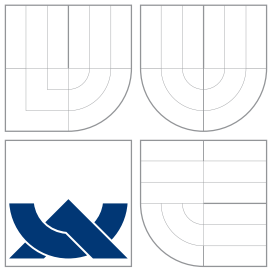
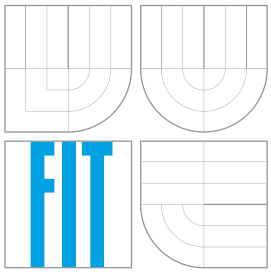


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

O VYMAZÁVACÍCH PRAVIDLECH V ŘÍZENÝCH GRAMATIKÁCH

ON ERASING RULES IN REGULATED GRAMMARS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PETR ZEMEK
AUTHOR

VEDOUCÍ PRÁCE — Prof. RNDr. ALEXANDER MEDUNA, CSc.
SUPERVISOR

BRNO 2010

Zadání diplomové práce/9183/2009/xzemek02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2009/2010

Zadání diplomové práce

Řešitel: **Zemek Petr, Bc.**
Obor: Informační systémy
Téma: **O vymazávacích pravidlech v řízených gramatikách**
On Erasing Rules in Regulated Grammars
Kategorie: Teorie informatiky

Pokyny:

1. Seznamte se s řízenými gramatikami.
2. Stanovte a studujte podmínky, za kterých je možné odstranit zkracující pravidla v těchto gramatikách.
3. Dokažte výsledky studia v předchozím bodu matematicky.
4. Objasněte význam dosažených výsledků pro syntaktickou analýzu.
5. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Meduna, A.: Automata and Languages, Springer, London, 2000
- Rozenberg, G. and Salomaa, A. (eds.): Handbook of Formal Languages, Volume 1 through 3, Springer, 1997, ISBN 3-540-60649-1
- Aho, A. V., Sethi, R., Ullman, J. D. : Compilers : principles, techniques, and tools, Addison-Wesley, 2nd ed., 2007, ISBN: 0321486811

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1-2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Meduna Alexander, prof. RNDr., CSc.**, UIFS FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

V této práci je diskutován vliv vymazávacích pravidel na generativní sílu řízených gramatik, což je velký otevřený problém teorie řízeného přepisování. Tato práce studuje možnost odstranění vymazávacích pravidel z těchto gramatik tak, že shromažďuje aktuální výsledky na toto téma a přináší novou podmínku, nazvanou k-limitované vymazávání, která zaručuje, že jsme bez vlivu na generovaný jazyk schopni odstranit všechna vymazávací pravidla z libovolné bezkontextové gramatiky řízené regulárním jazykem splňující tuto podmínku. Tento výsledek je částečným řešením výše zmíněného problému. Mimoto je prezentován nový algoritmus k odstranění vymazávacích pravidel z bezkontextových gramatik, který nepotřebuje předurčovat tzv. epsilon-neterminály (na rozdíl od standardního algoritmu používaného v učebnicích). V závěru je zhodnocen přínos těchto výsledků pro syntaktickou analýzu.

Abstract

This work discusses the effect of erasing rules to the generative power of regulated grammars, which is a big open problem in the theory of regulated rewriting. It studies the possibility of removal of erasing rules from regulated grammars by aggregation of current, up-to-date results concerning this elimination and by presentation of a new condition, called k-limited erasing, under which all erasing rules can be always removed from regularly controlled context-free grammars without affecting their generative power. This result partially solves the abovementioned problem. Moreover, a new algorithm for elimination of erasing rules from context-free grammars is presented. This algorithm does not require any predetermination of so called epsilon-nonterminals (in contrast to the standard algorithm used in textbooks). In the conclusion, a significance of these results concerning syntactical analysis is discussed.

Klíčová slova

bezkontextová gramatika, řízené gramatiky, bezkontextová gramatika řízená regulárním jazykem, odstraňování vymazávacích pravidel, limitované vymazávání

Keywords

context-free grammar, regulated grammars, regularly controlled context-free grammar, removal of erasing rules, limited erasing

Citace

Petr Zemek: On Erasing Rules in Regulated Grammars, diplomová práce, Brno, FIT VUT v Brně, 2010

On Erasing Rules in Regulated Grammars

Declaration

I hereby declare that this thesis is my own work that has been created under the supervision of prof. Alexander Meduna. Where other sources of information have been used, they have been duly acknowledged.

.....
Petr Zemek
May 4, 2010

Acknowledgements

Sections 4.2 and 5.2 are based on two upcoming papers which have been written jointly with prof. Alexander Meduna. I wish to thank prof. Alexander Meduna for his support during his supervision of this work, for valuable and inspiring consultations, and for his advice and recommendations from which I have benefited greatly. I also wish to thank ing. Jiří Koutný for his comments and suggestions towards my paper published on a student conference, which is a shortened version of Section 5.2. Last, but certainly not least, I wish to thank my parents for their constant support and patience during my work on this thesis.

Contents

Chapter 1

Introduction

Two of the most used and investigated formal models are indisputably regular and context-free grammars, because these types of grammars and the corresponding families of languages have a lot of nice properties [29, 50, 69], which make them very suitable for the purpose of parsing and compilation [3, 54]. However, these types of grammars are not able to cover all aspects which occur in modelling of phenomena by means of formal languages, like some common features of natural and programming languages [44, 57].

Obviously, one can consider using more powerful classical models to handle these aspects, like context-sensitive grammars, but this model is too complex and has some bad properties, like it can not be used as easily as context-free grammars (due to unnatural way how to model using context-sensitive rules [57]) and even some basic problems, like the emptiness problem, are undecidable [66]. Therefore, there has been a tendency to create models based on context-free rules (because they are simple and easy to use), but which have higher generative power than context-free grammars.

As one of the approaches, many regulated grammars were introduced [1, 2, 12, 24, 26, 30, 40, 64, 71, 72]. These grammars are mostly based on context-free rules and to achieve higher generative power, they control the derivation process. For example, it can be controlled by prescribing sequences of rules (if a rule from some sequence is applied, a rule following the applied rule in that sequence has to be used) or by placing context conditions on rules and sentential forms (with a rule, we associate some restrictions for sequential forms that have to be satisfied to apply the rule) [44].

In case of context-free grammars, it is possible to remove all erasing rules, i.e. rules with the empty string on their right-hand side, from any context-free grammar while preserving the generated language [50]. However, whether erasing rules can be eliminated from regulated grammars in general, is an open problem. While this question was answered for some types of regulated grammars (for example, one can remove erasing rules from indexed grammars without affecting their generative power [67]), it still remains unanswered[1] for some other types of regulated grammars, like regularly controlled grammars, matrix grammars, programmed grammars, and random context grammars [44].

[1] In [67, Theorem 2.15], it is claimed that this is impossible in case of regularly controlled grammars, matrix grammars, and programmed grammars. However, the given references do not contain a proof for this and in more recent publications, like [44] and [76], this is still considered to be an open problem.

The goals of this work are (1) to present current, up-to-date results concerning the exact effect of erasing rules to the generative power of regulated grammars and the possibility of elimination of such rules from these grammars, (2) to study new results in this area, and (3) to discuss a significance of these results concerning syntactical analysis. Thereby, this work contributes to the theory of regulated rewriting, which is an important field of the formal language theory.

1.1 Chapter Survey

This work is organized as follows. First, Chapter 2 gives preliminaries and basic definitions, including used notation. Then, in Chapter 3, some classical types of regulated grammars are defined along with their modifications. These grammars are of our special interest. Also, their generative power with regard to the presence or absence of erasing rules is mentioned in there.

Since many regulated grammars are based on context-free grammars, Chapter 4 reviews the standard algorithm for elimination of erasing rules from context-free grammars, which is based on a predetermination of so-called ε-nonterminals. It then presents a new algorithm for this task that does not need this predetermination. It formally verifies the new algorithm.

Chapter 5 first discusses why is the removal of erasing rules from regulated grammars different from the removal of erasing rules from context-free grammars. Then, it presents known results regarding elimination of erasing rules from regulated grammars. Apart from these results, it presents a new result: a condition, called k-limited erasing, under which all erasing rules can be always removed from regularly controlled grammars without affecting their generative power. This elimination is presented in the form of a transformation whose correctness is formally verified. In the last section of this chapter, a significance of these results concerning syntactical analysis is discussed.

In the conclusion of this work, given in Chapter 6, obtained results are summarized, some final remarks are made, and possible further research is discussed. It also states several open problems closely related to this work.

Relation to the Term Project

As this work is a direct continuation of my term project, the following parts of the term project are (in a modified and corrected form) used in this work:

- First part of Chapter 1,
- Sections 2.1, 2.2, and 2.3 in Chapter 2,
- Section 3.1 in Chapter 3,
- Chapter 4 (without the verification of correctness of Algorithm 4.1.2),
- Introduction to Chapter 5 and Section 5.2 (without the verification of correctness of Algorithm 5.2.1 and without examples).

Chapter 2

Preliminaries and Basic Definitions

In this chapter, the used terminology, notation, and fundamental terms from the area of mathematics and formal language theory are reviewed. However, the reader is assumed to have basic mathematical knowledge regarding elementary algebra and proof techniques, such as a proof by induction. In particular, Section 2.1 reviews sets, relations, functions, and closures. Then, in Section 2.2, the meaning of basic elements of the formal language theory is given, such as an alphabet, strings, languages, and operations over them. Section 2.3 reviews the basics of grammars, derivations, and classical language families. Finally, Section 2.4 describes a graphical representation of the structure of derivations—derivation trees. It also establishes a notation to simplify definitions and proofs. This chapter is based on [41, 50, 58, 66, 69].

2.1 Sets, Relations, Closures, and Functions

A *set*, Q, is a collection of differentiable elements taken from some *universe*, $\mathbb{U}$, without any structure other than membership. To indicate that x is an element (a *member*) of Q, we write $x \in Q$. The statement that x is not in Q is written as $x \notin Q$. If Q has a finite number of members, then Q is a *finite set*. Otherwise, Q is an *infinite set*. The set that has no members is the *empty set*, denoted by $\emptyset$. The *cardinality* of Q, denoted[1] by $card(Q)$, is, for finite sets, the number of members of Q. If Q is infinite, then we assume $card(Q) > k$ for every integer, k. Note that $card(\emptyset) = 0$.

Sets can specified by enclosing some description of its elements in curly brackets; for example, the set Q of three consecutive integers ($1, 2$ and 3) is denoted by

$$Q = \{0, 1, 2\}.$$

Ellipses can be used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lower-case letters of the English alphabet. When the need arises, we use more explicit notation, in which a set, Q, is specified by a property, σ, so Q contains all elements satisfying σ.

[1] In the literature, there is an alternative denotation of the cardinality of a set: $|Q|$. In this work, this notation is exclusively used to denote the length of a sequence.

This specification has the following format:

$$Q = \{x \mid \sigma(x)\}.$$

Let $\mathbb{N}$ denote the set of all nonnegative integers. Then, for example, the set of all even nonnegative integers can be defined as $\mathbb{N}_{even} = \{i \mid i \in \mathbb{N}, i \text{ is even}\}$.

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$), whose definition is given next.

Definition 2.1.1. Let Q_1 and Q_2 be two sets. Then

$$\begin{aligned} Q_1 \cup Q_2 &= \{x \mid x \in Q_1 \text{ or } x \in Q_2\}, \\ Q_1 \cap Q_2 &= \{x \mid x \in Q_1 \text{ and } x \in Q_2\}, \\ Q_1 - Q_2 &= \{x \mid x \in Q_1 \text{ and } x \notin Q_2\}. \end{aligned}$$

For n sets, $Q_1, Q_2, \ldots, Q_n$, instead of $Q_1 \cup Q_2 \cup \cdots \cup Q_n$ and $Q_1 \cap Q_2 \cap \cdots \cap Q_n$, we usually write $\bigcup_{1 \le i \le n} Q_i$ and $\bigcap_{1 \le i \le n} Q_i$, respectively. If there are infinitely many sets, we omit the upper bound, n.

Definition 2.1.2. Let Q be a set. A set, P, is said to be a *subset* of Q if every element of P is also an element of Q; we write this as

$$P \subseteq Q.$$

If $P \subseteq Q$, but Q contains one or more elements that are not in P, we say that P is a *proper subset* of Q; this is written as

$$P \subset Q.$$

If two sets, Q_1 and Q_2, have no common element, then they are said to be *disjoint*. If $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$, then Q_1 and Q_2 are said to be *identical* and we write $Q_1 = Q_2$ (they *coincide*). Otherwise, they are said to be *nonidentical* and we write $Q_1 \neq Q_2$. Using intersection and identity, we can write $Q_1 \cap Q_2 = \emptyset$ to indicate that Q_1 and Q_2 are disjoint.

Definition 2.1.3. Let Q be a set. The *power set* of Q, denoted by 2^Q, is the set of all subsets of Q. In symbols,

$$2^Q = \{U \mid U \subseteq Q\}.$$

Observe that 2^Q is a set of sets and sets of this kind are customarily called *families* of sets, rather than sets of sets.

For a finite set, $Q \subseteq \mathbb{N}$, let $max(Q)$ denote the smallest integer, m, such that m is greater or equal to all other members of Q. Similarly, $min(Q)$ denotes the greatest integer, n, such that n is lesser or equal to all other members of Q.

An *(ordered) sequence* is a list of elements. Contrary to a set, a sequence can contain an element more than once and the elements appear in a certain order. Elements in sequences are usually separated by a comma. As sets, sequences can be either *finite* or *infinite*. Finite sequences are also called *tuples*. More specifically, sequences of two, three, and four elements are called *pairs*, *triplets*, and *quadruples*, respectively.

Definition 2.1.4. A *Cartesian product* of two sets, Q_1 and Q_2, denoted by $Q_1 \times Q_2$, is a set of pairs defined as

$$Q_1 \times Q_2 = \{(x_1, x_2) \mid x_1 \in Q_1 \text{ and } x_2 \in Q_2\}.$$

Definition 2.1.5. A *binary relation* (or, simply, a *relation*), ρ, from a set, Q_1, to a set, Q_2, is any subset of their Cartesian product. That is,

$$\rho \subseteq Q_1 \times Q_2.$$

Instead of $(x, y) \in \rho$, we often write $x\rho y$; in other words, $(x, y) \in \rho$ and $x\rho y$ are used interchangeably. If $Q_1 = Q_2$, then we say that ρ is a *relation on* Q_1 or *relation over* Q_1. As relations are sets, all common operations over sets apply to relations as well. For a relation, ρ, if ρ is a finite set, then ρ is a *finite relation*; otherwise, ρ is an *infinite relation*.

Definition 2.1.6. Let ρ be a relation over a set, Q. ρ is a *partial order* if for all $x, y, z \in Q$, ρ satisfies the following three conditions:

(1)	$x\rho x$,	(*reflexivity*)
(2)	if $x\rho y$ and $y\rho x$, then $x = y$,	(*antisymmetry*)
(3)	if $x\rho y$ and $y\rho z$, then $x\rho z$.	(*transitivity*)

Definition 2.1.7. Let ρ be a relation over a set, Q. For every $k \geq 1$, the *k-fold product* of ρ, denoted by ρ^k, is recursively defined as

(1) $x\rho^1 y$ if and only if $x\rho y$,
(2) $x\rho^k y$ if and only if $x\rho z$ and $z\rho^{k-1} y$ for some $z \in Q$.

Definition 2.1.8. Let ρ be a relation over a set, Q. The *transitive closure* of ρ, denoted by ρ^+, is defined as $x\rho^+ y$ if and only if $x\rho^k y$ for some $k \geq 1$, and the *reflexive and transitive closure* of ρ, denoted by ρ^*, is defined as $x\rho^* y$ if and only if $x\rho^+ y$ or $x = y$.

Definition 2.1.9. A *function* (*mapping*), ψ, from Q_1 to Q_2, denoted by $\psi\colon Q_1 \to Q_2$, is a relation from Q_1 to Q_2 such that for every $x \in Q_1$,

$$card(\{y \mid y \in Q_2 \text{ and } (x, y) \in \psi\}) \leq 1.$$

If for every $y \in Q_2, card(\{x \mid x \in Q_1 \text{ and } (x, y) \in \psi\}) \leq 1$, ψ is an *injection*. If for every $y \in Q_2, card(\{x \mid x \in Q_1 \text{ and } (x, y) \in \psi\}) \geq 1$, ψ is a *surjection*. If ψ is both an injection and a surjection, ψ represents a *bijection*.

Definition 2.1.10. Let $\psi\colon Q_1 \to Q_2$ be a function. The *domain* of ψ, denoted by $domain(\psi)$, and the *range* of ψ, denoted by $range(\psi)$, are defined as

$$domain(\psi) = \{x \mid x \in Q_1 \text{ and } (x, y) \in \psi \text{ for some } y \in Q_2\}$$

and

$$range(\psi) = \{y \mid y \in Q_2 \text{ and } (x, y) \in \psi \text{ for some } x \in Q_1\}.$$

If $domain(\psi) = Q_1$, ψ is *total*; otherwise, ψ is *partial*. Instead of $(x, y) \in \psi$, we usually write $\psi(x) = y$.

2.2 Alphabet, Strings, and Languages

Definition 2.2.1. An *alphabet*, Σ, is a finite, nonempty set of elements called *symbols*.

Definition 2.2.2. Let Σ be an alphabet. A *string* (a *word*) over Σ is any finite sequence of symbols from Σ.

We omit all separating commas in strings; that is, for a string, $a_1, a_2, \ldots, a_n$, for some $n \geq 1$, we write $a_1a_2 \ldots a_n$ instead. The *empty string*, denoted by ε, is the string that is formed by no symbols, i.e. the empty sequence. Σ^* denotes the set of all strings over Σ (including ε). Set $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. Let x be a string over Σ, i.e. $x \in \Sigma^*$, and express x as $= a_1a_2 \ldots a_n$, where $a_i \in \Sigma$, for all $1 \leq i \leq n$, for some $n \geq 0$ ($x = \varepsilon$ if and only if $n = 0$). Then, $|w| = n$ denotes the *length* of x and $alph(x) = \{a_1, a_2, \ldots, a_n\}$ denotes the set of symbols occurring in x (note that $|\varepsilon| = 0$ and $alph(\varepsilon) = \emptyset$).

Definition 2.2.3. Let x and y be two strings over an alphabet, Σ. Then, xy is the *concatenation* of x and y.

Note that $x\varepsilon = \varepsilon x = x$.

Definition 2.2.4. Let x be a string over an alphabet, Σ. If x can be written in the form $x = uv$ for some strings $u, v \in \Sigma^*$, then u is called a *prefix* of x and v is called a *suffix* of x. If $0 < |u| < |x|$, then u is called a *proper prefix* of x; similarly, if $0 < |v| < |x|$, then v is called a *proper suffix* of x.

Definition 2.2.5. Let n be a nonnegative integer and x be a string over an alphabet, Σ. Then, the *nth power* of x, denoted by x^n, is a string over Σ recursively defined as

$$\begin{array}{ll} (1) & x^0 = \varepsilon, \\ (2) & x^n = xx^{n-1}. \end{array}$$

Definition 2.2.6. Let $x = a_1a_2 \ldots a_n$ be a string over Σ for some $n \geq 0$. The set of all *permutations* of x, $perm(x)$, is defined as

$$\begin{array}{c} perm(x) = \{b_1b_2 \ldots b_n \mid b_i \in alph(x), \text{ for all } 1 \leq i \leq n, \text{ and} \\ (b_1, b_2, \ldots, b_n) \text{ is a permutation of } (a_1, a_2, \ldots, a_n)\}. \end{array}$$

Note that $perm(\varepsilon) = \varepsilon$. Now, the important notion of a language is introduced.

Definition 2.2.7. Let Σ be an alphabet. A *language*, L, over Σ is any set of strings over Σ, i.e. $L \subseteq \Sigma^*$.

Σ^* is called the *universal language*, because it consists of all strings over Σ. If L is a finite set, then it is a *finite language*; otherwise, it is an *infinite language*. The *empty language* is denoted by $\emptyset$. As all languages are sets, all common operations over sets can be applied to them.

Definition 2.2.8. Let L_1 and L_2 be two languages over Σ. Then

$$\begin{array}{rcl} L_1 \cup L_2 & = & \{x \mid x \in L_1 \text{ or } x \in L_2\}, \\ L_1 \cap L_2 & = & \{x \mid x \in L_1 \text{ and } x \in L_2\}, \\ L_1 - L_2 & = & \{x \mid x \in L_1 \text{ and } x \notin L_2\}. \end{array}$$

There are also some special operations which apply only to languages.

Definition 2.2.9. Let L_1 and L_2 be two languages over Σ. The *concatenation* of L_1 and L_2, denoted by L_1L_2, is the set

$$L_1L_2 = \{x_1x_2 \mid x_1 \in L_1 \text{ and } x_2 \in L_2\}.$$

Note that $L\{\varepsilon\} = \{\varepsilon\}L = L$.

Definition 2.2.10. For a nonnegative integer, n, and a language, L, the *nth power* of L, denoted by L^n, is recursively defined as

$$\begin{array}{ll} (1) & L^0 = \{\varepsilon\}, \\ (2) & L^n = L^{n-1}L. \end{array}$$

Definition 2.2.11. The *star* (*Kleene closure*) of a language, L, denoted by L^*, is the set

$$L^* = \bigcup_{i \geq 0} L^i.$$

Definition 2.2.12. The *positive closure* of a language, L, denoted by L^+, is the set

$$L^+ = \bigcup_{i \geq 1} L^i.$$

Note that $L^+ = L^* - \{\varepsilon\}$ if and only if $\varepsilon \notin L$.

Definition 2.2.13. Let Σ and Γ be two alphabets and let $\varphi\colon \Sigma^* \to 2^{\Gamma^*}$ be a function. φ is said to be a *substitution* if it satisfies the following two conditions:

$$\begin{array}{ll} (1) & \varphi(\varepsilon) = \{\varepsilon\}, \\ (2) & \varphi(xy) = \varphi(x)\varphi(y), \text{ for every } x, y \in \Sigma^*. \end{array}$$

φ is said to be *regular* if $\varphi(a)$ is a regular language (see Section 2.3), for all $a \in \Sigma$.

Definition 2.2.14. Let Σ and Γ be two alphabets and let $\varphi\colon \Sigma^* \to \Gamma^*$ be a function. φ is said to be a *homomorphism* if it satisfies the following two conditions:

$$\begin{array}{ll} (1) & \varphi(\varepsilon) = \varepsilon, \\ (2) & \varphi(xy) = \varphi(x)\varphi(y), \text{ for every } x, y \in \Sigma^*. \end{array}$$

Note that a homomorphism is a special case of a substitution where each substituted language contains only one string.

2.3 Grammars and Language Families

In the theory of formal languages, the basic model for the description of a language is a grammar.

Definition 2.3.1. An *(unrestricted) grammar*, G, is a quadruple,

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

where

- N is an alphabet of *nonterminals*,
- T is an alphabet of *terminals* such that $N \cap T = \emptyset$,
- $S \in N$ is the *start nonterminal*, and
- P is a finite relation from $(N \cup T)^* N (N \cup T)^*$ to $(N \cup T)^*$.

Pairs $(u, v) \in P$ are called *productions* or *rewriting rules* (abbreviated *rules*) and are written as $u \to v$. Accordingly, P is called the *set of rules*. $V = N \cup T$ is the *total alphabet* of G. A rewriting rule $u \to v \in P$, where $v = \varepsilon$, is called an *erasing rule*. If there is no such rule in P, then we say that G is an *ε-free* (or *non-erasing*) grammar.

Definition 2.3.2. Let $G = (N, T, S, P)$ be a grammar. The *direct derivation* relation induced by G is a relation between strings over V, denoted by $\Rightarrow_G$, and defined as

$$x \Rightarrow_G y,$$

if and only if $x = x_1 u x_2, y = y_1 v y_2$, and $u \to v \in P$, where $x_1, x_2, y_1, y_2 \in V^*$.

Since $\Rightarrow_G$ is a relation, $\Rightarrow_G^k$ is the k-fold product of $\Rightarrow_G$ for $k \geq 1$, $\Rightarrow_G^+$ is the transitive closure of $\Rightarrow_G$, and $\Rightarrow_G^*$ is the reflexive and transitive closure of $\Rightarrow_G$. Furthermore, we extend $\Rightarrow_G^k$ to $k = 0$: $x \Rightarrow_G^0 y$ if and only if $x = y$. If $S \Rightarrow_G^* x$ for some $x \in V^*$, x is called a *sentential form*. If $x \in T^*$, then x is called a *sentence*.

Definition 2.3.3. Let $G = (N, T, S, P)$ be a grammar. The *language generated by G*, denoted by $L(G)$, is the set of all sentences defined as

$$L(G) = \{w \mid w \in T^*, S \Rightarrow_G^* w\}.$$

Definition 2.3.4. Let G and H be two grammars. If $L(G) - \{\varepsilon\} = L(H) - \{\varepsilon\}$, then we consider G and H to be *equivalent grammars*.

When no confusion exists, we simplify $x \Rightarrow_G y$ to $x \Rightarrow y$. If P contains rules $u \to v_1$, $u \to v_2$, $\ldots$, $u \to v_n$ for some $n \geq 1$, we use a more condensed notation $u \to v_1 \mid v_2 \mid \ldots \mid v_n$.

For brevity, we often denote $u \to v$ with a unique label, r, as $r\colon u \to v$, and instead of $u \to v \in P$, we simply write $r \in P$. For a rule, $r\colon u \to v \in P$, u and v represent the *left-hand side* of r, denoted by $lhs(r)$, and the *right-hand side* of r, denoted by $rhs(r)$, respectively. The notion of rule labels is formalized in the following definitions.

Definition 2.3.5. Let $G = (N, T, S, P)$ be a grammar. Let Ψ be a set of symbols called *rule labels* such that $card(\Psi) = card(P)$, and ψ be a bijection from P to Ψ.

