu,j,\kappa+1,v} \mid 1 \le \nu \le n, a_\nu \to v \in P_j\}, \emptyset)$$

for $1 \leq i \leq n$, $1 \leq j \leq r$, $1 \leq \kappa < k$, $a_i \to w \in P_j$;

$$(P_{i,j,k,w} : A_{i,j} \to g^{-1}(w), \{\mathrm{term}_\mu \,|\, 1 \leq \mu \leq m\} \cup \{f_{\nu,\rho,1} \,|\, 1 \leq \nu \leq n, 1 \leq \rho \leq r\}, \emptyset)$$

for $1 \leq i \leq n$, $1 \leq j \leq r$, $a_i \to w \in P_j$.

The crucial thing is that the k parallel replacements of G are simulated sequentially by $f_{i,j,\cdot}$ and $P_{i,j,\cdot,w}$, where the intermediate words due to that sequentialization cannot contribute to the language. □

From [18], we get as a corollary:

Corollary 6.4 For every $k \in \mathbb{N}$, we have

(i) $\mathcal{L}_{\mathrm{gen}}(k\mathrm{ulEPT0L}) = \mathcal{L}_{\mathrm{gen}}(k\mathrm{ulEPT0L,ex}) \subset \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF}-\lambda,\mathrm{ac}) \subset \mathcal{L}_{\mathrm{gen}}(\mathrm{CS})$

(ii) $\mathcal{L}_{\mathrm{gen}}(k\mathrm{ulET0L}) = \mathcal{L}_{\mathrm{gen}}(k\mathrm{ulET0L,ex}) \subset \mathcal{L}(\mathrm{REC}) \subset \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ac}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{RE})$

Proof. We only have to show the second case. By [18], we know that for $\mathcal{L}_{\mathrm{gen}}(\mathrm{M,CF}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF})$, the membership problem is decidable. Since every recursively enumerable language is the morphic image of a recursive language, we obtain by [18, Theorem 3], applied to the language class $\mathcal{L}_{\mathrm{gen}}(\mathrm{M,CF})$, the strictness of the inclusion $\mathcal{L}_{\mathrm{gen}}(\mathrm{M,CF}) \subset \mathcal{L}(\mathrm{REC})$. □

Let us turn to the discussion of accepting versus generating mode. First, we consider the exact mode.

Since the exact modes differs between the first and all other derivation steps, we may have a discussion "which should be the appropriate definition for the accepting mode?". To circumvent this, we take the next definition of generating/accepting exact $k$ulET0L systems which is obviously equivalent to the one considered up to now.

An exact $k$ulET0L system is a quintuple $G = (\Sigma, \Delta, \{P_1, \ldots, P_r\}, \{\omega_1, \ldots, \omega_s\}, k)$, where Σ is the total alphabet, $\Delta \subseteq \Sigma$ is the terminal alphabet, each P_j is a table as in ET0L systems, and $\{\omega_1, \ldots, \omega_s\} \subset \Sigma^+$ is a finite set of axioms. The yield relation $\Rightarrow$ is defined as follows: $x \Rightarrow y$ iff there is a table P_j and productions $a_\kappa \to w_\kappa \in P_j$ for each $1 \leq \kappa \leq k$ and words $x_0, \ldots, x_k$ such that $x = x_0 a_1 x_1 \cdots a_k x_k$ and $y = x_0 w_1 x_1 \cdots w_k x_k$. The language generated by G is $L_{\mathrm{gen}}(G) = \{w \in \Delta^* \,|\, (\exists \sigma \in \{1, \ldots, s\})(\omega_\sigma \overset{*}{\Rightarrow} w)\}$. Likewise, accepting exact $k$ulET0L systems are defined.

Theorem 6.5 For every $k \in \mathbb{N}$, we have
$\mathcal{L}_{\mathrm{gen}}(k\mathrm{ul[E][P][T]0L,ex}) = \mathcal{L}_{\mathrm{acc}}(k\mathrm{ul[E][P][T]0L,ex})$

Proof. Nearly, the concept of duality introduced for L systems above suffices to show the theorem. More precisely, if $G = (\Sigma, \Delta, \{P_1, \ldots, P_r\}, \{\omega_1, \ldots, \omega_s\}, k)$ is an exact generating $k$ulET0L system, its dual can be described as

$$G^d = (\Sigma, \Delta, \{P_1^d, \ldots, P_r^d\}, \{\omega_1, \ldots, \omega_s\}, k),$$

where, as general, $P^d = \{v \to w \mid w \to v \in P\}$. Then, $L_{\text{gen}}(G) = L_{\text{acc}}(G^d)$.

Namely, assume that $x \Rightarrow y$ in the generating system G. Then, $|x| \geq k$. Since some table P_i must have been applied successfully deriving y out of x, y contains (at least) k non-overlapping subwords which are right-hand sides of productions of P_i. Therefore, within the accepting system G^d, we find $y \Rightarrow x$, using table P_i^d.

Now, consider some direct derivation $y \Rightarrow x$ in the accepting dual system G^d. By definition, there is a table P_j^d and there are (at least) k non-overlapping subwords in y which are left-hand sides of productions of P_j^d. Hence, $|x| \geq k$, and employing table P_j, we find $x \Rightarrow y$ within the original system G.

On the other hand, if $G = (\Sigma, \Delta, \{P_1, \ldots, P_r\}, \{\omega_1, \ldots, \omega_s\}, k)$ is an exact accepting $k$ulET0L system, its dual may be described as

$$G^d = (\Sigma, \Delta, \{P_1^d, \ldots, P_r^d\}, \{\omega_1, \ldots, \omega_s\}, k)$$

and $L_{\text{acc}}(G) = L_{\text{gen}}(G^d)$. □

An alternative proof uses our metatheorem: Each production $v \to w$ in a table P_i of an exact $k$ul-system corresponds to a production $(v \to w, (V_G^*\{\#\})^k V_G^* \times \{1, \ldots, k\})$ in a simulating table P_i' of the corresponding cc grammar.

Note that this theorem allows also to carry over results on the synchronization degree from generating systems to accepting ones.

As regards Wätjen's mode, it depends on the precise definition of generating systems how to define accepting ul systems, since the original definition "replace exactly $\min\{k, |x|\}$ symbols of the word x to be rewritten" is not an appropriate one to be interpreted in accepting mode.

We discuss a few possible interpretations of the derivation $x \Rightarrow y$ in the following.

1. Select k symbols of the word x and replace them. If this is not possible, replace $k-1$ symbols of x. If this is not possible, replace $k-2$ symbols of x ...

2. Select k symbols of the word x and replace them. If this is not possible, interpret the tables as finite substitutions σ_j as in L systems and define $x \Rightarrow y$ iff $y \in \sigma_j(x)$ for some table σ_j.

3. Let x be the string to be rewritten. Denote by $L_k(x)$ the set of strings derived directly from x via G, where G is interpreted as an exact $k$ulET0L system. If $L_k(x) \neq \emptyset$, then $x \Rightarrow y$ iff $y \in L_k(x)$. If $L_k(x) = \emptyset$, then $x \Rightarrow y$ iff there is a partition $x = x_1 \cdots x_l$ with $l < k$ and there is a table in G containing productions $x_1 \to y_1$, ..., $x_l \to y_l$ such that $y = y_1 \cdots y_l$. Note that this definition is equally valid for generating and accepting mode. In generating mode, it coincides with Wätjen's definition.

The first reformulation does not lead to a unique interpretation in accepting mode. Taking as rules $aa \to z$ and $aaa \to y$ and $k = 3$, is $a^6 \Rightarrow azy$ allowed or not? If we select the subwords as indicated, especially applying $aaa \to y$, we do not find 3 subwords as required. But selecting other subwords, this is possible, namely in the derivation $a^6 \Rightarrow zzz$.

But also the second reformulation causes problems in accepting mode. Taking our example again, we find $a^6 \Rightarrow yy$ because of the interpretation via inverse substitutions as in L systems. Is this our intention? Maybe. But taking additionally a rule of the form $a \to b$, we may fail applying $aaa \to y$ as before, but then the L interpretation would yield $a^6 \Rightarrow b^6$, actually replacing 6 symbols.

Unfortunately, even the third definition does not lead to a situation where the generating and accepting mode are dual as in lrcc grammars. For example, in the generating system dual to the above example, $yy \Rightarrow a^6$, but $yy \notin L_3(a^6)$, hence $a^6 \Rightarrow yy$ is not valid in accepting mode.

Of course, one could try to consider the above variants separately as language description mechanisms and discuss them in detail. This is beyond the scope of this paper. We only mention that the third way of defining accepting ulET0L systems is at least as powerful as their generating counterparts, but we do not know whether or when the converse assertion holds.

Instead, and because of these difficulties, we define the yield relation given by a $k$ulET0L system G as follows:

Let x be the string to be rewritten. Again, denote by $L_k(x)$ the set of strings derived directly from x via G, where G is interpreted as an exact $k$ulET0L system. Now define $x \Rightarrow y$ iff $y \in L_k(x)$ or there is a partition $x = x_1 \cdots x_l$ with $l \leq k$ and there is a table in G containing productions $x_1 \to y_1, \ldots, x_l \to y_l$ such that $y = y_1 \cdots y_l$. Note that this definition is equally valid for generating and accepting mode. In generating mode, it coincides with Wätjen's definition.

Using this definition and the usual concept of dual systems (again, the employment of our metatheorem is possible), we find:

Theorem 6.6 For every $k \in \mathbb{N}$, we have $\mathcal{L}_{\text{gen}}(k\text{ul[E][P][T]0L}) = \mathcal{L}_{\text{acc}}(k\text{ul[E][P][T]0L})$ □

By our above considerations, we see that Wätjen's mode and exact mode do coincide within systems with extensions regarding their descriptive power when considering accepting systems, too.

Finally, we turn to deterministic ulET0L systems. Defining symmetrically deterministic systems as above, we can use nearly the same constructions as within Lindenmayer systems. Nevertheless, the results obtained for uniformly limited systems differ remarkably from our previous results. Since the dual of say an accepting ulESDT0L system is again a ulESDT0L system generating the same language, we now show that for any generating (accepting) ulEDT0L system, there is an equivalent generating (accepting) ulESDT0L system in order to prove the desired relation $\mathcal{L}_{\text{gen}}(k\text{ulEDT0L}) = \mathcal{L}_{\text{acc}}(k\text{ulEDT0L})$.

