BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

REGULATED GRAMMARS: CONCEPTS, PROPERTIES AND APPLICATIONS

REGULOVANÉ GRAMATIKY: KONCEPTY, VLASTNOSTI A VYUŽITÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE
Bc. PETR BEDNÁŘ

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

BRNO 2016

Master Thesis Specification/17977/2015/xbedna46

Brno University of Technology - Faculty of Information Technology

Department of Information Systems Academic year 2015/2016

Master Thesis Specification

For:	**Bednář Petr, Bc.**
Branch of study:	Information Systems
Title:	**Regulated Grammars: Concepts, Properties and Applications**
Category:	Theoretical Computer Science

Instructions for project work:

1. Study regulated grammars, their properties and applications. Consult this study with your advisor.
2. Introduce new regulated grammars.
3. Study the properties of grammars defined in 2. Compare their power with other grammars.
4. Discuss applications of grammars defined in 2. Consider complicated syntax structures that are non-context-free. Describe their parsing based on grammars from 2.
5. Implement and test the parsing methods developed in 4.
6. Summarize the results. Discuss the future investigation concerning this project.

Basic references:

- 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

Requirements for the semestral defense:
Parts 1 and 2.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

Supervisor:	**Meduna Alexander, prof. RNDr., CSc.**, DIFS FIT BUT
Beginning of work:	November 1, 2015
Date of delivery:	May 25, 2016

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

L.S.

[illegible]

Dušan Kolář
Associate Professor and Head of Department

Abstract

This thesis is investigating regulated grammars. Introduces new modifications of existing regulated grammars. Introduces parsing methods of newlz introduces modifications. Discusses problems of determinism in definition of grammars. It studies the expressive strength of these new modifications applied on regular-controlled grammars.

Abstrakt

Tato práce se zabývá regulovanými gramatikami. Zavádí nové modifikace existujících regulovaných gramatik. Pro tyto modifikace zavádí metody syntaktické analýzi. Diskutuje problémy determinismu v definici gramatik. Studuje sílu nově uvedených modifikací aplikovaných na regulárně regulované gramatiky.

Keywords

regulated grammars, determinism, syntax analysis, regular-controlled grammars, LL grammars

Klíčová slova

regulované gramatiky, determinismus, syntaktická analýza, regulárně regulované gramatiky, LL gramatiky

Reference

BEDNÁŘ, Petr. *Regulated Grammars:*
Concepts, Properties and Applications. Brno, 2016. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Meduna Alexander.

Regulated Grammars: Concepts, Properties and Applications

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of prof. RNDr. Alexandra Meduny, CSc. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

....
Petr Bednář
May 25, 2016

Acknowledgements

I would like to express my gratitude to my supervisor prof. RNDr. Alexander Meduna, CSc. for his guidance and kind approach.

Contents

1 Introduction 5
1.1 Focus . . . 6
1.2 Organization . . . 6

2 Basic Concepts 7
2.1 Basic Definitions . . . 7
2.1.1 Alphabets and Words . . . 7
2.1.2 Languages . . . 8
2.2 Grammars . . . 9
2.3 Language Families . . . 10
2.3.1 Recursively Enumerable Languages . . . 10
2.3.2 Context-Sensitive Languages . . . 10
2.3.3 Context-Free Languages . . . 11
2.3.4 Metalinear Languages . . . 11
2.3.5 Linear Languages . . . 12
2.3.6 Regular Languages . . . 12
2.3.7 Finite Languages . . . 13
2.3.8 Chomsky Hierarchy . . . 13
2.4 Automata . . . 13
2.4.1 Finite Automata . . . 13
2.4.2 Grammar Automata . . . 14
2.5 Derivation Trees . . . 16
2.6 Syntax Analysis . . . 16

3 Regulated Grammars 18
3.1 Reasons to Study . . . 18
3.1.1 Context-Free Grammars . . . 19
3.1.2 Context-Sensitive Grammars . . . 19
3.1.3 Computational History . . . 20
3.2 Basic Concepts . . . 21
3.3 Context-Based Regulation . . . 21
3.3.1 Random Context Grammars . . . 21
3.3.2 Context-Conditional Grammars . . . 22
3.3.3 Scatered Context Grammars . . . 22
3.3.4 Restricted Derivation Tree Grammars . . . 22
3.4 Rule-Based Regulation . . . 22
3.4.1 Regular-Controlled Grammars . . . 22
3.4.2 Matrix Grammars . . . 23

3.4.3 Programmed Grammars . . . 23
3.4.4 State Grammars . . . 23
3.5 On Adaptive Grammars . . . 24
3.6 Leftmost Modification . . . 24

4 Grammar Automata **26**
4.1 Positioned Grammar Automata . . . 26
4.2 Basic transformations . . . 26
4.3 Operation . . . 28

5 Switch-Paused Regulated Grammars **29**
5.1 Definition . . . 29
5.2 Other Regulations . . . 31
5.2.1 Rule-Based Regulations . . . 31
5.2.2 Context-Based Regulations . . . 32
5.3 Examples . . . 32
5.4 Generation . . . 34
5.4.1 Leftmost Modification . . . 35
5.5 Parsing . . . 35
5.5.1 Specific Transformations . . . 35
5.5.2 Analysis . . . 37
5.6 Properties . . . 37
5.6.1 Switch-Paused to Original . . . 37
5.6.2 Original to Switch-Paused . . . 38
5.6.3 Expressive Strength . . . 38

6 Determinism-Paused Regulated Grammars **40**
6.1 On Determinism in Generation . . . 40
6.1.1 Undeterministic Generation . . . 41
6.1.2 Ways to Determinism . . . 41
6.2 Definition . . . 47
6.2.1 Leftmost Modification . . . 49
6.3 Other Regulations . . . 49
6.3.1 Random Context Grammars . . . 49
6.3.2 Tree-Regulated Grammars . . . 49
6.4 Examples . . . 49
6.5 Generation . . . 52
6.6 Parsing . . . 52
6.6.1 Application of Limitations . . . 53
6.6.2 Specific Transformations . . . 53
6.7 Properties . . . 53
6.7.1 Original to Determinism-paused . . . 53
6.7.2 Expressive Strength . . . 54

7 Conclusions **57**

Bibliography **59**

Appendices **60**
List of Appendices 61

A Content of CD **62**

List of Figures

2.1 Chomsky hierarchy of languages 13

4.1 Single rules automata 27
4.2 Concatenation of M_1 and M_2 27
4.3 Union of M_1 and M_2 27
4.4 Iteration of M_1 27

5.1 Original initial state 36
5.2 Transformed initial state 36
5.3 Original off-switching state 36
5.4 Transformed off-switching state 36

6.1 Original state 53
6.2 Transformed state 53

Chapter 1

Introduction

Languges are the single meaning of a communication. Humans use natural languages to convey and store information. Natural languages are rich in their varieties and capabilities. They have also few crucial disadvantages. Mutual understanding of two different parties inherently depends on shared model of a world. Any differences can lead to information loss or change. In time, the model can be changed. This may result in loss of stored informations even with unharmed data, which represents it. The natural languages proven itselves to be hard to machine analyse. This is largely due to wide range of variations and minute differences, brought by any human user.

The disadvantages and hardship with using natural languages while interfacing computers, led to creation of formal languages. Now, formal languages are one of corner stones of theoretical informatics. They allow us, to record both data and algoriths, used to manipulate those data. Basic form of programming languages and communication protocols are usually defined using formal languages. Their rigid and formaly specified notation allow us to communicate safely, without the danger of misinterpretation.

The formal languages are classified into different language families. Families are created by similarities in instances of related instances of languages. They also pose a restrictions on languages. The formal language theory uses mainly two basic kinds of formal models: grammars, which defines a language by capability to generate its every word, and automata, which defines a by ability to accept its every word.

The grammar is a generative model of formal language. The start symbol is rewriten, untill a sentence of language is acquired. The rewrites are controlled by a set of rules, which are an only form of action a grammar can take.

One of the families of formal languages are context-free languages, specified by context-free grammars. The field of context-free grammars was studied for a long time. The goal was, to use them to specify both, a natural and programming lanugages. Context-free languages, and grammars, are comparably simple in their form. But they often fail in real applications, due to limits of their expressive strength. Not even commonly used programming languages can be specified by them.

Regulated languages aims to increase expressive power of their unregulated counterparts. This should be achieved without changing of basic form of production rules. In history, many models of regulated grammars has been introduced. They use regulations both imperatively, when describing order of productions, or declaratively, when deciding only by the combined effect, created by using number of production rules.

1.1 Focus

The goal of this thesis is creation of new models of regulated grammars. This new models are based on already existing models, not creating entirely new branch of regulated grammars. We introduce new modes of operation, for already existing regulated grammars.

For newly introduced models, a formal definition and an example of parsing technique, is presented. The computational power is investigated.

1.2 Organization

The theis is organized into 6 chapters. We will look closer to the content of each of them.

- Chapter 1, this chapter, serves as a simple introduction into field of study, outlines the focus of this work and describes its structure.
- Chapter 2 describes background definitions of basic terms, used later in this thesis. This includes terms on formal languages, grammars and automata. This thesis require the reader to be familiar with basic mathematical concepts and notation.
- Chapter 3 is a closer study of the current state of regulated grammars study. The regulated grammars are cathegorized acording to similarities of productions and means of conduct.
- Chapter 4 describes a basic model used for parsing of a newly introduced grammars.
- Chapter 5 introduces the first of newly proposed modes of regulation, which is called *pause-paused mode*. We define this mode for already existing language-controlled regulation model. Chapter describes a way, to generate a valid sentence under this mode and introduces a way for parsing a sentence using this modification.
- Chapter 6 introduces the second of newly proposed modes of regulation, this one is called *determinism-paused mode*. As for the first mode, we define this mode for already existing language-controlled regulation model. Chapter describes a way, to generate a valid sentence under this mode and introduces a way for parsing a sentence using this modification.
- Chapter 7 summarizes the results of this work, and discusses the several ways for further research of recently presented modes of regulated grammars.

Chapter 2

Basic Concepts

This chapter gives the survey of the fundamental terms,used in formal languages theory and later in this thesis. At first section 2.1, the formal languages and its parts are defined. Next two sections, 2.2 and 2.4, serves as an introduction to the models, used to specify formal languages. A section 2.3 is dedicated to establishing of hiearchical view of formal languages. A section 2.5 is explaining the term of derivation tree. Lastly, a section 2.6 describes a term of syntax analysis.

The definitions and formalisms in this chapter are based on [10], [11], [7] and [14].

2.1 Basic Definitions

In order to underestand terms, introduced in this thesis, we need to establish a common set of knowledge. Hence, we will introduce several terms, related to the formal languages. This terms are used later in this thesis.

The *set*, *tuple* and *sequence* are standard mathematical devices. Rigorous definitions can be studied in literature. It is expected from the reader, to be familiar with them. For sequence, the term *group* may be used interchangeably.

2.1.1 Alphabets and Words

Definition 2.1 (Symbol)
The *symbol* is a basic atomic unit in formal languages.

Definition 2.2 (Alphabet)
The *alphabet* is a finite, nonempty set of symbols.

Definition 2.3 (Size of alphabet)
Let Σ be an alphabet. Then the $|\Sigma|$ denotes the *size of alphabet* Σ. It is defined as a number of symbols contained in set Σ. The *empty alphabet* is an alphabet, containing no symbols.

Definition 2.4 (Word and string)
Let Σ be an alphabet. The *word* is then ordered group of symbols of any length. For a group $(x_0, x_1, \ldots, x_n)$, where $n \geq 1$, we can use shorter notation $x_0x_1 \ldots x_n$. Alternative name for a word is *string*.

- ε is an empty group, which contains no symbols. It is called an *empty word*.
- If x is a word over an alphabet Σ and $a \in \Sigma$, then xa is a string over an alphabet Σ.

Let Σ^* denote the set of all words over Σ and set $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

Definition 2.5 (Length of word)
Let x be a word over an alphabet Σ. Then *length of word* x, denoted as $|x|$, is defined as a number of symbols in a word.

- If $x = \varepsilon$, then $|x| = 0$.
- If $x = a_1a_2 \dots a_n$, where $a_i \in \Sigma$, for all $0 \leq i \leq n$ and some $n \geq 1$, then $|x| = n$.

Definition 2.6 (Concatenation of words)
Let x and y be two words over an alphabet Σ. Then, xy is the *concatenation* of x and y. For every word x, it holds $x\varepsilon = \varepsilon x = x$.

Definition 2.7 (Power of word)
Let x be a word over an alphabet Σ. Then, x^n denotes *nth power of word*, which is defined such:

- $x^0 = \varepsilon$,
- $x^n = xx^{n-1}$, where $n \geq 1$ and n is an integer.

Definition 2.8 (Subword and substring)
Let x and y be two words over an alphabet Σ. Then x is a *subword* of y if there exist two words z and z', over Σ, such that $zxz' = y$. If $x \notin \{\varepsilon, y\}$ then x is a *proper subword* of y.

Definition 2.9 (Prefix and suffix)
Let x, y and z be words over an alhabet Σ, such that $x = yz$. Then y is a *prefix* of x and z is a *suffix* of x. If $y \notin \{\varepsilon, x\}$, then y is a *proper prefix* of x. If $z \notin \{\varepsilon, x\}$, then z is a *proper suffix* of x.