For simplicity and brevity, to express that ψ maps a rule, $u \to v \in P$, to r, where $r \in \Psi$, we write $r\colon u \to v \in P$; in other words, $r\colon u \to v$ means $\psi(u \to v) = r$. Let P^* and Ψ^* denote the set of all sequences of rules from P and the set of all sequences of rule labels from Ψ, respectively. Set $P^+ = P^* - \{\varepsilon\}$ and $\Psi^+ = \Psi^* - \{\varepsilon\}$. As with strings, we omit all separating commas in these sequences.

Definition 2.3.6. Let $G = (N, T, S, P)$ be a grammar and Ψ be its set of rule labels. We extend ψ from P^* to Ψ^* as

$$\begin{array}{ll} (1) & \psi(\varepsilon) = \varepsilon, \\ (2) & \psi(r_1 r_2 \dots r_n) = \psi(r_1)\psi(r_2)\dots\psi(r_n), \end{array}$$

for any sequence of rules, $r_1 r_2 \dots r_n$, where $r_i \in P$, for all $1 \leq i \leq n$, for some $n \geq 1$.

Let $w_0, w_1, \dots, w_n$ be a sequence of strings, where $w_i \in V^*$, for all $0 \leq i \leq n$, for some $n \geq 0$. If $w_{j-1} \Rightarrow w_j$ in G according to a rule, $r_j \in P$, for $1 \leq j \leq n$, then we write $w_0 \Rightarrow^* w_n\ [\psi(r_1 r_2 \dots r_n)]$ ($w_0 \Rightarrow^* w_0\ [\varepsilon]$ if $n = 0$). $\psi(r_1 r_2 \dots r_n)$ is called the *sequence of rules (rule labels) used in the derivation of* w_n, or, more briefly, the *parse*[2] *of* w_n.

For any grammar, G, we automatically assume that V, N, T, S, P, and Ψ denote its total alphabet, the alphabet of nonterminal symbols, the alphabet of terminal symbols, the start symbol, the set of rules, and the set of rule labels, respectively. If there exists a danger of confusion, we mark V, N, T, S, P, Ψ with G as V_G, N_G, T_G, S_G, P_G, Ψ_G, respectively, to clearly relate these components to G (in particular, we make these marks when several grammars are simultaneously discussed).

Chomsky Hierarchy of Language Families

Noam Chomsky, the founder of the formal language theory, provided an initial classification of grammars according to the language families they generate, denoted by type 0 through type 3 [7]. Each grammar type has a different restriction placed on its set of rules.

Definition 2.3.7. A *phrase-structure (type* 0*) grammar* is a grammar,

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

with no additional restriction placed on P.

The family of all languages generated by this type of a grammar is the family of all *recursively-enumerable languages*, denoted by **RE**.

Definition 2.3.8. A *context-sensitive (type* 1*) grammar* is a grammar,

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

[2]Let us note that the notion of a parse represents a synonym to some other notions, including a *derivation word*, a *Szilard word*, and a *control word* [57].

such that each rule, $u \to v \in P$, satisfies either

$$u = x_1 A x_2,\ v = x_1 y x_2, \text{ where } x_1, x_2 \in V^*,\ A \in N,\ y \in V^+,$$

or

$$u = S,\ v = \varepsilon, \text{ and } S \text{ does not appear on any right-hand side of any rule.}$$

The family of all languages generated by this type of a grammar is the family of all *context-sensitive languages*, denoted by **CS**.

Definition 2.3.9. A *context-free (type 2) grammar* is a grammar,

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

such that each rule $u \to v \in P$ satisfies $u \in N$.

The family of all languages generated by this type of a grammar is the family of all *context-free languages*, denoted by **CF**.

Definition 2.3.10. A *regular (type 3) grammar* is a grammar,

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

such that each rule $u \to v \in P$ satisfies $u \in N$ and $v \in T(N \cup \{\varepsilon\})$.

The family of all languages generated by this type of a grammar is the family of all *regular languages*, denoted by **REG**.

For the families of languages generated by regular, context-free, context-sensitive, and phrase-structure grammars, respectively, the following theorem holds.

Theorem 2.3.1. *(See [50, Theorem 8.4.1])* $\boldsymbol{REG} \subset \boldsymbol{CF} \subset \boldsymbol{CS} \subset \boldsymbol{RE}$.

Thus, each language family of type i is a proper subset of the family of type $i - 1$.

A Note on the Presence of the Empty String

As outlined in Definition 2.3.4, ε-free grammars are not able to generate the empty string. To enable an ε-free grammar to generate the empty string, sometimes a single rule of the form $S \to \varepsilon$ is allowed (S then cannot appear on the right-hand side of any rule, as in Definition 2.3.8). Another approach is to state that the generated language is equal up to the presence of the empty string—this approach is used in this work.

For this reason, we consider two languages to be equal if and only if they differ only in the presence of the empty string. Inclusions between languages and between language families are understood in the same manner. However, sometimes the absence of the empty string is explicitly expressed to emphasize this fact.

2.4 Derivation Trees

A derivation tree graphically represents the structure of a derivation in a context-free grammar and suppresses the order in which individual rules are used (in case of context-free grammars, this piece of information is often irrelevant[3]). To define derivation trees, we first review the needed basics of the graph theory (for more information, see [17]).

Definition 2.4.1. Let V be a finite set. A *directed graph* is a pair, $G = (V, \rho)$, where ρ is a relation over V.

For brevity, we omit the adjective "directed" and write just *graph*. Members of V are called *nodes* and pairs in ρ are called *edges*. If $e = (a, b) \in \rho$, then e *leaves* node a and *enters* node b; at this point, a is a *direct predecessor* of b and b is a *direct descendant* of a. A sequence of nodes, $a_0, a_1, \ldots, a_n$, where $n \geq 1$, forms a *walk* from a_0 to a_n if $(a_{i-1}, a_i) \in \rho$ for all $1 \leq i \leq n$; if, in addition, $a_0 = a_n$, then $a_0, a_1, \ldots, a_n$ is a *cycle*. If there is no cycle in G, then G is an *acyclic graph*.

Definition 2.4.2. Let Q be a nonempty set. An *ordered labelled tree* is an acyclic graph, $G = (V, \rho)$, satisfying the following four conditions:

(1) there is exactly one specific node, called the *root*, which no edge enters,

(2) for each node $a \in V$ other than the root, there is exactly one walk from the root to a,

(3) every node is labelled with a member of Q (there is a total function from V to Q), and

(4) each node $a \in V$ has its direct descendants, $b_1, b_2, \ldots, b_n$, ordered from the left to the right, so b_1 is the leftmost descendant of a and b_n is the rightmost descendant of a.

As with graphs, for brevity, we omit both adjectives and write just *tree*. If there is no danger of confusion, we use nodes and their labels interchangeably. Let $G = (V, \rho)$ be a tree and $a \in V$ a node. If no edges leaves a, then a is a *leaf*. The *frontier* of G is the sequence of G's leaves ordered from the left to the right.

Definition 2.4.3. Let $G = (V, \rho)$ be a tree. A tree, $G' = (V', \rho')$, is a *subtree* of G if it satisfies these three conditions:

(1) $V' \subseteq V$ and $V' \neq \emptyset$,

(2) $\rho' = (V' \times V') \cap \rho$, and

(3) in G, no node in $V - V'$ is a direct descendant of a node in V'.

Let $G = (V, \rho)$ be a tree, $G' = (V', \rho')$ be its subtree, and $r \in V'$ be the root of G'. Then we say that G' is *rooted* at r.

Definition 2.4.4. Let $G = (N, T, S, P)$ be a context-free grammar and $S \Rightarrow^* w$ be a derivation of the form $S = w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n = w$, where $w_i \in V^*$, for all $1 \leq i \leq n$, for some $n \geq 1$. A *derivation tree* corresponding to this derivation is denoted by $\Delta(S \Rightarrow^* w)$ and defined as a tree with the following properties:

[3] One notable exception is syntactical analysis [54].

(1) nodes of the derivation tree are labelled with members of $V \cup \{\varepsilon\}$,

(2) the root of the derivation tree is labelled by S,

(3) for a direct derivation $w_{i-1} \Rightarrow w_i$, for all $1 \leq i \leq n$, where

- $w_{i-1} = xAz$, $x, z \in V^*$, $A \in N$,
- $w_i = xyz$, and
- $A \to y \in P$, where $y = Y_1 Y_2 \dots Y_k$, $Y_j \in N$, for all $1 \leq j \leq k$, for some $k \geq 0$ (if $k = 0$, then $y = \varepsilon$),

if $y \neq \varepsilon$, then there are exactly k edges, (A, Y_j), $1 \leq j \leq k$, leaving A, which are ordered from the left to the right in order $(A, Y_1), (A, Y_2), \dots, (A, Y_k)$. If $y = \varepsilon$, then there is only one edge leaving A, (A, ε).

A derivation subtree whose frontier is ε is called an *ε-subtree*. A derivation subtree whose frontier is different from ε is called a *+-subtree*.

Example 2.4.1. Let $G = (N, T, S, P)$ be a context-free grammar, where $N = \{S, A, B, C\}$, $T = \{a, b, c, d\}$ and P consists of the following rules:

$S \to ABC$,

$A \to a \mid \varepsilon$,

$B \to BbAc \mid CC$,

$C \to Cd \mid d \mid \varepsilon$.

Let $S \Rightarrow ABC \Rightarrow ABbAcC \Rightarrow ABbcC \Rightarrow ACCbcC \Rightarrow ACbcC \Rightarrow AbcC$ be a derivation in G. The corresponding derivation tree for this derivation is pictured in Figure 2.1.

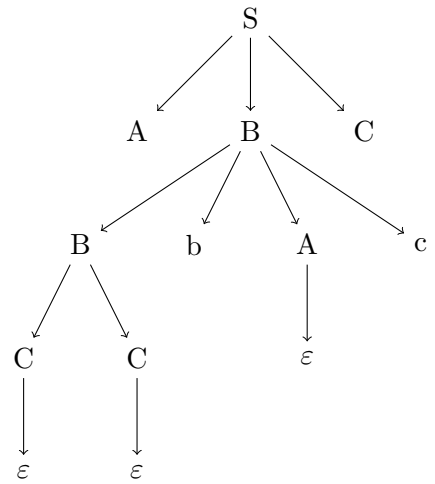


Figure 2.1: Graphical Representation of the Derivation From Example 2.4.1.

$\Delta(A \Rightarrow \varepsilon)$ and $\Delta(B \Rightarrow CC \Rightarrow C \Rightarrow \varepsilon)$ are two examples of ε-subtrees. $\Delta(B \Rightarrow BbAc \Rightarrow Bbc \Rightarrow CCbc \Rightarrow Cbc \Rightarrow bc)$ is an example of a +-subtree.

To simplify a definition in Section 5.1.1 and to make proofs in Sections 4.2 and 5.2 more readable, we formally introduce the following notation. This notation is then referenced from every place where it is used.

Notation 2.4.1. Let $G = (N, T, S, P)$ be a context-free grammar. Let $S \Rightarrow^* w$ in G be of the form $S \Rightarrow^* x_0X_1x_1X_2x_2 \dots X_nx_n \Rightarrow^* w$, where $x_i \in N^*$, $X_j \in V^*$, and $w \in T^*$, for all $0 \leq i \leq n$, $1 \leq j \leq n$, for some $n \geq 0$. We write $S \Rightarrow^* {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_n{}^{\varepsilon}x_n \Rightarrow^* w$ to express that

(1) either $x_i = \varepsilon$ or the $|x_i|$ subtrees in $\Delta(S \Rightarrow^* w)$ rooted at the symbols in x_i are all ε-subtrees (informally, it means that x_i is erased in the rest of the derivation),

(2) either $X_j \in T$ or $X_j \in N$ and there is a $+$-subtree rooted at X_j in $\Delta(S \Rightarrow^* w)$ (informally, it means that X_j is not erased in the rest of the derivation),

for all $0 \leq i \leq n$, $1 \leq j \leq n$.

This notation in also used in rules, where it has the same meaning. For example, a rule of the form $A \to {}^{\varepsilon}xB^{\varepsilon}y$, where $x, y \in V^*$ and $B \in N$, indicates that x and y are erased in the rest of the derivation and B is not erased. If we omit w from $S \Rightarrow^* {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_n{}^{\varepsilon}x_n \Rightarrow^* w$, as in $S \Rightarrow^* {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_n{}^{\varepsilon}x_n$, we automatically assume that there is one (we care only about the information which symbols are erased and which are not erased in the rest of the derivation).

Chapter 3

Regulated Grammars

The main idea behind regulated rewriting is to pick a simple model, which is not powerful enough (in terms of generative power), and somehow regulate the derivation process to increase its power. For this reason, regulated grammars are also known as "grammars with controlled derivations" [67]. Regulated grammars presented in this chapter use context-free grammars as the underlying model, but there also have been studies of other types of underlying models, see [19, 35, 74] and [11, Chapter 8].

This chapter defines different types of regulated grammars, including their modifications and their generative power. Since there are many regulated grammars, only those that are of our special interest are defined or mentioned (for other regulated grammars, refer to [11], [44], [67], or [69, Chapter 5]). Definitions of these grammars are roughly based on [44] and [67] and are defined in a unified way as context-free grammars with additional regulations. Bibliographical and historical remarks are mostly adopted from [67] and [69, Chapter 5].

Note: Majority of regulated grammars discussed in this chapter were introduced by their respective authors in special forms (for example, a programmed grammar, as it was originally introduced in [64], required that a rule is applied to the leftmost occurrence of its left-hand side). However, all models have been eventually studied in more general forms, like with the possibility of rewriting any occurrence of a nonterminal in a sentential form, with the presence of erasing rules, and with appearance checking (discussed later in this chapter). For brevity, regulated grammars in this chapter are presented in their most general forms and special cases are derived from these general definitions (except for regularly controlled grammars, where, to introduce and illustrate new concepts, we go in the opposite direction).

3.1 Regularly Controlled Grammar

First type of regulation is based on the prescription of a set of allowed sequences of rules. A generated sentence, x, belongs to the generated language only if there is a derivation of x matching a sequence from this set. In other words, we restrict the order in which rules can be applied. Note that in context-free grammars, there is no restriction on the order of rules. Every rule that is applicable can be applied. The idea of this type of regulation was introduced by Ginsburg and Spanier in [25].

Definition 3.1.1. A *regularly controlled (context-free) grammar*, see [44], is a pair,

$$H = (G, \Xi),$$

where

- $G = (N, T, S, P)$ is a context-free grammar, called *core grammar*, and
- $\Xi \subseteq \Psi^*$ is a regular language, called *control language*.

The *language generated by* H, denoted by $L(H)$, is defined as

$$L(H) = \{w \mid w \in T^*, S \Rightarrow^* w \ [\alpha] \text{ with } \alpha \in \Xi\}.$$

In other words, $L(H)$ consists of all strings $w \in T^*$ such that there is a derivation,

$$S \Rightarrow w_1 \ [r_1] \Rightarrow w_2 \ [r_2] \Rightarrow \ldots \Rightarrow w_n \ [r_n],$$

with

$$w = w_n \text{ and } r_1 r_2 \ldots r_n \in \Xi, \text{ for some } n \geq 1.$$

Note that if $\Xi = \Psi^*$, then there is no regulation, and thus $L(H) = L(G)$ in this case.

Example 3.1.1. Let $H = (G, \Xi)$ be a regularly controlled grammar, where $G = (N, T, S, P)$ is a context-free grammar with $N = \{S, A, B, C\}$, $T = \{a, b, c\}$, P consists of the following rules:

$$r_1 \colon S \to ABC, \qquad r_2 \colon A \to aA, \qquad r_5 \colon A \to \varepsilon,$$
$$r_3 \colon B \to bB, \qquad r_6 \colon B \to \varepsilon,$$
$$r_4 \colon C \to cC, \qquad r_7 \colon C \to \varepsilon,$$

and $\Xi = \{r_1\}\{r_2 r_3 r_4\}^*\{r_5 r_6 r_7\}$.

First, r_1 has to be applied. Then, r_2, r_3, and r_4 can be sequentially applied any number of times. The derivation is finished by applying r_5, r_6, and r_7. As a result, this grammar generates the language $L(H) = \{a^n b^n c^n \mid n \geq 0\}$, which is not context-free [50, Example 6.1.2].

For example, the sentence *aabbcc* is obtained by the following derivation:

$$S \Rightarrow ABC \ [r_1] \Rightarrow aABC \ [r_2] \Rightarrow aAbBC \ [r_3] \Rightarrow aAbBcC \ [r_4] \Rightarrow aaAbBcC \ [r_2] \Rightarrow$$
$$aaAbbBcC \ [r_3] \Rightarrow aaAbbBccC \ [r_4] \Rightarrow aabbBccC \ [r_5] \Rightarrow aabbccC \ [r_6] \Rightarrow aabbcc \ [r_7].$$

Now, let $H = (G, \Xi)$ be an arbitrary regularly controlled grammar and assume that we want to apply some sequence of rules, $s_1 s_2 \ldots s_n \in \Xi$, for some $n \geq 1$, and we have successfully applied rules $s_1 s_2 \ldots s_k$ for some $k < n$. What happens when there is no rewritable nonterminal in the current sentential form, i.e. $s_{k+1} \colon D \to y$, but there is no D in the sentential form? The derivation is blocked. And, according to the definition of a regularly controlled grammar, there is nothing we can do. We are forced to use the rule s_{k+1} independently of the symbols appearing in the sentential form. This brings us to the idea of *appearance checking*.

Definition 3.1.2. A *regularly controlled grammar with appearance checking*, see [44], is a triplet,

$$H = (G, \Xi, F),$$

where

- G and Ξ are defined as in a regularly controlled grammar and
- $F \subseteq \Psi$ is the *appearance checking set.*

We say that $x \in V^+$ directly derives $y \in V^*$ in *appearance checking mode* by application of $r\colon A \to w \in P$, written as $x \Rightarrow_{ac} y\ [r]$, if one of the following conditions hold:

$$x = x_1 A x_2 \text{ and } y = x_1 w x_2$$

or

$$A \notin alph(x), \text{ i.e. } A \text{ does not appear in } x,\ r \in F, \text{ and } x = y.$$

Let $\Rightarrow_{ac}^*$ be the reflexive and transitive closure of $\Rightarrow_{ac}$. The *language generated by* H, denoted by $L(H)$, is defined as

$$L(H) = \{w \mid w \in T^*, S \Rightarrow_{ac}^* w\ [\alpha] \text{ with } \alpha \in \Xi\}.$$

So, the only difference between a regularly controlled grammar with and without appearance checking is the derivation mode ($\Rightarrow$ versus $\Rightarrow_{ac}$). Note that when $F = \emptyset$, these two modes coincides, so a regularly controlled grammar (without appearance checking) is a special case of a regularly controlled grammar with appearance checking.

Example 3.1.2. (From [67]) Let $H = (G, \Xi, F)$ be a regularly controlled grammar with appearance checking, where $G = (N, T, S, P)$ is a context-free grammar with $N = \{S, A, X\}$, $T = \{a\}$, P consists of the following rules:

$r_1\colon S \to AA$, $\quad r_4\colon A \to X$,

$r_2\colon S \to X$, $\quad r_5\colon S \to a$,

$r_3\colon A \to S$,

$\Xi = (\{r_1\}^*\{r_2\}\{r_3\}^*\{r_4\})^*\{r_5\}^*$, and $F = \{r_2, r_4\}$.

Assume that we have the sentential form S^{2^m} for some $m \geq 0$, obtained by using a sequence from $(\{r_1\}^*\{r_2\}\{r_3\}^*\{r_4\})^*$. This holds for the start nonterminal ($m = 0$). Now, we can either repeat this sequence or finish the derivation by using r_5 until we have a^{2^m}. In the former case, we might apply r_1 any number of times. However, if we apply it only k many times, where $k < m$, then we have to use r_2, which blocks the derivation (there is no rule with X on its left hand side). This rule guarantees that every S is eventually rewritten to AA. Since $r_2 \in F$, after there are no S symbols in the sentential form, we can skip it (it

is not applicable), so we get $S^{2^m} \Rightarrow_{ac}^* (AA)^{2^m} = A^{2^{m+1}}$. Then, by the same reasoning, we have to rewrite each A to S, so we get $A^{2^{m+1}} \Rightarrow_{ac}^* S^{2^{m+1}}$, which is of the same form as the sentential form we started with. Therefore, this grammar generates the non-context-free language $L(H) = \{a^{2^n} \mid n \geq 0\}$ [67, Example 2.1].

For example, the sentence *aaaa* is obtained by the following derivation:

$$S \Rightarrow_{ac} AA\ [r_1] \Rightarrow_{ac} AS\ [r_3] \Rightarrow_{ac} SS\ [r_3] \Rightarrow_{ac} AAS\ [r_1] \Rightarrow_{ac} AAAA\ [r_1] \Rightarrow_{ac}$$
$$AASA\ [r_3] \Rightarrow_{ac} AASS\ [r_3] \Rightarrow_{ac} SASS\ [r_3] \Rightarrow_{ac} SSSS\ [r_3] \Rightarrow_{ac} SSSa\ [r_5] \Rightarrow_{ac}$$
$$aSSa\ [r_5] \Rightarrow_{ac} aaSa\ [r_5] \Rightarrow_{ac} aaaa\ [r_5].$$

We can disallow erasing rules in the underlying core grammar. This is formalized in the following definition.

Definition 3.1.3. Let $H = (G, \Xi)$ ($H = (G, \Xi, F)$) be a regularly controlled grammar (with appearance checking). If G is ε-free, then H is said to be an *ε-free regularly controlled grammar (with appearance checking).*

By $\mathbf{rC}_{ac}^{\varepsilon}$, $\mathbf{rC}_{ac}$, $\mathbf{rC}^{\varepsilon}$, and $\mathbf{rC}$, we denote the families of all languages generated by regularly controlled grammars with appearance checking, ε-free regularly controlled grammars with appearance checking, regularly controlled grammars, and ε-free regularly controlled grammars, respectively.

Theorem 3.1.1. *(See [44, Theorem 1])*

(1) $\boldsymbol{CF \subset rC \subset rC_{ac} \subset CS}$

(2) $\boldsymbol{CF \subset rC \subseteq rC^{\varepsilon} \subset rC_{ac}^{\varepsilon} = RE}$.

These relations are pictured in Figure 3.1. If two families are connected by a dashed arrow (solid arrow), then the upper family includes (includes properly) the lower family; if two families are not connected then they are not necessarily incomparable.

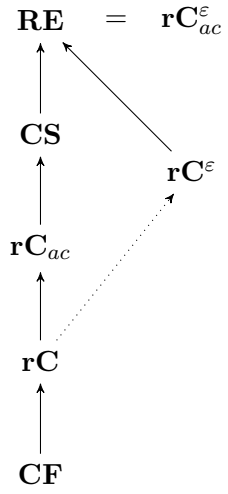


Figure 3.1: Graphical Representation of Theorem 3.1.1.

3.2 Matrix Grammar

Instead of complete sequences of rules (in the sense that each sequence of rules either lead to a sentence or not), which forms the type of regulation in regularly controlled grammars, we can consider shorter sequences. First, we can choose any sequence we want. However, after we pick some sequence, we have to apply every rule from it in the prescribed order (up to the application of appearance checking). Then we choose again and continue until we obtain a sentence and finish applying all rules from the last sequence we chose. Matrix grammars were introduced by Abraham in [1].

Contrary to the definition of a regularly controlled grammar, for brevity, in the following definition of a matrix grammar, we consider matrix grammars with appearance checking and define matrix grammars as matrix grammars without appearance checking.

Definition 3.2.1. A *matrix grammar with appearance checking*, see [44], is a triplet,

$$H = (G, M, F),$$

where

- $G = (N, T, S, P)$ is a context-free grammar, called *core grammar*,
- $M \subseteq \Psi^*$ is a finite language, whose elements are called *matrices*, and
- $F \subseteq \Psi$ is the *appearance checking set*.

For $m = m_1 m_2 \dots m_n \in M$, for some $n \geq 1$, and $x, y \in V^*$, we define $x \ {}_m\Rightarrow_H \ y$ as

$$x = x_0 \Rightarrow_{ac} x_1 \ [m_1] \Rightarrow_{ac} x_2 \ [m_2] \Rightarrow_{ac} \dots \Rightarrow_{ac} x_n = y \ [m_n],$$

where $x_i \in V^*$, for all $1 \leq i \leq n$, and the application of rules in the appearance checking mode is defined as in Definition 3.1.2.

The *language generated by* H, denoted by $L(H)$, is defined as the set of all strings $w \in T^*$ such that there is a derivation

$$S \ {}_{m_1}\Rightarrow_H \ w_1 \ {}_{m_2}\Rightarrow_H \ w_2 \ {}_{m_3}\Rightarrow_H \ \cdots \ {}_{m_n}\Rightarrow_H \ w_n = w,$$

where $m_i \in M$, $w_i \in V^*$, for all $1 \leq i \leq n$, for some $n \geq 1$.

Note that if $M = \Psi$, then there is no regulation, and thus $L(H) = L(G)$ in this case.

Definition 3.2.2. Let $H = (G, M, F)$ be a matrix grammar with appearance checking. We say that H is a *matrix grammar without appearance checking* (abbreviated *matrix grammar*) if and only if $F = \emptyset$.

Example 3.2.1. (From [44]) Let $H = (G, M, \emptyset)$ be a matrix grammar (without appearance checking), where $G = (N, T, S, P)$ is a context-free grammar with $N = \{S, A, B\}$, $T = \{a, b\}$, P consists of the following rules:

$r_1 \colon S \to AB$, $r_4 \colon A \to bA$, $r_7 \colon B \to a$,

$r_2 \colon A \to aA$, $r_5 \colon B \to bB$, $r_8 \colon A \to b$,

$r_3 \colon B \to aB$, $r_6 \colon A \to a$, $r_9 \colon B \to b$,

and $M = \{r_1, r_2r_3, r_4r_5, r_6r_7, r_8r_9\}$.