Lemma 6.7 For every $k \in \mathbb{N}$, we have
$\mathcal{L}_{\text{gen}}(k\text{ulEDT0L}) = \mathcal{L}_{\text{gen}}(k\text{ulESDT0L})$ and
$\mathcal{L}_{\text{gen}}(k\text{ulEDT0L,ex}) = \mathcal{L}_{\text{gen}}(k\text{ulESDT0L,ex})$ □

Instead of a proof, we only remark that — in comparison with Lemma 5.2 — in the exact mode within $k$ulET0L systems, one should introduce $a_i \to \Lambda_i^k x$ in table $SD(P)$ instead of $a_i \to \Lambda_i x$ in order to guarantee the termination of the simulation.

Lemma 6.8 For every $k \in \mathbb{N}$, we have $\mathcal{L}_{\text{acc}}(k\text{ulE[P]DT0L}) = \mathcal{L}_{\text{acc}}(k\text{ulE[P]SDT0L})$

Proof. Since the inclusion '$\supseteq$' of the claim is trivial, the other direction '$\subseteq$' remains to be shown. This proof is very similar to the one showing Lemma 5.2, but is also valid in the propagating case. Let $G = (V, V', \{P_1, \ldots, P_r\}, \omega)$ be an accepting deterministic k-uniformly-limited ET0L system with $V = \{a_1, \ldots, a_t\}$, $V' = \{a_{s+1}, \ldots, a_t\}$. For each table P in G, define

$$\text{conflict}(P) = \{x \to a_i \in P \,|\, (\exists y \neq x)(y \to a_i \in P)\} \quad \text{and}$$

$$\text{no-conflict}(P) = P \setminus \text{conflict}(P)$$

Let L be the maximal number of conflicts in G, i.e. $L(i,\rho) = \#\{x \to a_i \in P_\rho\}$, and $L = \max\{\max\{L(i,\rho) \,|\, 1 \leq i \leq t\} \,|\, 1 \leq \rho \leq r\}$. Introduce an indexing scheme in the set of conflicting productions, e.g., $\{x \to a_i \in P_\rho\} = \{x_{i,1} \to a_i, \ldots, x_{i,L(i,\rho)} \to a_i\}$. Let

$$F, b_{1,1}, \ldots, b_{1,L}, b_{2,1}, \ldots, b_{2,L}, \ldots, b_{t,1}, \ldots, b_{t,L}, F_{1,1}, \ldots, F_{1,L}, F_{2,1}, \ldots, F_{2,L}, \ldots, F_{t,1}, \ldots, F_{t,L}$$

be new symbols. Let $\overline{V}$ contain all these symbols, together with the symbols of V. Define

$$\begin{array}{rcl}
SD(P_\rho) & = & \text{no-conflict}(P_\rho) \cup \{x_{i,j} \to b_{i,j} \,|\, x_{i,j} \to a_i \in \text{conflict}(P_\rho), 1 \leq j \leq L(i,\rho)\} \\
 & \cup & \{F^{i+1} \to a_i \,|\, a_i \in V\} \\
 & \cup & \{F^{t+1+i} \to b_{i,L(i,\rho)+1}, F^{2t+1+i} \to b_{i,L(i,\rho)+2}, \ldots, F^{(L-L(i,\rho))t+1+i} \to b_{i,L} \,|\, a_i \in V\} \\
 & \cup & \{b_{i,j} \to F_{i,j} \,|\, 1 \leq i \leq t, 1 \leq j \leq L\} \cup \{F \to F\}, \quad \text{and} \\
T & = & \{b_{i,1} \to a_i, b_{i,2} \to b_{i,1}, \ldots, b_{i,L} \to b_{i,L-1} \,|\, 1 \leq i \leq t\} \cup \{F \to F\} \\
 & \cup & \{F_{i,j} \to F_{i,j} \,|\, 1 \leq i \leq t, 1 \leq j \leq L\} \cup \{F^{i+1} \to b_{i,L} \,|\, 1 \leq i \leq t\}.
\end{array}$$

Consider the symmetrically deterministic system

$$\overline{G} = (\overline{V}, V', \{SD(P_1), \ldots, SD(P_r), T\}, \omega).$$

It is easily seen that $L_{\text{acc}}(G) = L_{\text{acc}}(\overline{G})$. One application of say table P_ρ in G is simulated by one application of $SD(P_\rho)$ in $\overline{G}$, followed by at most L applications of T. □

In the preceding proof, observe that even if, in the derivation $v \Rightarrow w$ according to P_ρ, actually k subwords have been replaced, this is not the case any more for the applications of T in the simulating grammar, since symbols a_i — once introduced via T — do not change by subsequent applications of T. Hence, the above argument is only valid in Wätjen's mode.

By Corollary 6.2, we obtain:

Theorem 6.9 For every $k \in \mathbb{N}$, we have $\mathcal{L}_{\text{acc}}(k\text{ulEDT0L,ex}) = \mathcal{L}_{\text{gen}}(k\text{ulEDT0L,ex}) =$
$= \mathcal{L}_{\text{gen}}(k\text{ulEDT0L}) = \mathcal{L}_{\text{acc}}(k\text{ulEDT0L})$ □

Since Lemma 6.7 is only valid for the non-propagating case, we obtain:

Corollary 6.10 For every $k \in \mathbb{N}$, we have $\mathcal{L}_{\text{gen}}(k\text{ulEPDT0L}) \supseteq \mathcal{L}_{\text{acc}}(k\text{ulEPDT0L})$ □

We do not know whether the inclusion in the last corollary is strict.

Analyzing the proof of Lemma 5.3, we see that an analogue holds for uniformly limited systems. Note that in Wätjen's mode, we must add a failure symbol F' and, in $DET(P)$, the following productions $\{\Lambda_i \to F' \,|\, 1 \leq i \leq I\} \cup \{F' \to F'\} \cup \{x \to F' \,|\, (\exists y)(x \to y \in \text{conflict}(P))\}$. Otherwise, there might be for example 'shortcuts' in the simulating derivation process.

Lemma 6.11 For every $k \in \mathbb{N}$, we have
$\mathcal{L}_{\text{acc}}(k\text{ulET0L}) = \mathcal{L}_{\text{acc}}(k\text{ulEDT0L}) = \mathcal{L}_{\text{acc}}(k\text{ulEDT0L,ex})$ □

Corollary 6.12 For any $k$ulET0L language L,
$\text{Det}_{\text{acc}}(L, k\text{ulET0L}) = \text{Det}_{\text{gen}}(L, k\text{ulET0L}) = 1$ □

Chapter 7

Uniformly Limited ET0L Systems With Unique Interpretation

7.1 Introduction

The content of this section is accepted for publication [10]. We repeat some definitions already introduced in the preceding section for reasons of some subtle formal differences.

$k$ulET0L systems have been introduced by Wätjen and Unruh [36] in order to further the study of the influence of restricted forms of parallelism in the derivation process of systems (grammars) defining formal languages. As in klET0L systems [32, 4], they found interesting relations with programmed grammars which are another form of varying the idea of context-free parallel generation (acceptance) of words.

Unfortunately, Wätjen and Unruh did only show that $k$ulE(P)T0L systems are at most as powerful as programmed grammars (without erasing productions) with appearance checking, leaving the other possible inclusion open. In the previous section, we showed that $k$ulE(P)T0L systems are at most as powerful as programmed grammars (without erasing productions) without appearance checking, hence proving that the inclusion shown by Wätjen and Unruh is strict indeed (by [18, 9]). This leaves us with the natural question whether $k$ulE(P)T0L systems are at least as powerful as programmed grammars (without erasing productions) without appearance checking. Up to now, we were not able to solve this problem.

In this section, we introduce another class of limited parallel systems which look quite similar to $k$ulET0L systems in their definition. For this modification, we show that the above stated and other problems open for $k$ulET0L systems can be settled. We hope that this variation sheds some light on the still open problems for $k$ulET0L systems and supports the interrelation of restricted parallel and regulated rewriting.

For the convenience of the reader, below we repeat the definition of $k$ulET0L systems and unordered scattered context grammars without appearance checking.

$G = (\Sigma, \Delta, H, \omega, k)$ is a *k-uniformly-limited ET0L system* (abbreviated as $k$ulET0L system) if $k \in \mathbb{N}$ and $(\Sigma, \Delta, H, \omega)$ is (formally) an ET0L system with alphabet Σ, terminal alphabet $\Delta \subseteq \Sigma$, finite set of tables H (where a table is a finite substitution on Σ), and axiom $\omega \in \Sigma^+$. In a derivation of a $k$ulET0L system, at each step of the rewriting process, exactly $\min\{k, |w|\}$ symbols of the word w considered have to be rewritten, where $|w|$ is the length of w. As in ET0L systems, a system is called propagating if no table contains an erasing production $a \to \lambda$. We abbreviate such systems by $k$ulEPT0L.

An interesting variant of the above definition was introduced by K. Salomaa in [30] under the name of "k-context-free grammars" in the case of systems with just one table. In our setting, this variant reads as follows: formally, an *exact k-uniformly-limited ET0L system* $G = (\Sigma, \Delta, H, \Omega, k)$ is defined as a $k$ulET0L system (except Ω which is now a finite set of axioms $\omega_i \in \Sigma^+$), but in each derivation step of an exact $k$ulET0L system, exactly k occurrences of left-hand sides of productions of some table selected before are replaced by the corresponding right-hand sides within the string w to be rewritten.