2.1.2 Languages

Definition 2.10 (Language)
Let Σ be an alphabet. Then any set $L \subseteq \Sigma^*$ is a *language* over Σ. The set Σ^* is called the *universal language*. It contains all strings over the alphabet Σ. If $L = \varnothing$, then L is an *empty language*. If L contains single string, then L is a *unary language*.

Definition 2.11 (Concatenation of languages)
Let L_1 and L_2 be two languages. The *concatenation of languages* L_1 and L_2, denoted by L_1L_2, is defined as a language containing all possible concatenations of strings of both languages. $L_1L_2 = \{xy | x \in L_1 \wedge y \in L_2\}$.

Definition 2.12 (Union of languages)
Let L_1 and L_2 be two languages. The *union of languages* L_1 and L_2, denoted by $L_1 \cup L_2$, is defined as a language containing sentences of both languages. $L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$.

Definition 2.13 (Intersection of languages)
Let L_1 and L_2 be two languages. The *intersection of languages* L_1 and L_2, denoted by $L_1 \cap L_2$, is defined as a language containing sentences, which appears in both languages. $L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$.

Definition 2.14 (Difference of languages)
Let L_1 and L_2 be two languages. The *difference of languages* L_1 and L_2, denoted by $L_1 \setminus L_2$, is defined as a language L_1 without a sentences of language L_2. $L_1 \setminus L_2 = \{x | x \in L_1 \wedge x \notin L_2\}$.

Definition 2.15 (Complement of language)
Let L be a language over an alphabet Σ. The *complement of language* L, denoted by $\bar{L}$, is defined as $\bar{L} = \Sigma^* \setminus L$.

Definition 2.16 (Power of language)
Let L be a language. The *nth power of language* L, denoted by L^n, for $n \geq 0$, is defined as a number of language concatenations. The recursive definition is

- $L^0 = \{\varepsilon\}$,
- $L^n = LL^{n-1}$, where $n \geq 1$ and n is an integer.

Definition 2.17 (Class of languages)
The *class of languages* is defined as a set of languages.

2.2 Grammars

In the formal language theory, a grammar is a fundamental model for generating languages. When grammar defines a language, then all and no other string can be generated by this grammar.

Definition 2.18 (Phrase structure grammar)
A *phrase structure grammar* G is a quadruple

$$G = (N, T, P, S), \tag{2.1}$$

where

- N is an alphabet of *nonterminals*;
- T is an aplhabet of *terminals*, $N \cap T = \varnothing$;
- $P \subset (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ is a finite set of *rules*;
- $S \in N$ is the *start symbol*.

Pairs $(\alpha, \beta) \in P$ are also called *rewriting rules* or *production rules*.

Convention: Instead of $(\alpha, \beta) \in P$, the form $(\alpha \to \beta) \in P$ can be used.

The set $V = N \cup T$ is the *total alphabet* of G. A rewriting rule $\alpha \to \varepsilon \in P$ is called an *erasing rule*. If there is no erasing rule in P, then we say that G is a *propagating*, or *ε-free* grammar.

The word from V^* is called a *sentential form* and the word from T^* is called a *sentence*.

The G-based *direct derivation* is a relation over V^*. It is denoted by a symbol $\Rightarrow_G$ and defined as

$$x \Rightarrow_G y \tag{2.2}$$

if and only if $x = x_1\alpha x_2$, $y = x_1\beta x_2$, where $x_1, x_2 \in V^*$ and $\alpha \to \beta \in P$.

Since $\Rightarrow_G$ is a relation, $\Rightarrow_G^k$ is the kth power of $\Rightarrow_G$, for $k \geq 0$, $\Rightarrow_G^+$ is the transitive closure of $\Rightarrow_G$, and $\Rightarrow_G^*$ is the reflexive-transitive closure of $\Rightarrow_G$. Let D be a derivation $S \Rightarrow_G^* x$, if $x \in T^*$, then D is a *successful* (or *terminal*) *derivation.*

The *language* of G, denoted by $L(G)$, is the set of all sentences defined as

$$L(G) = \{w \in T^* | S \ \Rightarrow_G^* w\}. \tag{2.3}$$

For every phrase-structure grammar G, we define two sets, $F(G)$ and $\Delta(G)$. $F(G)$ contains all sentential forms of G. $\Delta(G)$ contains all sentential forms from which there is a derivation of a string in $L(G)$.

The grammar G can be writen as a quintuple (N, T, Ψ, P, S). In this form, the additional Ψ denotes a set of *labels.* For each rule there exists exactly one label, which is paired with a bijection ψ from Ψ to P.

2.3 Language Families

Language family is other name used for language class. This name is usualy used for systematically created class, not just arbitrary created one. With both language family or class, we sometimes use another term, an *expressive power*, or *computational power.* The model has an expressive power of a certain language class, when we are able to fully describe this class by this model. With expressive power we manipulate in a similar maner to sets. When a model has a power to express certain class $\mathcal{L}_1$, we expect it to have power to express any class $\mathcal{L}_2 \subseteq \mathcal{L}_1$ fully.

2.3.1 Recursively Enumerable Languages

Definition 2.19 (Recursively enumerable language)
A *recursive enumerable language* is a language, for which there exists a phrase-structured grammar.

Commonly used abbreviation is **RE** or *type 0.* Any language models that characterize **RE** are said to be *computationaly complete.* They have the same expressive strength as all possible language-defining procedures according to Church-Turing's thesis.

Convention: The family of recursively enumerable languages is denoted by $\mathcal{L}(RE)$.

2.3.2 Context-Sensitive Languages

Definition 2.20 (Context-sensitive grammar)
A *context-sensitive grammar* is a phrase-structure grammar

$$G = (N, T, P, S) \tag{2.4}$$

such that every $\alpha \to \beta \in P$ satisfies the form

$$\alpha = x_1 A x_2,\ \beta = x_1 y x_2,\ A \in N,\ \alpha, \beta \in (N \cup T)^*,\ y \in (N \cup T)^+. \tag{2.5}$$

Definition 2.21 (Context-sensitive language)
A *context-sensitive language* is a language generated by a context-sensitive grammar.

Commonly used abbreviation is **CS** or *type 1.*

Convention: The family of context-sensitive languages is denoted by $\mathcal{L}(CS)$.

2.3.3 Context-Free Languages

Definition 2.22 (Context-free grammar)
A *context-free grammar* is a phrase-structure grammar

$$G = (N, T, P, S) \tag{2.6}$$

such that every $\alpha \to \beta \in P$ satisfies the form

$$\alpha \in N,\ \beta \in (N \cup T)^*. \tag{2.7}$$

Definition 2.23 (Context-free language)
A *context-free language* is a language generated by a context-free grammar.

Commonly used abbreviation is **CF** or *type 2*.

Convention: The family of context-free languages is denoted by $\mathcal{L}(CF)$.

Ambiguity

For context-free languages we define few additional properties. These are directly tied to context-free grammars.

Definition 2.24 (Ambiguity)
Let $G = (N, T, P, S)$ be a context-free grammar. If there exists a word $x \in L(G)$ such that $S \Rightarrow_G^* x[\pi_1]$ and $S \Rightarrow_G^* x[\pi_2]$ with $\pi_1 \neq \pi_2$, then G is *ambiguous*; otherwise, G is *unambiguous*.

Definition 2.25 (Inherent ambiguity)
Let L be language. If every context-free grammar G satisfying $L(G) = L$ is ambiguous, then L is *inherently ambiguous*.

2.3.4 Metalinear Languages

Definition 2.26 (Metalinear grammar)
A *metalinear grammar* is a phrase-structure grammar

$$G = (N, T, P, S) \tag{2.8}$$

such that every $\alpha \to \beta \in P$ satisfies the form

$$\alpha \in N,\ \beta = x_1 B x_2,\ x_1, x_2 \in T^*,\ B \in (N \cup \varepsilon) \tag{2.9}$$

or

$$\alpha = S,\ \beta = x_0 B_1 x_2 B_2 \dots x_{n-1} B_n x_n,\ x_i \in T^*,\ B \in (N \setminus \{S\}). \tag{2.10}$$

Occurence of rule in second form implies, that no rule uses symbol S on its right-hand side.

Definition 2.27 (Metalinear language)
A *metalinear language* is a language generated by a metalinear grammar.

Commonly used abbreviation is **MLIN**.

Convention: The family of context-free languages is denoted by $\mathcal{L}(MLIN)$.

2.3.5 Linear Languages

Definition 2.28 (Linear grammar)
A *linear grammar* is a phrase-structure grammar

$$G = (N, T, P, S) \tag{2.11}$$

such that every $\alpha \to \beta \in P$ satisfies the form

$$\alpha \in N,\ \beta = x_1 B x_2,\ x_1, x_2 \in T^*,\ B \in (N \cup \varepsilon). \tag{2.12}$$

Definition 2.29 (Linear language)
A *linear language* is a language generated by a linear grammar.

Commonly used abbreviation is **LIN**.

Convention: The family of context-free languages is denoted by $\mathcal{L}(LIN)$.

2.3.6 Regular Languages

Definition 2.30 (Regular grammar)
A *regular grammar* is a phrase-structure grammar

$$G = (N, T, P, S) \tag{2.13}$$

such that every $\alpha \to \beta \in P$ satisfies the form

$$\alpha \in N,\ \beta = aB,\ a \in T,\ B \in (N \cup \varepsilon). \tag{2.14}$$

Definition 2.31 (Regular language)
A *regular language* is a language generated by a regular grammar.

Commonly used abbreviation is **REG** or *type 3*.

Convention: The family of regular languages is denoted by $\mathcal{L}(REG)$.

The regular language, is the weakest form of infinite languages we are using. If we want to write down any infinite language, we cannot list every possible sentence. We usually use grammar, automata or set of conditions, that specifies this language. For regular languages we also specify *regular expressions.* These are based on specification of sets of allowed substrings.

Used notation of regular expressions:

- $\varnothing$ is a regular expression denoting an *empty set*;
- ε is a regular expression denoting a set $\{\varepsilon\}$;
- a is a regular expression denoting a set $\{a\}$, for $a \in \Sigma$;
- if x and y are both regular expressions, denoting a sets X and Y, we may construct another expressions:
  - $x + y$ is a regular expression, denoting a set $X \cup Y$,
  - xy is a regular expression, denoting a set $X \cdot Y$,
  - x^* is a regular expression, denoting a set X^*,
  - x^+ is a regular expression, denoting a set $X \cdot X^*$,
  - $x^?$ is a regular expression, denoting a set $\varepsilon \cup X$.

2.3.7 Finite Languages

Definition 2.32 (Finite languages)
A *finite language* is a lanugage L with size $|L| = n$, where n is a finite integer. A finite lanugage can be completly listed in a finite ammount of time.

Commonly used abbreviation is **FIN**.

Convention: The family of finite languages is denoted by $\mathcal{L}(FIN)$.

2.3.8 Chomsky Hierarchy

Language classes can be classified into hierarchical language system. This system is organised by properties of languages. One of the well-known hierarchies was created by Noam Chomsky [3]. This hierarchy has proven itself to be usefull in qualification of concrete models of languages. It is specified with theorem 2.1 [4].

Theorem 2.1 (Chomsky hierarchy)
$\mathcal{L}(REG) \subset \mathcal{L}(CF) \subset \mathcal{L}(CS) \subset \mathcal{L}(RE)$

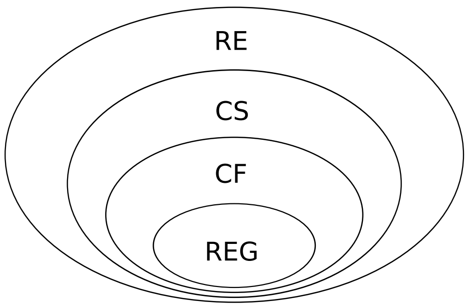

Figure 2.1: Chomsky hierarchy of languages

This hierarchy can be expanded by adding another classes of languages, to get expanded hierarchy 2.2 [7][Theorem 0.2.3].

Theorem 2.2 (Extended Chomsky hierarchy)
$\mathcal{L}(FIN) \subset \mathcal{L}(REG) \subset \mathcal{L}(LIN) \subset \mathcal{L}(CF) \subset \mathcal{L}(CS) \subset \mathcal{L}(RE)$

2.4 Automata

In the formal language theory, an antomata serves as a counterpart to grammars. Automata are models for accepting languages. When language is defined by an antumata, then this automata has to succesfully stop for all and no other string of this language. This section is limited to finite automata, pushdown automata and grammar automata because they serve as a models used later. Many other automata systems exists.

2.4.1 Finite Automata

Finite automata is one of basic automata models. It has advantage of constant space complexity, which also limits its expressive power.

Definition 2.33 (Finite automaton)
A finite automaton M is a quintuple

$$M = (Q, \Sigma, R, s, F), \tag{2.15}$$

where

- Q is a finite set of *states*;
- Σ is an *input alphabet*, $Q \cap \Sigma = \varnothing$,
- $R \subset Q(\Sigma \cup \{\varepsilon\}) \times Q$ is a finite set of *rules*;
- $s \in Q$ is the *initial state*;
- $F \subseteq Q$ is a set of *final states.*

Convention: Instead of $(pa, q) \in R$, the form $pa \to q \in R$ is preferred.