We start with the only applicable rule, r_1, from matrix r_1 and we get AB. Now, we can either

- terminate the derivation using the matrix r_6r_7 and obtain aa, or
- terminate the derivation using the matrix r_8r_9 and obtain bb, or
- rewrite AB to $aAaB$ by using the matrix r_2r_3, or
- rewrite AB to $bAbB$ by using the matrix r_4r_5.

If the derivation is not terminated, we can continue by the same reasoning. Clearly, this grammar generates the language $L(H) = \{ww \mid w \in \{a, b\}^+\}$, which is not context-free [50, Example 6.2.3]. For example, the sentence $aabaab$ is obtained by the following derivation:

$$S \Rightarrow_{r_1} AB \Rightarrow_{r_2r_3} aAaB \Rightarrow_{r_2r_3} aaAaaB \Rightarrow_{r_8r_9} aabaab.$$

As with regularly controlled grammars, we can disallow erasing rules in the underlying core grammar.

Definition 3.2.3. Let $H = (G, M, F)$ be a matrix grammar (with appearance checking). If G is ε-free, then H is said to be an *ε-free matrix grammar (with appearance checking).*

The families of all languages generated by matrix grammars with appearance checking, ε-free matrix grammars with appearance checking, matrix grammars, and ε-free matrix grammars, respectively, are denoted by $\mathbf{M}^{\varepsilon}_{ac}$, $\mathbf{M}_{ac}$, $\mathbf{M}^{\varepsilon}$, and $\mathbf{M}$, respectively. There is an interesting relation between matrix grammars and regularly controlled grammars.

Theorem 3.2.1. *(See [44, Theorem 2])*

(1) $\boldsymbol{M}^{\varepsilon}_{ac} = \boldsymbol{rC}^{\varepsilon}_{ac}$

(2) $\boldsymbol{M}_{ac} = \boldsymbol{rC}_{ac}$

(3) $\boldsymbol{M}^{\varepsilon} = \boldsymbol{rC}^{\varepsilon}$

(4) $\boldsymbol{M} = \boldsymbol{rC}$

So, the relations among families of languages generated by matrix grammars are the same as among languages generated by regularly controlled grammars. They are pictured in Figure 3.2, which, due to Theorem 3.2.1, corresponds to Figure 3.1. Again, if two families are connected by a dashed arrow (solid arrow), then the upper family includes (includes properly) the lower family; if two families are not connected then they are not necessarily incomparable.

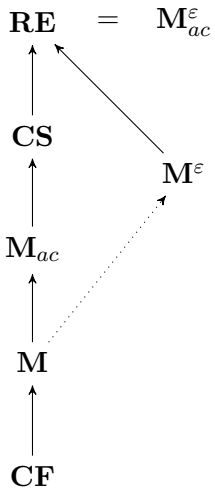


Figure 3.2: Relations Among Families of Languages Generated by Matrix Grammars.

3.3 Programmed Grammar

In preceding sections the allowed derivations were given in the form of prescribed sequences of rules. We now give a grammar, called programmed grammar, which differs from the preceding ones in the way it is regulated—the derivation is accompanied by computation which selects the allowed derivations. Informally, every context-free rule, r, has assigned two sets of rules. If r is applied successfully, then in the next step, a rule from the first set has to be applied. On the other hand, if r is not applicable, a rule from the second set has to be applied. Programmed grammars were introduced by Rosenkrantz in [64].

Definition 3.3.1. A *programmed grammar with appearance checking*, see [44], is a triplet,

$$H = (G, \sigma, \varphi),$$

where

- $G = (N, T, S, P)$ is a context-free grammar, called *core grammar*,
- $\sigma\colon \Psi \to 2^{\Psi}$ is a total function, called *success field*, and
- $\varphi\colon \Psi \to 2^{\Psi}$ is a total function, called *failure field*.

For a nonterminal, $A \in N$, and a string, $w \in V^*$, we write $A \Rightarrow_H^+ w$ if

$$A = w_0 \Rightarrow w_1\ [r_1] \Rightarrow w_2\ [r_2] \Rightarrow \ldots \Rightarrow w_n = w\ [r_n] \text{ in } G,$$

for some $n \geq 1$, where for each $r_i\colon A_i \to v_i \in P$, $v_i \in V^*$, for all $1 \leq i \leq n$, one of the following hold:

$$w_{i-1} = x_{i-1}A_iy_{i-1},\ w_i = x_{i-1}v_iy_{i-1},$$
$$\text{for some } x_{i-1}, y_{i-1} \in V^* \text{ and if } i < n, \text{ then } r_{i+1} \in \sigma(r_i),$$

or

$$A_i \notin alph(w_i),\ w_{i-1} = w_i, \text{ and if } i < n, \text{ then } r_{i+1} \in \varphi(r_i).$$

The *language generated by* H, denoted by $L(H)$, is then defined as $L(H) = \{w \mid w \in T^*, S \Rightarrow_H^+ w\}$.

Note that if for each $r \in \Psi$, $\sigma(r) = \Psi$, then there is no regulation and $L(G) = L(H)$ in this case.

Definition 3.3.2. Let $H = (G, \sigma, \varphi)$ be a programmed grammar with appearance checking. If for each $r \in \Psi$, $\varphi(r) = \emptyset$, then we say that H is a *programmed grammar without appearance checking* (abbreviated *programmed grammar*).

Definition 3.3.3. Let $H = (G, \sigma, \varphi)$ be a programmed grammar (with appearance checking). If G is ε-free, then H is said to be an *ε-free programmed grammar (with appearance checking).*

Let $H = (G, \sigma, \varphi)$ be a programmed grammar (with appearance checking), where $G = (N, T, S, P)$. For brevity, for a rule $r\colon A \to v \in P$, $v \in V^*$, we write $\lfloor r\colon A \to v, \sigma(r), \varphi(r) \rfloor \in P$.

Example 3.3.1. (From [11]) Consider the programmed grammar, $H = (G, \sigma, \varphi)$, with $G = (\{S, A\}, \{a\}, S, P)$, where P consists of the rules:

$$\lfloor r_1\colon S \to AA, \{r_1\}, \{r_2, r_3\} \rfloor,$$

$$\lfloor r_2\colon A \to S, \{r_2\}, \{r_1\} \rfloor,$$

$$\lfloor r_3\colon A \to a, \{r_3\}, \emptyset \rfloor.$$

Since $\sigma(r_i) = \{r_i\}$, for each $i \in \{1, 2, 3\}$, the rules r_1, r_2, and r_3 have to be used as many times as possible. Therefore, starting from S^n for some $n \geq 1$, we have to pass to A^{2n} and then, using r_2, to S^{2n}, or using r_3, to a^{2n}. Each such cycle consisting of the use of r_1 and r_2 doubles the number of symbols. In conclusion, we obtain the non-context-free language $L(H) = \{a^{2^n} \mid n \geq 1\}$ [11, Example 1.1.5]. Notice the similarity between H from this example and H from Example 3.1.2.

For example, the sentence *aaaa* is obtained by the following derivation:

$$S \Rightarrow AA\ [r_1] \Rightarrow AS\ [r_2] \Rightarrow SS\ [r_2] \Rightarrow AAS\ [r_1] \Rightarrow AAAA\ [r_1] \Rightarrow AASA\ [r_2] \Rightarrow$$
$$AASS\ [r_2] \Rightarrow SASS\ [r_2] \Rightarrow SSSS\ [r_2] \Rightarrow SSSa\ [r_3] \Rightarrow aSSa\ [r_3] \Rightarrow aaSa\ [r_3] \Rightarrow aaaa\ [r_3].$$

The families of all languages generated by programmed grammars with appearance checking, ε-free programmed grammars with appearance checking, programmed grammars, and ε-free programmed grammars, respectively, are denoted by $\mathbf{P}_{ac}^{\varepsilon}$, $\mathbf{P}_{ac}$, $\mathbf{P}^{\varepsilon}$, and $\mathbf{P}$, respectively. Programmed grammars are related to matrix grammars as matrix grammars are related to regularly controlled grammars.

Theorem 3.3.1. *(See [44, Theorem 5])*

(1) $\boldsymbol{P}^{\varepsilon}_{ac} = \boldsymbol{M}^{\varepsilon}_{ac}$

(2) $\boldsymbol{P}_{ac} = \boldsymbol{M}_{ac}$

(3) $\boldsymbol{P}^{\varepsilon} = \boldsymbol{M}^{\varepsilon}$

(4) $\boldsymbol{P} = \boldsymbol{M}$

So, the relations among families of languages generated programmed grammars are the same as among languages generated by matrix grammars and regularly controlled grammars (follows from Theorem 3.2.1 and from Theorem 3.3.1). They are pictured in Figure 3.3. Again, if two families are connected by a dashed arrow (solid arrow), then the upper family includes (includes properly) the lower family; if two families are not connected then they are not necessarily incomparable.

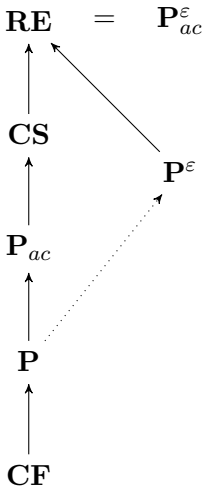


Figure 3.3: Relations Among Families of Languages Generated by Programmed Grammars.

3.4 Random Context Grammar

Up to now, we considered only regulation by prescribed or computed sequences of rules. The next possible type of regulation is based on sentential forms. With any rule, we associate some restrictions for sentential forms which have to be satisfied to apply that rule. More specifically, in the following type of a regulated grammar, we associate two sets of nonterminals with each rule. A rule can be applied only if the current sentential form contain some nonterminal from the first set and no nonterminal from the second set. This type of a regulated grammar is called a random context grammar and was introduced by van der Walt in [72].

Definition 3.4.1. A *random context grammar with appearance checking*, see [67], is a triplet,

$$H = (G, Per, For),$$

where

- $G = (N, T, S, P)$ is a context-free grammar, called *core grammar*,
- $Per\colon \Psi \to 2^N$ is a total function, called *permitting context*, and
- $For\colon \Psi \to 2^N$ is a total function, called *forbidding context*.

The *language generated by* H, denoted by $L(H)$, is defined as the set of all strings $w \in T^*$ such that there is a derivation

$$S = w_0 \Rightarrow w_1\ [r_1] \Rightarrow w_2\ [r_2] \Rightarrow \ldots \Rightarrow w_n = w\ [r_n],$$

for some $n \geq 1$, where for each $w_i \in V^*$ and $r_i \in \Psi$, for all $1 \leq i \leq n$, the following two conditions hold:

$$Per(r_i) \subseteq alph(w_{i-1})$$

and

$$For(r_i) \cap alph(w_{i-1}) = \emptyset.$$

Note that if for each $r \in \Psi$, $Per(r) = \emptyset$ (or $Per(r) = \{lhs(r)\}$) and $For(r) = \emptyset$, then there is no regulation and $L(G) = L(H)$ in this case.

Definition 3.4.2. Let $H = (G, Per, For)$ be a random context grammar with appearance checking. If for each $r \in \Psi$, $For(r) = \emptyset$, then we say that H is a *random context grammar without appearance checking* (abbreviated *random context grammar* or *permitting random context grammar*).

Definition 3.4.3. Let $H = (G, Per, For)$ be a random context grammar (with appearance checking). If G is ε-free, then H is said to be an *ε-free random context grammar (with appearance checking)*.

Let $H = (G, Per, For)$, be a random context grammar (with appearance checking), where $G = (N, T, S, P)$. For brevity, for a rule, $r\colon A \to v \in P$, $v \in V^*$, we write $\lfloor r\colon A \to v, Per(r), For(r) \rfloor \in P$.

Example 3.4.1. (From [58]) Consider the random context grammar $H = (G, Per, For)$ with $G = (\{S, A, B, C, A', B', C'\}, \{a, b, c\}, S, P)$, where P consists of the rules:

$\lfloor r_0\colon S \to ABC, \emptyset, \emptyset \rfloor$,
$\lfloor r_1\colon A \to aA', \{B\}, \emptyset \rfloor$,
$\lfloor r_2\colon B \to bB', \{C\}, \emptyset \rfloor$,
$\lfloor r_3\colon C \to cC', \{A'\}, \emptyset \rfloor$,
$\lfloor r_4\colon A' \to A, \{B'\}, \emptyset \rfloor$,
$\lfloor r_5\colon B' \to B, \{C'\}, \emptyset \rfloor$,
$\lfloor r_6\colon C' \to C, \{A\}, \emptyset \rfloor$,
$\lfloor r_7\colon A \to a, \{B\}, \emptyset \rfloor$,
$\lfloor r_8\colon B \to b, \{C\}, \emptyset \rfloor$,
$\lfloor r_9\colon C \to c, \emptyset, \emptyset \rfloor$.

Since only r_0 has S on its left-hand side, r_0 has to be applied first. Then, we can either use r_7, r_8, and r_9 to finish the derivation, thus obtaining abc, or continue with r_1, r_2, and r_3. In either case, these rules have to be applied in this specific order (for example, if we apply r_9 before r_8, we are not be able to finish the derivation, because r_8 can be applied only if there is some C in the sentential form). Rules r_5, r_6, and r_7 are used to rewrite primed nonterminals to their respective non-primed versions. In conclusion, we obtain the non-context-free language $L(H) = \{a^n b^n c^n \mid n \geq 1\}$ [58, Example 1].

Consider the sentence $aabbcc$. H generates this sentence in the following way:

$$S \Rightarrow ABC\ [r_0] \Rightarrow aA'BC\ [r_1] \Rightarrow aA'bB'C\ [r_2] \Rightarrow aA'bB'cC'\ [r_3] \Rightarrow aAbB'cC'\ [r_4] \Rightarrow aAbBcC'\ [r_5] \Rightarrow aAbBcC\ [r_6] \Rightarrow aabBcC\ [r_7] \Rightarrow aabbcC\ [r_8] \Rightarrow aabbcc\ [r_9].$$

The families of all languages generated by random context grammars with appearance checking, ε-free random context grammars with appearance checking, random context grammars, and ε-free random context grammars, respectively, are denoted by $\mathbf{RC}_{ac}^{\varepsilon}$, $\mathbf{RC}_{ac}$, $\mathbf{RC}^{\varepsilon}$, and $\mathbf{RC}$, respectively[1].

Theorem 3.4.1. *(See [67, Theorem 2.7])*

(1) $\boldsymbol{CF} \subset \boldsymbol{RC} \subset \boldsymbol{RC}_{ac} \subset \boldsymbol{CS}$

(2) $\boldsymbol{CF} \subset \boldsymbol{RC} \subseteq \boldsymbol{RC}^{\varepsilon} \subset \boldsymbol{RC}_{ac}^{\varepsilon} = \boldsymbol{RE}$

Also, four relations to the families of languages generated by matrix grammars (and thus also to the families of languages generated by regularly controlled grammars and programmed grammars, see Theorem 3.2.1 and Theorem 3.3.1) were established.

Theorem 3.4.2. *(See [67, Theorem 2.7])*

(1) $\boldsymbol{RC}_{ac}^{\varepsilon} = \boldsymbol{M}_{ac}^{\varepsilon}$

(2) $\boldsymbol{RC}_{ac} = \boldsymbol{M}_{ac}$

(3) $\boldsymbol{RC} \subseteq \boldsymbol{M}$

(4) $\boldsymbol{RC}^{\varepsilon} \subseteq \boldsymbol{M}^{\varepsilon}$

Relations among language families generated by random context grammars and matrix grammars are pictured in Figure 3.4. If two families are connected by a dashed arrow (solid arrow), then the upper family includes (includes properly) the lower family; if two families are not connected then they are not necessarily incomparable.

3.5 Scattered Context Grammar

Up to now, only a single rule was applied during a derivation step. A natural approach is to allow more than one rule to be applied in a single step. However, compared to formal

[1]Notice that the families of languages generated by regularly controlled grammars are denoted by **rC** (contrary to **RC**), which is a usual denotation in the literature [11, 44, 67].

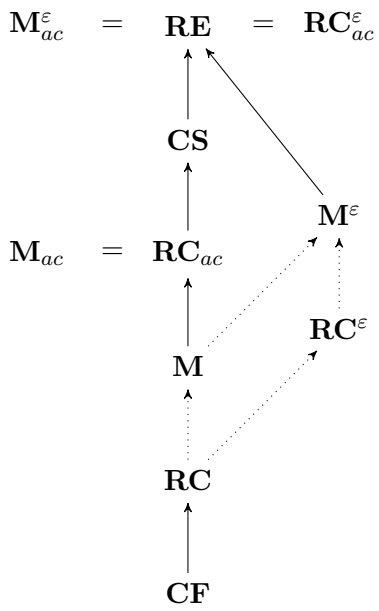


Figure 3.4: Relations Among Families of Languages Generated by Random Context Grammars and Matrix Grammars.

models with full parallelism (like L systems [65]), we obtain only partial parallelism (not all symbols are rewritten, especially not terminals).

The following type of a regulated grammar, called scattered context grammar, introduces compound rules of the form $(A_1 \to w_1, A_2 \to w_2, \ldots, A_n \to w_n)$, where all $A_i \to w_i$ are context-free rules, for some $n \geq 1$. This compound rule can be applied only if the current sentential form is of the form $x_0 A_1 x_1 A_2 x_2 \ldots x_{n-1} A_n x_n \ldots$, where every x_j is an arbitrary string. In a single derivation step, every A_i is rewritten w_i. As the name suggests, the context is scattered, rather than being completely random (as in a random context grammar). Also note that since all x_j are arbitrary strings, their length is not limited and they can contain any symbol (for example, x_0 can contain A_1). The only thing that matters is the context—every A_k has to appear before every A_{k+1}. Finally, the difference between matrix grammars and scattered context-grammars is that in matrix grammars, all rules in a sequence are applied sequentially, rather than in parallel, and there is no directly prescribed context between occurrences of nonterminals in consecutive applications of rules. Scattered context grammars were introduced by Greibach and Hopcroft in [26].

Definition 3.5.1. A *scattered context grammar*, see [67], is a pair,

$$H = (G, R),$$

where

- $G = (N, T, S, P)$ is a context-free grammar, called *core grammar*, and
- $R \subseteq P^*$ is a finite language.

For $x, y \in V^*$, we say that x directly derives y in H according to a compound rule, $r \in R$, written as $x \Rightarrow_H y\ [r]$ (or, more briefly, $x \Rightarrow_H y$), if and only if all of the following conditions are satisfied:

(1) $x = x_0A_1x_1A_2x_2 \dots A_nx_n$, where $x_i \in V^*$, $A_j \in N$, for all $0 \leq i \leq n$, $1 \leq j \leq n$, for some $n \geq 1$,

(2) $y = x_0w_1x_1w_2x_2 \dots w_nx_n$, where $w_i \in V^*$, for all $1 \leq i \leq n$,

(3) $r = (A_1 \to w_1, A_2 \to w_2, \dots, A_n \to w_n) \in R$.

Let $\Rightarrow_H^*$ be the reflexive and transitive closure of $\Rightarrow_H$. The *language generated by* H, denoted by $L(H)$, is defined as $L(H) = \{w \mid w \in T^*, S \Rightarrow_H^* w\}$.

Note that if $R = P$, then there is no regulation and $L(G) = L(H)$ in this case. Also note that there is no appearance checking in scattered context grammars.

Definition 3.5.2. Let $H = (G, R)$ be a scattered context grammar. If G is ε-free, then H is said to be an *ε-free scattered context grammar*[2].

Example 3.5.1. (Modified example from [57]) Consider the non-context-free language $L = \{a^nb^nc^n \mid n \geq 0\}$. This language can be generated by the scattered context grammar $H = (G, R)$, where $G = (\{S, A\}, \{a, b, c\}, S, P)$ is a context-free grammar with P containing the following rules:

$$S \to AAA,$$

$$A \to aA \mid bA \mid cA \mid \varepsilon,$$

and R contains the following three sequences:

$$(S \to AAA),$$

$$(A \to aA, A \to bA, A \to cA),$$

$$(A \to \varepsilon, A \to \varepsilon, A \to \varepsilon).$$

Clearly, the first sequence (containing only one rule) is the only applicable sequence to the start nonterminal. After three As are generated, the second and third sequence can be used to obtain any string of the form $a^nb^nc^n$, for some $n \geq 0$. For example, the sentence $aabbcc$ can be generated by the following derivation:

$$S \Rightarrow_H AAA \Rightarrow_H aAbAcA \Rightarrow_H aaAbbAccA \Rightarrow_H aabbcc.$$

The families of all languages generated by scattered context grammars and ε-free scattered context grammars, respectively, are denoted by $\mathbf{SC}^\varepsilon$ and $\mathbf{SC}$, respectively.

Theorem 3.5.1. *(See [67, Theorem 2.12])* $\boldsymbol{CF \subset SC \subseteq CS \subset SC^\varepsilon = RE}$

These relations are pictured in Figure 3.5. If two families are connected by a dashed arrow (solid arrow), then the upper family includes (includes properly) the lower family.

[2]Let us note that sometimes the term *propagating* is used instead of ε-free [49, 55, 56, 57].

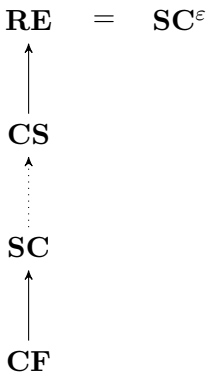


Figure 3.5: Graphical Representation of Theorem 3.5.1.

3.6 Other Types of Regulation

Previous sections covered some classical types of regulation. They also covered grammars that are discussed in Chapter 5 in further detail. However, there are many more types of regulation, so this section briefly mentions some of them. Since these grammars are not of our main interest, only informal description is given. Results regarding the possibility of erasing rules elimination from these grammars are also mentioned.

Tree and Path Controlled Grammars

Instead of requiring for the parse of a sentence to be in a control language (like in a regularly controlled grammar), we can control derivations by putting restrictions on derivation trees. The following two restrictions belong among the studied ways of derivation control regarding derivation trees.

(1) *(Horizontal control)* For a sentence, x, we require that there is a derivation of x and the corresponding derivation tree such that each string obtained by concatenating all symbols at any *level* (except the last one) from the left to the right is in the control language. Informally, a level in a derivation tree is a sequence of nodes that have the same distance from the root of the derivation tree. For example, there five levels in the derivation tree in Example 2.4.1 (one of them is B, b, A, c)[3]. Regulated grammars using this type of a restriction are called *tree controlled grammars* and were introduced by Culik and Maurer in [30].

(2) *(Vertical control)* For a sentence, x, we require that there is a derivation of x and the corresponding derivation tree such that there is a *path* (or more paths [37]) which, when we concatenate all symbols in this path, is in a control language. Informally, a path in a derivation tree is a sequence of nodes that begins in the root of the derivation tree and ends in a leaf marked with a terminal symbol. For example, there are two paths in the derivation tree in Example 2.4.1 (S, B, b and S, B, c). Regulated grammars using this type of a restriction are called *path controlled grammars* and were introduced by Marcus, Martín-Vide, Mitrana, and Păun in [42].

[3]Note that this derivation tree does not correspond to a sentence, but to a string, since its frontier contains nonterminal symbols.

It was proved that tree controlled grammars with erasing rules are more powerful than tree controlled grammars without them [11, Theorem 2.3.3]. However, the impact of the presence of erasing rules in path controlled grammars has not been studied by knowledge (in [42] and [43], core grammars of path controlled grammars are ε-free).

Ordered Grammar

Rather than prescribing sequences of rules (like in regularly controlled grammars or matrix grammars), one may consider to impose priorities on the set of rules so we can say whether one rule has greater, lower, or the same priority as some other rule. Then, when deciding which rules are applicable to some sentential form, these priorities are taken into account. A rule, $r\colon A \to x$, is applicable only if there is some A in the current sentential form and there is no other rule, $s\colon B \to y$, where B is also in the sentential form and s has a higher priority than r. According to this observation, we can order rules using these priorities and base the applicability on this ordering. Strictly speaking, we impose a partial order $\preccurlyeq$ on the set of rules, and we can apply a rule r only if there is no other applicable rule s such that $r \neq s$ and $r \preccurlyeq s$.

Regulated grammars using this type of regulation are called *ordered grammars* and were introduced by Friš in [24]. It is not known whether we can eliminate erasing rules from these grammar without affecting the generated language [44, Theorem 9].

Indian and Russian Parallel Grammars

In scattered context grammars, we have compound rules of the form $(A_1 \to w_1, A_2 \to w_2, \ldots, A_n \to w_n)$, for some $n \geq 1$, where every $A_i \to w_i$, for all $1 \leq i \leq n$, is an ordinary context-free rule. In a single derivation step, we rewrite A_1 to w_1, A_2 to w_2, and so on, where every A_{i+1} has to appear on the right from A_i, for all $1 \leq i < n$, and all nonterminals $A_1, A_2, \ldots, A_n$ are rewritten at once.

Another semi-parallel approach is to take a single context-free rule, $A \to w$, and rewrite all occurrences of A in the sentential form to w in a single step. A grammar using this type of regulation is called an *Indian parallel grammar* and was introduced by Siromoney and Krithivasan in [71]. If we split the set of rules, P, into two disjoint sets, $P_1, P_2 \subseteq P$, and require that all rules from P_1 have to be applied in a standard context-free way and all rules from P_2 have to be applied as in an Indian parallel grammar, we obtain a *Russian parallel grammar*, introduced by Levitina in [40].

Regarding erasing rules, we are able to convert any Indian parallel with erasing rules to an equivalent Indian parallel grammar without erasing rules [67, Theorem 2.9]. However, it is not known whether we can always remove all erasing rules from any Russian parallel grammar without affecting the generated language [11, Open Problem 2.4.2].