As usual, we write $w \Rightarrow v$ if there is a one-step-derivation in some sense[1]. The reflexive transitive closure of $\Rightarrow$ is written $\overset{*}{\Rightarrow}$. If $G = (\Sigma, \Delta, H, \omega, k)$ is a generating $k$ulET0L system (with an underlying generating ET0L system $(\Sigma, \Delta, H, \omega)$), then $L_{\text{gen}}(G) = \{w \in \Delta^* \mid \omega \overset{*}{\Rightarrow} w\}$, and the family of languages generated in such a way is denoted by $\mathcal{L}_{\text{gen}}(k\text{ulET0L})$. Similarly, for exact systems, we write $L_{\text{gen}}(G) = \{w \in \Delta^* \mid \exists \omega \in \Omega(\omega \overset{*}{\Rightarrow} w)\}$ and $\mathcal{L}_{\text{gen}}(k\text{ulET0L,ex})$. For exact systems, it is quite natural to consider them as language acceptors as well, supposing that the underlying ET0L system is accepting, too. In this case we define $L_{\text{acc}}(G) = \{w \in \Delta^* \mid \exists \omega \in \Omega(w \overset{*}{\Rightarrow} \omega)\}$ and write $\mathcal{L}_{\text{acc}}(k\text{ulET0L,ex})$. If the underlying ET0L system is propagating, we replace 'ET0L' by 'EPT0L' in our notation.

Syntactically, our modification looks like the just defined systems: $G = (\Sigma, \Delta, H, \Omega, k)$ is a *k-unique uniformly-limited ET0L system* (abbreviated as ku^{2}lET0L system) if G can be interpreted as an exact $k$ulET0L system.[2] In each derivation step of G, exactly k pairwise different productions of a chosen table are applied to the string to be rewritten. This means that exactly k occurrences of left-hand sides of pairwise different productions are replaced by the corresponding right-hand sides according to the selected productions. The generated/accepted language of a given ku^{2}lET0L system is defined as above, leading us to the language classes $\mathcal{L}_{\text{gen}}(k\text{u}^2\text{lET0L})$ and $\mathcal{L}_{\text{acc}}(k\text{u}^2\text{lET0L})$ and their propagating counterparts.

Observe that, due to our modified definition, any table within a ku^{2}lET0L system which contains less than k productions is never applicable.

Instead of proving the equivalence of programmed grammars and ku^{2}lET0L systems directly, we use unordered scattered context grammars instead, because they are also rewriting systems with restricted parallelism in a certain sense known to be equivalent to programmed grammars without appearance checking [5, Lemma 2.4.5 and Lemma 2.4.6] with or without erasing productions.

An *unordered scattered context grammar* is a quadruple $G = (\Sigma, \Delta, P, S)$, where Σ and Δ denote the total alphabet and the terminal alphabet, respectively, and $S \in \Sigma \setminus \Delta$

[1] It will be clear from the context which mechansism we choose.

[2] It is obviously possible to introduce this modification in the non-exact case, too, but this is not needed here.

is a start/goal symbol. P is a finite set of finite sequences of context-free productions, $P = \{p_1, \ldots, p_n\}$, written as

$$p_i : (v_{i,1}, v_{i,2}, \ldots, v_{i,r_i}) \rightarrow (w_{i,1}, w_{i,2}, \ldots, w_{i,r_i}), 1 \leq i \leq n.$$

The application of such a rule p_i to some $x \in \Sigma^+$ yields $y \in \Sigma^*$ (written as $x \Rightarrow y$), if there is a permutation $\pi : \{1, \ldots, r_i\} \rightarrow \{1, \ldots, r_i\}$ such that

$$x = x_1 v_{\pi(1)} x_2 v_{\pi(2)} \cdots x_{r_i} v_{\pi(r_i)} x_{r_i+1}, \quad \text{and}$$

$$y = x_1 w_{\pi(1)} x_2 w_{\pi(2)} \cdots x_{r_i} w_{\pi(r_i)} x_{r_i+1}.$$

Note that we have covered both the generating and the accepting case, since these notions are inherited via the underlying context-free productions. Denoting the reflexive transitive closure of $\Rightarrow$ by $\overset{*}{\Rightarrow}$, we define in the generating (accepting) case $L_{\text{gen}}(G) = \{w \in \Delta^* \mid S \overset{*}{\Rightarrow} w\}$ ($L_{\text{acc}}(G) = \{w \in \Delta^* \mid w \overset{*}{\Rightarrow} S\}$) and denote the language families by $\mathcal{L}_{\text{gen}}(\text{uSC})$ ($\mathcal{L}_{\text{acc}}(\text{uSC})$). Following [5], we attach $-\lambda$ to these denotations if we want to restrict ourselves to nonerasing productions. A uSC-grammar $G = (\Sigma, \Delta, P, S)$ is in 2-normal form if, for any $p_i \in P$, $r_i \leq 2$, where r_i denotes the length of the sequence of productions p_i as above.

Similarly, a *scattered context grammar* is a quadruple $G = (\Sigma, \Delta, P, S)$, where Σ and Δ denote the total alphabet and the terminal alphabet, respectively, and $S \in \Sigma \setminus \Delta$ is a start/goal symbol.[3] P is a finite set of finite sequences of context-free productions, $P = \{p_1, \ldots, p_n\}$, written as

$$p_i : (v_{i,1}, v_{i,2}, \ldots, v_{i,r_i}) \rightarrow (w_{i,1}, w_{i,2}, \ldots, w_{i,r_i}), 1 \leq i \leq n.$$

The application of such a rule p_i to some $x \in \Sigma^+$ yields $y \in \Sigma^*$ (written as $x \Rightarrow y$), if

$$x = x_1 v_1 x_2 v_2 \cdots x_{r_i} v_{r_i} x_{r_i+1}, \quad \text{and}$$

$$y = x_1 w_1 x_2 w_2 \cdots x_{r_i} w_{r_i} x_{r_i+1}.$$

Note that we have covered both the generating and the accepting case, since these notions are inherited via the underlying context-free productions. Denoting the reflexive transitive closure of $\Rightarrow$ by $\overset{*}{\Rightarrow}$, we define in the generating (accepting) case $L_{\text{gen}}(G) = \{w \in \Delta^* \mid S \overset{*}{\Rightarrow} w\}$ ($L_{\text{acc}}(G) = \{w \in \Delta^* \mid w \overset{*}{\Rightarrow} S\}$) and denote the language families by $\mathcal{L}_{\text{gen}}(\text{SC})$ ($\mathcal{L}_{\text{acc}}(\text{SC})$). Again, we attach $-\lambda$ to these denotations if we want to restrict ourselves to nonerasing productions.

7.2 Results

We first want to state a normal form result for unordered scattered context grammars which immediately follows from the proof of Lemma 2.4.6 in [5]. In the following lemmas 7.1,7.2,7.3, we exploit the fact that the proofs of Lemma 2.4.5 and Lemma 2.4.6 in [5] work for the accepting case as well.

[3] The general case of scattered context grammars is not considered in the paper [10].

Lemma 7.1 There is a Turing machine which, given a uSC-grammar G', constructs an equivalent uSC-grammar G in 2-normal form. If G' is non-erasing, G inherits this property. □

A further analysis of the proof of Lemma 2.4.6 shows that, in addition, one can restrict one's attention to uSC-grammars $G = (\Sigma, \Delta, P, S)$ in 2-normal form with the following separation property:

There is a partition $\Sigma_1 \cup \Sigma_2 = \Sigma$, $\Sigma_1 \cap \Sigma_2 = \emptyset$ of the total alphabet Σ of G such that, if $p_i : (v_1, v_2) \to (w_1, w_2)$ is an arbitrary sequence of two context-free productions in P, then $v_1, w_1 \in \Sigma_1^*$ and $v_2, w_2 \in \Sigma_2^*$.

With the help of this observation, we can show the following normal form result.

Lemma 7.2 There is a Turing machine which, given a uSC-grammar G, constructs an equivalent uSC-grammar $\hat{G} = (\hat{\Sigma}, \Delta, \hat{P}, \hat{S})$ such that

1. $\hat{G}$ in 2-normal form.
2. $\hat{G}$ has separation property.
3. None of the context-free productions in any of the sequences $p \in P$ has the form $A \to A$.

If G is non-erasing, $\hat{G}$ inherits this property.

Proof. Let $G = (\Sigma, \Delta, P, S)$ be given by the preceding lemma. If P does not contain a sequence p containing a production $A \to A$, we are done. Sequences of the form $(A) \to (A)$ can simply be left out. Otherwise, we introduce a new symbol A', replace the disturbing $A \to A$ by $A \to A'$, and introduce for every sequence $q \in P, q \neq p$ containing A a new sequence q', where each occurrence of A is replaced by A'. In this way, we obtain a grammar G' equivalent to G which satisfies (1) and (2) and in which the number of 'irking symbols' has been reduced by 1. Since Σ is finite, repeating this algorithm, we finally arrive at a grammar $\hat{G}$ with the desired properties. □

From the effective equivalences $\mathcal{L}_{\text{gen}}(\text{P,CF}) = \mathcal{L}_{\text{acc}}(\text{P,CF})$ and $\mathcal{L}_{\text{gen}}(\text{P,CF} - \lambda) = \mathcal{L}_{\text{acc}}(\text{P,CF} - \lambda)$, we immediately get:

Lemma 7.3 There is a Turing machine which, given an accepting (generating) uSC-grammar G', constructs an equivalent generating (accepting) uSC-grammar G. If G' is non-erasing, G inherits this property. □

Note that the last lemma can also be proved directly, using independence arguments developped in the case of so-called lrcc grammars [3] and L systems [11].

Now, we compare ku^{2}lE(P)T0L systems with unordered scattered context grammars in regard to their descriptional power.

Lemma 7.4 There is a Turing machine which, given an accepting (generating) $k\text{u}^2$lET0L system G', produces an equivalent accepting (generating) uSC-grammar G. If G' is propagating, then G is non-erasing.

Proof. Let $G' = \{\Sigma', \Delta, H, \Omega, k\}$ be a $k\text{u}^2$lET0L system. For any table $h \in H$, there are $(h, k) := \left(\begin{array}{c} \#h \\ k \end{array} \right)$ possible selections of productions in h which can be applied in parallel during one derivation step of G'. Choose (h, k) sequences $p_{h,i}$, $1 \leq i \leq (h, k)$ of productions of length k from h such that any of the (h, k) possible selections is uniquely identified. We write such a sequence as uSC-productions:

$$p_{h,i} : (v_{h,i,1}, v_{h,i,2}, \ldots, v_{h,i,k}) \to (w_{h,i,1}, w_{h,i,2}, \ldots, w_{h,i,k})$$

denotes such a possible selection $\{v_{h,i,j} \to w_{h,i,j} \,|\, 1 \leq j \leq k\} \subseteq h$.