If R has the properties of a function and it holds, that

$$\forall p \in Q \ \forall a \in \Sigma \ \exists q, q` \in Q : p \to q \in R \Rightarrow pa \to q` \notin R \tag{2.16}$$

then finite automata is said to be *deterministic.*

A *configuration* Ξ of M, is defined as $\Xi \in Q\Sigma^*$. Let $\xi = paw$ and $\xi` = qw$ be two configurations of $M$, where $w \in \Sigma^*$, $a \in \Sigma \cup \{\varepsilon\}$, and $p, q \in Q$. If $r : pa \to q \in R$ si a rule, then $M$ makes a *move* from $\xi$ to $\xi`$ according to r, writen as $\xi \vdash_M \xi`[r]$ or $\xi \vdash_M \xi`$.

The *language* $L(M)$ accepted by finite automaton M is defined as

$$L(M) = \{w \in \Sigma^* \mid sw \vdash_M^* f[\pi], f \in F\}. \tag{2.17}$$

Convention: Finite automata is abbreviated with FA, the deterministic variant by DFA. The family of languages accepted by FA is denoted by $\mathcal{L}(FA)$

Theorem 2.3 (Fininte automata strength)
$\mathcal{L}(FA) = \mathcal{L}(REG)$ [13]

2.4.2 Grammar Automata

The grammar automaton is an automaton builded upon a finite automaton. To this automaton is added a work string and input string. The automaton generates a working model of a sentential form in its work string. Using this template, it reduces the input string till no original input symbols are left.

Definition 2.34 (Grammar automaton)
The *grammar automaton GA*, see [2][Definition 4.1], is an 8-tuple

$$GA = (Q, \Sigma, \Gamma, R, s, S, F, \delta), \tag{2.18}$$

where

- $Q = Q_e \cup Q_r$, $Q_e \cap Q_r = \varnothing$, where Q_e is a finite set of *empty states*, which does not apply any rules and Q_r is a finite set of *non-empty states.* Every non-empty state t applies exactly one production rule $\rho_t : \Gamma^+ \to \Gamma^*$, when the automaton moves into this state;

- Σ is an input aplhabet;
- Γ is an alphabet of work symbols;
- $R \subseteq (Q \times \mathbb{N} \times f \times Q)$ is a finite set of *rules*, where $f : (\Gamma^* \times \Sigma^* \times \{0,1\})$ is a *feasibility function.* The rule, writen as $(p, v, f, q) \in R$, is *feasible* only when the feasibility function has a value 1 as its last element. The symbol v denotes the *priority* of the rule;
- $s \in Q$ is an *initial state*;
- $S \in \Gamma^+$ is an *initial work string*;
- $F \subseteq Q$ is a finite set of *final states*;
- $\delta : (\Gamma^* \times \Sigma^* \times \Gamma^* \times \Sigma^*)$ is a *normalization function*, which is applied after every application of the rule, to acquire a *normalized form* of a work and input string.

Convention: Instead of $(p, v, f, q) \in R$, the form $p(v, f) \to q \in R$ is preferred.

If feasibility function is constantly 1 and a priority is 0, the rule is called to be *joining.* The rule is r_1 is said to be *executable* only when

$$\begin{aligned} r_1 &: p(v_1, f_1) \to q_1, \\ r_2 &: p(v_2, f_2) \to q_2, \end{aligned} \tag{2.19}$$

where $r_1, r_2 \in R$, $v_1 \leq v_2$, $S_1 \in \Gamma^*$, $S_2 \in \Sigma^*$ and $f_1(S_1, S_2) = 1 \wedge f_2(S_1, S_2) = 0$

Let $s_0, s_1, \ldots, s_n$, for $n \geq 1$, be a *string of states* of grammar automata, then therealways exists at least one rule such that

$$s_m(v, f) \to s_{m+1} \in R \wedge 0 \leq m \leq n - 1. \tag{2.20}$$

The *configuration* of the grammar automata is a triplet (q, P, I), where $q \in Q$, $P \in \Gamma^*$ is the work string and $I \in \Sigma^*$ denotes the input string.

The grammar automaton is said to be *finished*, when in an actual configuration (q, P, I), the state $q \in F$ and both P and I equals ε.

The *transition* of a grammar automaton from a state p into a state q by a rule r is denoted by

$$p \vdash q[r]. \tag{2.21}$$

For joining rule, we can use a notation

$$p \vdash_\varepsilon q[r]. \tag{2.22}$$

The *step* $c[r]$ of a grammar automata from a state p_0 into a state p_2 by a rule r is a string of transistions, containing exactly one non-joining transition, such that

$$c: \ p_0 \vdash_\varepsilon^* p_1 \vdash p_2[r]. \tag{2.23}$$

For every state $p \in Q$ if a grammar automaton is defined a *joining closure* $C(p)$ such that

$$C(p) = \{q \mid q \in Q \wedge p \vdash_{eps}^* q\}. \tag{2.24}$$

Convention: Abbreviation of grammar automata is GA.

The grammar automata starts at initial configuration, which consists of initial state, initial work string and initial input. At every step, the automata applies a single rule, represented by an arriving into an automata state. This transition is permitted only when a feasibility function for current configuratio equals 1. There exists a hierarchy of rules, created by its priorities. Lower priority is always preferred. After each change of a work or input string, the normalization function is used, to acquire a mormalized state, ready for next step.

2.5 Derivation Trees

A derivation tree represents the structure of a derivation using a graph. This representation does preserve the structure of the derivation, but ommits an order in which individual rules were applied.

It is expected from a reader to be familiar with the basic terms of the graph theory. These includes *graph*, *directed graph*, *edge*, *path* and *tree*. For more information, see [8].

Definition 2.35 (Directed subtree)
Let $T = (V, \rho)$ be a directed tree. A tree $T‘ = (V‘, \rho‘)$ is a *directed subtree*, when

- $V' \subseteq V$ and $V' \neq \varnothing$;
- $\rho' = (V' \times V') \cap \rho$;
- there is no simple path from any node in V' to any node in $V \setminus V'$ in T

Definition 2.36 (Ordered directed tree)
Let $T = (V, \rho)$ be a directed tree. It is an *ordered directed tree*, when upon nodes $v_1, v_2 \dots v_n$, for $n \geq 0$, which are direct descendants of node u, there exists total order.

Definition 2.37 (Derivation tree)
Let $G = (N, T, P, S)$ be a context-free grammar. Then *derivation tree* is an ordered directed graph, such that:

- Nodes of a derivation tree are labelled with a member of $N \cup T \cup \{\varepsilon\}$;
- Root of a derivation tree is labelled with S;
- Let nodes $v_1, v_2 \dots v_n$, labeled with $y_1, y_2 \dots y_n$, be a direct descendants of a node u, labeled with x. The ordering $<$ is defined as $v_i < v_{i+1}$. Then there exists a rule $x \to y_1 y_2 \dots y_n$. Descendant labeled with a symbol ε is allowed only for an erasing rule.

2.6 Syntax Analysis

Grammars are used to generate strings of formal language. *Syntax analysis* [1], or parsing, is a reverse process with a simple goal. Take a concrete string as an input and decide, whethever this string is part of analysed language, or not.

The syntax analysis is performed by a syntax analyser. This syntax analyser is created on the basis of a concrete grammar. Its input is a sentence, for which it decides membership of a sentence to the language specified by the grammar.

We can classify syntax analysers by the properties of its analysis process. We can divide them based on a direction of derivation and a number of tried derivations.

Top-Down Approach

The top-down analysis is starting from an initial string, defined by the grammar. This strings is then derived using a derivation rules, also specified by the grammar. The derivations continue, until the derived string does not equal the input, in which case the output is positive.

Bottom-Up Approach

The bottom-up approach starts with the input string itself. On this string, it applies the derivation rules in a reverse, in an attempt to acquire the initial string of the grammar. When the analysis acquires the initial string, it proves, there is a derivation for an input sentence and ends.

Brute Force Approach

This method of syntax analysis is using some sort of blind search, to generate every possible derivation tree for a given input. It can use any way, to generate the combinations. We should note, that the grammars can contain a recursion, in which case, the depth-first search [6] may not be the best choice, because of infinite loop may occure.

Knowledge Based Approach

This approach is using an input string and possible an analysis of the grammar itself, to limit the number of generated possibilities. When the number is decreased down to 1, we say it is deterministic.

Chapter 3

Regulated Grammars

Regulate grammars are one of many types of grammars introduced throughout the history. Their main goal is to provide greater expressive power or easier reasoning about instances of grammars. They should achieve this goals without significantly raising complexity of their model.

Common practice for regulated grammars is to join two simpler models to achieve expresive power of a single, more complicated, model. This combination of models is usually implemented using a grammar, to generate a sentence by its rules, and an additional model to restrict the derivations. Based on type of used model, we characterize regulated grammar.

This chapter defines several different types of regulated grammars. Defined grammars are grouped according to type of model, used for regulation. Other types also exists, more complete list can be found in [7]. In section 3.1 we will discuss a reasons, we might consider in choosing a regulated grammar over an unregulated one. Section 3.2 defines a basic concepts used in field of regulated grammars. The folowing sections serve as a short list of definitions of concrete regulated grammars models. Section 3.6 introduces commonly used modification used upon regulated grammars.

We should note, that number of regulated grammars, discussed in this chapter were initialy introduced by in special forms. These was later studied and generalized.

The survey of various types of regulated grammars is mostly based on [14] and [7].

3.1 Reasons to Study

Ordinary grammars are using only their rules to transform start symbol into final sentence. In every production step, you have to choose a single rule of production from the same full set of rules. You always have a freedom in what rule you choose to use. Let us compare the two most used unregulated models.

Regulated grammars are trying to join smaller complexity of using simpler grammars with greater expresive power of their more complex relatives, by restricting grammars in their operation.

3.1.1 Context-Free Grammars

Example 3.1 (Simple context-free grammar)
Look at simple example of context-free grammar G_1.

$$
\begin{aligned}
G_1 &= (N_1, T_1, P_1, S) && (3.1)\\
N_1 &= \{S, A, B\} && (3.2)\\
T_1 &= \{a, b\} && (3.3)\\
P_1 &= \{r_1 : S \rightarrow A,\\
&\quad\ r_2 : S \rightarrow B,\\
&\quad\ r_3 : A \rightarrow aA,\\
&\quad\ r_4 : A \rightarrow aB,\\
&\quad\ r_5 : B \rightarrow bA,\\
&\quad\ r_6 : B \rightarrow bB,\\
&\quad\ r_7 : A \rightarrow \varepsilon,\\
&\quad\ r_8 : B \rightarrow \varepsilon\} && (3.4)
\end{aligned}
$$

We can easily see that grammar G_1 is generating language L_1, which is composed of sentences using only two symbols in any amount and in any order. In every derivation step, we specify the symbol generated in next derivation step, by choosing appropriate nonterminal symbol. And we can always choose any of them.

$$L_1 = \{x^n | x \in \{a, b\}, n \geq 0\} \tag{3.5}$$

Example 3.1 is a demonstration of a grammar with great freedom of choice. Every nonterminal symbol in sentence form can be anytime rewriten in more than one way. This freedom of choice results in generating a simple language without structure, the *universal language*.

Context-free grammars are using simple format of production rules. Each rule has exactly one nonterminal symbol on the left-hand side. Hence, all of these rules may be used to rewrite a single symbol to specified sequence of symbols without any limitations imposed by symbols surrounding them. This greatly limits expressive power of context-free grammars.

3.1.2 Context-Sensitive Grammars

Non-context-free grammars are taking into consideration surrounding symbols and thus improving expressive power of non-context-free languages. Yet, context-sensitive rules are relying on occurence of strict conditions, prescribed by their left-hand sides.

Standard context-sensitive grammars are using single finite string to define left-hand side of its rules. This bears a certain complication, when we want to react on conditions, that had occured in a different part of sentence. Source of this complication is the fact, that while left-hand side of a rule is finite, the substring that stands between context-sensitive parts of a string might be always greater.

Example 3.2 (Simple context-sensitive grammar)
Consider the contex-sensitive grammar G_2.

$$
\begin{aligned}
G_2 &= (N_2, T_2, P_2, S) && (3.6)\\
N_2 &= \{S, Q\} && (3.7)\\
T_2 &= \{a, b, c\} && (3.8)\\
P_2 &= \{r_1 : S \ \rightarrow abc,\\
&\quad\ r_2 : S \ \rightarrow aSQ,\\
&\quad\ r_3 : bQc \rightarrow bbcc,\\
&\quad\ r_4 : cQ \rightarrow Qc\} && (3.9)
\end{aligned}
$$

$$L_2 = \{a^n b^n c^n | n \geq 1\} \qquad (3.10)$$

The grammar G_2 is generating canonical language L_2[9, Fig. 15.1]. It is using the rule r_2 to prolong current string, by adding another terminal symbol a. Rule r_4 serves only for shuffling reminder symbol Q to its final place between group of symbols b and c. Next special rule r_3 transforms this symbol to its terminal symbols.