Indexed Grammar

Yet another way of derivation control is to have an additional set—the set of *indices*—where each index is a set of context-free rules. Then, every occurrence of a nonterminal in a sentential form is followed by a sequence of indices (in certain sense, this sequence describes

the history of the nonterminal), and an index can only be erased by rules contained in the index (where the erasing is done in the reversed order of the appearance) [67]. For example, if the current sentential form is $aaBgfcc$, where a, b, c are terminals, B is a nonterminal, and $f = \{B \to b\}$, $g = \{B \to bB\}$ are two indices, then we can erase the index g by an application of a rule from g (in this case, there is only one), thus obtaining $aabBfcc$. Then, we erase f in the same manner (again, in this case, there is only one rule in f), thus obtaining the sentence $aabbcc$.

An *indexed grammar* was introduced by Aho in [2] and it was proved that the family of all languages generated by indexed grammars with erasing rules coincides the family of all languages generated by indexed grammars without erasing rules [67, Theorem 2.9].

Petri Net Controlled Grammar

Consider regularly controlled grammars. Instead of a regular grammar, we can consider other types of formalisms to control the derivation process. One of such formalisms is a *Petri net* [63]. A Petri net is a modeling language for the description of distributed (parallel) systems. Visually, Petri nets are represented by a *bipartite graph*, which is a graph where the set of nodes, V, can can be divided into two disjoint sets, V_1 and V_2, such that every edge connects either a vertex in V_1 to one in V_2 or vice versa. Basically, a Petri net consists of *places* (first type of nodes, drawn as circles), *transitions* (second type of nodes, drawn as boxes), *directed arcs* (edges, drawn as arrows from places to transitions and from transitions to places), and *tokens* (drawn as small solid dots inside places). The behaviour of a Petri net is defined in terms of transition of tokens between places. Figure 3.6 shows an example of a Petri net (there are four places, A, B, C, D, three transitions, $T1, T2, T3$, and three tokens).

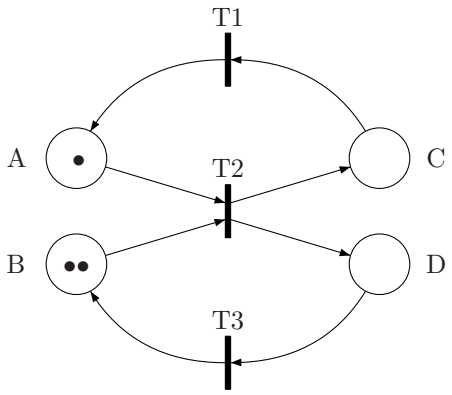


Figure 3.6: An Example of a Petri Net.

Essentially, a *Petri net controlled grammar* is a context-free grammar equipped with a Petri net and a function which maps transitions of the net to rules of the grammar [16]. Petri net controlled grammars have been intensively studied in the last three years [12, 13, 14, 15, 16, 76, 77]. One of the results concerns the presence of erasing rules in these grammar: it was proved that Petri net controlled grammars with and without erasing rules generate the same family of languages [76, Theorem 3].

Chapter 4

Removal of Erasing Rules from Context-Free Grammars

It is very well known that one can remove all erasing rules from any context-free grammar by converting such grammar into an equivalent ε-free context-free grammar, thus without affecting the generated language [50, Theorem 5.1.3.2.4]. From Chapter 3 it is clear that many regulated grammars are based on context-free grammars (in the sense that they use context-free rules), but they regulate the application of these rules. So, it is desirable to know algorithms that can be used to remove erasing rules from context-free grammars and try to apply them to regulated grammars. To this end, this chapter presents two algorithms for the purpose of elimination of erasing rules from context-free grammars: a standard one (used in textbooks), described in Section 4.1, and a new one, presented in Section 4.2.

4.1 Standard Algorithm

The standard algorithm is based on a predetermination of so-called ε-nonterminals. These are nonterminals from which the empty string can be derived.

Definition 4.1.1. Let $G = (N, T, S, P)$ be a context-free grammar. A nonterminal, $A \in N$, is said to be an *ε-nonterminal* if and only if $A \Rightarrow^* \varepsilon$ in G.

The following algorithm determines the set of all ε-nonterminals in a given context-free grammar.

Algorithm 4.1.1. *(See [50, Algorithm 5.1.3.2.1]) Determination of ε-nonterminals in a context-free grammar.*

Input: A context-free grammar, $G = (N, T, S, P)$.

Output: The set of all ε-nonterminals in G, $N_\varepsilon = \{A \mid A \in N \text{ and } A \Rightarrow^* \varepsilon\}$.

Method: Initially, set $N_\varepsilon = \{A \mid A \to \varepsilon \in P\}$. Now, apply the following step until N_ε cannot be extended:

> **If** $A \to X_1X_2 \dots X_n \in P$, where $X_i \in N_\varepsilon$, for all $1 \leq i \leq n$, for some $n \geq 1$
> **then** $N_\varepsilon = N_\varepsilon \cup \{A\}$.

Main Idea. N_ε is first initialized to nonterminals from which the empty string can be derived in a single step. Then, if there is a rule which have only nonterminals from N_ε on its right-hand side, include the left-hand side of this rule to N_ε. This step is repeated until N_ε cannot be extended.

Example 4.1.1. Let $G = (N, T, S, P)$ be a context-free grammar, where $N = \{S, A, B, C\}$, $T = \{b, c\}$, and P consists of the following rules:

$S \to AB \mid BC,$

$A \to \varepsilon,$

$B \to A \mid b,$

$C \to c.$

With G on its input, Algorithm 4.1.1 produces the following set of ε-nonterminals: $N_\varepsilon = \{S, A, B\}$.

Lemma 4.1.1. *(See [50, Lemma 5.1.3.2.2]) Algorithm 4.1.1 is correct, i.e. with a context-free grammar, G, on its input, it halts and correctly produces the set of all ε-nonterminals in G.*

Now, using Algorithm 4.1.1, we can define the standard algorithm for elimination of erasing rules from context-free grammars.

Algorithm 4.1.2. *(See [29, Section 7.3.1]) Standard elimination of erasing rules from context-free grammars.*

Input: A context-free grammar, $G = (N, T, S, P_G)$.

Output: An ε-free context-free grammar, $H = (N, T, S, P_H)$, such that $L(H) = L(G) - \{\varepsilon\}$.

Method: Use Algorithm 4.1.1 to compute N_ε from G. Initially, set $P_H = \emptyset$. Now, to compute P_H, apply the following step until P_H cannot be extended:

> **If** $A \to x_0X_1x_1X_2x_2 \dots X_nx_n \in P_G$, where $x_i \in N_\varepsilon^*$, $X_j \in V$, for all $0 \leq i \leq n$, $1 \leq j \leq n$, for some $n \geq 1$
> **then** add $A \to X_1X_2 \dots X_n$ to P_H.

Main Idea. Consider the context-free grammar, $G = (N, T, S, P)$, where $A \to aAbB, A \to \varepsilon, B \to \varepsilon \in P$, $A, B \in N$, and $a, b \in T^*$. Notice that $A, B \in N_\varepsilon$. The idea behind Algorithm 4.1.2 is that if some nonterminal, A, is erased during a derivation (in arbitrary number of steps), then it does not need to be present in the sentential form, so why derive it in the first place. Hence, the corresponding ε-free context-free grammar additionally has rules $A \to abB$, $A \to aAb$ and $A \to ab$ in its set of rules (additionally, because it also has to contain the original rule, $A \to aAbB$, in case there are some non-erasing rules, $A \to x$ and $B \to y$, in P, which can lead into a derivation of a string of terminal symbols). This procedure is done for all rules in P.

Also, notice that if we transform a rule, $A \to BB$, with $B \in N_\varepsilon$, then H has only $A \to BB$ and $A \to B$ in its set of rules, because it does not make a difference if the first occurrence of B is erased, or the second one. We also do not want to include the erasing rule $A \to \varepsilon$ (by the **if** condition, there has to be at least one symbol that is chosen as a not-to-be-erased symbol).

Example 4.1.2. Let $G = (N, T, S, P_G)$ be a context-free grammar, where $N = \{S, A, B, C\}$, $T = \{a, b, c\}$ and P_G consists of the following rules:

$S \to ABC$,

$A \to a \mid BB$,

$B \to bB \mid \varepsilon$,

$C \to cBcC \mid \varepsilon$.

With G on its input, Algorithm 4.1.2 produces an ε-free context-free grammar, $H = (N, T, S, P_H)$, where P_H contains these rules:

$S \to ABC \mid AB \mid BC \mid AC$,

$A \to a \mid BB \mid B$,

$B \to bB \mid b$,

$C \to cBcC \mid cBc \mid ccC \mid cc$.

Lemma 4.1.2. *(See [29, Theorem 7.9]) Algorithm 4.1.2 is correct, i.e. with a context-free grammar, G, on its input, it halts and correctly produces an ε-free context-free grammar, H, such that $L(H) = L(G) - \{\varepsilon\}$.*

Theorem 4.1.1. *Let G be a context-free grammar. Then, there is an ε-free context-free grammar, H, such that $L(H) = L(G) - \{\varepsilon\}$.*

Proof. This theorem follows from Algorithm 4.1.2 and Lemma 4.1.2. □

4.2 New Algorithm

In this section, we describe an alternative elimination of erasing rules without any predetermination of ε-nonterminals.

Algorithm 4.2.1. *Elimination of erasing rules from context-free grammars without any predetermination of ε-nonterminals.*

Input: A context-free grammar, $G = (N_G, T, S_G, P_G)$.

Output: An ε-free context-free grammar, $H = (N_H, T, S_H, P_H)$, such that $L(H) = L(G) - \{\varepsilon\}$.

Method: Let us note that in what follows, symbols $\langle$ and $\rangle$ are used to clearly unite more symbols into a single compound symbol. Initially, set:

$N_H = \{\langle X, U\rangle \mid X \in V_G, U \subseteq N_G\}$;
$S_H = \langle S_G, \emptyset\rangle$;
$P_H = \{\langle a, \emptyset\rangle \to a \mid a \in T\}$.

Now, apply the following steps until P_H cannot be extended:

(1) **If** $A \to x_0X_1x_1X_2x_2 \dots X_nx_n \in P_G$, where $x_i \in N_G^*$, $X_j \in V_G$, for all $0 \leq i \leq n$, $1 \leq j \leq n$, for some $n \geq 1$
then for all $U \subseteq N_G$, add $\langle A, U\rangle \to \langle X_1, U \cup alph(x_0x_1 \dots x_n)\rangle\langle X_2, \emptyset\rangle \dots \langle X_n, \emptyset\rangle$ to P_H.

(2) **If** $\langle X, U\rangle \in N_H$ and $A \to x \in P_G$, where $A \in U$ and $x \in N_G^*$
then add $\langle X, U\rangle \to \langle X, (U - \{A\}) \cup alph(x)\rangle$ to P_H.

Main Idea. Let $G = (N_G, T, S_G, P_G)$ be a context-free grammar and $A \in N_G$ a be nonterminal which derives ε. This derivation can be expressed in the following, step-by-step way:

$$A \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \dots \Rightarrow x_n \Rightarrow \varepsilon,$$

where $x_i \in N_G^*$, for all $1 \leq i \leq n$, for some $n \geq 0$ ($n = 0$ means $A \Rightarrow \varepsilon$). If there is more than one occurrence of A in a sentential form, every occurrence of A can be erased in a similar way—the form of x_i is not important. Based on this observation, the algorithm introduces compound nonterminals of the form $\langle X, U\rangle$, where $X \in V_G$ is a single symbol that is not erased during the rest of the derivation and $U \subseteq N_G$ is a set of nonterminals, which are going to be erased during the rest of the derivation (these are chosen strongly nondeterministically). During a derivation, the algorithm simulates the erasure of nonterminals in U within the compound nonterminal in the way sketched above. Observe that as U is a set, U contains no more than one occurrence of any nonterminal, because there is no need to record several occurrences of the same nonterminal; indeed, as already pointed out, all these occurrences can be erased in the same way.

So, for a rule, $B \to aAb \in P_G$, $a, b \in T$, $A \in N_G$, the following two new rules are introduced:

$$\langle B, U\rangle \to \langle a, U\rangle\langle A, \emptyset\rangle\langle b, \emptyset\rangle$$

and

$$\langle B, U\rangle \to \langle a, U \cup \{A\}\rangle\langle b, \emptyset\rangle,$$

for every $U \subseteq N_G$. Then, if there is a rule, $A \to x \in P_G$, where $x \in N_G^*$, we can either rewrite the A in $\langle A, \emptyset\rangle$ by a rule introduced in (1) or the A in $\langle a, U \cup \{A\}\rangle$, introduced in (2). In the first case, this is the same as the case with $B \to aAb$ above; it also covers the situation where $x \in V_G^*$. In the latter case, A is rewritten inside the second component.

Since even terminals are at first enclosed in angle brackets (they are part of a compound nonterminal, too), to derive terminals from these nonterminals, P_H contains rules of the form $\langle a, \emptyset\rangle \to a$, for every $a \in T$, introduced in the initialization part of the algorithm. Note that in this case the second component of the compound nonterminal has to be the empty set; otherwise, there are still some nonterminals to be erased.

Example 4.2.1. Consider the context-free grammar, $G = (N_G, T, S_G, P_G)$, where $N_G = \{S_G\}$, $T = \{a, b\}$, and P_G consists of the following rules:

$$S_G \to aS_Gb,$$

$$S_G \to \varepsilon.$$

The generated language is clearly $L(G) = \{a^n b^n \mid n \geq 0\}$. Algorithm 4.2.1 produces an ε-free context-free grammar $H = (N_H, T, S_H, P_H)$, where $N_H = \{\langle S_G, \emptyset\rangle, \langle a, \emptyset\rangle, \langle b, \emptyset\rangle, \langle S_G, \{S_G\}\rangle, \langle a, \{S_G\}\rangle, \langle b, \{S_G\}\rangle\}$, $S_H = \langle S_G, \emptyset\rangle$, and P_H contains these rules:

$$\langle S_G, \emptyset\rangle \to \langle a, \emptyset\rangle\langle S_G, \emptyset\rangle\langle b, \emptyset\rangle, \qquad \langle b, \{S_G\}\rangle \to \langle b, \emptyset\rangle,$$

$$\langle S_G, \emptyset\rangle \to \langle a, \{S_G\}\rangle\langle b, \emptyset\rangle, \qquad \langle a, \emptyset\rangle \to a,$$

$$\langle a, \{S_G\}\rangle \to \langle a, \emptyset\rangle, \qquad \langle b, \emptyset\rangle \to b.$$

For example, for

$$S_G \Rightarrow aS_Gb \Rightarrow aaS_Gbb \Rightarrow aabb \text{ in } G,$$

there is

$$\langle S_G, \emptyset\rangle \Rightarrow \langle a, \emptyset\rangle\langle S_G, \emptyset\rangle\langle b, \emptyset\rangle \Rightarrow \langle a, \emptyset\rangle\langle a, \{S_G\}\rangle\langle b, \emptyset\rangle\langle b, \emptyset\rangle \Rightarrow \langle a, \emptyset\rangle\langle a, \emptyset\rangle\langle b, \emptyset\rangle\langle b, \emptyset\rangle \Rightarrow^* aabb \text{ in } H.$$

Lemma 4.2.1. *Algorithm 4.2.1 is correct, i.e. with a context-free grammar, G, on its input, it halts and correctly produces an ε-free context-free grammar, H, such that $L(H) = L(G) - \{\varepsilon\}$.*

Proof. Clearly, the algorithm always halts. Since P_H does not contain any erasing rules, H is ε-free. To prove that $L(H) = L(G) - \{\varepsilon\}$, we first prove the following claim, where we use Notation 2.4.1

Claim 4.2.1.

$$S_G \Rightarrow^{m\ \varepsilon} x_0 X_1{}^{\varepsilon} x_1 X_2{}^{\varepsilon} x_2 \ldots X_h{}^{\varepsilon} x_h \text{ in } G$$

if and only if

$$\langle S_G, \emptyset\rangle \Rightarrow^n \langle X_1, U_1\rangle\ \langle X_2, U_2\rangle \ldots \langle X_h, U_h\rangle \text{ in } H,$$

where $x_i \in N_G^*$, $X_j \in V_G$, for all $0 \leq i \leq h$, $1 \leq j \leq h$, and $\bigcup_{1 \leq i \leq h} U_i \subseteq \bigcup_{0 \leq i \leq h} alph(x_i)$, for some $m, n \geq 0$ and $h \geq 1$.

Proof. Only If: This is established by induction on the length m of derivations.

Basis: Let $m = 0$. For $S_G \Rightarrow^0 S_G$ in G there is $\langle S_G, \emptyset\rangle \Rightarrow^0 \langle S_G, \emptyset\rangle$ in H, so the basis holds.

Induction Hypothesis: Suppose that the *Only If* part of Claim 4.2.1 holds for all derivations of length m or less, for some $m \geq 0$.

Induction Step: Consider any derivation of the form

$$S_G \Rightarrow^{m+1} w \text{ in } G,$$

where $w \in V_G^+$. Since $m+1 > 0$, this derivation can be expressed as

$$S_G \Rightarrow^{m} x \Rightarrow w,$$

where $x \in V^+$. Let $x = {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_h{}^{\varepsilon}x_h$, where $x_i \in N_G^*$, $X_j \in V_G$, for all $0 \leq i \leq h$, $1 \leq j \leq h$, for some $h \geq 1$, so

$$S_G \Rightarrow^{m} {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_h{}^{\varepsilon}x_h \text{ in } G.$$

Then, by the induction hypothesis,

$$\langle S_G, \emptyset \rangle \Rightarrow^{n} \langle X_1, U_1 \rangle \langle X_2, U_2 \rangle \dots \langle X_h, U_h \rangle \text{ in } H,$$

where $\bigcup_{1 \leq i \leq h} U_i \subseteq \bigcup_{0 \leq i \leq h} alph(x_i)$, for some $n \geq 0$. Now, let us consider all possible forms of $x \Rightarrow y$ in G:

(i) Let $X_j \rightarrow {}^{\varepsilon}y_0Y_1{}^{\varepsilon}y_1 \dots Y_q{}^{\varepsilon}y_q \in P_G$, where $y_i \in N_G^*$, for all $0 \leq i \leq h$, $Y_i \in V_G$, for all $1 \leq i \leq h$, for some $1 \leq j \leq h$ and $q \geq 1$, so

$$\begin{gathered} {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_j{}^{\varepsilon}x_j \dots X_h{}^{\varepsilon}x_h \Rightarrow \\ {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_{j-1}{}^{\varepsilon}x_{j-1}{}^{\varepsilon}y_0Y_1{}^{\varepsilon}y_1 \dots Y_q{}^{\varepsilon}y_q{}^{\varepsilon}x_jX_{j+1}{}^{\varepsilon}x_{j+1} \dots X_h{}^{\varepsilon}x_h \text{ in } G. \end{gathered}$$

By (1) in Algorithm 4.2.1, there is $\langle X_j, U_j \rangle \rightarrow \langle Y_1, U_j \cup alph(y_0y_1 \dots y_q) \rangle \langle Y_2, \emptyset \rangle \dots \langle Y_q, \emptyset \rangle \in P_H$, so

$$\begin{gathered} \langle X_1, U_1 \rangle \langle X_2, U_2 \rangle \dots \langle X_j, U_j \rangle \dots \langle X_h, U_h \rangle \Rightarrow \\ \langle X_1, U_1 \rangle \langle X_2, U_2 \rangle \dots \langle X_{j-1}, U_{j-1} \rangle \langle Y_1, U_j \cup alph(y_0y_1 \dots y_q) \rangle \\ \langle Y_2, \emptyset \rangle \dots \langle Y_q, \emptyset \rangle \langle X_{j+1}, U_{j+1} \rangle \dots \langle X_h, U_h \rangle \text{ in } H. \end{gathered}$$

Clearly, $(\bigcup_{1 \leq i \leq h} U_i) \cup (\bigcup_{0 \leq i \leq q} alph(y_i)) \subseteq (\bigcup_{0 \leq i \leq h} alph(x_i)) \cup (\bigcup_{0 \leq i \leq q} alph(y_i))$.

(ii) Let $x_j = x_j'Ax_j''$ and $A \rightarrow {}^{\varepsilon}y \in P_G$, where $y \in N_G^*$ and $x_j', x_j'' \in N_G^*$, so

$$\begin{gathered} {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_j{}^{\varepsilon}x_j \dots X_h{}^{\varepsilon}x_h \Rightarrow \\ {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_{j-1}{}^{\varepsilon}x_{j-1}X_j{}^{\varepsilon}x_j'{}^{\varepsilon}y{}^{\varepsilon}x_j''X_{j+1}\ {}^{\varepsilon}x_{j+1} \dots X_h{}^{\varepsilon}x_h \text{ in } G. \end{gathered}$$

If $A \notin \bigcup_{1 \leq i \leq h} U_i$, then

$$\begin{gathered} \langle X_1, U_1 \rangle \langle X_2, U_2 \rangle \dots \langle X_h, U_h \rangle \Rightarrow^0 \\ \langle X_1, U_1 \rangle\ \langle X_2, U_2 \rangle \dots \langle X_h, U_h \rangle \text{ in } H \end{gathered}$$

and clearly $\bigcup_{1 \leq i \leq h} U_i \subseteq (\bigcup_{0 \leq i \leq h, i \neq j} alph(x_i)) \cup alph(x_j'yx_j'')$, so assume that $A \in \bigcup_{1 \leq i \leq h} U_i$. By (2) in Algorithm 4.2.1, there is $\langle X_k, U_k \rangle \rightarrow \langle X_k, (U_k - \{A\}) \cup alph(y) \rangle \in P_H$, where $U_k = U_k' \cup \{A\}$, $U_k' \subseteq N_G$, for some $1 \leq k \leq h$, so

$$\langle X_1, U_1\rangle\langle X_2, U_2\rangle \dots \langle X_k, U_k\rangle \dots \langle X_h, U_h\rangle \Rightarrow$$
$$\langle X_1, U_1\rangle\langle X_2, U_2\rangle \dots \langle X_k, (U_k - \{A\}) \cup alph(y)\rangle \dots \langle X_h, U_h\rangle \text{ in } H.$$

Clearly, $(\bigcup_{1\leq i\leq h, i\neq k} U_i) \cup (U'_k \cup alph(y)) \subseteq (\bigcup_{0\leq i\leq h, i\neq j} alph(x_i)) \cup alph(x'_j y x''_j)$.

Observe that these two cases cover all possible derivations of the form $x \Rightarrow w$ in G. Thus, the *Only If* part of Claim 4.2.1 holds.

If: This is also established by induction, but in this case on n.

Basis: Let $n = 0$. For $\langle S_G, \emptyset\rangle \Rightarrow^0 \langle S_G, \emptyset\rangle$ in H there is $S_G \Rightarrow^0 S_G$ in G, so the basis holds.

Induction Hypothesis: Suppose that the *If* part of Claim 4.2.1 holds for all derivations of length n or less, for some $n \geq 0$.

Induction Step: Consider any derivation of the form

$$\langle S_G, \emptyset\rangle \Rightarrow^{n+1} w \text{ in } H,$$

where $w \in N_H^+$. Since $n + 1 > 0$, this derivation can be expressed as

$$\langle S_G, \emptyset\rangle \Rightarrow^n x \Rightarrow w,$$

where $x \in N_H^+$. Let $x = \langle X_1, U_1\rangle\langle X_2, U_2\rangle \dots \langle X_h, U_h\rangle$, where $X_i \in V_G$, $U_i \in N_G^*$, for all $1 \leq i \leq h$, for some $h \geq 1$. By the induction hypothesis, there is

$$S_G \Rightarrow^m {}^{\varepsilon}x_0 X_1 {}^{\varepsilon}x_1 X_2 {}^{\varepsilon}x_2 \dots X_h {}^{\varepsilon}x_h \text{ in } G,$$

where $x_i \in N_G^*$, for all $1 \leq i \leq h$, such that $\bigcup_{1\leq i\leq h} U_i \subseteq \bigcup_{0\leq i\leq h} alph(x_i)$, for some $m \geq 0$. Now, let us consider all possible forms of $x \Rightarrow w$ in H:

(i) Let $\langle X_j, U_j\rangle \to \langle Y_1, W\rangle\langle Y_2, \emptyset\rangle \dots \langle Y_q, \emptyset\rangle \in P_H$, where $W \subseteq N_G$ and $Y_i \in N_G$, for all $1 \leq i \leq q$, for some $q \geq 1$, so

$$\langle X_1, U_1\rangle\langle X_2, U_2\rangle \dots \langle X_j, U_j\rangle \dots \langle X_h, U_h\rangle \Rightarrow$$
$$\langle X_1, U_1\rangle\langle X_2, U_2\rangle \dots \langle X_{j-1}, U_{j-1}\rangle\langle Y_1, W\rangle\langle Y_2, \emptyset\rangle$$
$$\dots \langle Y_q, \emptyset\rangle\langle X_{j+1}, U_{j+1}\rangle \dots \langle X_h, U_h\rangle \text{ in } H.$$

By (1) in Algorithm 4.2.1, W is of the form $W = U_j \cup alph(y_0 y_1 \dots y_q)$, where $y_i \in N_G^*$, for all $1 \leq i \leq q$, so there is $X_j \to {}^{\varepsilon}y_0 Y_1 {}^{\varepsilon}y_1 \dots Y_q {}^{\varepsilon}y_q \in P_G$, so

$${}^{\varepsilon}x_0 X_1 {}^{\varepsilon}x_1 X_2 {}^{\varepsilon}x_2 \dots X_j {}^{\varepsilon}x_j \dots X_h {}^{\varepsilon}x_h \Rightarrow$$
$${}^{\varepsilon}x_0 X_1 {}^{\varepsilon}x_1 X_2 {}^{\varepsilon}x_2 \dots X_{j-1} {}^{\varepsilon}x_{j-1} {}^{\varepsilon}y_0 Y_1 {}^{\varepsilon}y_1 \dots Y_q {}^{\varepsilon}y_q {}^{\varepsilon}x_j X_{j+1} {}^{\varepsilon}x_{j+1} \dots X_h {}^{\varepsilon}x_h \text{ in } G.$$

Clearly, $(\bigcup_{1\leq i\leq h} U_i) \cup (\bigcup_{0\leq i\leq q} alph(y_i)) \subseteq (\bigcup_{0\leq i\leq h} alph(x_i)) \cup (\bigcup_{0\leq i\leq q} alph(y_i))$.