Let S be a new symbol. In the accepting (generating) case, we define goal (start) productions $r_\omega : (\omega \to S)$ $(r_\omega : (S \to \omega))$. Now, the simulating uSC-grammar is defined as $G = (\Sigma' \cup \{S\}, \Delta, \{p_{h,i} \,|\, h \in H, 1 \leq i \leq (h, k)\} \cup \{r_\omega \,|\, \omega \in \Omega\}, S)$. □

First of all, let us mention two consequences of [18, 9] and [5, 31] due to the effective equivalence of uSC-grammars and matrix grammars without appearance checking [5]:

Lemma 7.5 There is a Turing machine which, given an accepting (generating) uSC-grammar G and some word w, decides whether $w \in L_{\text{acc}}(G)$ $(w \in L_{\text{gen}}(G))$. □

Lemma 7.6 There is a Turing machine which, given an accepting (generating) uSC-grammar G, decides whether there exists some word w with $w \in L_{\text{acc}}(G)$ $(w \in L_{\text{gen}}(G))$.□

Lemma 7.7 There is a Turing machine TM which, given an accepting (generating) uSC-grammar G' and a natural number $k \geq 2$, produces an equivalent $k\text{u}^2$lET0L system G. If G' is non-erasing, then G is propagating.

Proof. In order to avoid clumsy formulations, we restrict ourselves to the generating case. W.l.o.g, we can assume that the language described by $G' = (\Sigma', \Delta, P, S)$ is not empty. Furthermore, assume G' to be in the normal form of Lemma 7.2. First, by the decidability of the word problem for uSC-grammars, TM can collect all words (sentential forms) over Σ' of length $\leq \max\{2 \cdot k, 2 \cdot l\}$ (where l denotes the length of the longest right-hand-side of context-free productions contained in G') described by G' into a set Ω. Hence, Ω contains all possible 'starting points' of simulations of the simulating $k\text{u}^2$lET0L system $G = (\Sigma, \Delta, H, \Omega, k)$ which TM wants to construct. Let $\Sigma = \Sigma' \cup \{F\}$, where F is a special failure symbol.

The set of production strings P can be split into $P_1 = \{(A) \to (w) \,|\, A \in \Sigma \setminus \Delta, w \in \Sigma^*\} \cap P$ and $P_2 = P \setminus P_1$. For every production string of the form $(A) \to (w) \in P_1$, we introduce a table $\{A \to w\} \cup \{a \to a \,|\, a \in \Sigma\}$, yielding a set of tables H_1. For every production string of the form $(A_1, A_2) \to (w_1, w_2) \in P_2$ and every $(k-2)$-element subset $\hat{\Sigma}$ of Σ',[4] we introduce a table $\{A_1 \to w_1, A_2 \to w_2\} \cup \{A \to F \,|\, A \in \Sigma \setminus \hat{\Sigma}\} \cup \{a \to a \,|$

[4] Here, we need $k \geq 2$.

$a \in \hat{\Sigma}\}$. Hence, from P_2 we get a set of tables H_2. Let $H = H_1 \cup H_2$. It is obvious how a derivation step in G' is simulated by G. Note that the normal form assumption is needed in order to show that any 'successful' derivation in G' can be simulated by G. □

We clarify our construction with the help of the following example. Consider the uSC-grammar $G' = (\Sigma', \{a,b,c\}, \{(S) \to (AX), (A,X) \to (Aa, bXc), (A,X) \to (a, bc)\})$ with $\Sigma' = \{A, X, S, a, b, c\}$. Let $\Sigma = \{A, X, S, F, a, b, c\}$. Obviously, $L_{\text{gen}}(G') = \{a^n b^n c^n \mid n \in \mathbb{N}\}$. Let $k = 3$. We have, using the abbreviations from the above proof, $\Omega = \{S, AX, AabXc, abc, aabbcc\}$, $H_1 = \{\{S \to AX\} \cup \{Y \to Y \mid Y \in \Sigma\}\}$, and $H_2 = \{\{A \to Aa, X \to bXc, Y \to Y\} \cup \{Z \to F \mid Z \in \Sigma \setminus \{Y\}\}, \{A \to a, X \to bc, Y \to Y\} \cup \{Z \to F \mid Z \in \Sigma \setminus \{Y\}\} \mid Y \in \Sigma'\}$. Now, we have $G = (\Sigma, \{a,b,c\}, H_1 \cup H_2, \Omega, 3)$. A possible derivation is $AabXc \Rightarrow AaabbXcc \Rightarrow aaabbbccc$, where in the first derivation step we use one of the tables $\{A \to Aa, X \to bXc, Y \to Y\} \cup \{Z \to F \mid Z \in \Sigma \setminus \{Y\}\}$ and in the second one we use one of the tables $\{A \to a, X \to bc, Y \to Y\} \cup \{Z \to F \mid Z \in \Sigma \setminus \{Y\}\}$ (with $Y \in \{a,b,c\}$ in both cases). Note that because of the particular structure of sentential forms derivable in G', most of the tables in $H_1 \cup H_2$ are never applicable, and in addition, some of them are superfluous and harmless.

Combining our lemmas with other known results proved above or contained in [5], we obtain:

Theorem 7.8 For each $k \geq 2$:
$\mathcal{L}_{\text{gen}}(2\text{u}^2\text{lET0L}) = \mathcal{L}_{\text{acc}}(2\text{u}^2\text{lET0L}) = \mathcal{L}_{\text{gen}}(k\text{u}^2\text{lET0L}) = \mathcal{L}_{\text{acc}}(k\text{u}^2\text{lET0L}) =$
$\mathcal{L}_{\text{gen}}(\text{uSC}) = \mathcal{L}_{\text{acc}}(\text{uSC}) = \mathcal{L}_{\text{gen}}(\text{P,CF}) = \mathcal{L}_{\text{acc}}(\text{P,CF})$.
All these language families properly include the family of context-free languages (which in turn equals $\mathcal{L}_{\text{gen}}(1\text{u}^2\text{lET0L}) = \mathcal{L}_{\text{acc}}(1\text{u}^2\text{lET0L})$), even more, they include $\mathcal{L}_{\text{gen}}(k\text{ulET0L})$.

The class of recursive languages properly contains all these language families.

There is a Turing machine which, given an arbitrary generating (accepting) $k'\text{u}^2\text{lET0L}$ system G with $k' \geq 1$, decides whether $L_{\text{gen}}(G)$ ($L_{\text{acc}}(G)$) is empty or not.

Moreover, each $k\text{u}^2\text{lETOL}$-language can be described by a $2\text{u}^2\text{lET0L}$-system with a special failure symbol F satisfying the following normal form for each table h

- $\#\{u \to v \in h \mid u \neq v \wedge (u \neq F \vee v \neq F)\} \leq 2$
- $u \to v \in h$ implies $|u|, |v| \leq 2$
- If $u_1 \to v_1$ and $u_2 \to v_2$ are two distinct productions in h such that $u_i \neq v_i$ $(i = 1, 2)$ and $F \notin \{u_1, v_1, u_2, v_2\}$, then neither $u_1 = u_2$ nor $v_1 = v_2$. □

The second requirement of the normal form can be readily established, the other two follow directly from the uSC-normal forms used in our above constructions. Note that this normal form resembles the ones encountered in regulated rewriting (cf. [5, p. 49]) and limited ET0L systems [19].

Theorem 7.9 For each $k \geq 2$:
$\mathcal{L}_{\text{gen}}(2\text{u}^2\text{lEPT0L}) = \mathcal{L}_{\text{acc}}(2\text{u}^2\text{lEPT0L}) = \mathcal{L}_{\text{gen}}(k\text{u}^2\text{lEPT0L}) = \mathcal{L}_{\text{acc}}(k\text{u}^2\text{lEPT0L}) =$
$\mathcal{L}_{\text{gen}}(\text{uSC} - \lambda) = \mathcal{L}_{\text{acc}}(\text{uSC} - \lambda) = \mathcal{L}_{\text{gen}}(\text{P,CF} - \lambda) = \mathcal{L}_{\text{acc}}(\text{P,CF} - \lambda)$.

All these language families properly include the family of λ-free context-free languages (which in turn equals $\mathcal{L}_{\text{gen}}(1\text{u}^2\text{lEPT0L}) = \mathcal{L}_{\text{acc}}(1\text{u}^2\text{lEPT0L})$), even more, they include $\mathcal{L}_{\text{gen}}(k\text{ulEPT0L})$.

The class of λ-free context-sensitive languages properly and effectively contains all these language families.

Moreover, each $k\text{u}^2\text{lEPTOL}$-language can be described by a $2\text{u}^2\text{lEPT0L}$-system satisfying the normal form described in the preceding theorem. □

Observe that we could prove Lemma 7.3 also directly using our metatheorem. We show this technique in the following for scattered context grammars in general. For any production $(v_{i1}, \ldots, v_{ir_i}) \to (w_{i1}, \ldots, w_{ir_i})$ of a scattered context grammar, we introduce a table t_i in the simulating cc grammar consisting of the productions $(v_{ij} \to w_{ij}, (V_G^*\{\#\})^{r_i} V_G^* \times \{j\})$ for each $1 \leq j \leq r_i$.

Theorem 7.10 • $\mathcal{L}_{\text{gen}}(\text{SC}) = \mathcal{L}_{\text{acc}}(\text{SC}) = \mathcal{L}(\text{RE})$

- $\mathcal{L}_{\text{gen}}(\text{SC} - \lambda) = \mathcal{L}_{\text{acc}}(\text{SC} - \lambda)$

Proof. We still have to show $\mathcal{L}_{\text{gen}}(\text{SC}) = \mathcal{L}(\text{RE})$.[5] By [16, Corollary on page 245], the enumerable languages can be characterized as homomorphic images of $\mathcal{L}_{\text{gen}}(\text{SC} - \lambda)$. This result yields the claim almost immediately. □

7.3 Conclusions

From previous works [2, 11, 32], we know that

$$\mathcal{L}_{\text{gen}}(k\text{ulET0L}) = \mathcal{L}_{\text{gen}}(k\text{ulET0L,ex}) = \mathcal{L}_{\text{acc}}(k\text{ulET0L,ex}),$$

hence we can say that this language family is somewhat robust as far as small changes within its defining mechanism are concerned.