The way of operation of grammar G_2 from example 3.2 may be seen as obfuscation of its real goal. The real goal is, in short, to generate strings with three parts of equal length. In order to do just that, it needs to generate two sequences and later convert one of them into two sequences of equal length. For that reason, it employs a remainder symbol and a shuffling rule to move it along the sentence. The remainder symbols is used to mark unfinished production or computation.

On this simple example, we may see, that computation of a final position of a single symbol in a sentential form might need a great number of separate derivations.

3.1.3 Computational History

In the light of the possibility to shuffle symbols, we have to consider another disadvantage of using context-sensitive grammars. We cannot easily form a derivation tree for every sentence generated by any context-sensitive grammar. Let us look at a derivation tree as a directional continuous non-cyclic graph, where every node of the graph has atmost one inbound edge. Every rule, which has string of a left-hand side longer than 1, would result in creating a node of a tree with more than one inbound edge, resulting from rewriting more than one symbol. We cannot remove this disadvantage. This contradicts a definition of a derivation tree. Result is unability to use a standard derivation tree to represent a sentence of context-sensitive laguage.

For contex-free grammars, the derivation tree represents a record of history of computation. Every manipulation with sentence, from entering a start symbol to acquisition of complete final sentence, is recorded inside a derivation tree. It can not only be viewed as a record of history, but also as a record of a membership relation of a concrete terminal symbol with a subtree of derivation tree. More different rules may have generated single terminal symbol and in derivation tree, we may find complete path of used derivations from start symbol to a terminal symbol. One thing missing in normal derivation tree is an order of used derivations.

This records of history may prove useful, when we are not tasked with simple generation or accepting of a strings from grammar-defined language, but with practical parsing of its sentences, or using them for practical purposes. As an example for this could serve semantic control used during parsing of programming languages in compilers. The information about a membership of a sentence into a language is not enough. We also need to read the sentence and extract informations, contained within.

For context-sensitive grammars we are unable to use simple syntax tree, but we may use simple directional continuous non-cyclic graph to record whole computational history. But this representation has one major disadvantage. Its size may quickly grow, when grammar is moving a lot of information around. Look at example 3.2. For n, of already accepted symbols c, we need to apply the rule r_4 n-times.

3.2 Basic Concepts

Regulated grammars are joining multiple, more or less separate models into a single one. One of these models has to be grammar, so it can be called a regulated grammar. This grammar is usually called *core grammar*. This grammar is usually modified by changing the way of its usual operation. Either by modifying definition of single derivation or its use in definition of language. This is usually accompanied by adding annotations to original rules of core grammar.

Regulated grammars are usually using context-free grammars as underlaying grammars. This has simple underlaying reason.

We are trying to keep complexity of both models as low as possible. Wildly used regulating models have complexity of regular language. To serve as example, we can take Regular-controlled grammars, which have by definition regulating language regular. As next example may serve Programmed grammars or State grammars, both of which are based around finite automata.

When we would use context-sensitive grammars as underlying grammars, we would easily achieve greater expressive strength. But the price for it would be more complex reasoning about generated language.

When we would join two models of regular language expressive power together into single one, we would not be able to increase combined power to context-free, much less context-sensitive. Hence regulated grammars, using only regular grammar compliant rules, are of limited use.

3.3 Context-Based Regulation

Context-based grammatical regulation places context-related restrictions upon their sentential forms. This contextual conditions must be met by whole current sentential form. We will analyse a short list of grammars in this category.

3.3.1 Random Context Grammars

Random context grammars are adding two sets of symbols to every rule. One of them specifies symbols that has to be present in current sentential form and the other specifies symbols that must not be present. If this two conditions are met, then the production rule may be used. Any rule with satisfied conditions is alowed to be used.

3.3.2 Context-Conditional Grammars

Context-conditional grammars are using the same mechanism as Random context grammars. But insted of mere symbols, the sets contains whole strings, with the same meaning as in random context grammars.

3.3.3 Scatered Context Grammars

Scatered context grammar is changing notion of a derivation as application of single production rule. It uses n-tupples of simple production rules. Every n-tupple is handled as single rule. Therefore all simple rules of selected n-tupple must be used simultaneously and in the same relative position to each other.

3.3.4 Restricted Derivation Tree Grammars

Whole range of *grammars with restricted derivation trees*[12] was introduced. This type of regulation is using regulating language to check context independently on rules used to produce sentential form. This language is used to check the form of a constructed derivation tree, rather then simply using current sentential form for restricting subsequent rewrite rules.

3.4 Rule-Based Regulation

This type of regulation places restrictions on the use of rules during derivation. Restrictions defines subsets of rules that are allowed for next derivation step. We will analyse a short list of grammars that can be placed in this category.

3.4.1 Regular-Controlled Grammars

Regular-controled grammars are one of integral pieces of theory of regulated rewriting. They use control language defined over a set of rules of a core grammar. The control language specifies allowed sequences of rules in G used to generate a sentence.

Definition 3.1 (Regular-controlled grammar)
A *regular-controlled (context-free) grammar* [14, Definition 5.1.1], is a pair

$$H = (G, \Xi), \tag{3.11}$$

where

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

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

$$L(H) = \{w \in T^* | S \ \Rightarrow_G^* w[\alpha] \wedge \alpha \in \Xi\} \tag{3.12}$$

Convention: The abbreviation for regular-controlled grammar is **RC** and the family of languages generated by it is denoted by $\mathcal{L}(RC)$.

3.4.2 Matrix Grammars

Matrix Grammar is a pair $H = (G, M)$, where *core grammar* G is extended by a finite set M of sequences of rules. This grammar can be viewed as a special case of a regular-controlled grammar, where the control language is specified to be an iteration over a finite language.

3.4.3 Programmed Grammars

Programmed grammar is a context-free grammar, in which two sets, σ_r and φ_r are attached to each rule r. Both of these sets are subsets of all rules of underlying grammar. If a rule r is used, then one of rules in σ_r must be used next. If a rule r could not be used, then one of rules in φ_r has to be used next.

Definition 3.2 (Programmed grammar)
A *programmad grammar* is a quintuple

$$G = (N, T, Psi, P, S) \tag{3.13}$$

where

- N, T, Ψ and S are defined as in a context-free grammar;
- $P \subseteq \Psi \times N \times (N \cup T)^* \times 2^\Psi \times 2^\Psi$ is a finite relation, called the set of *rules*.

Convention: Instead of $(r, A, x, \sigma_r, \varphi_r) \in P$, we write $(r : A \to x, \sigma_r, \varphi_r) \in P$. A is refferd to as the *left-hand side* of r, and x is reffered to as the *right-hand side* of r.

Let $V = N \cup T$ be the *total alphabet*. The *direct derivation relation*, symbolically denoted by $\Rightarrow_G$, is defined over $V^* \times \Psi$ as follows: for $(x_1, r), (x_2, s) \in V^* \times \Psi$,

$$(x_1, r) \Rightarrow_G (x_2, s) \tag{3.14}$$

if and only if either

$$x_1 = yAz,\ x_2 = ywz,\ (r : A \to w, \sigma_r, \varphi_r) \in P,\ \text{and } s \in \sigma_r \tag{3.15}$$

or

$$x_1 = x_2,\ (r : A \to w, \sigma_r, \varphi_r) \in P,\ x_1 \neq yAz \text{ for any } y \text{ and } z,\ \text{and } s \in \varphi_r \tag{3.16}$$

The $(w, s) \in V^* \times \Psi$ is called a *configuration.* The *language* of G is denoted by $L(G)$ and defined as

$$L(G) = \{w \in T^* | (S, r) \Rightarrow_G^* (w, s), forsomer, s \in \Psi\} \tag{3.17}$$

Convention: The abbreviation for programmed grammar is **P** and the family of languages generated by it is denoted by $\mathcal{L}(P)$.

3.4.4 State Grammars

State grammar is a context-free grammar extended by finite-state mechanism. Each rule is enhanced by source and destination state from a finite set of states. At each derivation step, leftmost nonterminal symbol, for which there exists a rule with source state equal to curent state, is rewriten, using this rule.

3.5 On Adaptive Grammars

Rule-based regulated grammars are using a controlling model to select subset of production rules of core grammar that are allowed to be used in next production. But the original set of rules, from which they pick, does not change. The grammar remains constant. This behaviour may result in large set of rules in core grammar. This set is then greatly reduced for every production step.

There also exists related model of *Adaptive grammars*[5]. For this concept was historicaly used many names, which includes *Extensible*, *Modifiable* and *Dynamic*. This grammmars are using additional mechanisms to add, remove or modify rules in core grammar. They then use whole grammar at any given time.

The addition and modification of rules distinquishes the class of adaptive grammars from the class of regulated grammars. Adaptive grammars can be used to specify dynamic languages by single grammar without additional semantical checking.

3.6 Leftmost Modification

With unregulated grammars we sometimes use leftmost modification of original grammars. This may be viewed as a regulation of some sort. The usage of this modification can greatly affect the outcom lanugage of a grammar.

This modification can be defined as a regulation of both, used nonterminal and terminal. Although the leftmost usage of terminals is usually defined only in terms of automata.

Let us define this modification of languages [7][54, type I].

Definition 3.3 (Unregulated leftmost modification)
Under *unregulated leftmost modification*, the leftmost occurence of a nonterminal has to be rewritten.

Convention: When a class of grammars G is using unregulated leftmost modification, we add a symbols UL to its upper index, to form G^{UL}.

This modification greatly diminishes the number of usable nonterminals to at most 1 in every derivation step. This takes also a great toll on expressive power of modified grammar. Even with control model in place, we do not surpass the original model.

Lemma 3.1 (Strength under unregulated leftmost modification)

$$\mathcal{L}(P^{UL}) = \mathcal{L}(CF) \text{ [7][Theorem 1.4.1]} \tag{3.18}$$

$$\mathcal{L}(RC^{UL}) = \mathcal{L}(CF) \text{ [7][Theorem 1.4.1]} \tag{3.19}$$

When we are dealing with regulated grammars, the original definition of leftmost modification may prove itself too restrictive, because not every time, we are able to derive every nonterminal of a sententian form, due to diminished set of rules we are able to use in the next derivation step [7][54, type II].

Definition 3.4 (Regulated leftmost modification)
Under *regulated leftmost modification*, the leftmost occurence of nonterminal, which can be currently rewritten, has to be rewritten.

Convention: When a class of grammars G is using regulated leftmost modification, we add a symbol L to its upper index, to form G^L.

We can see the sutle difference in formulation. This definition is giving us a freedom to follow the regulation, withou crushing due to missing nonterminals. This makes a big difference, because we are free to obey the control model and we have acquired a way to limit the nonterminal, on which it is used.

Lemma 3.2 (Strength under unregulated leftmost modification)

$$\mathcal{L}(P^L) = \mathcal{L}(RE) \text{ [7][Theorem 1.4.3]} \tag{3.20}$$

$$\mathcal{L}(RC^L) = \mathcal{L}(RE) \text{ [7][Theorem 1.4.4]} \tag{3.21}$$

The left-most modification serves as an example of extreme-based nonterminal rewrite. In similar maner a right-most modification may be introduced.

Chapter 4

Grammar Automata

In this chapter we will introduce a reworked version of a grammar automata. This version is allowing us, to derive the nonterminal symbol at specified position.

The section 4.1 contains definition of automata itself. In section 4.2, we will describe basic techniques to transfer regular-language regulation into an automata. Basic version of evaluation of automata is presented in senction 4.3.

4.1 Positioned Grammar Automata

Definition 4.1 (Positioned grammar automaton)
The *positioned grammar automaton PGA*, is an 8-tuple

$$GA = (Q, \Sigma, \Gamma, \Delta, R, s, S, F, \delta), \tag{4.1}$$

where

- $Q, \Sigma, \Gamma, s, S, F$ and δ is defined as in *grammar automata*;
- $\rho_t : \mathbb{N}^* \times \Gamma^+ \rightarrow \Gamma^*$ is a function applying a production rule at state t;
- $R \subseteq (Q \times \mathbb{N} \times f \times Q)$ is a finite set of *rules*, which differ from the original version in a *feasibility function* $f : (\Gamma^* \times \Sigma^* \times \mathbb{N}^*)$. The rule, is *feasible* only when the feasibility function has outputted one or more numbers, symbolising possible positions. The joining rule can have any priority and after transition with this rule, the production rule is not applied.

4.2 Basic transformations

We will introduce basic transformations of regular control language into a grammar automaton. We will base this transformations on the fact, that every regular language can be writen with the help of regular expressions and regular expressions can be represented with a single symbols, and its concatenations, unions and iterations. We have to keep in mind, that the symbol of the control language is a label of a rule, which has to be applied.

As opposed to the ordinary finite automata, the grammar automata accepts its string not by an edges it had to travel, but the states, it had to visit. Every automata resulting from basic transformation will have single start state and single final state, this will help us to systematically build an automata from separate pieces.

Let M_1 be a automata, with a start state q_{11} and a final state q_{12}, for a regular expression r_1, and M_2 be a automata, with a start state q_{21} and a final state q_{22}, for a regular expression r_2. We will use 2 as a standard priority.

Symbol

Application of a single rule is represented with two states q_0 (start state) and q_1 (final state). The final state applies the production rule $p : \ l \to r$, with a function $\rho(n, s_l l s_r) = s_l r s_r \wedge |s_l| \in n$. Between this two states exists a single rule $q_0(n, f) \to q_1$, where n is a standard priority and f is applicability function, defined by the grammar.