(ii) Let $\langle X_j, U_j\rangle \to \langle X_j, W\rangle \in P_H$ for some $1 \leq j \leq h$, where $W \subseteq N_G$, so

$$\langle X_1, U_1\rangle\langle X_2, U_2\rangle \dots \langle X_j, U_j\rangle \dots \langle X_h, U_h\rangle \Rightarrow$$
$$\langle X_1, U_1\rangle\langle X_2, U_2\rangle \dots \langle X_j, W\rangle \dots \langle X_h, U_h\rangle \text{ in } H.$$

By (2) in Algorithm 4.2.1, W is of the form $W = (U_j - \{A\}) \cup alph(z)$, where $A \in N_G$ and $z \in N_G^*$, such that there is $A \to {}^{\varepsilon}z \in P_G$. Recall that $\bigcup_{1 \leq i \leq h} U_i \subseteq \bigcup_{0 \leq i \leq h} alph(x_i)$ by the induction hypothesis. Since $A \in \bigcup_{1 \leq i \leq h} U_i$, some x_k has to be of the form $x_k = x'_k A x''_k$, where $x'_k, x''_k \in N_G^*$, so

$$
\begin{gathered}
{}^{\varepsilon}x_0 X_1 {}^{\varepsilon}x_1 X_2 \, {}^{\varepsilon}x_2 \ldots X_k {}^{\varepsilon}x_k \ldots X_h {}^{\varepsilon}x_h \Rightarrow \\
{}^{\varepsilon}x_0 X_1 {}^{\varepsilon}x_1 X_2 {}^{\varepsilon}x_2 \ldots X_{k-1} {}^{\varepsilon}x_{k-1} X_k {}^{\varepsilon}x'_k {}^{\varepsilon}z {}^{\varepsilon}x''_k X_{k+1} {}^{\varepsilon}x_{k+1} \ldots X_h \, {}^{\varepsilon}x_h \text{ in } G.
\end{gathered}
$$

Clearly, $(\bigcup_{1 \leq i \leq h, i \neq j} U_i) \cup (U_j - \{A\}) \cup alph(z) = (\bigcup_{1 \leq i \leq h, i \neq k} alph(x_i)) \cup (alph(x_k) - \{A\}) \cup alph(z)$.

Observe that these two cases cover all possible derivations of the form $x \Rightarrow w$ in H. Thus, the *If* part of Claim 4.2.1 also holds. Hence, Claim 4.2.1 holds. □

Now, consider a special case of Claim 4.2.1 when $x_i = \varepsilon$, $X_j \in T_G$, for all $0 \leq i \leq h$, $1 \leq j \leq h$. Then,

$$S_G \Rightarrow^m X_1 X_2 \ldots X_h \text{ in } G$$

if and only if

$$\langle S_G, \emptyset \rangle \Rightarrow^n \langle X_1, \emptyset \rangle \langle X_2, \emptyset \rangle \ldots \langle X_h, \emptyset \rangle \text{ in } H,$$

for some $m, n \geq 0$ and $h \geq 1$. By the initialization part of Algorithm 4.2.1, there are rules $\langle X_i, \emptyset \rangle \to X_i \in P_H$, for all $1 \leq i \leq h$, so

$$\langle X_1, \emptyset \rangle \langle X_2, \emptyset \rangle \ldots \langle X_h, \emptyset \rangle \Rightarrow^h X_1 X_2 \ldots X_h \text{ in } H.$$

Therefore, Lemma 4.2.1 holds. □

Using results from this section, we obtain another way how to prove Theorem 4.1.1.

Theorem 4.2.1. *Let G be a context-free grammar. Then there is an ε-free context-free grammar, H, such that $L(H) = L(G) - \{\varepsilon\}$.*

Proof. This theorem follows from Algorithm 4.2.1 and Lemma 4.2.1. □

Chapter 5

Removal of Erasing Rules from Regulated Grammars

From Chapter 4, we know that we can always eliminate erasing rules from any context-free grammar. In Chapter 3, we see that most regulated grammars are based on context-free grammars, so one might think that we can use the same approach in regulated grammars. However, there is a slight difference between context-free grammars and regulated grammars—the regulation. The reason why these techniques work in case of context-free grammars is that there is no restriction in regard which rule we should apply and when we should apply. When we introduce regulation, the order of rules become crucial. Also, the number of occurrences of a nonterminal that are erased in the rest of the derivation may be important. With incautious changes to the grammar, one could end up with a grammar generating a different language.

The type of problems we might run into depends on the type of a regulated grammar. For the purpose of this introduction, let us consider regularly controlled grammars. Nevertheless, since regularly controlled grammars, matrix grammars, and programmed grammars are somewhat similar (see Chapter 3), the same kind of reasoning apply to these grammars as well. Let $H = (G, \Xi)$ be a regularly controlled grammar, where $G = (N, T, S, P)$, and let us consider the following idea. We run Algorithm 4.1.2 or Algorithm 4.2.1 from Chapter 4 on the core grammar G and, at the same time, we accordingly modify the control language. Depending of the algorithm we use, there are two cases:

(1) We use the standard algorithm. Recall that this algorithm works by introducing more rules for a single rule, depending on the number of ε-nonterminals present on the right-hand side of that rule. Let $r\colon A \to bBcC \in P$ and $B, C \in N_\varepsilon$, so the algorithm introduces rules $r_1\colon A \to bBcC$, $r_2\colon A \to bcC$, $r_3\colon A \to bBc$, and $r_4\colon A \to bc$. Then, we substitute the use of r in the original control language with $\{r_1, r_2, r_3, r_4\}$. However, if we use r_2 and, later on, the control language prescribes us to use a rule with B on its left-hand side, there might be no B in the sentential form. If we use r_2, we should also substitute (or remove) rules with B on their left-hand side from the control language and so on. But, as you can see, this is not really an easy thing to formalize and to proof, because we have to be certain that the generated language of the resulting ε-free grammar is the same as the language generated by the original grammar.

(2) We use the new algorithm which introduces nonterminals of the form $\langle X, U\rangle$, where $X \in V$ and $U \subseteq N$. Since U is a set, several occurrences of a nonterminal are stored as only one occurrence. Let $r\colon A \to BBBB$ and $s\colon B \to \varepsilon$ be rules in P and $\langle C, \{A\}\rangle$ be a nonterminal in the resulting ε-free grammar. If we rewrite A in $\langle C, \{A\}\rangle$ by using r, $\langle C, \{A\}\rangle \Rightarrow \langle C, \{B\}\rangle$ $[r]$, there might be a sequence of four s rules in the control language, but we have only one B in the second component. By the same reasoning as in the first case, with incautious changes to the control language, we might end up with a different language.

So, there are problems in both cases. It is not known whether these approaches can be made working (Chapter 3 or [44]).

Section 5.1 contains present results regarding the elimination of erasing rules from regulated grammars. Then, a new result on this topic is presented in Section 5.2. Finally, Section 5.3 discusses a significance of these results concerning syntactical analysis.

5.1 Present Results

To summarize the results regarding elimination of erasing rules from regulated grammars from Chapter 3,

- we are able to eliminate erasing rules from all indexed grammars and all Indian parallel grammars,
- there exist scattered context grammars and tree controlled grammars with erasing rules which do not have their equivalent ε-free versions, and
- it is not known whether we are able to remove all erasing rules from regularly controlled grammars, matrix grammars, programmed grammars, random context grammars, ordered grammars, and Russian parallel grammars.

Of course, all eliminations have to be done without any affection of the generated language. Also, from Theorem 3.2.1 and Theorem 3.3.1, we obtain the following corollary.

Corollary 5.1.1.

(1) $\boldsymbol{rC} = \boldsymbol{rC^{\varepsilon}}$ *if and only if* $\boldsymbol{M} = \boldsymbol{M^{\varepsilon}}$,

(2) $\boldsymbol{M} = \boldsymbol{M^{\varepsilon}}$ *if and only if* $\boldsymbol{P} = \boldsymbol{P^{\varepsilon}}$.

(3) $\boldsymbol{P} = \boldsymbol{P^{\varepsilon}}$ *if and only if* $\boldsymbol{rC} = \boldsymbol{rC^{\varepsilon}}$.

Thus, by solving one of these inclusions, we solve them all. The next sections present known techniques, results, and consequences regarding the possibility of elimination of erasing rules from regulated grammars.

5.1.1 Limited Erasing in Scattered Context Grammars

As stated in Chapter 3 (more precisely, in Theorem 3.5.1), there are scattered context grammars with erasing rules which cannot be converted to equivalent ε-free scattered context grammars. However, there is a condition, called *k-limited erasing in scattered context grammars*, introduced by Meduna and Techet in [56] (see also [57, Section 4.2]), which guarantees that we can eliminate all erasing rules from any scattered context grammar which satisfies this condition. Informally, a scattered context grammar erases its nonterminals in a k-limited way, where $k \geq 0$, if for every sentence there exists a derivation such that in every sentential form of that derivation, each of its substrings consisting of nonterminals from which the grammar derives empty strings later in the derivation is of length k or less [56].

To define the condition formally, an auxiliary notion of the underlying derivation by a core grammar is needed.

Definition 5.1.1. Let $H = (G, R)$ be a scattered context grammar, where $G = (N, T, S, P)$, and let

$$v \Rightarrow_H w \ [r]$$

be a derivation in H, where $v = u_1A_1u_2A_2 \dots u_nA_nu_{n+1}$, $w = u_1x_1u_2x_2 \dots u_nx_nu_{n+1}$, and $r = (A_1 \to x_1, A_2 \to x_2, \dots, A_n \to x_n)$, $u_i \in V^*$, for all $1 \leq n+1$, for some $n \geq 1$. The *partial m-step context-free simulation* of this derivation step by G is denoted by $cf_m(v \Rightarrow_H w)$ and defined as an m-step derivation in G of the form

$$\begin{array}{ll} & u_1A_1u_2A_2 \dots u_nA_nu_{n+1} \\ \Rightarrow_G & u_1x_1u_2A_2 \dots u_nA_nu_{n+1} \\ \Rightarrow_G^{m-1} & u_1x_1u_2x_2 \dots u_mx_mu_{m+1}A_{m+1} \dots u_nA_nu_{n+1}, \end{array}$$

where $m \leq n$. The *context-free simulation* of a derivation step of H, denoted by $cf(v \Rightarrow_H w)$, is the partial n-step context-free simulation of this step. Let $v = v_1 \Rightarrow_H^* v_n = w$ be of the form

$$v_1 \Rightarrow_H v_2 \Rightarrow_H \dots \Rightarrow_H v_n.$$

The context-free simulation of $v \Rightarrow_H^* w$ by G is denoted by $cf(v \Rightarrow_H^* w)$ and defined as

$$v_1 \Rightarrow_G^* v_2 \Rightarrow_G^* \dots \Rightarrow_G^* v_n$$

such that for all $1 \leq i \leq n-1$, $v_i \Rightarrow_G^* v_{i+1}$ is the context-free simulation of $v_i \Rightarrow_H v_{i+1}$.

To make the following definition more readable, we use Notation 2.4.1.

Definition 5.1.2. Let $H = (G, R)$ be a scattered context grammar, where $G = (N, T, S, P)$, and let $k \geq 0$. *H erases its nonterminals in a k-limited way*, if for every $y \in L(H) - \{\varepsilon\}$, there exists a derivation

$$S \Rightarrow_H^* y$$

such that every sentential form x in $cf(S \Rightarrow_H^* y)$ can be expressed in the form $x = {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_n{}^{\varepsilon}x_n$, $x_i \in N^*$, $X_j \in V$, which satisfies $|x_i| \leq k$, for all $0 \leq i \leq n$, $1 \leq j \leq n$, for some $n \geq 1$.

Example 5.1.1. Consider the scattered context grammar H from Example 3.5.1. Observe that every derivation leading to a nonempty sentence, $a^nb^nc^n$, for some $n \geq 1$, is of the form

$$S \Rightarrow_H AAA \Rightarrow_H aAbAcA \Rightarrow_H \dots \Rightarrow_H a^nAb^nAc^nA \Rightarrow_H a^nb^nc^n.$$

From this observation it is clear that H erases its nonterminals in a 1-limited way (the only sentential form, where some nonterminals are started to be erased, is $a^nAb^nAc^nA$, where all three As are erased, and since $n \geq 1$, the condition from Definition 5.1.2 is satisfied).

For a more complex example regarding this condition, refer to [56] or [57, Section 4.2].

Theorem 5.1.1. *(See [57, Theorem 4.15]) For every scattered context grammar, H, that erases its nonterminals in a k-limited way, there exists an ε-free scattered context grammar, O, such that $L(O) = L(H) - \{\varepsilon\}$.*

5.1.2 Recursive Erasing in Programmed Grammars

Křivka in [39] placed a condition on rules in programmed grammars, called *recursively nonterminal-erasing set*, and proved that all erasing rules satisfying this condition, called *recursively erasing rules*, can be removed from such a programmed grammar. Informally, a recursively erasing rule is a rule that if we start to erase some nonterminal with it, then we have to erase all occurrences of this nonterminal in the current sentential form [39].

Before we introduce this condition formally and give an example, we need the following auxiliary definition.

Definition 5.1.3. Let $H = (G, \sigma, \varphi)$ be a programmed grammar (with appearance checking), where $G = (N, T, S, P)$. If there is a sequence of rules, $\alpha = \lfloor r_1 \colon A_1 \to w_1, \sigma(r_1), \varphi(r_1) \rfloor \dots \lfloor r_n \colon A_n \to w_n, \sigma(r_n), \varphi(r_n) \rfloor \in P^+$, for some $n \geq 1$, such that $A_1 \Rightarrow_H^+ \varepsilon\ [\alpha]$, then α is called *A_1-erasing sequence of rules*.

Definition 5.1.4. Let $H = (G, \sigma, \varphi)$ be a programmed grammar (with appearance checking), where $G = (N, T, S, P)$. For every A_1-erasing sequence of rules, $\alpha = \lfloor r_1 \colon A_1 \to w_1, \sigma(r_1), \varphi(r_1) \rfloor \dots \lfloor r_n \colon A_n \to w_n, \sigma(r_n), \varphi(r_n) \rfloor \in P^+$, for some $n \geq 1$, let $Q_{A_1} \subseteq P$ be a set of rules such that

(1) $\lfloor r_1 \colon A_1 \to w_1, \sigma(r_1), \varphi(r_1) \rfloor \in Q_{A_1}$,

(2) there is no derivation, $A_1 \Rightarrow_H^+ x\ [r_1\beta]$, where $x \in V^*TV^*$, $\beta \in P^*$, and

(3) $s \in \sigma(r_n)$ imply $\lfloor s \colon A_1 \to y, \sigma(s), \varphi(s) \rfloor \in Q_{A_1}$.

Q_{A_1} is called *recursively A_1-erasing set of rules*.

Set[1]

$$_{\text{rec-}\varepsilon}Q_H = \bigcup_{A \in (N - \{S\})} Q_A,$$

$$_{\text{rec-}\varepsilon}\Theta_H = \{r_1 \dots r_n \mid r_1 \in {}_{\text{rec-}\varepsilon}Q_H, rhs(r_n) = \varepsilon, \text{ and } r_1, \dots, r_n \in P \text{ for some } n \geq 1\},$$

$$_{\text{rec-}\varepsilon}P_H = \bigcup_{\alpha \in {}_{\text{rec-}\varepsilon}\Theta_H} alph(\alpha).$$

Example 5.1.2. (Modified example from [39]) Consider the programmed grammar with appearance checking, $H = (G, \sigma, \varphi)$, where $G = (N, T, S, P)$ is a context-free grammar with $N = \{S, A, B\}$, $T = \{a\}$, and P consisting of the following rules:

$$\lfloor r_1 \colon S \to ABAB, \{r_1\}, \{r_2, r_3\} \rfloor,$$

$$\lfloor r_2 \colon A \to S, \{r_2\}, \{r_1\} \rfloor,$$

$$\lfloor r_3 \colon B \to \varepsilon, \{r_3\}, \{r_4\} \rfloor,$$

$$\lfloor r_4 \colon A \to a, \{r_4\}, \emptyset \rfloor.$$

Clearly, $L(H) = \{a^{2^n} \mid n \geq 1\}$ (observe the similarity with Example 3.3.1). Since only r_3 satisfies the condition of recursive erasing (once we use it, we have to erase all occurrences of B in the current sentential form), $_{\text{rec-}\varepsilon}P_H = \{\lfloor r_3 \colon B \to \varepsilon, \{r_3\}, \{r_4\} \rfloor\}$.

Theorem 5.1.2. *(See [39, Theorem 2]) Let H be programmed grammar with appearance checking. Then, there is a programmed grammar with appearance checking, O, such that $L(H) = L(O)$ and $_{\text{rec-}\varepsilon}P_O = \emptyset$.*

The same result holds also for programmed grammars without appearance checking (see [39, Corollary 3]). In the proof of Theorem 5.1.2 in [39], a similar idea to the one of Algorithm 4.2.1 is used.

5.1.3 Generation of Extended Languages by ε-Free Grammars

In addition to the result presented in Section 5.1.1, there is another way how to transform scattered context grammars to ε-free scattered context grammars. Let H be a scattered context grammar. We can replace the empty string in H's rules with a special terminal symbol, usually denoted by \$ or #, which does not appear in any string that H generates, thus obtaining an ε-free scattered context grammar H'. However, the strings generated by H' have to be of some special form so the new symbol is not placed randomly in generated sentences. For example, one may consider to generate sentences followed by a sequence of \$s. If we then remove this sequence of \$s from the end of the *extended sentence*, we obtain the original sentence. Generated languages containing strings of these special forms are called *extended languages* throughout this section[2].

[1] Let us note that we use a slightly different notation than the one used in [39] to avoid confusion with our denotation of the set of rule labels by Ψ.

[2] Let us note that our notion of extension is not related to so-called *extended scattered context grammars* (see [49] or [57, Section 5.6]) in any way.

Two types of generation of extended languages are presented in this section: coincidental extensions and quotients of languages generated by ε-free scattered context grammars. However, note that this concept is more general and can be applied to other grammars as well.

Coincidental Extension

For a symbol, #, and a string, $x = a_1a_2 \dots a_{n-1}a_n$, any string of the form $\#^i a_1 \#^i a_2 \#^i \dots \#^i a_{n-1} \#^i a_n \#^i$, where $i \geq 0$, represents a coincidental #-extension of x. A language, K, is a coincidental #-extension of L if every string in K represents a coincidental #-extension of a string in L and the deletion of all #s in K results in L. This type of extension was introduced by Meduna in [52] (see erratum [53]).

The notion of a coincidental extension is formalized in the following definition.

Definition 5.1.5. Let V be an alphabet and $\# \notin V$ a symbol. A *coincidental #-extension* of ε is any string of the form $\#^i$, where $i \geq 0$. A coincidental #-extension of $a_1a_2 \dots a_{n-1}a_n \in V^+$ is any string of the form $\#^i a_1 \#^i a_2 \#^i \dots \#^i a_{n-1} \#^i a_n \#^i$, where $n \geq 1$, $a_j \in V$, for all $1 \leq j \leq n$, for some $i \geq 0$. For any language, $L \subseteq V^*$, ${}_{\#}CE(L)$ denotes the set of all *c*oincidental #-*e*xtensions of strings in L. Define the homomorphism, $\omega \colon (V \cup \{\#\})^* \to (V \cup \{\#\})^*$, as $\omega(a) = a$ for every $a \in (V - \{\#\})$ and $\omega(\#) = \varepsilon$. Let $K \subseteq (V \cup \{\#\})^*$. K is a *coincidental #-extension* of L, symbolically written as $L \, {}_{\#}\blacktriangleleft \, K$, if $K \subseteq {}_{\#}CE(L)$ and $L = \{\omega(x) \mid x \in K\}$.

Example 5.1.3. (From [52]) Consider three languages, $K = \{\#^i a \#^i b \#^i \mid i \geq 5\} \cup \{\#^i c^n \#^i d^n \#^i \mid n, i \geq 0\}$, $L = \{ab\} \cup \{c^n d^n \mid n \geq 0\}$, and $M = \{\#^i a \#^i b \#^i \mid i \geq 5\} \cup \{\#^i c^n \#^i d^n \#^{i+1} \mid n, i \geq 0\}$. Observe that $L \, {}_{\#}\blacktriangleleft \, K$, but there exists no language, N, satisfying $N \, {}_{\#}\blacktriangleleft \, M$.

Theorem 5.1.3. *(See [52, Corollary 2]) Let $K \in \boldsymbol{RE}$. Then, there exists an ε-free scattered context grammar, H, such that $K \, {}_{\#}\blacktriangleleft \, L(H)$.*

Since $\mathbf{SC}^{\varepsilon} = \mathbf{RE}$ (from Theorem 3.5.1), we obtain the following corollary.

Corollary 5.1.2. *For every scattered context grammar, G, there is an ε-free scattered context grammar, H, such that $L(G) \, {}_{\#}\blacktriangleleft \, L(H)$.*

Quotients

Instead of placing special symbols in between other symbols (like a coincidental extension does), one can place them in front of or in back of a sentence. For example, the sentence *aaabbb* can be extended to *aaabbb*\$\$\$\$ by a special symbol, \$. When we omit the suffix \$\$\$\$, we get a so-called symbol-exhaustive right quotient of *aaabbb*\$\$\$\$.

The notion of quotients is formalized in the next two definitions.

Definition 5.1.6. Let L_1 and L_2 be two arbitrary languages. The *right quotient* of L_1 with respect to L_2, denoted by L_1/L_2, is defined as

$$L_1/L_2 = \{w \mid wx \in L_1 \text{ for some } x \in L_2\}.$$

Similarly, the *left quotient* of L_1 with respect to L_2, denoted by $L_1 \backslash L_2$, is defined as

$$L_1 \backslash L_2 = \{w \mid xw \in L_2 \text{ for some } x \in L_1\}.$$

Definition 5.1.7. Let L_1 and L_2 be two arbitrary languages. The *exhaustive right quotient* of L_1 with respect to L_2, denoted by $L_1 // L_2$, is defined as

$$L_1 // L_2 = \{w \mid w \in L_1 / L_2 \text{ and } w \text{ is not a proper prefix of any string in } L_1 / L_2\}.$$

Similarly, the *exhaustive left quotient* of L_1 with respect to L_2, denoted by $L_2 \backslash\backslash L_1$, is defined as

$$L_2 \backslash\backslash L_1 = \{w \mid w \in L_1 \backslash L_2 \text{ and } w \text{ is not a proper suffix of any string in } L_1 \backslash L_2\}.$$

Let $L_2 = \{\$\}^*$, where \$ is a symbol. Then $L_1 // L_2$ is the *symbol-exhaustive right quotient* of L_1 with respect to \$ and $L_2 \backslash\backslash L_1$ is the *symbol-exhaustive left quotient* of L_1 with respect to \$.

Example 5.1.4. Consider two languages, $K = \{a^n b^n c^n \mid n \geq 0\}$ and $L = \{a^n b^n c^n \$^m \mid m, n \geq 0\}$. Observe that K is the symbol-exhaustive right quotient of L with respect to \$, i.e. $K = L // \{\$\}^*$.

The following result was first proved by Ehrenfeucht and Rozenberg in [18]. Later, it was improved by Meduna in [49] in terms of the number of nonterminals needed in the resulting grammar.

Theorem 5.1.4. *(See [49, Theorem 4]) Let $K \in$ **RE** and let \$ be a symbol such that \$ $\notin alph(w)$ for all $w \in K$. Then, there is an ε-free scattered context grammar, H, such that $K = L(H) // \{\$\}^*$.*

Again, since $\mathbf{SC}^{\varepsilon} = \mathbf{RE}$ (from Theorem 3.5.1), we obtain the following corollary.

Corollary 5.1.3. *For every scattered context grammar, G, and a symbol, \$ $\notin alph(w)$ for all $w \in L(G)$, there is an ε-free scattered context grammar, H, such that $L(G) = L(H) // \{\$\}^*$.*

Similar results hold also for the symbol-exhaustive left quotient [49, Theorem 4].

In [55] and [57, Chapter 7], instead of an insignificant symbol, \$, Meduna and Techet considered generation of sentences followed by their parses. For example, let $G = (N, T, S, P)$ by a context-free grammar. For a successful derivation in G, $S \Rightarrow y$ $[\alpha]$, where $y \in L(G)$, $y\alpha$ is the sentence y followed by its parse. These languages consisting of sentences followed by their parses are referred to as *extended Szilard languages* in [11, Section 7.2]. Meduna and Techet managed to prove similar result to Theorem 5.1.4 in terms of generation of sentences followed by their parses (let us note that Dassow and Păun in [11, Section 7.2] did not considered generation of extended Szilard languages of grammars with erasing rules by ε-free grammars; they only considered generation of Szilard languages of grammars with erasing rules by ε-free grammars, i.e. only rule labels were generated, no original terminals).