This is probably not true for the modification of $k\text{u}^2\text{lET0L}$ systems discussed in this note. Via the established connections of our mechanism with well-known families from the theory of regulated rewriting, one might get a better insight in the subtle difficulties encountered within uniformly limited systems.

Below, we list some properties of $k\text{u}^2\text{E(P)T0L}$ languages inherited from the work on programmed grammars whose state is presently unknown for $k\text{ulE(P)T0L}$ languages. Hence, this list might help to settle the conjectured strictness of the inclusion $\mathcal{L}_{\text{gen}}(k\text{ulE(P)T0L}) \subseteq \mathcal{L}_{\text{gen}}(k\text{u}^2\text{lE(P)T0L})$.

- The language class $\mathcal{L}_{\text{gen}}(k\text{u}^2\text{lE(P)T0L})$ is closed under the following operations:
  - concatenation

[5] Note that this also supplements [5, page 147].

  - intersection with regular languages
  - substitution by context-free languages
  - (λ-free) gsm and inverse gsm mappings
  - permutations
  - quasiintersection

- The language class $\mathcal{L}_{\text{gen}}(k\text{u}^2\text{lE(P)T0L})$ is not closed under the following operations:
  - intersection
  - complementation

In [30], Salomaa only considered systems with one table in the exact mode. In that case, 'T' is omitted in our abbreviations. (The fact that Salomaa's (k-)context-free grammars start from only one string does not matter, since it is possible to 'blow' this string up with a symbol Λ immediately deriving to λ such that we can replace an exact $k$ulE0L system $(\Sigma, \Delta, \{h\}, \Omega, k)$ by an equivalent k-context-free grammar $(\Sigma \cup \{S, \Lambda\}, \Delta, h \cup \{\Lambda \to \lambda\} \cup \{S \to \omega \mid \omega \in \Omega\}, S\Lambda^{k-1})$. Moreover, Wätjen and Unruh showed in [36] the equivalence of k-context-free grammars and $k$ulE0L systems.) Salomaa showed that one obtains an infinite strict hierarchy

$$\mathcal{L}_{\text{gen}}(1\text{ulE0L}) \subsetneq \mathcal{L}_{\text{gen}}(2\text{ulE0L}) \subsetneq \mathcal{L}_{\text{gen}}(3\text{ulE0L}) \subsetneq \dots$$

The corresponding relations are unclear in the case of an arbitrary number of tables or when restricting to propagating systems. On the contrary, the same situation is clear for k-uniquely uniformly limited systems, where the hierarchy nearly collapses:

$$\mathcal{L}_{\text{gen}}(1\text{u}^2\text{lE(P)T0L}) \subsetneq \mathcal{L}_{\text{gen}}(2\text{u}^2\text{lE(P)T0L}) = \mathcal{L}_{\text{gen}}(3\text{u}^2\text{lE(P)T0L}) = \dots$$

Finally, observe that the systems introduced in the present paper may be also seen as a variation of so-called 1lET0L systems [32] (see below). In a 1lET0L system, a derivation step according to some chosen table h is defined as follows:

- Replace one occurrence of each symbol according to h, if this is possible![6]

We may see a derivation step according to the table h of a 2u^{2}lET0L system as follows:

- Replace one occurrence of each symbol according to h, such that exactly 2 characters are replaced in total![7]

[6] i.e. if the string to be rewritten contains at least one occurrence of the symbol under question; if it does not contain that particular symbol, this does not affect the rewriting of the other symbols

[7] Again, by inspecting the proof of Lemma 2.4.6 in [5], one sees that the system constructed in Lemma 7.7 has the property desired by these new restrictions also in the case of tables of type H_1.

So, this note may also help to get a better understanding of the relationship between limited and uniformly limited rewriting. This would probably add to the comprehension of programmed grammars, too. Whilst it is clear from [31] that in the non-erasing case the class $\mathcal{L}_{\rm gen}({\rm P,CF}-\lambda,{\rm ut})$ of languages generable by so-called programmed grammars with unconditional transfer does not coincide with the class $\mathcal{L}_{\rm gen}({\rm P,CF}-\lambda)$ (which in turn equals $\mathcal{L}_{\rm gen}({\rm uSC}-\lambda)$), it is still unknown whether there holds an inclusion relation, and if there is such a relation, which direction is true. Even more, literally nothing seems to be known in the corresponding case with erasing productions.[8] Now, $\mathcal{L}_{\rm gen}({\rm P,CF}-\lambda,{\rm ut})\supseteq\mathcal{L}_{\rm gen}({\rm 1lEPT0L})$, and $\mathcal{L}_{\rm gen}({\rm P,CF,ut})=\mathcal{L}_{\rm gen}({\rm 1lET0L})$ [4], whilst on the other hand $\mathcal{L}_{\rm gen}({\rm P,CF}-\lambda)\supseteq\mathcal{L}_{\rm gen}(k{\rm ulEPT0L})$ and $\mathcal{L}_{\rm gen}({\rm P,CF})\supseteq\mathcal{L}_{\rm gen}(k{\rm ulET0L})$.

[8] New results are contained in [8].

Chapter 8

Limited L Systems

In [32], another similar type of parallel derivation was examined, so-called limited L systems. In such systems, say klE0L systems with a fixed k, in each derivation step, k occurrences of each symbol have to be rewritten, if possible. In this paper, we examine corresponding accepting systems. But what should this mean for klE0L systems? "Replace k occurrences of non-overlapping subwords for each left-hand side of a production" would lead to the same problems as discussed above within the uniformly limited systems if there are not k subwords of the required form. Furthermore, there are different ideas how to handle situations like in the string $abba$ to be derived and productions $abb \to a$ and $ba \to b$, where there is some kind of mutual exclusion of the application of those productions. Similar problems are encountered within variants of limited L systems as considered in [6, 14].

Since it is the easiest case, we restrict ourselves to 1lET0L systems in the following as first introduced by Frings in [12]. At first, we give precise definitions of such systems [32].

A 1-*limited ET0L system* (abbreviated as 1lET0L system) is a quintuple $G = (V, V', \{P_1, \ldots, P_r\}, \omega, 1)$ such that $(V, V', \{P_1, \ldots, P_r\}, \omega)$ is an ET0L system (either generating or accepting). The notion of propagating and that of deterministic systems is inherited from L systems. According to G, $x \Rightarrow y$ (for $x, y \in V^*$) iff there is a table P_i and partitions $x = x_0\alpha_1x_1 \cdots \alpha_nx_n$, $y = x_0\beta_1x_1 \cdots \beta_nx_n$ such that $\alpha_\nu \to \beta_\nu \in P_i$ for each $1 \leq \nu \leq n$, $\alpha_\nu \neq \alpha_\mu$ for $\nu \neq \mu$, and each left-hand side z of a production of P_i is either equal to some α_ν or not contained in $\mathrm{Sub}(x_0) \cup \mathrm{Sub}(x_1) \cup \cdots \cup \mathrm{Sub}(x_n)$.

The language generated by a generating 1lET0L system G is $L_{\mathrm{gen}}(G) = \{w \in V'^* \mid \omega \overset{*}{\Rightarrow} w\}$. The language accepted by an accepting 1lET0L system G is $L_{\mathrm{acc}}(G) = \{w \in V'^* \mid w \overset{*}{\Rightarrow} \omega\}$.

Theorem 8.1 $\mathcal{L}_{\mathrm{acc}}(\mathrm{1lEPT0L}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{CS})$

Proof. We transfer the proof of Theorem 3.5, taking into account the modifications indicated in the proof of Theorem 3.15. The simulation of an accepting 1lEPT0L system by a linear bounded automaton should be clear and is therefore omitted.

It is easily seen that $\mathcal{L}_{\mathrm{acc}}(\mathrm{1lEPT0L})$ is closed with respect to finite union and contains all finite languages. Therefore, for the proof of $\mathcal{L}_{\mathrm{acc}}(\mathrm{1lEPT0L}) \supseteq \mathcal{L}_{\mathrm{gen}}(\mathrm{CS})$, it is sufficient to show that $\{a\}M\{b\} \in \mathcal{L}_{\mathrm{acc}}(\mathrm{1lEPT0L})$ for $M \in \mathcal{L}_{\mathrm{gen}}(\mathrm{CS})$, $\lambda \notin M$. Let $G = (V_N, V_T, P, S)$ be a context-sensitive grammar without λ-productions in Kuroda normal form generating M. Let us assume a unique label r being attached to any rule of the form $XU \to YZ$ (the set of lables is denoted by Lab). We construct a 1lEPT0L system $G' = (V, V', \{\mathrm{init}_a, \mathrm{init}_b, \mathrm{term}\} \cup \{\text{simul-cf}_{X \to x} \mid X \to x \in P\} \cup \{\text{simul-cs}_{r,i} \mid r \in Lab, 1 \leq i \leq 6\}, S', 1)$ accepting $\{a\}M\{b\}$ as follows. Let

$$V = V_N \cup \{A, B, S', F_l, F_r\} \cup \{(A,r), [A,r], [B,r], (Y,r), [Z,r] \mid r : XU \to YZ \in P\} \cup V'$$

(the unions being disjoint), where $V' = V_T \cup \{a, b\}$.

For brevity, we leave out productions of the form $F_l \to x$ which have to be added for right-hand sides x not present in the table specifications listed above in order to fulfill the completeness restriction for tables inherited from L systems.