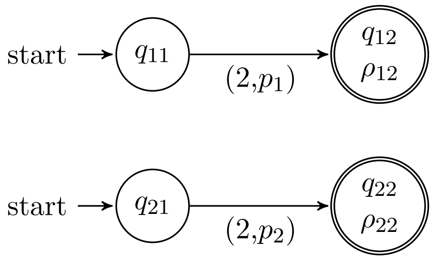

Figure 4.1: Single rules automata

Concatenation

The concatenation r_1r_2 is created by joining two smaller automata, r_1 and r_2, by merging the states q_{12} and q_{21} into a single state, while preserving the properties of the state q_{12} and edges of both states. The resulting automaton will use q_{11} as its start state and g_{22} as its final state.

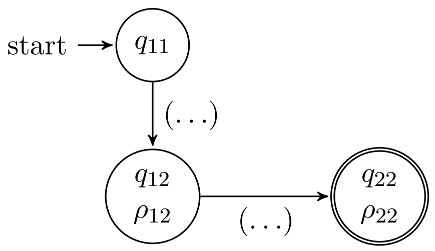

Figure 4.2: Concatenation of M_1 and M_2

Union

The union $r_1 + r_2$ is created by joining two smaller automata for r_1 and r_2. Both start states are merged, while preserving all their edges. New final state is created and new joining edges from original final states into a new final state is created.

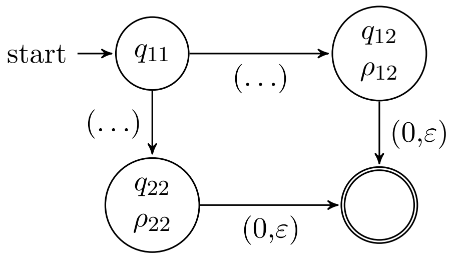

Figure 4.3: Union of M_1 and M_2

Iteration

The r_1* is created by looping original automata for r_1 into a start state. A new joining edge is created from a final state into a start state. A new final state is introduced, connected with a start state with a joining edge of a normal priority.

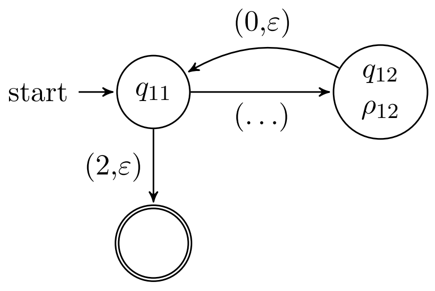

Figure 4.4: Iteration of M_1

4.3 Operation

We can create a generic algorithm for running a positioned grammar automaton 4.1. This algorithm outputs a string of visited non-empty states. The algorithm can end either successfully, with $FINISH$, or unsuccessfully, with $FAIL$.

Later, we will use few common definitions, unless said otherwise. The work alphabet contains two border symbols ⟨ and ⟩, which are not defined by the grammar. The application of production rule δ, for production rule $(p : l \to r)$ chooses one of offered positions and symbol of a work string on that location rewrites with $\langle r \rangle$. The normalizing function $\delta(work, input)$ is working as accepting function. If work string, after filtering out border symbols, equals the input string, both are completly emptied. Otherwise is nothing changed.

Algorithm 4.1 Positioned grammar automaton run

```
Input: A positioned grammar automaton $A = (Q, \Sigma, \Gamma, R, s, S, F, d)$
Input: Input string $Input$
Output: String of used non-empty states and $FINISH$ or $FAIL$
 1: $config \leftarrow (s, S, Input)$
 2: while $config \neq (f, \varepsilon, \varepsilon) \wedge f \in F$ do
 3:     $(state, work, input) \leftarrow config$
 4:     $trans \leftarrow \{(v, n, p) \mid state(v, f) \leftarrow n \wedge p = f(work, input) \neq \varnothing\}$
 5:     $trans \leftarrow \{(v, n, p) \mid (v, n, p) \in trans \wedge v$ is minimal$\}$
 6:     if $trans = \varnothing$ then
 7:         $FAIL$, no possible transition
 8:     end if
 9:     Choose $(v, next, pos)$ as one of $trans$
10:     $state \leftarrow next$
11:     if $state \in Q_r$ then
12:         Emit $state$
13:         $work \leftarrow \rho_{state}(work, pos)$
14:         $(work, input) \leftarrow \delta(work, input)$
15:     end if
16:     $config \leftarrow (state, work, input)$
17: end while
18: $FINISH$
```

Chapter 5

Switch-Paused Regulated Grammars

Standard regulated grammars are using their regulating mechanism in every production step. This mode of operation may result in excessive control over each step of production. We can use regulated grammars, when we want to have a structure of our grammar and subsequently languages, more obvious.

If we are forced to write every used production rule into regulating language, the true purpose of this language may be obscured by its other parts.

The first version of paused modes is using predefined two sets of rules to serve as on-switches and off-switches for operation of regulating model. Every time, an on-switching rule is being used, the regulating mechanism is used. Regulation continues till next off-switching rule is used.

This mode of operation is usefull in cases, where we want to regulate only certain portion of sentence. This allows us to keep regulating model simpler and more focused.

This chapter introduces a new modification of regulated models. In sections 5.1 and 5.2, it is defined for various regulated grammars. Few examples of grammars are in section 5.3 Sections 5.4 and 5.5 are dealing with generation and parsing of sentences. Lastly, the section 5.6 is investigating properties of this modifcation.

5.1 Definition

We can define switch-paused regulation for regular-controlled grammars. This definition is directly adding two sets of switching rule labels to the definition of the original non-switching version of the grammar. The definition of generated language is then modified to accomodate the switching nature of newly defined grammar.

Definition 5.1 (Switch-paused regular-controlled grammar)
A *switch-paused regular-controlled (context-free) grammar*, is a 4-tuple

$$H = (G, \Xi, \Psi_{ON}, \Psi_{OFF}), \tag{5.1}$$

where

- $G = (N, T, \Psi, P, S)$ is a context-free grammar, called *core grammar*;
- $\Xi \subseteq (\Psi_{FREE} + (\Psi_{JON}\overline{\Psi_{OFF}}^{*}\Psi_{OFF}))^{*}(\Psi_{JON}\overline{\Psi_{OFF}})^{?}$ is a regular language, called *control language*;

- $\Psi_{JON} = \Psi_{ON} \setminus \Psi_{OFF}$ is set of rules, which are only on-switching;
- $\Psi_{FREE} = \Psi_{ON} \cap \Psi_{OFF}$ is a set of rules, which serve both switching functions;
- N is a finite set of nonterminal symbols;
- T is a finite set of terminal symbols;
- $P \subseteq (N \times (N \cup T)^*)$ is a set of *rules*;
- $S \in N$;
- Ψ is a set of symbols called *rule labels* such that $|\Psi| = |P|$ and there exists a bijection ψ from Ψ to P;
- $\Psi_{ON} \subseteq \Psi$ is a set of *on-switching rules*;
- $\Psi_{OFF} \subseteq \Psi$ is a set of *off-switching rules*;

The sentential form can be in two of the possible states $States = \{ON, OFF\}$.
The *language* of H, denoted by $L(H)$, is defined as

$$\begin{aligned} L(H) = \{w \in T^* \mid \exists i \geq 1 : \forall 0 \leq j < i : S \; &\Rightarrow_G^* w[\alpha] \\ &\wedge \alpha \in n_0 g_1 n_1 g_2 n_2 \dots g_{i-1} n_{i-1} g_i \\ &\wedge n_j \in \overline{\Psi_{ON}}^* \wedge g_1 g_2 \dots g_i \in \Xi \\ &\wedge g_j \in (\Psi_{FREE} + (\Psi_{JON} \overline{\Psi_{OFF}}^* \Psi_{OFF})) \\ &\wedge g_i \in (\Psi_{JON} \overline{\Psi_{OFF}})^? \} \end{aligned} \tag{5.2}$$

Convention: The switch-paused regular-controlled grammar is abbreviated as RC^{SP} and the family of languages generated by it is denoted by $\mathcal{L}(RC^{SP})$

From definition, we can see that rule can be both on-switching and off-switching. In that case, it serves both functions. On-switching semantics is applied before its off-switching counterpart. This rule is then allowed to stand freely by itself in control language.

There are two main differences from standard definition of regular-controlled grammars. First is stricter form of the control language. It can be empty, which is not allowed in ordinary regulated grammars. Empty control language would result in generating only a single sentence containing only starting symbol. Empty control language, or empty string of control language, in switch-stopped regular-controlled grammars have a simple meaning. Non of on-switching rules may be used.

When string of control language is not empty, its first rule has to be on-switching rule. It is a consequence of two properties of switch-paused modification. The controll language does contain switching rules and the language-driven controll is switched off at the start of generating of a sentence.

Second difference is in generated language. Not every production need to be writen in the controll language. Conclusion is, that during production, more rules can be used, than specified by controll language.

For usage during derivation run, the control language sentences are sliced into separate groups. Every group starts with on-switching rule and ends with off-switching. Inside group, there could be any number of rules which are not off-switching. Therefore even more on-switching rules. The group can also be formed by single rule. This rule has to be in both switching set. This rule is therefore called *free standing*.

The definition 5.1 is concerned only with regular language as controling language. We do not need to be limited to a single class of langages. We can use the original definition as a base for more general definition. This one allows the controll language to be of more expressive class, than regular language.

Definition 5.2 (General switch-paused modification)
A *switch-paused L-controlled grammar*, is a 4-tuple

$$H = (G, \Xi, \Psi_{ON}, \Psi_{OFF}),$$

where H is a switch-paused regular-controlled grammar, and Ξ is a language of class L.

This definition further limits format of language Ξ. It still has to maintain a format, where any non-empty string of controll language has to be on-switching rule and after any off-switching rule, which is not lats rule of controll sequence, has to be an on-switching rule. And additionaly, the controll language has to adhere to limitations of certain class, we want to achieve.

5.2 Other Regulations

This is not an exhaustive list of formal definitions of switch-paused mode of operation for any regulated grammars model. This list contains merely a descriptions of propositions, that supports the idea of more universal usage of switch-paused mode to opearation of regulated grammars. There exists more regulated grammars, which are not mentioned in this section.

5.2.1 Rule-Based Regulations

The rule-based regulations are generaly described using languages or automata. We defined switch-paused mode for regular-controlled grammar and expanded it to controll by any class of languages.

Matrix Grammars

Matrix grammars can be viewed as a close relative to regular controlled grammars. The matrix regulation can be directly translated to regular language regulation. Every matrix can be viewed as rigid string of rule labels and control language is then iteration over this strings. The switch-paused regulation of matrix grammar then has a similar definition. The matrices can serve as a self contained groups, where every matrix starts with a on-switching rule and ends with an off-switching one.

State Grammars

State grammars are using rules to travel between different states. When we would applied a switch-paused mode, we would ignore those transitions and conditions, while in off state. We would hold the last state acquired by the last off-switching rule that transitioned from the on-state to the off-state. The use of on-switching rule, with start state equal to the remembered one, would command a new state to be remembered, together with transition into an on-state.

5.2.2 Context-Based Regulations

So far, we limited this mode of operation only for rule-based regulations. Context-based regulations does not directly specifies orders of rules. Context-based regulations are based on decisions, tied to state of current derivation, or history of its states. The limitations on rules used to generate the final sentence are direct consequence of that.

The case of scattered-context grammars is straight forward. In this case, the definition of singular rules is completly ommited, together with core grammar, from the definition. We cannot use them as a single rules anymore.

Even when we would recreate the original core grammar, to obtain original rules, we would hit another obstacle. The scattered context grammars are using multiple rules at the same time, in parallel to each other. The switching sets would have to be defined using this newly created rules, because when the controll mechanism is turned off, we cannot use compound rules to switch it on. The compound rule would then combine all semantic functions, in terms of off-swithing, of all rules it uses.

5.3 Examples

In this section, we will investigate a few examples of switch-paused regulated grammars. We will show examples from both ends of a spectra of ammount of regulation.