5.1.4 Erasing in Petri Net Languages and Matrix Grammars

As Petri nets are related to specific models from the formal language theory [28], one can use particular results from the theory of Petri nets and apply them in the formal language theory. For example, using deep results of the theory of Petri nets and a relation between matrix grammars and Petri nets, it was shown that all languages generated by matrix grammars over one-symbol alphabets are regular [28, Theorem 5.3].

Recently, Zetche in [76] and [77] proved two results concerning Petri nets and Petri net controlled grammars, which are related to the topic of this work. The first one sheds a new light on the question whether it is possible to eliminate erasing rules from any matrix grammar (and thus also from any regularly controlled grammar and from any programmed grammar). The second one shows that if a particular conjecture regarding decidability of a Petri nets problem holds, then we obtain an interesting property of the family of all matrix languages. These two results are briefly discussed in the two following sections.

Petri Net Controlled Grammars and Matrix Grammars

Let **PN** and $\mathbf{PN}^{\varepsilon}$ denote the family of all languages generated by Petri net controlled grammars without erasing rules and with erasing rules, respectively. In [12], it is shown that Petri net controlled grammars with erasing rules generate the same language family as matrix grammars with erasing rules and without appearance checking.

Theorem 5.1.5. *(See [12])* $\boldsymbol{PN^{\varepsilon} = M^{\varepsilon}}$

For completeness, in [76], as stated in Section 3.6, it is shown that Petri net controlled grammars have the same generative power regardless of the presence of erasing rules.

Theorem 5.1.6. *(See [76, Theorem 3])* $\boldsymbol{PN^{\varepsilon} = PN}$

It is also shown there that when we restrict to so-called *linear Petri net controlled grammars*[3], their language class coincides with **M**. Let **LinPN** denote the family of all languages generated by linear Petri net controlled grammars.

Theorem 5.1.7. *(See [76, Theorem 2])* $\boldsymbol{M = LinPN}$

Since every linear Petri net controlled grammar is a special case of a Petri net controlled grammar, i.e. **LinPN** $\subseteq$ **PN** [76, Definition 4], from Theorems 5.1.5, 5.1.6, and 5.1.7, we obtain the following corollary.

Corollary 5.1.4. $\boldsymbol{M = M^{\varepsilon}}$ *if and only if* $\boldsymbol{LinPN = PN}$.

Thus, by proving the exact relation between **LinPN** and **PN**, we also prove the exact relation between **M** and $\mathbf{M}^{\varepsilon}$. Note that by Corollary 5.1.1, we can straightforwardly reformulate Corollary 5.1.4 in terms of regularly controlled grammars and programmed grammars.

[3] Since this is beyond the scope of this work, please refer to [76] for details.

Parikh Images of Matrix Languages

First, we make some notions and definitions to facilitate the upcoming discussion. For a string, w, over some arbitrary alphabet, Σ, let $|w|_a$ denote the number of occurrences of a symbol, $a \in \Sigma$, in w. Informally, a Parikh image of a string is a string in which we consider only the relative number of occurrences of symbols, not the order in which they appear in the string. This idea, which is due to Parikh [61], is formalized in the following definition.

Definition 5.1.8. Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be an alphabet, where $n \geq 1$ and the order of $a_1, a_2, \ldots, a_n$ is arbitrary, but fixed (i.e. one order is chosen, but which one does not matter). For a string, $w \in \Sigma^*$, the *Parikh image* of w, denoted by $\Phi(w)$, is defined as

$$\Phi(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n}).$$

When referring to the Parikh image of a language (language family), this should be taken to mean the image applied to all the strings in that language (to all the languages in that family). This idea is expressed in the next definition.

Definition 5.1.9. Let Σ be an alphabet. For a language, $L \subseteq \Sigma^*$, let $\Phi(L) = \{\Phi(w) \mid w \in L\}$. For a language family, $\mathscr{L}$, let $\Phi(\mathscr{L}) = \{\Phi(L) \mid L \in \mathscr{L}\}$.

Example 5.1.5. Let $\Sigma = \{a, b, c\}$ be an alphabet and let the order of a, b, and c be lexicographical, i.e. (a, b, c). For a string, $x = ccbbbbc$, and two languages, $K = \{a^n b^n c^n \mid n \geq 0\}$ and $L = \{ww \mid w \in \Sigma^*\}$, we have $\Phi(x) = (0, 4, 3)$, $\Phi(K) = \{(i, i, i) \mid i \geq 0\}$, and $\Phi(L) = \{(2i, 2j, 2k) \mid i, j, k \geq 0\}$.

Zetche proved that if so-called *Hack's Conjecture* (see [27]) holds, then the families $\Phi(\mathbf{M})$ and $\Phi(\mathbf{M}^\varepsilon)$ coincides. This conjecture says that the reachability problem for Petri nets is decidable[4].

Theorem 5.1.8. *(See [77, Theorem 7]) If Hack's Conjecture holds, then* $\Phi(\boldsymbol{M}) = \Phi(\boldsymbol{M}^\varepsilon)$.

Note that $\Phi(\mathbf{M}) = \Phi(\mathbf{M}^\varepsilon)$ does not imply $\mathbf{M} = \mathbf{M}^\varepsilon$ nor $\mathbf{M} \neq \mathbf{M}^\varepsilon$. It just shows a property of the family of all matrix languages, regardless of the presence of erasing rules.

5.2 New Result: Limited Erasing in Regularly Controlled Grammars

The main problem with the second approach from the introduction to this chapter is that several occurrences of the same nonterminal are reduced into a single occurrence. One way to solve this problem is to record every occurrence of the same nonterminal, thus keeping strings of nonterminals rather than sets of nonterminals. However, since there can potentially be an infinite number of occurrences of nonterminals in a sentential form and we only consider finite alphabets, we need to place some limit on the number of occurrences of nonterminals which are erased during the rest of the derivation. Then, we are able to

[4] Again, since this is beyond the scope of this work, for more information about decidability, please refer to [75, Chapter 10], and for more information about the reachability problem in Petri nets, refer to [27].

remove erasing rules from regularly controlled grammars meeting this limit. The following definition is a formalization of such a limit.

Throughout this section, for any nonnegative rational number, i, $\aleph(i)$ denotes the greatest nonnegative integer smaller or equal to i.

Definition 5.2.1. Let k be a nonnegative integer and $H = (G, \Xi)$ be a regularly controlled grammar, where $G = (N, T, S, P)$. *H erases its nonterminals in a k-limited way* provided that it satisfies this implication: if $S \Rightarrow^* y$ in G is a derivation of the form $S \Rightarrow^* x \Rightarrow^* y$, where $x \in V^+$ and $y \in L(H) - \{\varepsilon\}$, then in $\Delta(S \Rightarrow^* y)$, there are at most $\aleph(k|x|/(k+1))$ ε-subtrees rooted at the symbols of x.

Note that when $k = 0$, then in any derivation leading to a nonempty sentence, no nonterminals are erased.

Example 5.2.1. Consider the regularly controlled grammar H from Example 3.1.1. H erases its nonterminals in a 1-limited way, because for every sentential form, x, in any successful derivation of the form $S \Rightarrow^* x \Rightarrow^* y$ leading to a nonempty sentence, y, $\Delta(S \Rightarrow^* y)$ contains at most $\aleph(|x|/2)$ ε-subtrees rooted at the symbols of x. In other words, no more than half of all symbols in any sentential form are erased.

Example 5.2.2. Let $H = (G, \Xi)$ be a regularly controlled grammar, where $G = (N, T, S, P)$ is a context-free grammar with $N = \{S, A, B\}$, $T = \{a, b, \#\}$, P consisting of the following rules:

$r_1 \colon S \to A\#B$,
$r_2 \colon A \to aA$,
$r_3 \colon B \to aB$,
$r_4 \colon A \to bA$,
$r_5 \colon B \to bB$,
$r_6 \colon A \to \varepsilon$,
$r_7 \colon B \to \varepsilon$,

and $\Xi = \{r_1\}\{r_2r_3, r_4r_5\}^*\{r_6r_7\}$. Clearly, the generated language is $L(H) = \{w\#w \mid w \in \{a, b\}^*\}$. Observe that H does not erase its nonterminals in a 1-limited way, because there is a successful derivation,

$$S \Rightarrow A\#B\ [r_1] \Rightarrow \#B\ [r_6] \Rightarrow \#\ [r_7],$$

leading to a nonempty sentence, #, where $\aleph(|A\#B|/2) = 1$, but both A and B represent ε-subtrees in $\Delta(S \Rightarrow^* \#)$. However, it is easy to verify that H erases its nonterminals in a 2-limited way.

Example 5.2.3. Let $G = (N, T, S, P)$ be a context-free grammar with $N = \{S\}$, $T = \{a\}$, and P consisting of the following three rules:

$r_1 \colon S \to SS$, $r_2 \colon S \to a$, $r_3 \colon S \to \varepsilon$.

Now, consider two regularly controlled grammars, $H_1 = (G, \Xi_1)$ and $H_2 = (G, \Xi_2)$, where $\Xi_1 = \{r_1\}^*\{r_2\}^*\{r_3\}^*$ and $\Xi_2 = \{r_1\}^*\{r_2r_3\}^*$. Notice that $L(H_2) = L(H_1) - \{\varepsilon\}$. H_1 does not erase its nonterminals in a k-limited way for any $k \geq 0$, because there is a successful derivation of the form

$$S \Rightarrow^* S^{m+n+1} \Rightarrow S^m a S^n \Rightarrow^* a\ [\alpha],$$

where $\alpha \in \Xi_1$, $m, n \geq 0$, with no limit on m and n. On the other hand, H_2 erases its nonterminals in a 1-limited way, because every application of the erasing rule r_3 is preceded by the application of r_2.

The following algorithm shows how to eliminate erasing rules from regularly controlled grammars meeting the condition from Definition 5.2.1. It uses the idea outlined in the beginning of this section.

Algorithm 5.2.1. *Elimination of erasing rules from any regularly controlled grammar that erases its nonterminals in a k-limited way.*

Input: A context-free grammar, $G = (N_G, T_G, S_G, P_G)$, and a regular grammar, $H = (N_H, T_H, S_H, P_H)$, such that the regularly controlled grammar $I = (G, L(H))$ erases its nonterminals in a k-limited way.

Output: An ε-free context-free grammar, $O = (N_O, T_O, P_O, S_O)$, and a regular grammar, $Q = (N_Q, T_Q, P_Q, S_Q)$, such that $L(M) = L(I) - \{\varepsilon\}$ for the regularly controlled grammar $M = (O, L(Q))$.

Method: Let us note that in what follows, symbols $\langle$, $\rangle$, $\lfloor$, $\rfloor$, $\lceil$, and $\rceil$ are used to clearly unite more symbols into a single compound symbol. Without any loss of generality, assume that $Z \notin (V_H \cup \Psi_O)$. Initially, set:

$k' = k + max(\{|rhs(r)| \mid r \in \Psi_G\})$;
$T_O = T_G$;
$V_O = T_O \cup \{\langle X, y\rangle \mid X \in V_G, y \in N_G^*, 0 \leq |y| \leq k'\}$;
$S_O = \langle S_G, \varepsilon\rangle$;
$\Psi_O = \{\lfloor\langle a, \varepsilon\rangle \to a\rfloor \mid a \in T_G\}$;
$P_O = \{\lfloor\langle a, \varepsilon\rangle \to a\rfloor \colon \langle a, \varepsilon\rangle \to a \mid a \in T_G\}$;
$T_Q = \Psi_O$;
$V_Q = T_Q \cup N_H \cup \{Z\}$;
$S_Q = S_H$;
$P_Q = \{Z \to \lfloor\langle a, \varepsilon\rangle \to a\rfloor B \mid B \in \{Z, \varepsilon\}, a \in T_G\}$.

Repeat (1) through (3), given next, until none of the sets $\Psi_O, P_O, T_Q, N_Q, P_Q$ can be extended in this way.

(1) **If** $r \colon A \to x_0 X_1 x_1 X_2 x_2 \ldots X_n x_n \in P_G$ and $\langle A, w\rangle, \langle X_1, w x_0 x_1 \ldots x_n\rangle \in N_O$, where $w \in N_G^*$, $x_i \in N_G^*$, $X_j \in V_G$, for all $0 \leq i \leq n$, $1 \leq j \leq n$, for some $n \geq 1$
then add $s = \lfloor r, x_0, X_1 x_1, X_2 x_2, \ldots, X_n x_n\rfloor$ to Ψ_O and to T_Q;
add $s \colon \langle A, w\rangle \to \langle X_1, w x_0 x_1 \ldots x_n\rangle\langle X_2, \varepsilon\rangle \ldots \langle X_n, \varepsilon\rangle$ to P_O;
for each $B \to r \in P_H$, add $B \to sZ$ to P_Q;
for each $B \to rC \in P_H$, $C \in N_H$, add $B \to sC$ to P_Q.

(2) **If** $r \colon A \to w \in P_G$ and $\langle X, uAv\rangle, \langle X, uwv\rangle \in N_O$, where $X \in V_G$, $u, v, w \in N_G^*$
then add $s = \lfloor X, uAv, uwv, r\rfloor$ to Ψ_O and to T_Q;
add $s \colon \langle X, uAv\rangle \to \langle X, uwv\rangle$ to P_O;
for each $B \to r \in P_H$, add $B \to sZ$ to P_Q;
for each $B \to rC \in P_H$, $C \in N_H$, add $B \to sC$ to P_Q.

(3) **If** $\langle X, uAv\rangle$, $\langle Y, w\rangle$, $\langle Y, wA\rangle \in N_O$, where $X, Y \in V_G$, $A \in N_G$, $u, v, w \in N_G^*$
then add $r = \lfloor X, u, A, v, Y, w\rfloor$ and $s = \lfloor X, u, A, v, Y, w, A\rfloor$ to Ψ_O and to T_Q;
add $r\colon \langle X, uAv\rangle \to \langle X, uv\rangle$ and $s\colon \langle Y, w\rangle \to \langle Y, wA\rangle$ to P_O;
for each $B \in N_H$, add $C = \lceil B, r, s\rceil$ to N_Q and add $B \to rC$ and $C \to sB$ to P_Q.

Main Idea. The resulting grammar M uses compound nonterminals of the form $\langle X, y\rangle$, where X is a symbol that is not erased in the rest of derivation and y is a string of nonterminals that are erased in the rest of the derivation (contrary to Algorithm 4.2.1, where the second component is a set). The length of y is limited to $k' = k + p$, where p is the length of the longest right-hand side of a rule from P_G (to allow application of rules which have more than k nonterminals on their right-hand sides).

Rules introduced in (1) are used to simulate a derivation step in I in which one nonterminal is rewritten to a string of symbols, where at least one of them is not erased in rest of the derivation. Since these symbols are chosen strongly nondeterministically during a derivation, every possible combination is considered. On the other hand, rules introduced in (2) are used to simulate a derivation step in I in which some to-be-erased nonterminal is rewritten to a string of to-be-erased nonterminals or to the empty string (this rewrite is done in the second component). Since there might not be enough space to do such rewrite, rules introduced in (3) are used to move nonterminals between the second components. Indeed, M can move the nonterminals between the second components at will because these nonterminals are to be erased anyway, so it is completely irrelevant where they occur in the sentential forms (this is the key idea behind this algorithm). In addition, as I erases its nonterminals in a k-limited way, there is always enough space to accommodate all these to-be-erased nonterminals.

At the very end of any successful derivation, rules of the form $\langle a, \varepsilon\rangle \to a$, for all $a \in T_G$, introduced in the initialization part of the algorithm, are used to obtain terminal symbols from compound nonterminals, just like in Algorithm 4.2.1. This can be done only if the rightmost nonterminal in Q is the new nonterminal, Z. This nonterminal indicates that M is at the end of a derivation simulation of I, so, in the sentential form, there should be only nonterminals of the form $\langle a, \varepsilon\rangle$, $a \in T$. When the very last nonterminal is about to be rewritten to a terminal, $Z \to r \in P_Q$, $r \in \Psi_O$, introduced in the initialization part of the algorithm, is used to finish the simulation by "getting rid of" Z.

Example 5.2.4. Consider again the regularly controlled grammar from Example 3.1.1. We repeat its definition with regard to Algorithm 5.2.1. Let $G = (N_G, T_G, S_G, P_G)$ be a context-free grammar with $N_G = \{S, A, B, C\}$, $T_G = \{a, b, c\}$, and P_G consisting of the following rules:

$r_1\colon S \to ABC$, $r_2\colon A \to aA$, $r_5\colon A \to \varepsilon$,
$r_3\colon B \to bB$, $r_6\colon B \to \varepsilon$,
$r_4\colon C \to cC$, $r_7\colon C \to \varepsilon$.

Now, let $H = (N_H, T_H, S_H, P_H)$ be a regular grammar with $N_H = \{S_H, X, X_1, X_2, Y_1, Y_2\}$, $T_H = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$, and P_H consiting of the following rules:

$$
\begin{array}{lll}
S_H \to r_1 X, & X \to r_2 X_1, & X \to r_5 Y_1, \\
 & X_1 \to r_3 X_2, & Y_1 \to r_6 Y_2, \\
 & X_2 \to r_4 X, & Y_2 \to r_7.
\end{array}
$$

From Example 5.2.1, we know that the regularly controlled grammar $I = (G, L(H))$ erases its nonterminals in a 1-limited way. With G and H on its input, Algorithm 5.2.1 produces an ε-free context-free grammar, O, and a regular grammar, Q, such that $L(M) = L(I) - \{\varepsilon\}$ for a regularly controlled grammar $M = (O, L(Q))$. Due to a high number of nonterminals and rules in O and Q, their definitions are omitted[5].

For example, for the following derivation of abc in I,

$$
\begin{array}{c}
S_G \Rightarrow ABC\ [r_1] \Rightarrow aABC\ [r_2] \Rightarrow aAbBC\ [r_3] \Rightarrow aAbBcC\ [r_4] \\
\Rightarrow abBcC\ [r_5] \Rightarrow abcC\ [r_6] \Rightarrow abc\ [r_7],
\end{array}
$$

one of the corresponding derivations in M is

$$
\begin{array}{lll}
\langle S_G, \varepsilon\rangle & \Rightarrow \langle A, \varepsilon\rangle\langle B, \varepsilon\rangle\langle C, \varepsilon\rangle & [\lfloor r_1, \varepsilon, A, B, C\rfloor] \\
 & \Rightarrow \langle a, A\rangle\langle B, \varepsilon\rangle\langle C, \varepsilon\rangle & [\lfloor r_2, \varepsilon, aA\rfloor] \\
 & \Rightarrow \langle a, A\rangle\langle b, B\rangle\langle C, \varepsilon\rangle & [\lfloor r_3, \varepsilon, bB\rfloor] \\
 & \Rightarrow \langle a, A\rangle\langle b, B\rangle\langle c, C\rangle & [\lfloor r_4, \varepsilon, cC\rfloor] \\
 & \Rightarrow \langle a, \varepsilon\rangle\langle b, B\rangle\langle c, C\rangle & [\lfloor a, A, \varepsilon, r_5\rfloor] \\
 & \Rightarrow \langle a, \varepsilon\rangle\langle b, \varepsilon\rangle\langle c, C\rangle & [\lfloor b, B, \varepsilon, r_6\rfloor] \\
 & \Rightarrow \langle a, \varepsilon\rangle\langle b, \varepsilon\rangle\langle c, \varepsilon\rangle & [\lfloor c, C, \varepsilon, r_7\rfloor] \\
 & \Rightarrow a\langle b, \varepsilon\rangle\langle c, \varepsilon\rangle & [\lfloor\langle a, \varepsilon\rangle \to a\rfloor] \\
 & \Rightarrow ab\langle c, \varepsilon\rangle & [\lfloor\langle b, \varepsilon\rangle \to b\rfloor] \\
 & \Rightarrow abc & [\lfloor\langle c, \varepsilon\rangle \to c\rfloor].
\end{array}
$$

Lemma 5.2.1. *Algorithm* 5.2.1 *is correct, i.e. with a context-free grammar, G, and a regular grammar, H, on its input, such that the regularly controlled grammar $I = (G, L(H))$ erases its nonterminals in a k-limited way, it halts and produces an ε-free context-free grammar, O, and a regular grammar, Q, such that $L(M) = L(I) - \{\varepsilon\}$ for the regularly controlled grammar $M = (O, L(Q))$.*

Proof. Clearly, the algorithm always halts. Since P_O does not contain any erasing rules, O is ε-free. To show that $L(O, L(Q)) = L(G, L(H)) - \{\varepsilon\}$, we first introduce some notions needed later in the proof.

Consider (1) and (2) in Algorithm 5.2.1. To briefly express that the rule labeled by s is constructed from a rule labeled by r in (1) or (2), we just write s_r. Let $\Omega \subseteq \Psi_O$ be the set of all rule labels introduced in (3) in Algorithm 5.2.1. Define the regular substitution σ from Ψ_G^* to $2^{\Psi_O^*}$ as $\sigma(r) = \Omega^*\{s \mid s \in \Psi_O \text{ and } s_r\}$, for all $r \in \Psi_O$.

Define the mapping $\delta_{G,H}$ from Ψ_G^* to $2^{N_H \cup \{Z\}}$ as follows (note that $\Psi_G = T_H$):

- $\delta_{G,H}(\varepsilon) = \{\varepsilon\}$;

[5]There are 2387 nonterminals in N_O.

- if $S_H \Rightarrow^+ \alpha B$ in H, where $\alpha \in \Psi_G^+$, $B \in N_H$, then $B \in \delta_{G,H}(\alpha)$;
- if $S_H \Rightarrow^+ \alpha r$ in H, where $\alpha \in \Psi_G^*$, $r \in \Psi_G$, then $Z \in \delta_{G,H}(\alpha r)$.

Define the mapping $\delta_{O,Q}$ from Ψ_O^* to 2^{N_Q} by analogy with $\delta_{G,H}$.

Claim 5.2.1. *Let $S_G \Rightarrow^+ x\ [\alpha]$ in G and $\langle S_G, \varepsilon\rangle \Rightarrow^+ y\ [\gamma]$ in O, where $x \in V_G^+$, $y \in N_O^+$, $\alpha \in \Psi_G^+$, and $\gamma \in \Psi_O^+$ such that $\gamma \in \sigma(\alpha)$, $\alpha\beta_1 \in L(H)$, and $\gamma\beta_2 \in L(Q)$ for some $\beta_1 \in \Psi_G^*$ and $\beta_2 \in \Psi_O^*$. Then, $\delta_{O,Q}(\gamma) = \delta_{G,H}(\alpha)$.*

Proof. Since $\alpha \in \Psi_G^+$ and $\gamma \in \Psi_O^+$, α and γ can be expressed as $\alpha = \alpha' r$ and $\gamma = \gamma' s$, respectively, where $\alpha' \in \Psi_G^*$, $\gamma' \in \Psi_O^*$, $r \in \Psi_G$, and $s \in \Psi_O$. From $\gamma' s \in \sigma(\alpha' r)$ and by the definition of σ, s_r, so by (1) and (2), we see that for every $A \to rB \in P_H$, $B \in N_H$, there is $A \to sB \in P_Q$, and for every $C \to r \in P_H$, there is $C \to sZ \in P_Q$, so $\delta_{O,Q}(\gamma) = \delta_{G,H}(\alpha)$. Rigorous proof by induction is left to the reader. □

In the following claim, we use Notation 2.4.1.

Claim 5.2.2.

$$S_G \Rightarrow^m {}^{\varepsilon}x_0 X_1{}^{\varepsilon}x_1 X_2{}^{\varepsilon}x_2 \dots X_h{}^{\varepsilon}x_h\ [\alpha] \text{ in } G$$

if and only if

$$\langle S_G, \varepsilon\rangle \Rightarrow^n \langle X_1, u_1\rangle\langle X_2, u_2\rangle \dots \langle X_h, u_h\rangle\ [\gamma] \text{ in } O,$$

where $x_i \in N_G^*$, $X_j \in V_G$, for all $0 \le i \le h$, $1 \le j \le h$, $u_1 u_2 \dots u_h \in perm(x_0 x_1 \dots x_h)$, $\alpha \in \Psi_G^+$, $\gamma \in \Psi_O^+$, $\gamma \in \sigma(\alpha)$, $\alpha\beta_1 \in L(H)$, and $\gamma\beta_2 \in L(Q)$ for some $\beta_1 \in \Psi_G^*$, $\beta_2 \in \Psi_O^*$, and $h, m, n \ge 1$.

Proof. Only If: This is established by induction on the length m of derivations.

Basis: Let $m = 1$. The only applicable rule to S_G in G is $r\colon S_G \to {}^{\varepsilon}y_0 Y_1{}^{\varepsilon}y_1 Y_2{}^{\varepsilon}y_2 \dots Y_h{}^{\varepsilon}y_h$, where $y_i \in N_G^*$, $Y_j \in V_G$, for all $0 \le i \le h$, $1 \le j \le h$, for some $h \ge 1$, so

$$S_G \Rightarrow {}^{\varepsilon}y_0 Y_1{}^{\varepsilon}y_1 Y_2{}^{\varepsilon}y_2 \dots Y_h{}^{\varepsilon}y_h\ [r] \text{ in } G$$

and there is $S_H \to rB \in P_H$, where $B \in (N_H \cup \{\varepsilon\})$. By (1), there are $s = \lfloor r, y_0, Y_1 y_1, Y_2 y_2, \dots, Y_h y_h\rfloor \in \Psi_O$ and $s\colon \langle S_G, \varepsilon\rangle \to \langle Y_1, y_0 y_1 \dots y_h\rangle\langle Y_2, \varepsilon\rangle \dots \langle Y_h, \varepsilon\rangle$, so

$$\langle S_G, \varepsilon\rangle \Rightarrow \langle Y_1, y_0 y_1 \dots y_h\rangle\langle Y_2, \varepsilon\rangle \dots \langle Y_h, \varepsilon\rangle\ [s] \text{ in } O$$

and there is $S_Q \to sC \in P_Q$, $C \in \{B, Z\}$ (recall that $S_Q = S_H$). Clearly, $y_0 y_1 \dots y_h \in perm(y_0 y_1 \dots y_h)$, and since s_r, $s \in \sigma(r)$, so the basis holds.