1. start/termination/context-free rules:

   (a) $\mathrm{init}_a = \{a \to A\} \cup \{x \to F_r \mid x \in V \setminus V'\}$

   (b) $\mathrm{init}_b = \{b \to B\} \cup \{x \to F_r \mid x \in V \setminus (V' \cup \{A\})\}$

   (c) $\text{simul-cf}_{X \to x} = \{x \to X\} \cup \{(A,r) \to F_r, [A,r] \to F_r \mid r \in Lab\}$ for context-free rules $X \to x \in P$;

   (d) $\mathrm{term} = \{ASB \to S'\} \cup \{x \to F_r \mid x \in V\}$

2. For each context-sensitive rule $r : XU \to YZ \in P$, we introduce the next tables:

   (a) $\text{simul-cs}_{r,1} = \{A \to [A,r], B \to [B,r]\} \cup \{[A,s] \to F_r, (A,s) \to F_r, [B,s] \to F_r \mid s \in Lab\}$;

   (b) $\text{simul-cs}_{r,2} = \{Y \to [Y,r]\} \cup \{A \to F_r, B \to F_r, (A,s) \to F_r, [A,s'] \to F_r, [B,s'] \to F_r, [T,s] \to F_r, (T,s) \to F_r \mid s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N\}$;

   (c) $\text{simul-cs}_{r,3} = \{Z \to (Z,r)\} \cup \{A \to F_r, B \to F_r, (A,s) \to F_r, [A,s'] \to F_r, [B,s'] \to F_r, [T,s'] \to F_r, (T,s) \to F_r \mid s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N\}$;

   (d) $\text{simul-cs}_{r,4} = \{[A,r] \to (A,r)\} \cup \{A \to F_r, B \to F_r, (A,s) \to F_r, [A,s'] \to F_r, [B,s'] \to F_r, [T,s'] \to F_r, (T,s') \to F_r, [Y,r]y \to F_r \mid s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N, z \in V \setminus \{[Y,r]\}, y \in V \setminus \{(Z,r)\}\}$;

   (e) $\text{simul-cs}_{r,5} = \{[B,r] \to B\} \cup \{A \to F_r, B \to F_r, (A,s') \to F_r, [A,s] \to F_r, [B,r] \to F_r, [T,s'] \to F_r, (T,s') \to F_r, z(Z,r) \to F_r \mid s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N, z \in V \setminus \{[Y,r]\}, y \in V \setminus \{(Z,r)\}\}$;

   (f) $\text{simul-cs}_{r,6} = \{(A,r) \to A\} \cup \{(Z,r) \to U\} \cup \{[Y,r] \to X\} \cup \{A \to F_r, [A,s] \to F_r, [B,s] \to F_r, (A,s') \to F_r, [T,s'] \to F_r, (T,s') \to F_r \mid s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N\}$. □

Alternatively, we could have given a proof parallelling [4]. Note that in this case, some technical subtleties must be circumvented.

A third proof alternative is, using our meta-observation, to notice that the proof given in [7, Lemma 3.10] showing that $\mathcal{L}_{\text{gen}}(\text{O,CF}-\lambda) \subseteq \mathcal{L}_{\text{gen}}(\text{1lEPT0L})$ is valid in the erasing and accepting cases, too, which allows us to apply Theorem 3.5 and Corollary 3.7 directly.

Hence, we can derive the following.

Corollary 8.2 $\mathcal{L}_{\text{acc}}(\text{1lET0L}) = \mathcal{L}_{\text{gen}}(\text{RE})$ □

Our theorem largely solves the relation between generating and accepting mode in 1lET0L systems.

Corollary 8.3 (i) $\mathcal{L}_{\text{gen}}(\text{1lEPT0L}) \subset \mathcal{L}_{\text{acc}}(\text{1lEPT0L})$

(ii) $\mathcal{L}_{\text{gen}}(\text{1lET0L}) \subseteq \mathcal{L}_{\text{acc}}(\text{1lET0L})$

Proof. Dassow showed in [4] the inclusion $\mathcal{L}_{\text{gen}}(\text{1lEPT0L}) \subseteq \mathcal{L}_{\text{gen}}(\text{P,CF}-\lambda\text{,ut})$. In [31], the strictness of the inclusion $\mathcal{L}_{\text{gen}}(\text{P,CF}-\lambda\text{,ut}) \subset \mathcal{L}_{\text{gen}}(\text{CS})$ is proved. Hence, by our above Theorem 8.1, we know the first inclusion to be strict. □

We remark on point (ii) that, in spite of the solvability of the emptiness problem for 1lET0L systems,[1] it may be that the inclusion (ii) is not strict, but the method turning a, say type-0-grammar, into a generating 1lET0L system is not effective. Similarly, $\mathcal{L}_{\text{gen}}(\text{1lET0L})$ may be closed under intersection with regular languages, but not effectively closed.[2] These questions are thoroughly discussed in [9].

Observe that the simulation of generating 1lET0L systems via accepting ones is quite indirect in the preceding corollary: Firstly, we sequentialize 1lET0L systems using, e.g., type-0-grammars, secondly, we use parallelism and accepting mode in order to mimic the context checks possible in phrase structure grammars. It is instructive to try a direct proof of the corresponding inclusions. The main problems are the following three: (1) if you employ the dual G^d of a generating 1lET0L system G in order to simulate G's derivations,

[1] By [4], we can effectively turn a 1lET0L system into a programmed grammar with unconditional transfer. For generating (P,CF$-\lambda$,ut)-grammars, the emptiness problem is decidable by [31]. This implies, by the construction given in Lemma 3.9, the solvability of the emptiness problem for generating 1lET0L systems and generating (P,CF,ut)-grammars.

[2] This effectivity should have been added into the formulation of [18, Theorem 3], since otherwise we would have the following interesting 'proof' for the strictness of the inclusion in question: Assume to the contrary $\mathcal{L}_{\text{gen}}(\text{1lET0L}) = \mathcal{L}_{\text{acc}}(\text{1lET0L})$. Then, $\mathcal{L}_{\text{gen}}(\text{1lET0L})$ is closed under intersection with regular languages, since $\mathcal{L}_{\text{acc}}(\text{1lET0L}) = \mathcal{L}_{\text{acc}}(\text{RE})$ is so. Applying [18, Theorem 3(b)], we can find a language in $\mathcal{L}_{\text{acc}}(\text{1lET0L}) \setminus \mathcal{L}_{\text{gen}}(\text{1lET0L})$, contradicting our assumption. Hence, $\mathcal{L}_{\text{gen}}(\text{1lET0L}) \subset \mathcal{L}_{\text{acc}}(\text{1lET0L})$, but we cannot show up with a concrete example as in [18]. But note that the construction given above proves the following rather strange situation for generating 1lET0L systems: Either the inclusion in question is strict, i.e. 1lET0L systems are not computationally universal, or, in case $\mathcal{L}_{\text{gen}}(\text{1lET0L}) = \mathcal{L}_{\text{acc}}(\text{1lET0L})$, the family of 1lET0L systems is not effectively closed under intersection with regular languages, i.e. either the family of languages $\mathcal{L}_{\text{gen}}(\text{1lET0L})$ is not closed under intersection with regular languages or there is no algorithm that, given a 1lET0L system G and say a finite automaton A, delivers another 1lET0L system G' such that $L(G') = L(G) \cap L(A)$.

because there might be applicable productions $a \to a, b \to a$ in G, not any derivation of G may be simulated by G^d; (2) there are also problems with derivations possible in G^d but not in G (consider the 'dual' example of two productions $a \to a$, $a \to b$ in G); (3) instead of employing additional productions, G^d might also 'forget' productions to be simulated, e.g., with the set of productions $\{a \to aa, b \to ab\}$ in G and the axiom ab, the word aab would be accepted by G^d even in two different ways. (1) and (2) can be circumvented enhancing the alphabet. Difficulty (3) might be overcome using regulation of k-limited systems as introduced in [34] for the generating case. Of course, one might use additional tables in order to check whether a simulation step has been correct or not (like in the proof of the last theorem). It is instructive that case (3) also prevents the application of the idea of symmetrically deterministic systems, although a corresponding normal form does exist both in the generating and in the accepting mode. The problem is again that the simple idea of duality does not apply in the case of limited Lindenmayer systems.

Nevertheless, it is easily seen by a direct argument that

$$\mathcal{L}_{\text{acc}}(\text{1lEDT0L}) = \mathcal{L}_{\text{acc}}(\text{1lET0L}),$$

hence showing again that the syntactical concept of determinism does not decrease the descriptive power of accepting parallel systems with extension mechanism.

We were not able to do the necessary simulations without the help of additional tables; hence, we do not know whether $\mathcal{L}_{\text{gen}}(\text{1lE0L}) \subseteq \mathcal{L}_{\text{acc}}(\text{1lE0L})$ is true or not. Nonetheless, using arguments similar to [33], one can convince oneself that the inclusion $\mathcal{L}_{\text{acc}}(\text{1lE(P)0L}) \subseteq \mathcal{L}_{\text{acc}}(\text{1lE(P)T0L})$ is strict.

Note that it is possible to reason about the degree of synchronization in accepting 1lET0L systems using the same ideas as in [35], therefore, the degree of synchronization of an arbitrary accepting 1lET0L language equals two.

In the case of generating systems, the exact relation between the generative power of klE[P]T0L and $k$ulE[P]T0L systems is still open [36]. In the case of accepting systems, we readily infer the next theorem.

Theorem 8.4 $\bigcup_{k\in\mathbb{N}} \mathcal{L}_{\text{acc}}(k\text{ulE[P]T0L}) \subset \mathcal{L}_{\text{acc}}(\text{1lE[P]T0L})$. □

In [8], we recently showed that $\mathcal{L}_{\text{gen}}(\text{ulET0L})$ and $\mathcal{L}_{\text{gen}}(\text{1lET0L})$ cannot coincide, since $\mathcal{L}_{\text{gen}}(\text{1lE[P]T0L})$ contains non-recursive languages. Moreover, for $\mathcal{L} = \mathcal{L}_{\text{gen}}(\text{ulET0L})$, the following predicate is true.

- For each homomorphism h and language $L \in \mathcal{L}$, $L \subseteq \Sigma^*$, there is a Turing machine $T_{h,L}$ such that $T_{h,L}$ computes the function

$$f_{h,L} : \Sigma^* \to \{0,1\}, w \mapsto 1 \text{ iff } w \in h(L)$$

On the other hand, we showed in [8] that there is a homomorphism h and a language $L \in \mathcal{L}_{\text{gen}}(\text{1lEPT0L})$ such that the above predicate is false, i.e. the corresponding $f_{h,L}$

is uncomputable. Hence, $\mathcal{L}_{\text{gen}}(\text{ulEPT0L})$ and $\mathcal{L}_{\text{gen}}(\text{1lEPT0L})$ cannot coincide, and, furthermore, there is a language $L \in \mathcal{L}_{\text{gen}}(\text{1lEPT0L}) \setminus \mathcal{L}_{\text{gen}}(\text{ulEPT0L})$.