Example 5.1 ($\{a^nb^nc^n|n \geq 1\}$)
Consider the switch-paused regular-controlled grammar $H_1 = (G, \Xi, \Psi_{ON}, \Psi_{OFF})$.

$$\begin{aligned}
G &= (N, T, \Psi, P, S) && (5.3)\\
N &= \{S, A, B, C\} && (5.4)\\
T &= \{a, b, c\} && (5.5)\\
P &= \{r_1 : S \rightarrow ABC, && \\
&\quad r_2 : A \rightarrow aA, \qquad r_5 : A \rightarrow a, && \\
&\quad r_3 : B \rightarrow bB, \qquad r_6 : B \rightarrow b, && \\
&\quad r_4 : C \rightarrow cC, \qquad r_7 : C \rightarrow c\} && (5.6)\\
\Xi &= \{(r_2r_3r_4)^*(r_5r_6r_7)\} && (5.7)\\
\Psi_{ON} &= \{r_2, r_5\} && (5.8)\\
\Psi_{OFF} &= \{\} && (5.9)
\end{aligned}$$

$$L = \{a^nb^nc^n|n \geq 1\} \qquad (5.10)$$

The grammar H is generating canonical language L. The control lanugeage is created by iterations of phrases of three rules. Every phrase injects three eparate terminals, ensuring equal numbers. Only derivation, which happens outside of regulation is the initial usage of rule r_1. Let us explore an example of a derivation for a concrete example.

$$\begin{aligned} S &\Rightarrow ABC[r_1] \\ &\Rightarrow aABC[r_2] \\ &\Rightarrow aAbBC[r_3] \\ &\Rightarrow aAbBcC[r_4] \\ &\Rightarrow aabBcC[r_5] \\ &\Rightarrow aabbcC[r_6] \\ &\Rightarrow aabbcc[r_7] \end{aligned} \tag{5.11}$$

Example 5.2 (Random interleaving)
Consider the switch-paused regular-controlled grammar $H_2 = (G, \Xi, \Psi_{ON}, \Psi_{OFF})$.

$$G = (N, T, \Psi, P, S) \tag{5.12}$$

$$N = \{S, X, Y\} \tag{5.13}$$

$$T = \{a, b, c\} \tag{5.14}$$

$$\begin{aligned} P = \{r_1 &: S \rightarrow XY, \\ r_2 &: X \rightarrow aX, \qquad & r_6 &: Y \rightarrow aY, \\ r_3 &: X \rightarrow bX, & r_7 &: Y \rightarrow bY, \\ r_4 &: X \rightarrow cX, & r_8 &: Y \rightarrow cY, \\ r_5 &: X \rightarrow \varepsilon, & r_9 &: Y \rightarrow \varepsilon\} \end{aligned} \tag{5.15}$$

$$\Xi = \{((r_3r_7) + (r_4r_8))^*\} \tag{5.16}$$

$$\Psi_{ON} = \{r_3, r_4, r_7, r_8\} \tag{5.17}$$

$$\Psi_{OFF} = \{r_4, r_8\} \tag{5.18}$$

$$L = \{w_1w_2 | w = \{b, c\}^* \wedge w_1, w_2 \text{ are versions of } w \text{ interleaved with} a\} \tag{5.19}$$

The grammar H_2 is using regulating model only for positioning of tuples of symbols. The interleaving is uncontrolled. Let us explore an example of a derivation for a concrete example.

$$\begin{aligned} S &\Rightarrow XY[r_1] \\ &\Rightarrow aXY[r_2] \\ &\Rightarrow abXY[r_3] \\ &\Rightarrow abXbY[r_7] \\ &\Rightarrow abXbaY[r_6] \\ &\Rightarrow abXbaaY[r_6] \\ &\Rightarrow abcXbaaY[r_4] \\ &\Rightarrow abcXbaacY[r_8] \\ &\Rightarrow abcbaacY[r_5] \\ &\Rightarrow abcbaac[r_9] \end{aligned} \tag{5.20}$$

5.4 Generation

The grammar is a generative model for language it specifies. In case of ordinary grammars, we can simple pick a nonterminal in current sentential form and any matching rule, derive the selected symbol using selected rule and repeat this steps, until final sentence is acquired. This simple algoritm is not sufficient, when we need to implement more complicated model.

Algorithm 5.1 Generation of a string with RC^{SP} grammar

```
Input: $RC^{SP}$ grammar $H = (G, \Xi, \Psi_{ON}, \Psi_{OFF})$, where $G = (N, T, \Psi, P, S)$
Output: Sentence $w \in L(H)$ or Error
 1: $State \leftarrow OFF$
 2: $w \leftarrow S$
 3: $Possible \leftarrow \{(p, |s_l|) \mid s_l l s_r = w \wedge l \in N \wedge (p : l \rightarrow r) \in P\}$
 4: $Possible \leftarrow \{(p, i) \mid (p, i) \in Possible \wedge (p \notin \Psi_{ON} \vee (pd \in \Xi \wedge d \in \Psi^*))\}$
 5: $Track \leftarrow [(State, w, \varepsilon, Possible)]$
 6: while $w \notin T^*$ do
 7:     if empty($Track$) then
 8:         Error, no derivation possible
 9:     end if
10:     $(State, w, c, Possible) \leftarrow head(Track)$
11:     if $Possible = \emptyset$ then
12:         $Track = tail(Track)$
13:     else
14:         Choose $(p, i) \in Possible$
15:         $Track \leftarrow (State, w, c, Possible \setminus \{(p, i)\}) + tail(Track)$
16:         if $p \in \Psi_{ON}$ then
17:             $State \leftarrow ON$
18:         end if
19:         if $State = ON$ then
20:             $c \leftarrow cp$
21:         end if
22:         if $p \in \Psi_{OFF}$ then
23:             $State \leftarrow OFF$
24:         end if
25:         $w \leftarrow s_l r s_r$, for $w = s_l l s_r, l \in N, |s_l| = i, (p : l \rightarrow r) \in P$
26:         $Possible \leftarrow \{(p, |s_l|) \mid s_l l s_r = w \wedge l \in N \wedge (p : l \rightarrow r) \in P\}$
27:         if $State = ON$ then
28:             $Possible \leftarrow \{(p, i) \mid (p, i) \in Possible \wedge cpd \in \Xi \wedge d \in \Psi^*\}$
29:         else
30:             $Possible \leftarrow \{(p, i) \mid (p, i) \in Possible \wedge (p \notin \Psi_{ON} \vee (cpd \in \Xi \wedge d \in \Psi^*))\}$
31:         end if
32:         $Track \leftarrow (State, w, c, Possible) + Track$
33:     end if
34: end while
```

In algorithm 5.1, we use few variables to help with tracking of current state of simulated model. The *State* variable represents one of two possible states, the algorithm can be in. The *Track* variable serves as a history of states. It is a list. On this variable we use a few methods that are helping us to work with this list. The *head* method is accesing the first item in the list, the *tail* method is providing a sublist of its input list, without its head, and the *empty* method is checking, whethever the list contains no items.

The algorithm is nondeterministic in selecting next rule used in production. This non-determinism is the heart of generation of more than a single sentence. Error is raised, when language, defined by input grammar is empty. This algorithm uses a depth-first type exhaustive search in possible state-space of sentential forms. If there is no more states to search and no solution was found, the error is risen.

The algorithm uses a pushdown stack to remember unfinished paths. Every cell of this virtual stack is formed of state, sentential form of a generated language, already accepted prefix of controll language and set of unfinished paths, represented with a set of rules and plaes to apply them.

This algorithm is implementing a depth-first search, without checking of possible loops. The order of applied rules is dependant on a choice, which is handled outside of this algorithm. Therefore, for a recursive grammars, there could exist aa infinite sequence of choices, which leads to an infinite loop.

The algorithm is using always exactly on top-most cell of its stack. The cell of its stack can grow based on the length of a sentential form and a prefix of the control language. Due to this, we are unable to fully implement this algorithm in a space limited linearly by the length of final sentence.

We might decrease a real memory footprint, by working on a single instance of a sentential form. In this case we would memorise a used rule, instead of whole string. We can also interchange a prefix of control language with a state of automata, equivalent to the control language. Both of these limits a complexity to constant values. But we still remain with an unboulded length of stack.

5.4.1 Leftmost Modification

We treat the switch-paused regulation as a modification mode of original regulation. Similarly we can treat the leftmost modification, which can serve us as an eample of another modification appliable to most, if not all, of regulation models.

The left-most modification simply limits the applicability of rules to left most nonterminal only. This does not impeed with pausing of regulation in any way. This modifications can be combined on the same grammar to further limit its derivations.

In the case of combining left-most modification and switch-paused modification, we need to modify the way of selecting nontermial symbol of algorithm 5.1 to select only left most nonterminal symbol.

5.5 Parsing

Parsing of a sentence using particular grammar can answer the question of membership to the language, for a concrete sentence. This is usual operation, performed when working with a sentence in a compiler.

5.5.1 Specific Transformations

Using the basic transformations, we can express every regular-controlled grammar as a grammar automata. We need to add another transformations specific to the switch-paused nature of RC^{SP} grammars.

For following graphical examples, we will use rules and language

$$\begin{aligned} P &= \{p_1 : S \rightarrow aSb, p_2 : S \rightarrow ba, p_3 : S \rightarrow aSc\}; \\ \Psi_{ON} &= \{p_1\}; \\ \Psi_{OFF} &= \{p_1\}; \\ \Xi &= p_1 p_1. \end{aligned} \tag{5.21}$$

The feasibility function $c(p)$ for a production rule p returns every applicable position of a rule p.

Start State

Every automata, generated by basic transformations 4.2 has an empty state as a start state. The RC^{SP} grammars starts at an off-state. We have to allow usage of off-switching rules. A single state loops for all non on-switching production rules are created.

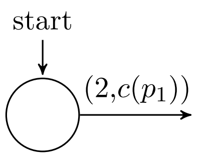

Figure 5.1: Original initial state

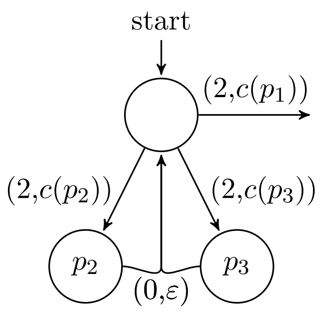

Figure 5.2: Transformed initial state

Middle Group

For each occurence of a state, created from an off-switching rule, is inserted a new empty state q_e. Only input edge to q_e is from its off-switching state. Every edge transitioning from the off-switching state is rewriten, to originate from q_e. A simple one-state loops are created upon the state q_e for every rule which is not on-switching.

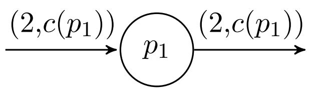

Figure 5.3: Original off-switching state

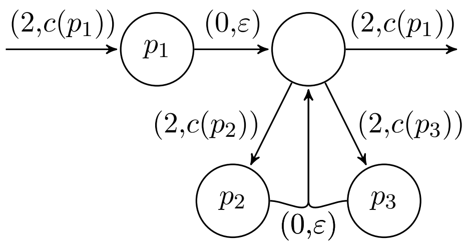

Figure 5.4: Transformed off-switching state

5.5.2 Analysis

After we created the grammar automaton, using transformations from previous section, for our desired switch-paused regular-controlled grammar, we have to simply run the generic algorithm 4.1 for the grammar automaton, to commence a syntax-analysis of our input sentence.

The algorithm 4.1 is not said to work deterministically. The same applies to the grammar itself.

5.6 Properties

We will investigate the properties of switch-paused regulated grammars. We will compare them with theirs non-paused versions.

In language controlled regulated grammars, we usually deal with two variables, that define final power of regulated model. First is the type of core grammar. If not noted otherwise, we will use context-free grammar as a core grammar. The second variable is type of controll language. This language does not have to be only regular. We will investigate several types of languages used to controll the core grammar.

5.6.1 Switch-Paused to Original

We will construct a transformation of RC^{SP} grammar into original RC grammar. This transformation 5.2 should preserve the generated language of an input language. It is based on interleaving of self-contained on-switched groups with iterations over not on-switching rules.

Algorithm 5.2 Transformation of RC^{SP} into RC

Input: RC^{SP} grammar $H_I = (G, \Xi_I, \Psi_{ON}, \Psi_{OFF})$
Output: RC grammar $H_O = (G, \Xi_O)$, where $G = (N, T, \Psi, P, S)$

```
1: $\Xi_O \leftarrow \{s_0 g_1 s_1 g_2 \ldots s_{n-1} g_n \mid g_1 g_2 \ldots g_n \in \Xi_I \wedge s_j \in \overline{\Psi_{ON}}^* \ \wedge \ n \geq 1 \ \wedge \ 0 \leq j < n$
       $\wedge\, g_j \in (\Psi_{FREE} + (\Psi_{JON} \overline{\Psi_{OFF}}^* \Psi_{OFF}))$
       $\wedge\, g_n \in (\Psi_{JON} \overline{\Psi_{OFF}})^?\}$
```

Lemma 5.1 (Correct transformation) Transformation algorithm 5.2 is correct.

Proof 5.1
Let us investigate properties of the transformation.

- Core grammars are equal.
- Every time, the regulation mechanism is switched off, a free iteration group is inserted into the control lanugage.
- All inserted groups contains all rules allowed in off-switched state withous on-switching the regulation.
- Every inserted group is a regular language. Regular languages are closed under concatenation. Final control language is also regular.

5.6.2 Original to Switch-Paused

The transformation of a RC grammar into RC^{SP}, which preserves the generated language, is straight forward. We can preserve the control language in the expense of never using the advantages of RC^{SP} grammars. Since we want to track every rule and never turn of the control, we wil not use the off-switching set and fill on-switching set with all rules. This transformation is described by the algorithm 5.3.

Algorithm 5.3 Transformation of RC into RC^{SP}

```
Input: RC grammar $H_I = (G, \Xi)$, where $G = (N, T, \Psi, P, S)$
Output: RC^SP grammar $H_O = (G, \Xi, \Psi_{ON}, \Psi_{OFF})$
1: $\Psi_{ON} \leftarrow \Psi$
2: $\Psi_{OFF} \leftarrow \varnothing$
```

Lemma 5.2 (Correct transformation) Transformation algorithm 5.3 is correct.

Proof 5.2
Let us investigate properties of the transformation.