Induction Hypothesis: Suppose that the *Only If* part of Claim 5.2.2 holds for all derivations of length m or less, for some $m \ge 1$.

Induction Step: Consider any derivation of the form

$$S_G \Rightarrow^{m+1} y \text{ in } G,$$

where $y \in V_G^+$. Since $m + 1 > 1$, there is some $x \in V_G^+$ and $r \in \Psi_G$ such that

$$S_G \Rightarrow^m x\ [\alpha] \Rightarrow y\ [r] \text{ in } G,$$

for some $\alpha \in \Psi_G^+$. Let $A \in \delta_{G,H}(\alpha)$ such that there is some $A \to rA' \in P_H$, where $A' \in (N_H \cup \{\varepsilon\})$. Note that $A \neq Z$, because there is no rule with Z on its left-hand side in P_H.

Let $x = {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \ldots X_h{}^{\varepsilon}x_h$, where $x_i \in N_G^*$, $X_j \in V_G$, for all $0 \leq i \leq h$, $1 \leq j \leq h$, for some $h \geq 1$, so

$$S_G \Rightarrow^m {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \ldots X_h{}^{\varepsilon}x_h\ [\alpha] \text{ in } G.$$

Then, by the induction hypothesis,

$$\langle S_G, \varepsilon\rangle \Rightarrow^n \langle X_1, u_1\rangle\langle X_2, u_2\rangle \ldots \langle X_h, u_h\rangle\ [\gamma] \text{ in } O,$$

where $u_1u_2 \ldots u_h \in perm(x_0x_1 \ldots x_h)$ and $\gamma \in \Psi_O^+$ such that $\gamma \in \sigma(\alpha)$. By Claim 5.2.1, $\delta_{O,Q}(\gamma) = \delta_{G,H}(\alpha)$.

The initialization part of Algorithm 5.2.1 sets $k' = k + p$, where $p = max(\{|rhs(t)| \mid t \in \Psi_G\})$. Clearly, $p > 0$; otherwise, there is no possible derivation

$$\langle S_G, \varepsilon\rangle \Rightarrow^+ \langle X_1, u_1\rangle\langle X_2, u_2\rangle \ldots \langle X_h, u_h\rangle\ [\gamma] \text{ in } O,$$

for any $h \geq 1$ (observe that $p = 0$ if and only if either $P_G = \emptyset$ or every rule in P_G is an erasing rule). Since G with control language $L(H)$ erases its nonterminals in a k-limited way, $|x_0x_1 \ldots x_h| \leq kh$, and since $k' = k + p$ and $p > 0$, $|x_0x_1 \ldots x_h| < k'h$. Recall that $u_1u_2 \ldots u_h \in perm(x_0x_1 \ldots x_h)$, so $|u_1u_2 \ldots u_h| < k'h$ as well. Hence, there has to be at least one nonterminal $\langle X_e, u_e\rangle$ with $|u_e| < k'$ for some $1 \leq e \leq h$. By (3), there are rules $t_1 \colon \langle X_g, d_1Dd_2\rangle \to \langle X_g, d_1d_2\rangle$, $t_2 \colon \langle X_e, u_e\rangle \to \langle X_e, u_eD\rangle \in P_O$ and rules $A \to t_1C$, $C \to t_2A \in P_Q$, where $d_1, d_2 \in N_G^*$, $D \in N_G$, $t_1, t_2 \in \Omega$, $C = \lceil A, t_1, t_2\rceil \in N_Q$, for some $1 \leq g \leq h$, by which it is possible to consecutively derive

$$\langle X_1, u_1\rangle\langle X_2, u_2\rangle \ldots \langle X_h, u_h\rangle \Rightarrow^* \langle X_1, v_1\rangle\langle X_2, v_2\rangle \ldots \langle X_h, v_h\rangle\ [\chi] \text{ in } O$$

for any $v_1v_2 \ldots v_h \in perm(u_0u_1 \ldots u_h)$ such that $|v_i| \leq k$, for all $1 \leq i \leq h$ (recall that $|u_i| \leq k$, for all $1 \leq i \leq h$). Clearly, $\chi = \varepsilon$ if and only if either $u_1u_2 \ldots u_h = \varepsilon$ or $v_1v_2 \ldots v_h = u_1u_2 \ldots u_h$. Since all rules from χ are introduced in (3), $\chi \in \Omega^*$. Furthermore, $A \in \delta_{O,Q}(\gamma\chi)$ by (3) (recall that $A \in \delta_{O,Q}(\gamma)$).

Now, let us consider all possible forms of $x \Rightarrow y\ [r]$ in G:

(i) Let $r \colon X_f \to {}^{\varepsilon}y_0Y_1{}^{\varepsilon}y_1Y_2{}^{\varepsilon}y_2 \ldots Y_q{}^{\varepsilon}y_q \in P_G$, where $y_i \in N_G^*$, $Y_j \in V_G$, for all $0 \leq i \leq q$, $1 \leq j \leq q$, for some $q \geq 1$ and $1 \leq f \leq h$, so

$$\begin{aligned}{}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \ldots X_f{}^{\varepsilon}x_f \ldots X_h{}^{\varepsilon}x_h \ \Rightarrow\ & {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \ldots X_{f-1}{}^{\varepsilon}x_{f-1} \\ & {}^{\varepsilon}y_0Y_1{}^{\varepsilon}y_1Y_2{}^{\varepsilon}y_2 \ldots Y_q{}^{\varepsilon}y_q{}^{\varepsilon}x_fX_{f+1}{}^{\varepsilon}x_{f+1} \ldots X_h{}^{\varepsilon}x_h\ [r] \text{ in } G.\end{aligned}$$

By (1), there are $s = \lfloor r, y_0, Y_1y_1, Y_2y_2, \ldots, Y_qy_q \rfloor \in \Psi_O$ and $s\colon \langle X_f, v_f\rangle \to \langle Y_1, v_fy_0y_1 \ldots y_q\rangle\langle Y_2, \varepsilon\rangle \ldots \langle Y_q, \varepsilon\rangle \in P_O$. Now, recall that there are $A \to rA' \in P_H$ and $\delta_{O,Q}(\gamma\chi) = \delta_{G,H}(\alpha)$. Then, by (1), there is $A \to sC \in P_Q$, where $C = Z$ if and only if $A' = \varepsilon$ and $C = A'$ otherwise, so

$$\begin{gathered}\langle X_1, u_1\rangle\langle X_2, u_2\rangle \ldots \langle X_f, u_f\rangle \ldots \langle X_h, u_h\rangle \Rightarrow^* \\ \langle X_1, v_1\rangle\langle X_2, v_2\rangle \ldots \langle X_f, v_f\rangle \ldots \langle X_h, v_h\rangle\ [\chi] \Rightarrow \\ \langle X_1, v_1\rangle\langle X_2, v_2\rangle \ldots \langle X_{f-1}, v_{f-1}\rangle\langle Y_1, v_fy_0y_1 \ldots y_q\rangle \\ \langle Y_2, \varepsilon\rangle \ldots \langle Y_q, \varepsilon\rangle\langle X_{f+1}, v_{f+1}\rangle \ldots \langle X_h, v_h\rangle\ [s] \text{ in } O.\end{gathered}$$

Let $w = y_0y_1 \ldots y_q$ and recall that $v_1v_2 \ldots v_f \ldots v_h \in perm(u_1u_2 \ldots u_f \ldots u_h)$ and $u_1u_2 \ldots u_f \ldots u_h \in perm(x_0x_1 \ldots x_f \ldots x_h)$. Clearly, $v_1v_2 \ldots v_fw \ldots v_h \in perm(x_0x_1 \ldots wx_f \ldots x_h)$. Since $\chi \in \Omega^*$ and s_r, $\chi s \in \sigma(r)$. Now, recall that $\gamma \in \sigma(\alpha)$ by the induction hypothesis. Therefore, $\gamma\chi s \in \sigma(\alpha r)$ and the induction step is completed for this case.

(ii) Let $r\colon D \to {}^{\varepsilon}w \in P_G$, where $w \in N_G^*$, and let $x_f = d_1Dd_2$, where $d_1, d_2 \in N_G^*$, for some $0 \leq f \leq h$, so

$$\begin{gathered}{}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \ldots X_f{}^{\varepsilon}d_1{}^{\varepsilon}D^{\varepsilon}d_2 \ldots X_h{}^{\varepsilon}x_h \Rightarrow \\ {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \ldots X_f{}^{\varepsilon}d_1{}^{\varepsilon}w^{\varepsilon}d_2 \ldots X_h{}^{\varepsilon}x_h\ [r] \text{ in } G.\end{gathered}$$

By (2), there are $s = \lfloor X_g, z_1Dz_2, z_1wz_2, r\rfloor \in \Psi_O$ and $s\colon \langle X_g, z_1Dz_2\rangle \to \langle X_g, z_1wz_2\rangle \in P_O$, where $v_g = z_1Dz_2$ and $z_1, z_2 \in N_G^*$, for some $1 \leq g \leq h$. Now, recall that there is $A \to rA' \in P_H$ and $\delta_{O,Q}(\gamma\chi) = \delta_{G,H}(\alpha)$. Then, by (2), there is $A \to sC \in P_Q$, where $C = Z$ if and only if $A' = \varepsilon$ and $C = A'$ otherwise, so

$$\begin{gathered}\langle X_1, u_1\rangle\langle X_2, u_2\rangle \ldots \langle X_g, u_g\rangle \ldots \langle X_h, u_h\rangle \Rightarrow^* \\ \langle X_1, v_1\rangle\langle X_2, v_2\rangle \ldots \langle X_g, z_1Dz_2\rangle \ldots \langle X_h, v_h\rangle\ [\chi] \Rightarrow \\ \langle X_1, v_1\rangle\langle X_2, v_2\rangle \ldots \langle X_g, z_1wz_2\rangle \ldots \langle X_h,\ v_h\rangle\ [s] \text{ in } O.\end{gathered}$$

Recall that $v_1v_2 \ldots v_{g-1}z_1Dz_2v_{g+1} \ldots v_h \in perm(u_1u_2 \ldots u_h)$ and $u_1u_2 \ldots u_h \in perm(x_0x_1 \ldots x_{f-1}d_1Dd_2x_{f+1} \ldots x_h)$. Clearly, $v_1v_2 \ldots v_{g-1}z_1wz_2v_{g+1} \ldots v_h \in perm(x_0x_1 \ldots x_{f-1}d_1wd_2x_{f+1} \ldots x_h)$. Since $\chi \in \Omega^*$ and s_r, $\chi s \in \sigma(r)$. Now, recall that $\gamma \in \sigma(\alpha)$ by the induction hypothesis. Therefore, $\gamma\chi s \in \sigma(\alpha r)$, which completes the induction step for this case.

Observe that these two cases cover all possible derivations of the form $x \Rightarrow y\ [r]$ in G. Thus, the *Only If* part of Claim 5.2.2 holds.

If: This is also established by induction, but in this case on n.

Basis: Let $n = 1$. The only applicable rule to $S_O = \langle S_G, \varepsilon\rangle$ in O is $s\colon \langle S_G, \varepsilon\rangle \to \langle X_1, w\rangle\langle X_2, \varepsilon\rangle \ldots \langle X_h, \varepsilon\rangle \in P_O$, where $w \in N_G^*$, $X_i \in V_G$, for all $1 \leq i \leq h$, for some $h \geq 1$, and there is $S_Q \to sB \in P_Q$, where $B \in N_Q$, so

$$\langle S_G, \varepsilon\rangle \Rightarrow \langle X_1, w\rangle\langle X_2, \varepsilon\rangle \ldots \langle X_h, \varepsilon\rangle\ [s] \text{ in } O.$$

Since s is introduced in (1), there is $r\colon S_G \to {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \ldots X_h{}^{\varepsilon}x_h \in P_G$, where $x_i \in N_G^*$, for all $1 \leq i \leq h$, such that $w = x_0x_1 \ldots x_h$ and there is $S_H \to rC \in P_H$ for some $C \in (\{B, \varepsilon\} - \{Z\})$, so

$$S_G \Rightarrow {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_h{}^{\varepsilon}x_h \ [r] \text{ in } G.$$

Clearly, $w \in perm(w)$, and since s_r, $s \in \sigma(r)$, so the basis holds.

Induction Hypothesis: Suppose that the *If* part of Claim 5.2.2 holds for all derivations of length n or less, for some $n \geq 1$.

Induction Step: Consider any derivation of the form

$$\langle S_G, \varepsilon \rangle \Rightarrow^{n+1} y \text{ in } O,$$

where $y \in N_O^+$. Since $n + 1 > 1$, this derivation can be expressed as

$$\langle S_G, \varepsilon \rangle \Rightarrow^{n} x \ [\gamma] \Rightarrow y \ [s] \text{ in } O,$$

where $x \in N_O^+$, $\gamma \in \Psi_O^+$, and $s \in \Psi_O$. Let $B \in \delta_{O,Q}(\gamma)$ such that there is $B \to sB' \in P_Q$, where $B' \in N_Q$. Note that $B \neq Z$; otherwise, there is no possible derivation $x \Rightarrow y \ [s]$ in O with $y \in N_O^+$ (if $B = Z$, the only applicable rules in O are of the form $\langle a, \varepsilon \rangle \to a$, where $a \in T_O$, but $a \notin N_O$).

Let $x = \langle X_1, u_1 \rangle \langle X_2, u_2 \rangle \dots \langle X_h, u_h \rangle$, where $X_i \in V_G$, $u_i \in N_G^*$, for all $1 \leq i \leq h$, for some $h \geq 1$. By the induction hypothesis, there is

$$S_G \Rightarrow^{m} {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_h{}^{\varepsilon}x_h \ [\alpha] \text{ in } G$$

such that $u_1u_2 \dots u_h \in perm(x_0x_1 \dots x_h)$ and $\gamma \in \sigma(\alpha)$, for some $m \geq 1$. Now, let us consider all possible forms of $x \Rightarrow y \ [s]$ in O:

(i) Let $s\colon \langle X_f, u_f \rangle \to \langle Y_1, u_f w \rangle \langle Y_2, \varepsilon \rangle \dots \langle Y_q, \varepsilon \rangle \in P_O$ be a rule introduced in (1), where $w = y_0y_1 \dots y_q$, $s = \lfloor r, y_0, Y_1y_1, Y_2y_2, \dots, Y_qy_q \rfloor$, $y_i \in N_G^*$, $Y_j \in V_G$, for all $0 \leq i \leq q$, $1 \leq j \leq q$, for some $1 \leq f \leq h$ and $q \geq 1$. Therefore,

$$\begin{gathered}\langle X_1, u_1 \rangle \langle X_2, u_2 \rangle \dots \langle X_f, u_f \rangle \dots \langle X_h, u_h \rangle \Rightarrow \\ \langle X_1, u_1 \rangle \langle X_2, u_2 \rangle \dots \langle X_{f-1}, u_{f-1} \rangle \langle Y_1, u_f w \rangle \langle Y_2, \varepsilon \rangle \\ \dots \langle Y_q, \varepsilon \rangle \langle X_{f+1}, u_{f+1} \rangle \dots \langle X_h, u_h \rangle \ [s] \text{ in } O.\end{gathered}$$

Since $\delta_{O,Q}(\gamma) = \delta_{G,H}(\alpha)$ by Claim 5.2.1 and s_r, by (1), there has to be some $r\colon X_f \to {}^{\varepsilon}y_0Y_1{}^{\varepsilon}y_1Y_2{}^{\varepsilon}y_2 \dots Y_q{}^{\varepsilon}y_q$ and $B \to rC \in P_H$, where $C = \varepsilon$ if and only if $B' = Z$ and $C = B'$ otherwise, so

$$\begin{gathered}{}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_{f-1}{}^{\varepsilon}x_{f-1}X_f{}^{\varepsilon}x_fX_{f+1}{}^{\varepsilon}x_{f+1} \dots X_h{}^{\varepsilon}x_h \Rightarrow \\ {}^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_{f-1}{}^{\varepsilon}x_{f-1}{}^{\varepsilon}y_0Y_1{}^{\varepsilon}y_1Y_2{}^{\varepsilon}y_2 \\ \dots Y_q{}^{\varepsilon}y_q{}^{\varepsilon}x_fX_{f+1}{}^{\varepsilon}x_{f+1} \dots X_h{}^{\varepsilon}x_h \ [r] \text{ in } G.\end{gathered}$$

Recall that $u_1u_2 \dots u_f \dots u_h \in perm(x_0x_1 \dots x_f \dots x_h)$. Clearly, $u_1u_2 \dots u_fw \dots u_h \in perm(x_0x_1 \dots wx_f \dots x_h)$. Now, recall that $\gamma \in \sigma(\alpha)$ by the induction hypothesis and s_r. Therefore, $\gamma s \in \sigma(\alpha r)$, and the induction step is completed for this case.

(ii) Let $s\colon \langle X_f, d_1Dd_2 \rangle \to \langle X_f, d_1wd_2 \rangle \in P_O$ be a rule introduced in (2), where $s = \lfloor X_f, d_1Dd_2, d_1wd_2, r \rfloor$, $d_1, d_2 \in N_G^*$, and $D \in N_G$ such that $u_f = d_1Dd_2$ for some $1 \leq f \leq h$. Therefore,

$$\langle X_1, u_1\rangle\langle X_2, u_2\rangle \dots \langle X_f, d_1Dd_2\rangle \dots \langle X_h, u_h\rangle \Rightarrow$$
$$\langle X_1, u_1\rangle\langle X_2, u_2\rangle \dots \langle X_f, d_1wd_2\rangle \dots \langle X_h,\ u_h\rangle[s] \text{ in } O.$$

By the induction hypothesis, $x_g = z_1Dz_2$, where $z_1, z_2 \in N_G^*$, for some $0 \leq g \leq h$. Since $\delta_{O,Q}(\gamma) = \delta_{G,H}(\alpha)$ by Claim 5.2.1 and s_r, by (2), there has to be some $r\colon D \to w \in P_G$ and $B \to rC \in P_H$, where $C = \varepsilon$ if and only if $B' = Z$ and $C = B'$ otherwise, so

$$^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_g{}^{\varepsilon}z_1{}^{\varepsilon}D^{\varepsilon}z_2 \dots X_h{}^{\varepsilon}x_h \Rightarrow$$
$$^{\varepsilon}x_0X_1{}^{\varepsilon}x_1X_2{}^{\varepsilon}x_2 \dots X_g{}^{\varepsilon}z_1{}^{\varepsilon}w^{\varepsilon}z_2 \dots X_h{}^{\varepsilon}x_h\ [r] \text{ in } G.$$

Recall that $u_1u_2 \dots u_{f-1}d_1Dd_2u_{f+1} \dots u_h \in perm(x_0x_1 \dots x_{g-1}z_1Dz_2x_{g+1} \dots x_h)$. Clearly, $u_1u_2 \dots u_{f-1}d_1wd_2u_{f+1} \dots u_h \in perm(x_0x_1 \dots x_{g-1}z_1wz_2x_{g+1} \dots x_h)$. Now, recall that $\gamma \in \sigma(\alpha)$ by the induction hypothesis and s_r. Therefore, $\gamma s \in \sigma(\alpha r)$, and the induction step is completed for this case.

(iii) Let $s\colon \langle X_f, d_1Dd_2\rangle \to \langle X_f, d_1d_2\rangle \in P_O$ be a rule introduced in (3), where $d_1, d_2 \in N_G^*$ and $D \in N_G$ such that $u_f = d_1Dd_2$ for some $1 \leq f \leq h$. Therefore,

$$\langle X_1, u_1\rangle\langle X_2, u_2\rangle \dots \langle X_f, d_1Dd_2\rangle \dots \langle X_h, u_h\rangle \Rightarrow$$
$$\langle X_1, u_1\rangle\langle X_2, u_2\rangle \dots \langle X_f, d_1d_2\rangle \dots \langle X_h, u_h\rangle\ [s] \text{ in } O.$$

By (3), the only applicable rule is now $t\colon \langle X_g, w\rangle \to \langle X_g, wD\rangle$, where $w \in N_G^*$, for some $1 \leq g \leq h$, so

$$\langle X_1, u_1\rangle\langle X_2, u_2\rangle \dots \langle X_f, d_1d_2\rangle \dots \langle X_h, u_h\rangle \Rightarrow$$
$$\langle X_1, v_1\rangle\langle X_2, v_2\rangle \dots \langle X_h, v_h\rangle \text{ in } O,$$

where $v_1v_2 \dots v_h \in perm(u_1u_2 \dots u_h)$. Since $st \in \Omega^*$, $\delta_{O,Q}(\gamma st) = \delta_{O,Q}(\gamma)$, and $v_1v_2 \dots v_h \in perm(x_1x_2 \dots x_h)$ (recall that $u_1u_2 \dots u_h \in perm(x_1x_2 \dots x_h)$), we can now reconsider cases (i) through (iii).

Observe that these three cases cover all possible derivations of the form $x \Rightarrow y\ [s]$ in O. Thus, the *If* part of Claim 5.2.2 also holds. Hence, Claim 5.2.2 holds. □

Now, consider a special case of Claim 5.2.2 when $x_i = \varepsilon$, $X_j \in T_G$, for all $0 \leq i \leq h$, $1 \leq j \leq h$. Then,

$$S_G \Rightarrow^m X_1X_2 \dots X_h\ [\alpha] \text{ in } G$$

if and only if

$$\langle S_G, \varepsilon\rangle \Rightarrow^n \langle X_1, \varepsilon\rangle\langle X_2, \varepsilon\rangle \dots \langle X_h, \varepsilon\rangle\ [\gamma] \text{ in } O,$$

where $\alpha \in \Psi_G^+$, $\gamma \in \Psi_O^+$, $\gamma \in \sigma(\alpha)$, $\alpha\beta_1 \in L(H)$ and $\gamma\beta_2 \in L(Q)$ for some $\beta_1 \in \Psi_G^*$, $\beta_2 \in \Psi_O^*$, and $h, m, n \geq 1$.

Since $\alpha \in \Psi_G^+$ and $\gamma \in \Psi_O^+$, α and γ can be expressed as $\alpha = \alpha' r$ and $\gamma = \gamma' s$, respectively, where $\alpha' \in \Psi_G^*$, $\gamma' \in \Psi_O^*$, $r \in \Psi_G$, and $s \in \Psi_O$. Furthermore, from $\gamma' s \in \sigma(\alpha' r)$ and by the definition of σ, s_r. Since $X_i \in T_G$, for all $1 \leq i \leq h$, no rule from P_G is now applicable in

G. To fulfill $\alpha\beta_1 \in L(H)$, $\beta_1 = \varepsilon$, so there has to be $A \to r \in P_H$, where $A \in \delta_{G,H}(\alpha')$. By Claim 5.2.1, $\delta_{O,Q}(\gamma) = \delta_{G,H}(\alpha)$, and by the definition of $\delta_{G,H}$, $Z \in \delta_{G,H}(\alpha' r) = \delta_{G,H}(\alpha)$, so $Z \in \delta_{O,Q}(\gamma)$.

By the initialization part of Algorithm 5.2.1, P_O contains $\lfloor\langle X_1, \varepsilon\rangle \to X_1\rfloor : \langle X_1, \varepsilon\rangle \to X_1$, $\lfloor\langle X_2, \varepsilon\rangle \to X_2\rfloor : \langle X_2, \varepsilon\rangle \to X_2$, ..., $\lfloor\langle X_h, \varepsilon\rangle \to X_h\rfloor : \langle X_h, \varepsilon\rangle \to X_h$ and P_Q contains $Z \to \lfloor\langle X_1, \varepsilon\rangle \to X_1\rfloor Z$, $Z \to \lfloor\langle X_1, \varepsilon\rangle \to X_2\rfloor Z$, ..., $Z \to \lfloor\langle X_h, \varepsilon\rangle \to X_h\rfloor$. As a result, there is a derivation

$$\begin{array}{lll} \langle X_1, \varepsilon\rangle\langle X_2, \varepsilon\rangle \dots \langle X_h, \varepsilon\rangle & \Rightarrow X_1\langle X_2, \varepsilon\rangle \dots \langle X_h, \varepsilon\rangle & [\lfloor\langle X_1, \varepsilon\rangle \to X_1\rfloor] \\ X_1\langle X_2, \varepsilon\rangle \dots \langle X_h, \varepsilon\rangle & \Rightarrow X_1 X_2 \dots \langle X_h, \varepsilon\rangle & [\lfloor\langle X_2, \varepsilon\rangle \to X_2\rfloor] \\ & \dots & \\ X_1 X_2 \dots \langle X_h, \varepsilon\rangle & \Rightarrow X_1 X_2 \dots X_h & [\lfloor\langle X_h, \varepsilon\rangle \to X_h\rfloor] \end{array}$$

in O. Therefore, $\gamma\lfloor\langle X_1, \varepsilon\rangle \to X_1\rfloor\lfloor\langle X_2, \varepsilon\rangle \to X_2\rfloor \dots \lfloor\langle X_h, \varepsilon\rangle \to X_h\rfloor \in L(Q)$ and since $X_i \in T_O$, for all $1 \leq i \leq h$, $X_1 X_2 \dots X_h \in L(O, L(Q))$, so we have $X_1 X_2 \dots X_h \in L(G, L(H))$ if and only if $X_1 X_2 \dots X_h \in L(O, L(Q))$. Hence, Lemma 5.2.1 holds. □

Theorem 5.2.1. *Let I be a regularly controlled grammar that erases its nonterminals in a k-limited way. Then, there is an ε-free regularly controlled grammar, M, such that $L(M) = L(I) - \{\varepsilon\}$.*

Proof. This theorem follows from Algorithm 5.2.1 and Lemma 5.2.1. □

Observe that Definition 5.2.1, Algorithm 5.2.1, Lemma 5.2.1, and Theorem 5.2.1 can be easily reformulated in terms of regularly controlled grammars with appearance checking. Reconsider Algorithm 5.2.1. Observe that during the simulation of a derivation in I, in the two-component nonterminals, M records all the symbols that occur in the corresponding sentential form of I. As a result, M has all the necessary information concerning the appearance checking mechanism available, so it can simply apply the same appearance checking mechanism just like in I. Leaving a fully rigorous description of Algorithm 5.2.1 modified in this straightforward way to the reader, we next reformulate Theorem 5.2.1 in terms of appearance checking.