Further note that our results on accepting limited Lindenmayer systems underline the relationship of this parallel language class with the regulated language class of programmed grammars with unconditional transfer, a relationship already observed by Dassow [4, 6] in the generating case. We summarize these relations in the following corollary. In the generating case, the strictness of one inclusion is still open.

Corollary 8.5 (i) $\bigcup_{k\in\mathbb{N}} \mathcal{L}_{\text{gen}}(k\text{lEPT0L}) \subseteq \mathcal{L}_{\text{gen}}(\text{P,CF}-\lambda\text{,ut})$

(ii) $\bigcup_{k\in\mathbb{N}} \mathcal{L}_{\text{gen}}(k\text{lET0L}) = \mathcal{L}_{\text{gen}}(\text{1lET0L}) = \mathcal{L}_{\text{gen}}(\text{P,CF,ut})$

(iii) $\mathcal{L}_{\text{acc}}(\text{1lEPT0L}) = \mathcal{L}_{\text{acc}}(\text{P,CF}-\lambda\text{,ut})$

(iv) $\mathcal{L}_{\text{acc}}(\text{1lET0L}) = \mathcal{L}_{\text{acc}}(\text{P,CF,ut})$ □

Chapter 9

Summary

In this section, we want to summarize the results obtained in this report and in other papers. In some sense, the following diagrams supplement the diagram on page 146 in [5]. In the diagrams, solid lines indicate strict inclusion, where the larger language class is near the arrow-tip, dashed lines indicate an inclusion relation where the strictness is unknown, and dotted lines mean that the inclusion relation which is indicated by the arrow tip cannot hold. Incomparability of language classes is indicated by a dotted line without arrows between them. In order to shorten the diagrams, we simply write $\mathcal{L}$ if the language accepting capacity equals the language generating capacity of the grammar families. As usual, we define

$$\mathcal{L}_{\mathrm{acc/gen}}(\text{[u]lE[P]T0L}) = \bigcup_{k\geq 1} \mathcal{L}_{\mathrm{acc/gen}}(k\text{[u]lE[P]T0L}).$$

We try to refer to proofs of any claim which is not proved in [5, 27] or is one of the many results comparing generating and accepting devices in this paper.

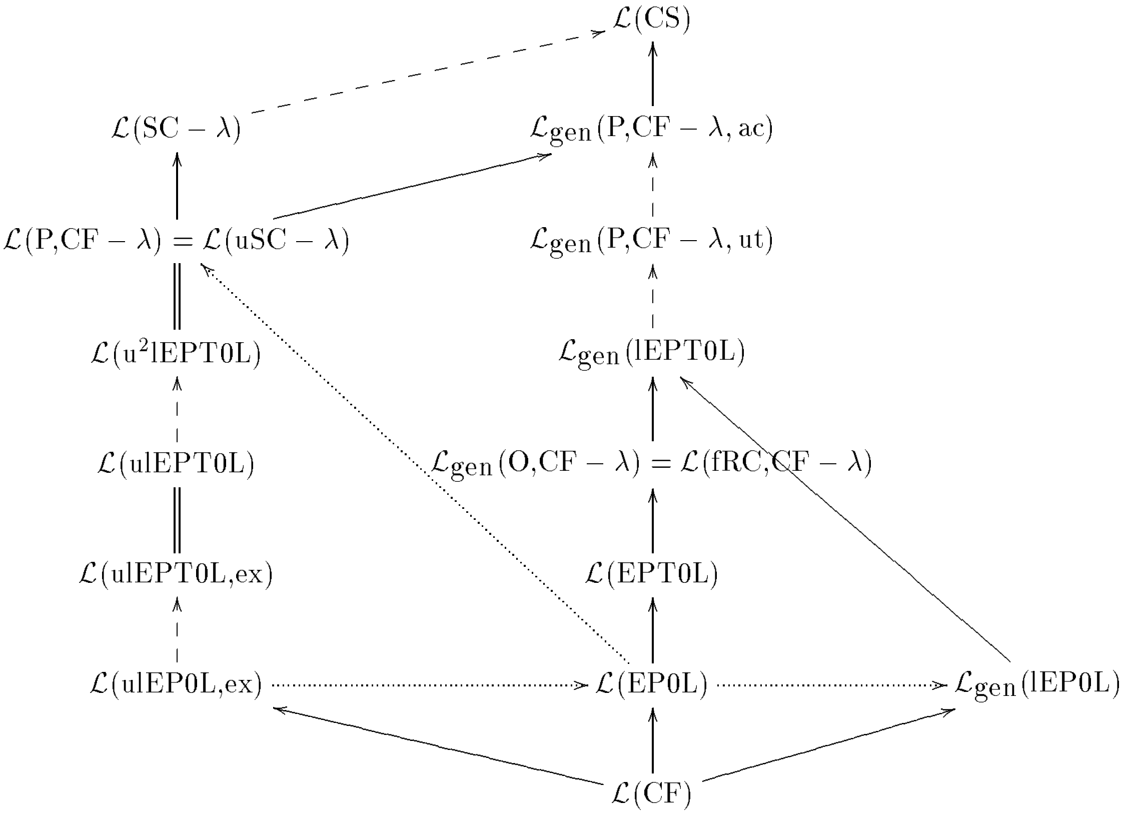
This section can also be understood as an invitation to the reader to help to clarify the yet unsolved and open relationships between the listed families of languages.

9.1 The Non-Erasing Case

Reference of claims:

- $\mathcal{L}_{\mathrm{gen}}(\text{ulEPT0L}) \subseteq \mathcal{L}_{\mathrm{gen}}(\text{P,CF}-\lambda)$ see Theorem 6.3
- In [17], it is shown that $\{a^{2^n} \mid n \in \mathbb{N}\} \notin \mathcal{L}_{\mathrm{gen}}(\text{P,CF})$. Hence, there are EP0L languages which are not in $\mathcal{L}_{\mathrm{gen}}(\text{P,CF})$. This naturally applies also to superfamilies of $\mathcal{L}_{\mathrm{gen}}(\text{EPT0L})$ and to subfamilies of $\mathcal{L}_{\mathrm{gen}}(\text{P,CF})$. Especially, this partially answers a question raised in [36] concerning the relationship of uniformly limited L systems and L systems.
- The same example shows the strictness of the inclusion between $\mathcal{L}_{\mathrm{gen}}(\text{P,CF}-\lambda)$ and $\mathcal{L}_{\mathrm{gen}}(\text{P,CF}-\lambda,\text{ac})$. Confer also to [18].

- Currently, we do not trust the proof given in [31] showing the strictness of the inclusion $\mathcal{L}_{\text{gen}}(\text{P,CF}-\lambda,\text{ut}) \subseteq \mathcal{L}_{\text{gen}}(\text{P,CF}-\lambda,\text{ac})$.
- In [8, Theorem 5.2], we showed $\mathcal{L}_{\text{gen}}(\text{O,CF}-\lambda) \subsetneq \mathcal{L}_{\text{gen}}(\text{1lEPT0L})$.
- The strictness of the inclusion $\mathcal{L}_{\text{gen}}(\text{CF}-\lambda) \subset \mathcal{L}_{\text{gen}}(\text{ulEP0L})$ can be seen by Example 2.1 in [36], where also the incomparability results concerning EP0L are proved.
- In [32, Theorem 4.4] together with [33], it is especially shown that $\mathcal{L}_{\text{gen}}(\text{lEP0L})$ does not contain $\mathcal{L}_{\text{gen}}(\text{EP0L})$. Similarly, [33] shows the strictness of the inclusion $\mathcal{L}_{\text{gen}}(\text{lEP0L}) \subset \mathcal{L}_{\text{gen}}(\text{lEPT0L})$.
- The inclusion $\mathcal{L}_{\text{gen}}(\text{lEPT0L}) \subseteq \mathcal{L}_{\text{gen}}(\text{P,CF}-\lambda,\text{ut})$ is proved in [4, 6].
- The strictness of the inclusion $\mathcal{L}(\text{uSC}-\lambda) \subset \mathcal{L}(\text{SC}-\lambda)$ is seen as follows: any recursively enumerable language is the homomorphic image of some $\mathcal{L}(\text{SC}-\lambda)$-language (see the proof of Theorem 7.10), but homomorphic images of $\mathcal{L}(\text{uSC}-\lambda)$-languages are always recursive by [18].



$\mathcal{L}(\text{CS})$
$\mathcal{L}(\text{SC}-\lambda)$
$\mathcal{L}_{\text{gen}}(\text{P,CF}-\lambda,\text{ac})$
$\mathcal{L}(\text{P,CF}-\lambda) = \mathcal{L}(\text{uSC}-\lambda)$
$\mathcal{L}_{\text{gen}}(\text{P,CF}-\lambda,\text{ut})$
$\mathcal{L}(\text{u}^2\text{lEPT0L})$
$\mathcal{L}_{\text{gen}}(\text{lEPT0L})$
$\mathcal{L}(\text{ulEPT0L})$
$\mathcal{L}_{\text{gen}}(\text{O,CF}-\lambda) = \mathcal{L}(\text{fRC,CF}-\lambda)$
$\mathcal{L}(\text{ulEPT0L,ex})$
$\mathcal{L}(\text{EPT0L})$
$\mathcal{L}(\text{ulEP0L,ex})$
$\mathcal{L}(\text{EP0L})$
$\mathcal{L}_{\text{gen}}(\text{lEP0L})$
$\mathcal{L}(\text{CF})$

Furthermore, we know that

$\mathcal{L}(\mathrm{CS}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF} - \lambda, \mathrm{ac}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF} - \lambda, \mathrm{ut}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{1lEPT0L}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{O,CF} - \lambda)$.

Finally, the problem remains where to place the chain

$$\begin{tikzcd}\mathcal{L}(\mathrm{CF}) \arrow[r] & \mathcal{L}(\mathrm{RC,CF} - \lambda) \arrow[r, dashed] & \mathcal{L}(\mathrm{P,CF} - \lambda)\end{tikzcd}$$

within the sketched diagram.