- Control languages are equal.
- With first application of any rule, the switching mechanism is turned on.
- The on-switching rule is always part of the control language.
- Regulation is never turned off, because no off-switching rule exists.
- Because every derivation happens under regulation, core grammars are equal and control languages are equal, produced languages are equal.

5.6.3 Expressive Strength

Theorem 5.1
$\mathcal{L}(RC_L^{SP}) = \mathcal{L}(RC_L)$, controll language $L \in \{REG, CF, CS, RE\}$

Proof 5.3
We will prove equality with original model by transformations.

- $\mathcal{L}(RC_L^{SP}) \subseteq \mathcal{L}(RC_L)$
  - There exists a transformation 5.2 of any RC_L^{SP} grammar into RC_L grammar, defining the same language.
- $\mathcal{L}(RC_L) \subseteq \mathcal{L}(RC_L^{SP})$
  - There exists a transformation 5.3 of any RC_L grammar into RC_L^{SP} grammar, defining the same language.
- $RC_L^{SP} = RC_L$
  - Equality of expressive strength of both models is direct conclusion of $RC_L^{SP} \subseteq RC_L$ and $RC_L \subseteq RC_L^{SP}$.

Theorem 5.2
$\mathcal{L}(CF) \subset \mathcal{L}(RC^{SP}) \subset \mathcal{L}(RE)$. See [14][Theorem 5.1.6]

Theorem 5.3
$\mathcal{L}(RC_L) \subset \mathcal{L}(RC_L^{SP})$, control language $L \in \{FIN\}$

Proof 5.4
We will prove superiority of a switch-paused model for finite control languages.

- Length of a right-side of any rule is finite.
- Number of sentences of control languages in FIN is finite.
- Hence $RC_L = FIN$.
- Consider a switch-paused finite-controlled grammar $H^{SP} = (G, \Xi^{SP}, \Psi_{ON}, \Psi_{OFF})$, where $G = (N, T, \Psi, P, S)$ and its non-paused version $H = (G, \Xi)$.
- If $\Psi_{ON} = \varnothing$ and $\Psi_{OFF} = \Psi$, the regulation is never used.
- If regulation is never used, the expressive power is determined by the grammar.
- The grammar is context-free. $\mathcal{L}(FIN) \subset \mathcal{L}(CF)$.

From discovered properties, we can see, that we do not suffer from diminished expressive strength, while using switch-paused regulated grammars. In case of finite controll language, we can increase the expressive power of a grammar by using switch-paused regulated grammar model. The expressive power is directly comparable to non-switching version of regular-controlled grammars.

Every non-paused regulated grammar can be directly translated into switch-paused version. This translation is executed by creating on-switching set equal to whole set of rule labels and an empty off-switching set. The grammar, constructed by this simple translation, is valid switch-paused regulated grammar, but does not use any of its different properties to its advantage.

Chapter 6

Determinism-Paused Regulated Grammars

When designing regulated grammars, we may face a problem of complicated control language. Standard regulated grammars are using their regulating mechanism in every production step. The construction of a correct controll language is not always a trivial task. We may want to use controll language to define only the parts, that needs explicit regulation.

As the switch-paused mode of regulation, the determinism-paused mode tries to use control language with shorter sentences, by ommiting production steps, which are already fully defined by the core grammar. The switch-paused mode used explicit sets, to define entrance points between realm of regulation and non-regulation.

The determinism-paused version of paused mode of rule-based regulation is using determinism of usage of production rules, to pause regulating model. When the core grammar can make deterministic step, it makes it, without checking of control model. This mode of operation may be useful, when we are dealing with nondeterminism only in few well defined places in every sentence of generated lanugage.

This chapter introduces a new modification of regulated models. The section 6.1 serves as an introduction into a determinism in grammars. In sections 6.2 and 6.3, it is defined for various regulated grammars. Few examples of grammars are in section 6.4 Sections 6.5 and 6.6 are dealing with generation and parsing of sentences. Lastly, the section 6.7 is investigating properties of this modifcation.

6.1 On Determinism in Generation

Grammars are, in oposition to its coresponding automata, intended to be primarily generative tools for specified languages. And as a generative tool, they should be able to generate every string, that belongs to the specified language, and not any other string.

With deterministic-paused regulation, we are introducing concept of determinism into generation of the target language itself. In parsing, the determinism is well understood term. Any action is deterministic, when the algoritm has a single path to follow. It does not have to choose a single one of more possible paths, to continue.

But in generating of strings by grammars, we are facing undeterministic decisions at almost every step. Yet we are using concept of deterministic applications of rules in paused modification of regulated grammars. This makes generation of strings using this modification even harder.

6.1.1 Undeterministic Generation

Process of generating single string of target language, of ordinary uncontrolled grammars is simple. We can describe it with this short algorithm 6.1. This algorithm is a result of a definition of the generated language, by chaining derivations, until final sentence is acquired.

Algorithm 6.1 Generation of string by unrestricted grammar

```
Input: Grammar $G = (N, T, P, S)$
Output: $w \in T^*$ is a generated string
1: $w \leftarrow S$
2: while $w \notin N^*$ do
3:     $Possible \leftarrow \{(l, r, |s_l|) | s_l l s_r = w, (l \to r) \in P\}$
4:     if $|Possible| = 0$ then goto 1
5:     end if
6:     Choose $(l, r, n) \in Possible$
7:     $s_l l s_r \leftarrow w$, such that $|s_l| = n$
8:     $w \leftarrow s_l r s_r$
9: end while
```

We write down start symbol of our grammar. Then we apply any applicable rule of specified set of rules. We repeat this step until all symbols in our string are terminal symbols.

This algorithm 6.1 alone is able to generate all strings of language, that is defined by our uncontrolled grammar. In this simple algorithm, we may find ourselves in dead-end, when no rule is applicable. This algoritm will then reset to its initial state and lets us choose different path. For just now, we can ignore infinite loops, that may occure during run of this algorithm.

We can also write down an algorithm that outputs whole language, not just single string. But when lanugage is not finite, that algorithm would newer stop. This makes it impractical.

With rule-based regulation applied to a grammar, our job of generating a valid sentence for a given language, becomes a little bit harder. We need to adhere to the specified order of rule applications. This will limit the set of rules, we are able to use at any given time. This limitations can change between every derivation step. But when we adhere to this limits, we can use similar algorithm 6.2, to the one, we have used for the uncontrolled grammars.

As with uncontrolled grammars, we write down a start symbol. We have added a string of used rules, this string we initialize on emptystring. We filter applicable rules with controll language, so that application of a selected rule would not fall outside of controll language. When we apply the selected rule, we also add its name into controll string.

In preseted algorithm, we allow to use any applicable rule, for current sentence form. We never allow using a production rule, which is not usable with specified control language. We are allowed to take any path, that is allowed by the controlled grammar, therefore, we can generate any string from language, genrated by a given grammar.

6.1.2 Ways to Determinism

Only truly deterministic step in generation of a string, can be seen in a situation, where current sentential form contains a single occurence of a left-hand side of a single usable

Algorithm 6.2 Generation of string by regulated grammar

Input: Grammar $H = (G, \Xi)$, where $G = (N, T, \Psi, P, S)$
Output: $w \in T^*$ is a generated string

```
1:  $w \leftarrow S$
2:  $c \leftarrow \varepsilon$
3:  while $w \notin N^*$ do
4:      $Possible \leftarrow \{(p, l, r, |s_l|) | s_l l s_r = w, (p : l \to r) \in P, q \in \Psi^*, cpq \in \Xi\}$
5:      if $|Possible| = 0$ then goto 1
6:      end if
7:      Choose $(p, l, r, n) \in Possible$
8:      $s_l l s_r \leftarrow w$, such that $|s_l| = n$
9:      $w \leftarrow s_l r s_r$
10:     $c \leftarrow cp$
11: end while
```

production rule. In this situation we are unable to make any other choice, then use this single rule, to acquire next sentential form. In all other cases, we are forced to make a choice.

We should note, that determinism is not equal to disambiguity. With both, we are allowed to use only a single path, to generate a concrete sentence. When using determinism, we add another condition. When we have to choose a path, we can have only a single one. We are not allowed to try, fail and try again.

We can choose multiple paths, to acquire generation with determinism. We will discuss two of them. The *blind path* and the *controlled path*. Both paths are using different mechanisms, but both paths are working on a principle of restricting original model.

Blind Path

The blind path can be, for example, used to define deterministic context-free language 6.1.

Definition 6.1 (Deterministic context-free language)
A lanugage L is a *deterministic context-free lanugage*, when there exists context-free grammar G, which describes it, and there exists a deterministic pushdown automata, which accepts language L.

We may see, that this definition combines two very different models for the language, it describes. It uses both grammars and automata. Since we know, that pushdown automata are equivalent to context-free grammar, we can ommit the notion of a grammar from this definition, without compromising the result. After this, we only need to define deteminism for a puhdown automata. That is not our goal, since the determinism for parsing is already defined.

When we stick with using grammars in definition of deterministic lanugage, we still are not telling anything about grammar itself, only that its language is deterministic. To that conclusion, we arrive by using some automata, to check afterwards. The grammar itself can behave undeterministically. We use grammar blindly, without any notion of determinism, during the production of a sentence.

Based on the definition of a deterministic context-free language 6.1, we may attempt to create a definition of deterministic context-free grammar, that constructs the deterministic language by itself. To handle this, we may use a transformation algorithm, to transform grammar into automata. Then we may define a deterministic context-free grammar as a context-free grammar, which by using particular transformation algorithm, yields deterministic pushdown automata.

Construction of such algorithm may not be trivial matter. Especialy for a more complicated models, then context-free grammars. Even then, we still do not specify deterministic steps of production directly, but indirectly, by transformation.

Controlled Path

The controlled path to acquire determinism of generation, is build on top of a different concept from the blind path. We define a grammar itself, and a derivation used to produce strings, as a determistic. We will use additional control over them, to meet our target. This additional control will work side by side with the production rules.

Main problem, we face, is the limited definition, of the atomic step of acquiring sentence of lanugage. It is single derivation. If we want to construct deterministic grammar, we need to limit the number of possible derivations to exactly one. But when we raise this extreme kind of limit, we will limit ourselves to at most single valid sentence in language. This is implied by choosing a single path in every step. We cannot deviate.

We also do not want to take any path we want, and then check the result, if we realy was deterministic in our choices, that is the blind path.

We want to restrict the derivation in such a way, that we will leter be able to say, that we acted deterministically, without actually acting deterministically. We can easily achieve that, by stating, that we are acting deterministically. This statement alone is very powerfull. With this statement, we are saying, we had exactly single option to act, at a certain step of production.

If, we would just created a statement, without retention of any supporting information, we would not be correct about defending it.

Example 6.1
Consider a context-free grammar $G = (N, T, P, S)$, where

$$\begin{aligned}
N &= \{S, R\}, && (6.1)\\
T &= \{a, b\}, && (6.2)\\
P &= \{p_1 : S \ \rightarrow RbS, \\
&\qquad p_2 : R \ \rightarrow aS, \\
&\qquad p_3 : S \ \rightarrow \varepsilon\}. && (6.3)
\end{aligned}$$

Lets look at two separate derivations, that may occure. Every derivation is marked using a tuple. First part of this tuple is a rule, used in this step, and the second part is a statement, whethever it was applied deterministically (D), or not (N).

$$\begin{aligned}
S \ &\Rightarrow RbS[(p_1, D)] \\
&\Rightarrow aSbS[(p_2, D)] \\
&\Rightarrow aaSbS[(p_2, D)] \\
&\Rightarrow aabS[(p_3, N)] \\
&\Rightarrow aab[(p_3, D)]
\end{aligned}
\qquad\qquad
\begin{aligned}
S \ &\Rightarrow aS[(p_2, D)] \\
&\Rightarrow aRbS[(p_1, D)] \\
&\Rightarrow aaSbS[(p_2, D)] \\
&\Rightarrow aaSb[(p_3, N)] \\
&\Rightarrow aab[(p_3, D)]
\end{aligned}$$

In example 6.1, we can see a problem, with a simple statement, about limiting number of available productions, when we do not take into an account all already executed derivations. The grammar outputted the same string, using two different sequences of derivations, while both of them started deterministically. This is a direct contradiction with the semantics of determinism during parsing. When we are conducting deterministic parsing, we can never take two different paths daterministcally, from the same initial conditions.

In order, to generate a language, we have to choose to behave deterministically. We cannot create statements, and later ignore them. When we are stating, that we made a deterministic step, we are creating adhoc artificial limitation. This limitation states, that we choosed, to have no other choice.

All of those limitations has to apply not only for current sentential form, but also for all consequent ones. Hence with every production, we are increasing a set of limitations. And when we are choosing next rule, we have to respect all already created limitations.

We have to create limitations. Those limitations has to be recorded. We can use a natural lanugage to express them, but that would not be easy to transform into any practically usable form. We can use a formal lanugages, to express them. For every rule, we then create a limitation in form of a language. This language will then help us to specify the cases of deterministic usage.

By using a production rule, we say, that final sentence will have to obey a form, prescribed by limiting language of the selected rule. When we use rule deterministically, no other limiting lanugage may be applicable to the final sentence. Also when we specificaly state, that rule was used nondeterministically, then all next sentential forms has to obey more than one of limiting languages, specified by rules of a grammar.