Theorem 5.2.2. *Let I be a regularly controlled grammar with appearance checking that erases its nonterminals in a k-limited way. Then, there is an ε-free regularly controlled grammar with appearance checking, M, such that $L(M) = L(I) - \{\varepsilon\}$.* □

5.3 Significance to Syntactical Analysis

Most current compilers use parsers based on context-free grammars, because there is a well-researched underlying theory (for details and/or all unexplained notions, please refer to [3, 4, 8, 9, 54, 59, 62]), which includes top-down parsing based on *LL(k) grammars* and *pushdown automata* (*recursive decent*, *table-based/predictive parsers*) and bottom-up parsing based on *LR(k) grammars* and *extended pushdown automata* (*SLR* or *LALR parsers*).

From a practical point of view, erasing rules are often very useful. When designing a grammar for a programming language, like authors of programming languages or compilers do, one can with advantage use erasing rules for optional parts in that language. For

example, consider the fragment of a grammar for some hypothetical C-like [31] programming language in Figure 5.1, which describes a function definition. Aside from implementation limits, a function can have arbitrary number of parameters, including no parameters. The same holds for the number of statements in function bodies. The easiest way how to cope with this is to use erasing rules. For example, *params* can either contain a list of parameters or no parameters at all.

function-def $\rightarrow$ *type-specifier ID (params)* { *statements* }

params $\rightarrow$ *param-list* $\mid$ ε

statements $\rightarrow$ *statement-list* $\mid$ ε

Figure 5.1: Fragment of a C-like Language Grammar.

On the other hand, the presence of erasing rules has also a few drawbacks. Three of them are discussed next.

I. Erasing rules can cause ambiguities and conflicts [54]. For example, consider the fragment of a grammar in Figure 5.2. This grammar is certainly not LL(1), because if the top nonterminal is *statement* and the input token is ID, the parser has no way how to determine what rule it should apply (both *label* and *assign-statement* begin with ID and *label* is optional).

statement $\rightarrow$ *label statement-type*

statement-type $\rightarrow$ *assign-statement* $\mid$ *return-statement* $\mid$ ...

label $\rightarrow$ *ID :* $\mid$ ε

assign-statement $\rightarrow$ *ID = expression ;*

Figure 5.2: Fragment of a C-like Language Grammar Which is Not LL(1).

II. The presence of erasing rules can result into more complex parser construction. For example, consider the table-driven (or predictive) LL parser. If the input grammar does contain any erasing rules, to construct the LL table it only suffices to compute the FIRST set. Otherwise, more sets have to be computed (like EMPTY and FOLLOW) [54].

III. There are parsing techniques which do not work with grammars containing erasing rules. For example, consider the *operator-precedence parser*, which is a simple parser suitable for the parsing of expressions. This parser cannot be—by design—used to parse sentences generated by grammars with erasing rules [54].

So, while erasing rules are nice during grammar design, they might cause problems when building a parser for such grammars. Here is where elimination of erasing rules might help. However, as you will see, problems arise. Since this work studies regulated grammars and previous paragraphs deal mainly with parsers based on classical context-free grammars, we also discuss parsing of regulated grammars and relevance of the presented results regarding elimination of erasing from regulated grammars.

I. If we eliminate erasing rules from a grammar of some type, the resulting grammar might not be of that type or might not be suitable for parsing. For example, consider LL(k) grammars. If we eliminate all erasing rules from an LL(k) grammar, the consequent ε-free grammar might not be LL(k) [54].

II. As far as we were able to determine, there has been no systematic research in the direction of parsing methods based on regulated grammars in general. Several papers and theses on this topic were written, but there is no complete and usable parsing theory based on regulated grammars by knowledge. Moreover, proposed methods sometimes lack determination of wanted properties, like their computational complexity[6] and precise generative power, i.e. what they can actually parse (if such methods parse only some restricted subset of regulated grammars).

In [5], [36], and [60], parsing algorithms based on regulated grammars are proposed. In [38], Křivka discusses a use of regulated grammars (especially programmed grammars) in programming languages. In [11], a sketch of a parser based on programmed grammars is described. Bravo and Neto in [6] proposed a method for building context-sensitive parsers from regularly controlled grammars. Rußmann in [68] introduced a *dynamic LL(k) parser* based on so-called *dynamic context-free grammars* and managed to characterize LR(1) grammars by a grammar model involving leftmost derivations which can be seen as the deterministic counterpart of matrix grammars with strong leftmost derivations [10]. Recently, Jirák [32, 33] and Kolář [34] considered parsing of restricted scattered context grammars.

III. Transformations of grammars to equivalent ε-free grammars often result into a large number of nonterminals and rules, i.e. in a high syntactic complexity[7]. For example, if the input regularly controlled grammar of Algorithm 5.2.1 contains n nonterminals, t terminals, the length of the longest right-hand side of a rule is p, and the grammar erases its nonterminals in a k-limited way, the output grammar contains $(n+t)\sum_{i=0}^{k+p} n^i$ nonterminals (recall how the total alphabet of the resulting grammar is computed). Other transformations presented in Section 5.1, like elimination of recursively erasing rules from programmed grammars, suffer from the same problem as well.

In recent years, there has been a trend to study regulated grammars in terms of their syntactic complexity, see [20, 21, 22, 23, 45, 46, 47, 48, 51, 73] and [57, Chapter 6]. However, there is often a trade-off between the number of nonterminals and the number or form of rules (the less nonterminals are required, the more and complicated rules are required and vice versa). What we need are (1) transformations that remove useless nonterminals and rules and (2) transformations that simultaneously reduce the number of nonterminals and rules.

To conclude the significance of elimination of erasing rules from regulated grammars to syntactical analysis, the two main problems are the lack of advanced, throughly examined, and generally usable parsing methods based on regulated grammars and high syntactic complexity of resulting ε-free grammars. Further research is these two directions is necessary.

[6] By computational complexity we mean time and space complexity, see [70, Part 3].

[7] By syntactic complexity of a grammar we mean the size of the description of the grammar, e.g. the number of nonterminals, the number of rules, and the form of rules. Other widely used synonyms are descriptive complexity or Kolgomorov complexity.

Chapter 6

Conclusion

In this work, the elimination of erasing rules from regulated grammars and its consequences are studied. The goal is to gather current results and present new results on the impact of the presence of erasing rules to the generative power of regulated grammars. As Chapter 3 shows, many regulated grammars are based on context-free grammars. Thus, it is of our interest to study algorithms used to remove erasing rules from context-free grammars and try to apply them on regulated grammars. To this end, Chapter 4 presents a standard algorithm and a new, alternative algorithm, which has the advantage that is does not need any predetermination of ε-nonterminals. Both of these algorithms prove that we are always able to remove erasing rules from context-free grammars. However, in the case of regulated grammars, the usage of these algorithm is not that simple, because, as outlined in the beginning of Chapter 5, there is another element, which is not present in context-free grammars—the regulation.

Whereas the (im)possibility of erasing rules elimination from some types of regulated grammars was proven, there are still some regulated grammars where the possibility of erasing rules removal is still an open problem (the summary is given below). Apart from current results presented in Chapter 3 and in Section 5.1, Section 5.2 presents a new result: elimination of erasing rules from regularly controlled grammars which erase their nonterminals in a k-limited way.

As outlined in Section 5.3, until there is more underlying theory regarding parsing techniques based on regulated grammars and algorithms for syntactic complexity reduction, the value of the results concerning the elimination of erasing rules from regulated grammars is mainly theoretical. However, the alternative Algorithm 4.2.1 might be interesting from a pedagogical point of view because one does not need to introduce ε-nonterminals and their computation.

Finally, let us make five remarks regarding the new results proved in this work.

I. Algorithm 5.2.1 and Lemma 5.2.1 represent a partial solution to the problem concerning the effect of erasing rules to the generative power of regularly controlled grammars grammars. Indeed, if these grammars erase their nonterminals in a k-limited way, they are equally powerful with or without erasing rules. Consequently, to solve this problem completely, the formal language theory can narrow its attention only to these grammars that do not erase their nonterminals in this way because if they do, this work has answered the problem.

II. By analogy with regularly controlled grammars that erase their nonterminals in a k-limited way, we can reconsider the study given in this work in terms of other regulated grammars. Specifically, this study and its results can be straightforwardly reformulated in terms of matrix, programmed, or random-context grammars that erase their nonterminals in a k-limited way.

III. Consider regularly controlled grammars and scattered context grammars, both erasing their nonterminals in a k-limited way. Apart from a different type of regulation (in scattered context grammars, more context-free rules are applied in a single step, there is no prescribed order in which rules have to be applied, and there are contextual dependencies between symbols in a sentential form), there is a main difference between these two types of limited erasing. While the k-limiting condition in scattered context grammars requires that between any two symbols that are not erased, there are at most k occurrences of nonterminals that are erased (regardless of the length of the sentential form), the k-limiting condition in regularly controlled grammars requires that there are is a certain balance between the number of to-be-erased nonterminals and not-to-be-erased symbols in the entire sentential form, so the position of to-be-erased nonterminals does not matter. Moreover, this balance depends on k and on the length of the sentential form.

IV. Consider regularly controlled grammars that erase their nonterminals in a k-limited way and recursive erasing in programmed grammars. On the one hand, the recursive erasing condition requires that all occurrences of the same nonterminal are erased, while this is not required by the k-limiting condition. On the other hand, using the idea of recursive erasing, we are able to remove erasing rules from regularly controlled grammars like H_1 from Example 5.2.3 that does not erase its nonterminals in a k-limited way.

Thus, both ideas are orthogonal in the sense that the recursive erasing condition can be straightforwardly reformulated in terms of regularly controlled grammars and vice versa, see the second remark in this section and [39, Section 4]. By combining these two approaches, we are able to eliminate more erasing rules from these grammars.

V. Consider the new Algorithm 4.2.1 for elimination of erasing rules from context-free grammars. It can be shown that this algorithm can be used even if the input context-free grammar derives in a so-called *semi-parallel derivation mode* (in a single derivation step, any number of occurrences of nonterminals might be rewritten at once, like in scattered context grammars, but without any contextual dependencies) or in a so-called *parallel derivation mode* (in every derivation step, all symbols have to be rewritten at once, like in EOL systems [65]). Since this is beyond the scope of this work, the proof of this claim is omitted[1].

6.1 Open Problems

This section summarizes open problems related to this work, where most of them are outlined in Chapters 3 and 5. These problems are suggested for further research.

- One of the main open problems in the theory of regulated rewriting remains unresolved. Is the inclusion between $\mathbf{rC}$ and $\mathbf{rC}^{\varepsilon}$, $\mathbf{M}$ and $\mathbf{M}^{\varepsilon}$, $\mathbf{P}$ and $\mathbf{P}^{\varepsilon}$, $\mathbf{RC}$ and $\mathbf{RC}^{\varepsilon}$,

[1] Curious reader is advised to use the approach used to establish Lemma 4.2.1.

respectively, proper? In other words, are we able to remove all erasing rules from any regularly controlled grammar, matrix grammar, programmed grammar, and random context grammar, respectively, without affecting the generated language? Is this possible in Russian parallel grammars?

- What is the relation between **CS** and $\mathbf{rC}^{\varepsilon}$, $\mathbf{M}^{\varepsilon}$, $\mathbf{P}^{\varepsilon}$, or $\mathbf{RC}^{\varepsilon}$? Is the inclusion between **RC** and **M**, $\mathbf{RC}^{\varepsilon}$ and $\mathbf{M}^{\varepsilon}$, **SC** and **CS**, respectively, proper?

- Section 5.1 shows that we can transform any scattered context grammar, H, to an ε-free scattered context grammar, O, such that O either generates a coincidental extension of $L(H)$ or $L(H)$ represents the symbol-exhaustive left (right) quotient of $L(O)$. Can any of these result be proved in terms of other regulated grammars, like regularly controlled grammars, matrix grammars, or programmed grammars?

- In Section 5.2, it is shown that we are able to eliminate all erasing rules from any regularly controlled grammar which erases its nonterminals in a k-limited way. Is there an algorithmic way how to determine whether a regularly controlled grammar erases its nonterminals in a k-limited way? In other words, is it decidable whether a given regularly controlled grammar erases its nonterminals in a k-limited way? Similar question is uttered in terms of scattered context grammars that erase their nonterminals in a k-limited way in [57, Chapter 4]. If it is decidable, what is the computational complexity of such determination[2]? What are the computational complexities of Algorithm 5.2.1 and Algorithm 4.2.1?

- As Section 5.3 outlined, the transformation of a regularly controlled grammar or a scattered context grammar that erases its nonterminals in a k-limited way into an equivalent ε-free grammar results into a large number of nonterminals and rules, where many of them are useless, i.e. they are never used. Also, there is the same problem with the new Algorithm 4.2.1 and elimination of recursively erasing rules from programmed grammars. Can these transformations be improved in terms of syntactic complexity? Is there an algorithm that can remove useless nonterminals and rules from transformed regulated grammars?

[2]For a highlight of the complexity checking of some basic decidability properties of regulated grammars, see [44, Page 269].

Bibliography

[1] S. Abraham. Some questions of language theory. In *Proceedings of the 1965 conference on Computational linguistics*, pages 1–11, Morristown, NJ, USA, 1965. Association for Computational Linguistics.

[2] A. V. Aho. Indexed grammars—an extension of context-free grammars. *Journal of the ACM*, 15(4):647–671, 1968.

[3] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd ed.)*. Addison-Wesley, 2006. ISBN 0-201-10088-6.

[4] H. Bal, D. Grune, C. Jacobs, and K. Langendoen. *Modern Compiler Design*. John Wiley & Sons, Inc., 2000. ISBN 0-471-97697-0.

[5] P. Blatný. Syntaktická analýza založená na gramatikách s rozptýleným kontextem. Master's thesis, Vysoké učení technické v Brně, Fakulta informačních technologií, 2004.

[6] C. Bravo and J. J. Neto. Building context-sensitive parsers from CF grammars with regular control language. In *Implementation and Application of Automata 8th International Conference*, pages 306–308. Springer, 2003.

[7] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[8] K. D. Cooper and L. Torczon. *Engineering a Compiler*. Elsevier, 2004. ISBN 1-55860-699-8.

[9] R. Cytron, C. Fischer, and R. LeBlanc. *Crafting a Compiler*. Addison-Wesley, 2009. ISBN 0-13-606705-0.

[10] J. Dassow, H. Fernau, and G. Păun. On the leftmost derivation in matrix grammars. *International Journal of Foundations of Computer Science*, 10(1):61–80, 1999.

[11] J. Dassow and G. Păun. *Regulated Rewriting in Formal Language Theory*. Springer, New York, 1989. ISBN 0-38751-414-7.

[12] J. Dassow and S. Turaev. Arbitrary Petri net controlled grammars. In *ForLing '08: Proceedings of the 2nd International Workshop „Non-Classical Formal Lanuages in Linguistics“*, pages 27–39, Tarragona, Spain, 2008.

[13] J. Dassow and S. Turaev. k-Petri net controlled grammars. In *LATA '08: Proceedings of the 2nd International Conference on Language and Automata Theory and Applications*, pages 209–220. Springer, 2008.

[14] J. Dassow and S. Turaev. Grammars controlled by special Petri nets. In *LATA '09: Proceedings of the 3rd International Conference on Language and Automata Theory and Applications*, pages 326–337. Springer, 2009.

[15] J. Dassow and S. Turaev. Petri net controlled grammars: the case of special Petri nets. *Journal of Universal Computer Science*, 15(14):2808–2835, 2009.

[16] J. Dassow and S. Turaev. Petri net controlled grammars: the power of labeling and final markings. *Romanian Journal of Information Science and Technology*, 12(12):191–207, 2009.

[17] R. Diestel. *Graph Theory (3rd ed.)*. Springer, 2005. ISBN 3-540-26182-6.

[18] A. Ehrenfeucht and G. Rozenberg. An observation on scattered grammars. *Information Processing Letters*, 9(2):84–85, 1979.

[19] B. Farwer, M. Jantzen, M. Kudlek, H. Rölke, and G. Zetzsche. Petri net controlled finite automata. *Fundamenta Informaticae*, 85(1–4):111–121, 2008.

[20] H. Fernau. Nonterminal complexity of programmed grammars. *Theoretical Computer Science*, 296(2):225–251, 2003.

[21] H. Fernau, R. Freund, M. Oswald, and K. Reinhardt. Refining the nonterminal complexity of graph-controlled, programmed, and matrix grammars. *Journal of Automata, Languages and Combinatorics*, 12(1–2):117–138, 2007.

[22] H. Fernau and A. Meduna. A simultaneous reduction of several measures of descriptional complexity in scattered context grammars. *Information Processing Letters*, 86(5):235–240, 2003.

[23] R. Freund and G. Păun. On the number of non-terminal symbols in graph-controlled, programmed and matrix grammars. In *MCU '01: Proceedings of the Third International Conference on Machines, Computations, and Universality*, pages 214–225. Springer, 2001.

[24] I. Friš. Grammars with partial ordering of the rules. *Information and Control*, 12(5/6):415–425, 1968.

[25] S. Ginsburg and E. H. Spanier. Control sets on grammars. *Theory of Computing Systems*, 2(2):159–177, 1968.

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

[27] M. Hack. Decidability questions for Petri nets. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1976.

[28] D. Hauschildt and M. Jantzen. Petri net algorithms in the theory of matrix grammars. *Acta Informatica*, 31(8):719–728, 1994.

[29] J.E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd ed.)*. Addison-Wesley, 2006. ISBN 0-321-45536-3.

[30] K. Culik II and H. A. Maurer. Tree controlled grammars. *Computing*, 19(2):129–139, 1977.

[31] ISO/IEC 9899:1999. Programming Languages – C (ISO and ANSI C99 Standard). Technical Report WG14 N1124, ISO/IEC, 1999. Available on URL: `http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1124.pdf` (April 2010).

[32] O. Jirák. Delayed execution of scattered context grammar rules. In *Proceedings of the 15th Conference and Competition STUDENT EEICT 2009 Volume 4*, pages 405–409, Brno, CZ, 2009. Faculty of Information Technology BUT.

[33] O. Jirák. Table-driven parsing of scattered context grammar. In *Proceedings of the 16th Conference Student EEICT 2010 Volume 5*, pages 171–175, Brno, CZ, 2010. Faculty of Information Technology BUT.

[34] D. Kolář. Scattered context grammars parsers. In *Proceedings of the 14th International Congress of Cybernetics and Systems of WOCS*, pages 491–500, Wroclaw, PL, 2008. Wroclaw University of Technology.

[35] D. Kolář and A. Meduna. Regulated pushdown automata. *Acta Cybernetica*, 2000(4):653–664, 2000.

[36] M. Kot. Řízené gramatiky. Master's thesis, VŠB – Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky, 2002.

[37] J. Koutný. On n-path-controlled grammars. In *Proceedings of the 16th Conference and Competition STUDENT EEICT 2010 Volume 5*, pages 176–180, Brno, CZ, 2010. Faculty of Information Technology BUT.

[38] Z. Křivka. Využití řízených gramatik v programovacích jazycích [online]. Projekt do předmětu Teorie programovacích jazyků doktorského studijního programu, Brno, CZ, 2002. Fakulta informačních technologií VUT. Available on URL: `http://www.fit.vutbr.cz/study/courses/TJD/public/0506TJD-Krivka.pdf` (April 2010).

[39] Z. Křivka. Recursive erasing in programmed grammars. In *Pre-Proceedings of the 1st Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 139–144, Znojmo, CZ, 2005. Faculty of Informatics MU.

[40] M. K. Levitina. On some grammars with rules of global replacement (in Russian). *Scientific-Technical Information*, 2(3):32–36, 1972.

[41] P. Linz. *An Introduction to Formal Languages and Automata (4th ed.)*. Jones and Bartlett Publishers, 2006. ISBN 0-7637-3798-4.

[42] S. Marcus, C. Martín-Vide, V. Mitrana, and G. Păun. A new-old class of linguistically motivated regulated grammars. In *Computational Linguistics in the Netherlands*, pages 111–125, 2000.

[43] C. Martín-Vide and V. Mitrana. Further properties of path-controlled grammars. In *FG-MoL '05: The 10th Conference on Formal Grammar and The 9th Meeting on Mathematics of Language*, pages 221–232, 2005.

[44] C. Martín-Vide, V. Mitrana, and G. Păun, editors. *Formal Languages and Applications*, chapter 13, pages 249–274. Springer, 2004. ISBN 3-540-20907-7.

[45] T. Masopust. On the descriptional complexity of scattered context grammars. *Theoretical Computer Science*, 410(1):108–112, 2009.

[46] T. Masopust and A. Meduna. Descriptional complexity of grammars regulated by context conditions. In *LATA '07 Pre-proceedings. Reports of the Research Group on Mathematical Linguistics 35/07, Universitat Rovira i Virgili*, pages 403–411, Tarragona, Spain, 2007.

[47] T. Masopust and A. Meduna. On descriptional complexity of partially parallel grammars. *Fundamenta Informaticae*, 87(3):407–415, 2008.

[48] T. Masopust and A. Meduna. Descriptional complexity of three-nonterminal scattered context grammars: An improvement. In *Proceedings of 11th International Workshop on Descriptional Complexity of Formal Systems*, pages 235–245. Otto-von-Guericke-Universität Magdeburg, 2009.

[49] A. Meduna. Syntactic complexity of scattered context grammars. *Acta Informatica*, 1995(32):285–298, 1995.

[50] A. Meduna. *Automata and Languages: Theory and Applications*. Springer, 2000. ISBN 1-85233-074-0.

[51] A. Meduna. Generative power of three-nonterminal scattered context grammars. *Theoretical Computer Science*, 2000(246):279–284, 2000.

[52] A. Meduna. Coincidental extention of scattered context languages. *Acta Informatica*, 39(5):307–314, 2003.

[53] A. Meduna. Erratum: Coincidental extension of scattered context languages. *Acta Informatica*, 39(9):699, 2003.

[54] A. Meduna. *Elements of Compiler Design*. Auerbach Publications, 2007. ISBN 1-4200-6323-5.

[55] A. Meduna and J. Techet. Canonical scattered context generators of sentences with their parses. *Theoretical Computer Science*, 2007(389):73–81, 2007.

[56] A. Meduna and J. Techet. Scattered context grammars that erase nonterminals in a generalized k-limited way. *Acta Informatica*, 45(7):593–608, 2008.

[57] A. Meduna and J. Techet. *Scattered Context Grammars and their Applications*. WIT Press, 2010. ISBN 1-84564-426-3.

[58] A. Meduna and M. Švec. *Grammars with Context Conditions and Their Applications*. Wiley, 2005. ISBN 0-471-71831-9.

[59] S. S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann, 1997. ISBN 1-55860-320-4.

[60] P. Navrátil. Syntaktická analýza založená na řízených gramatikách. Master's thesis, Vysoké učení technické v Brně, Fakulta informačních technologií, 2004.

[61] R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

[62] T. W. Parsons. *Introduction to Compiler Construction*. Computer Science Press, Inc., 1992. ISBN 0-7167-8261-8.

[63] W. Reisig. *Petri Nets: An Introduction*. Springer, 1985. ISBN 0-387-13723-8.

[64] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the ACM*, 16(1):107–131, 1969.

[65] G. Rozenberg and A. Salomaa. *Mathematical Theory of L Systems*. Academic Press, 1980. ISBN 0-12-597140-0.

[66] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages, Vol. 1: Word, Language, Grammar*. Springer, 1997. ISBN 3-540-60420-0.

[67] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages, Vol. 2: Linear Modeling: Background and Application*, chapter 3, pages 101–154. Springer, 1997. ISBN 3-540-61486-9.

[68] A. Rußmann. Dynamic LL(k) parsing. *Acta Informatica*, 34(4):267–289, 1997.

[69] A. Salomaa. *Formal Languages*. Academic Press, 1973. ISBN 0-12-615750-2.

[70] M. Sipser. *Introduction to the Theory of Computation (2nd ed.)*. Course Technology, 2006. ISBN 0-534-95097-3.

[71] R. Siromoney and K. Krithivasan. Parallel context-free languages. *Information and Control*, 24(2):155–162, 1974.

[72] A. P. J. van der Walt. Random context grammars. In *Proceedings of Symposium on Formal Languages*, pages 163–165, 1970.

[73] G. Vaszil. On the descriptional complexity of some rewriting mechanisms regulated by context conditions. *Theoretical Computer Science*, 330(2):361–373, 2005.

[74] D. Wätjen. Regulation of k-limited ETOL systems. *International Journal of Computer Mathematics*, 47(1):29–41, 1993.

[75] D. Wood. *Theory of Computation: A Primer*. Addison-Wesley Longman Publishing Co., Inc., 1987. ISBN 0-06-047208-1.

[76] G. Zetzsche. Erasing in Petri net languages and matrix grammars. In *DLT '09: Proceedings of the 13th International Conference on Developments in Language Theory*, pages 490–501. Springer, 2009.

[77] G. Zetzsche. A note on Hack's conjecture, Parikh images of matrix languages and multiset grammars. Bericht des Fachbereichs Informatik FBI-HH-B-289/09, Universität Hamburg, Germany, 2009.