9.2 The Erasing Case

With the references given in the preceding subsection, most of the connections given in the following diagram should be clear.

We add only two references:

- The strictness of the inclusion $\mathcal{L}_{\mathrm{gen}}(\mathrm{lE0L}) \subset \mathcal{L}_{\mathrm{gen}}(\mathrm{lET0L})$ was shown in [33].
- We showed in 3.10 that ordered grammars have a solvable membership problem. Furthermore, $\mathcal{L}_{\mathrm{gen}}(\mathrm{O,CF})$ is closed under homomorphism. If each context-sensitive language were generable by an ordered grammar, then also any homomorphic image of a context-sensitive language were generable by an ordered grammar, which would finally imply the recursiveness of any enumerable language.

We know that the following language classes characterize $\mathcal{L}(\mathrm{RE}) = \mathcal{L}(\mathrm{P,CF,ac})$:

- $\mathcal{L}_{\mathrm{acc}}(\mathrm{O,CF})$
- $\mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF,ut}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ac}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF,ac})$
- $\mathcal{L}_{\mathrm{acc}}(\mathrm{1lET0L})$
- $\mathcal{L}(\mathrm{SC})$
- $\mathcal{L}(\mathrm{u}^2\mathrm{lET0L})$

Moreover,

$$\begin{tikzcd}
 & \mathcal{L}(\mathrm{RE}) & \\
 & \mathcal{L}(\mathrm{REC}) \arrow[u] & \mathcal{L}_{\mathrm{gen}}(\mathrm{lET0L}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ut}) \arrow[ul, dashed] \arrow[l, dotted] \\
\mathcal{L}(\mathrm{P,CF}) = \mathcal{L}(\mathrm{uSC}) \arrow[ur] & \mathcal{L}(\mathrm{CS}) \arrow[u] \arrow[r, dotted] & \mathcal{L}_{\mathrm{gen}}(\mathrm{O,CF}) = \mathcal{L}(\mathrm{fRC,CF}) \arrow[ul] \arrow[u] \arrow[dd, dotted] \\
\mathcal{L}(\mathrm{ulET0L,ex}) \arrow[u, dashed] & \mathcal{L}(\mathrm{ET0L}) \arrow[u] \arrow[ur] & \\
\mathcal{L}(\mathrm{ulE0L,ex}) \arrow[u, dashed] \arrow[r, dotted] & \mathcal{L}(\mathrm{E0L}) \arrow[u] \arrow[uul, dotted] \arrow[r, dotted] & \mathcal{L}_{\mathrm{gen}}(\mathrm{lE0L}) \arrow[uuu] \\
 & \mathcal{L}(\mathrm{CF}) \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Let us point the reader to the following interesting observation [8, Theorem 5.5]: $\mathcal{L}_{\mathrm{gen}}(\mathrm{CS}) \subseteq \mathcal{L}_{\mathrm{gen}}(\mathrm{1lET0L})$ iff $\mathcal{L}_{\mathrm{gen}}(\mathrm{1lET0L}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ut}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{RE})$.

Finally, the problem remains where to place the chain

$$\begin{tikzcd}
\mathcal{L}(\mathrm{CF}) \arrow[r] & \mathcal{L}(\mathrm{RC,CF}) \arrow[r, dashed] & \mathcal{L}(\mathrm{P,CF})
\end{tikzcd}$$

within the sketched diagram.

Note that we did not list the classical equivalences of various types of regulated rewriting which also transfer to the accepting case, as shown in the paper. As one representative, we refer to programmed grammars in this section.

Chapter 10

Conclusions

In our introductory section, we listed three cases of possible relations between the language families generated and the language families accepted by some device X. Indeed, we found representatives for all these cases. We list these cases in detail in the case of parallel rewriting below.

- In the case of Lindenmayer and uniformly limited systems with extension mechanism, we found a trivial equivalence between accepting and generating mode.
- In the case of uniformly limited systems in Wätjen's mode (together with the extension mechanism), this equivalence was proved using a more involved technique.
- In the case of 1-limited systems with extension mechanism, the accepting mode is strictly more powerful than the generating one.

In the case of $\mathcal{L}_{\text{gen}}(X) \subset \mathcal{L}_{\text{acc}}(X)$, we mostly found this by proving that $\mathcal{L}_{\text{acc}}(X) = \mathcal{L}_{\text{gen}}(\text{CS})$, whereas the class $\mathcal{L}_{\text{gen}}(X)$ is known to be a proper subset of $\mathcal{L}_{\text{gen}}(\text{CS})$. Besides verifying the relationship between the families $\mathcal{L}_{\text{acc}}(X)$ and $\mathcal{L}_{\text{gen}}(X)$, these results give an interesting, somewhat practical characterization of the class of context-sensitive languages: the family $\mathcal{L}_{\text{gen}}(\text{CS})$ consists exactly of all languages which can be parsed (accepted) by a nonerasing context-free matrix grammar with appearance checking, or by a nonerasing context-free programmed grammar with unconditional transfer, or by an ordered grammar with λ-free context-free rules, etc. In this connection, it might also be interesting to investigate programmed, matrix, ... grammars with, e.g., linear or right-linear λ-free rules as accepting devices, examining in more detail the reason why context-free productions give such a descriptive power when considered in accepting mode.

We did not find examples for the following two cases:

- Is there a grammar family such that the generating mode is strictly more powerful than the accepting mode?
- Is there a grammar family such that the language families corresponding to generating and accepting mode are incomparable?

As regards the first case, we found a possible candidate, namely deterministic k-uniformly-limited EPT0L systems in Wätjen's mode, but we could not prove the strictness of the corresponding inclusion. For the second case, we will find examples when considering pure grammars. We present these results in another paper.

Bibliography

[1] S. Abraham. Some questions of phrase-structure grammars. *Comput. Linguistics*, 4:61–70, 1965.

[2] H. Bordihn and H. Fernau. Accepting grammars and systems. Technical Report 9/94, Universität Karlsruhe, Fakultät für Informatik, 1994.

[3] H. Bordihn and H. Fernau. Accepting grammars with regulation. *International Journal of Computer Mathematics*, 53:1–18, 1994.

[4] J. Dassow. A remark on limited 0L systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 24(6):287–291, 1988.

[5] J. Dassow and G. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science*. Berlin: Springer, 1989.

[6] H. Fernau. On function-limited Lindenmayer systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 27(1):21–53, 1991.

[7] H. Fernau. Adult languages of propagating systems with restricted parallelism. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 29(5):249–267, 1993.

[8] H. Fernau. Membership for limited ET0L languages is not decidable. Technical Report 34/94, Universität Karlsruhe, Fakultät für Informatik, Dec. 1994.

[9] H. Fernau. On grammar and language families. To appear in Fundamenta Informaticae, 1994.

[10] H. Fernau. A note on uniformly limited ET0L systems with unique interpretation. To appear in Information Processing Letters, 1995.

[11] H. Fernau and H. Bordihn. Remarks on accepting parallel systems. *International Journal of Computer Mathematics*, 57(3+4), 1995.

[12] M. Frings. Systeme mit eingeschränkter paralleler Ersetzung. Master's thesis, TU Braunschweig, D-3300 Braunschweig, 1985.

[13] I. Friš. Grammars with partial orderings of the rules. *Information and Control (now Information and Computation)*, 12:415–425, 1968.

[14] S. Gärtner. Partitions-limitierte Lindenmayersysteme. In W. Thomas, editor, *2. Theorietag 'Automaten und Formale Sprachen'*, volume 9220 of *Technische Berichte*, pages 23–27. Universität Kiel, 1992.

[15] S. Ginsburg and E. H. Spanier. Control sets on grammars. *Math. Syst. Th.*, 2:159–177, 1968.

[16] S. Greibach and J. Hopcroft. Scattered context grammars. *Journal of Computer and System Sciences*, 3:233–247, 1969.

[17] D. Hauschildt and M. Jantzen. Petri net algorithms in the theory of matrix grammars. *Acta Informatica*, 31:719–728, 1994.

[18] F. Hinz and J. Dassow. An undecidability result for regular languages and its application to regulated rewriting. *EATCS Bulletin*, 38:168–173, 1989.

[19] K. Landskron. On k-limited L forms. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 30(1):19–27, 1994.

[20] E. Navrátil. Context-free grammars with regular conditions. *Kybernetika*, 2:118–126, 1970.

[21] M. Nielsen. E0L systems with control devices. *Acta Informatica*, 4:373–386, 1975.

[22] G. Păun. On the generative capacity of conditional grammars. *Information and Control (now Information and Computation)*, 43:178–186, 1979.

[23] G. Păun. A variant of random context grammars: semi-conditional grammars. *Theoretical Computer Science*, 41:1–17, 1985.

[24] M. Penttonen. One-sided and two-sided context in formal grammars. *Information and Control (now Information and Computation)*, 25:371–392, 1974.

[25] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the Association for Computing Machinery*, 16(1):107–131, 1969.

[26] G. Rozenberg. Extension of tabled 0L-systems and languages. *International Journal of Computer and Information Sciences*, 2(4):311–336, 1973.

[27] G. Rozenberg and A. K. Salomaa. *The Mathematical Theory of L Systems.* Academic Press, 1980.

[28] A. Salomaa. *Jewels of formal language theory.* Rockville, Md.: Computer Science Pr., 1981.

[29] A. K. Salomaa. *Formal Languages.* Academic Press, 1973.

[30] K. Salomaa. Hierarchy of k-context-free languages. *International Journal of Computer Mathematics*, 26:69–90,193–205, 1989.

[31] E. Stotskii. Управление выводом в формальных грамматиках. Проблемы передачи информации, VII(3):87–102, 1971.

[32] D. Wätjen. k-limited 0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 24(6):267–285, 1988.

[33] D. Wätjen. A weak iteration theorem for k-limited E0L systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 28(1):37–40, 1992.

[34] D. Wätjen. Regulation of k-limited ET0L systems. *International Journal of Computer Mathematics*, 47:29–41, 1993.

[35] D. Wätjen and E. Unruh. On the degree of synchronization of klET0L systems. *Information Processing Letters*, 29:87–89, 1988.

[36] D. Wätjen and E. Unruh. On extended k-uniformly-limited T0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 26(5/6):283–299, 1990.