Take a closer look at a form of each context-free rule. Its left-hand side contains single nonterminal symbol. Its right-hand side is formed by a single string of nonterminals and terminals. This should be the basic for our limiting language. We will use a simple transformation from a production rule to a limiting lanuguage. This transformation should characterise the right-hand side of a rule. Every string, that could be generated starting this rule, should be in the limiting lanugage. In other case, we would allow us to fall into a trap, by raising limitations, which could never be met.

Our limitations will be created by analysing every rule in context of a whole grammar and finding a language, generated by this rule. This lanugage consists of a strings of terminal symbols, that are results of rewriting right-hand side of a rule, by any rule in the analysed grammar. Due to the context-free nature of every rule, the limiting language cannot specify conditions outside of a substring, generated directly of nonterminal, rewriten by this rule.

This easy transformation of a rule into a produced language, can be performed with context-free rules, because they do not allow symbols to change relative position to each other. When two instancess x and y of symbols in a sentence $kxlym$, are in this particular order, then it will never change, even when strings k, l and m are changed. The only exception of this rule is a case, when one of the symbols is rewriten. In this case, the relation of a position holds with whole string, that came to be from the rewriten symbol. This holds, because the subtree of a derivation tree, rooted with a rewriten symbol, stays at the original position of a rewriten symbol.

Extreme Examples

We will analyze two extreme cases of limiting languages. We will demonstrate this limitations on a grammar from previous example 6.1.

The first example uses very liberal definition of a limiting lanugage. It is trying to be the least specific, it can be.

Example 6.2 (Nonspecific limitation)
Consider a context-free grammar G from example 6.1. Consider a limiting language $L(p) = \{w|w \in T^*\}$, for every rule $p \in P$. For brevity, we write a rule p in format $p_n : l \to r[L(p_n)]$.

$$\begin{aligned} P = \{p_1 : S &\to RbS[(a+b)^*], \\ p_2 : R &\to aS[(a+b)^*], \\ p_3 : S &\to \varepsilon[(a+b)^*]\}. \end{aligned} \tag{6.4}$$

All limiting lanugages are the same. They are all supersets of lanugages, that are actualy producable, from any of the rules. The limiting languages are universal languages.

This transformation can be used to produce a limiting lanugage, but we are not able to define a deterministic limitation, using it. When we say, that one of the rules is not usable, non of them are. This limiting language can achieve only nondeterministic derivations, when sentential form contains more than a single occurence of nonterminal symbol. Even then, for deterministic operation, we would require exactly one rule for a nonterminal.

Transformation of this model, to parser is straight forward. You may apply any rule, any time. Due to the fact, that every possible substring, that could appear in a concrete sentence, is valid for any of the rules.

The second example goes to the farthes end of a spectra from the first example. The limiting lanugage is the most specific possible.

Example 6.3 (Maximal limitation)
Consider a context-free grammar G from example 6.1. Consider a limiting language $L(l \to r) = \{w|w \in T^*, r \Rightarrow_G^* w\}$, for every rule $p = (l \to r) \in P$. For brevity, we write a rule p in format $p_n : l \to r[L(p_n)]$.

$$\begin{aligned} P = \{p_1 : S &\to RbS[a^+(ba^*)^+], \\ p_2 : R &\to aS[a^+(ba^*)^*], \\ p_3 : S &\to \varepsilon[\varepsilon]\}. \end{aligned} \tag{6.5}$$

All limiting lanugages are constructed as an exact languages producable from a right-hand side of a rule. This transformation may yield a limiting language of the same class as the grammar itself.

This type of transformation is the worst case. We need to perform the deepest possible analysys on a whole string of right-hand side of a rule and finds all possible subsentences, which start by using of a rule. Therefore this is an ultimate tool to check for deterministic decision on picking up a rule, because we can be certain, it is both applicable and only possible.

Transformation of this model, to a parser is also straight forward. But its runtime complexity is much greater. In the first example, we had to check nothing. In this case, we have to check everything to the last symbol of a string we want to generate from a current symbol. We may see, that in the first derivation, we are trying to match the initial symbol with complete input string, therefore in order to pick the rule, to be used, we need to complete the syntax analysis itself.

Other Examples

First two examples was exploring the farthes borders of a spectra of limiting languages. We can also make a compromise between the precision and an ammount of analysis needed.

Example 6.4 (Leftmost 1)
Consider a context-free grammar $G_2 = (N, T, P, S)$, where

$$
\begin{aligned}
N &= \{S, R\}, && (6.6)\\
T &= \{a, b, c\}, && (6.7)\\
P &= \{p_1 : S \rightarrow RSb,\\
&\quad p_2 : S \rightarrow RSc,\\
&\quad p_3 : S \rightarrow aR,\\
&\quad p_4 : S \rightarrow bR,\\
&\quad p_5 : R \rightarrow \varepsilon\}. && (6.8)
\end{aligned}
$$

Consider a limiting language $L(l \rightarrow r) = \{ac | r \Rightarrow_G^* aw \wedge ((aw = \varepsilon \wedge c = \varepsilon) \vee (a \in T \wedge c \in T^*))\}$. This limitation takes all first symbols, that may appear in a generated language and concatenates them with universal language, if that symbol exists.

$$
\begin{aligned}
P = \{p_1 : S &\rightarrow RSb[(a+b)(a+b+c)^*],\\
p_2 : S &\rightarrow RSc[(a+b)(a+b+c)^*],\\
p_3 : S &\rightarrow aR[a(a+b+c)^*],\\
p_4 : S &\rightarrow bR[b(a+b+c)^*],\\
p_5 : R &\rightarrow \varepsilon[\varepsilon]\}. && (6.9)
\end{aligned}
$$

This is a metric used in $LL(1)$ parsers, checking only a single symbol at the start of a subtree-generated substring.

Example 6.5 (Natural limitation)
Consider a context-free grammar G_2 from the example 6.4 and a limitation, which constructs regular limiting lanugage from a rule. Every always erased symbol is replaced with ε. Every other erasable nonterminal symbol is replaced with universal lanugage. Every non-erasable one with universal lanugage without empty string. We call this the *natural limitation*.

$$
\begin{aligned}
P = \{p_1 : S &\rightarrow RSb[(a+b+c)^+b],\\
p_2 : S &\rightarrow RSc[(a+b+c)^+c],\\
p_3 : S &\rightarrow aR[a],\\
p_4 : S &\rightarrow bR[b],\\
p_5 : R &\rightarrow [\varepsilon]\}. && (6.10)
\end{aligned}
$$

In the dfference from the previous example, we can see, that for rules p_1 and p_2, the most specific parts of a limiting language moved from the first symbol to the last symbol. Other rules becomes the most specific langages possible.

6.2 Definition

We will define a determinism-paused regulation mode for regular-controlled grammars. With this mode of paused regulated grammar, we need to closely monitor, which productions was made deterministically and which not. Those, made deterministically, are not included in a string of controll language. This is a mechanism chosen for shortening of strings of control language.

Definition 6.2
A *determinism-paused language-controlled (context-free) grammar*, is a pair

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

where

- $G = (N, T, \Psi, P, S)$ is a context-free grammar, called *core grammar*;
- $\Xi \subseteq \Psi^*$ is a language over rules, called *control language*;
- N is a finite set of nonterminal symbols;
- T is a finite set of terminal symbols;
- $P \subseteq (N \times (N \cup T)^*)$ is a set of *rules*;
- $S \in N$;
- Ψ is a set of symbols called *rule labels* such that $|\Psi| = |P|$ and ψ is a bijection from Ψ to P;

The *mapping function* $\alpha : (\Psi, \{0,1\}) \rightarrow \Psi^*$ is defined as

$$\alpha(r,k) = \begin{cases} r & \text{iff } k \ = 1 \\ \varepsilon & \text{otherwise.} \end{cases} \tag{6.11}$$

The *filtering function* $\beta : (D, (N \cup T \cup D)^*) \rightarrow (N \cup T)^*$ is defined as

$$\beta(d,tv) = \begin{cases} \varepsilon & \text{iff } tv = \varepsilon \\ \beta(d,v) & \text{iff } |t| = 1 \wedge t \in d \\ t\beta(d,v) & \text{iff } |t| = 1 \wedge t \notin d \end{cases} \tag{6.12}$$

The *formating function* $\gamma : P \rightarrow f \subseteq T^*$ is the natural limitation (see example 6.5).The rule $p_1 : l_1 \rightarrow r_1 \in P$ is *inherently nondeterministic* iff there exists rule $p_2 : l_2 \rightarrow r_2 \in P$, such that $l_1 = l_2$, $r_1 \neq r_2$ and $\gamma(l_1 \rightarrow r_1) = \gamma(l_2 \rightarrow r_2)$.

The *configuration* is a tuple (w, s), where w is a sentential form and l is a set of limitations, which are defined as a tuple $(\dagger, l)$, $\dagger$ is a *border symbol* and l is a *limiting lanugage.* Each nonterminal symbol k is upon insertion into a sentential form inserted into envelope of two new identical border symbols, which together form tightly packed group $\dagger k \dagger$, without any interleaving symbols. Between border symbols is then contained a subtree, gained from the original symbol. The limiting language then forms a superset of language, allowed to apear between the belonging border symbols after application of the filtering function β, to remove all inside border symbols. Default limiting language for a single symbol is universal language.

The configuration (w, s) is *valid*, when none of the limiting lanugage in s is empty language and for every substring $x = x_1\dagger_0 v\dagger_0 x_2$ of w, where $v = v_0\dagger_1 u_1\dagger_1 v_1 \dots v_{n-1}\dagger_n u_n\dagger_n v_n$, $v_i \in T^*$, $B = \{\dagger|(\dagger, l) \in s\}$ and $(\dagger_i, L_i) \in s$ for $i \geq 0$, the limiting language $L_0 \cap \{\beta(B, v_0 L_1 v_1 \dots v_{n-1} L_n v_n)\} \neq \varnothing$.

For the rule $(p : l \to r) \in P$ to be *applicable* on a configuration $(x_l\dagger_l l\dagger_l x_r, s)$, consequent configuration $(x_l\dagger r\dagger x_r, \{(\dagger, K)|(\dagger, L) \in s \wedge K = L \cap \gamma(p) \text{ if } \dagger = \dagger_l, L \text{ otherwise}\})$ has to be a valid configuration. The application of the rule does not create impossible configuration of limitations.

A *single derivation* $\Rightarrow_H [(p, d)]$, using applicable production rule $p : l \to r$ is a relation between two valid configurations, defined as

$$
\begin{aligned}
& c_i = (w_i = x_l\dagger_p l\dagger_p x_r, s_i) \Rightarrow_H [(p, d)](w_{i+1} = x_l\dagger_p r\dagger_p x_r, s_{i+1}) = c_{i+1} : \\
& \text{if } d = 0 : \\
& \quad s'_{i+1} = \{(\dagger, K)|(\dagger, L) \in s_i \wedge K = L \cap \gamma(p) \text{ if } b = \dagger_p, \text{ otherwise} L\}, \\
& \quad s_{i+1} = s'_{i+1} \text{ where all limiting languages} L \text{ for any reachable single nonterminal } n \\
& \qquad \text{are chaged into } L \cap \bigcap\{\overline{\gamma(q)}|(q : n \to o) \in (P \setminus p)\} \\
& \text{if } d = 1 \wedge \text{ exists any other configuration } c_i \Rightarrow_H c_j \text{ using any rule :}, \\
& \quad s_{i+1} = s_i\{(\dagger, K)|(\dagger, L) \in s_i \wedge K = L \cap \gamma(p) \text{ if } b = \dagger_p, \text{ otherwise} L\}
\end{aligned}
\tag{6.13}
$$

The *language* of H, denoted by $L(H)$, consists of all strings w, for which there is a derivation in H, such that $\alpha(r_1, d_1)\alpha(r_2, d_2) \dots \alpha(r_n, d_n) \in \Xi$ for some $n \geq 1$ where

$$
\begin{aligned}
(S, \varnothing) & = c_0 \Rightarrow_H c_1[(r_1, d_1)] \Rightarrow_H c_2[(r_2, d_2)] \Rightarrow_H \cdots \Rightarrow_H c_n[(r_n, d_n)] = (w_n, s_n), \\
& w = \beta(\{s|(s, l) \in s_n\}, w_n) \in T^*, \\
& d_i \in \{0, 1\}
\end{aligned}
\tag{6.14}
$$

In definition 6.2, we can see a couple of differences, from previously introduced switch-paused mode of regulation. The determinism-paused mode does not add another sets do definition of grammar. Neither does it add another restrictions to the controll language. The definition of the grammar itself is unchanged from he original regular-controlled grammar. The change are mustly in the definition of derivation and generated language.

The configuration contains serialized leafs of a derivation tree and applied limitations for subtrees. Derivations between this configurations can be either deterministic or nondeterministic.

A simple mapping function α was added. This function is used in a definition of generated language. Every production has to be either deterministic or non-deterministic. We need to track all of them, to create a full chain leading up to final string. When we compare string of productions with controll language, we use only non-deterministic ones. This function efectively drops the productions, which are not supposed to be in control lanaguage.

Next, a filtering function β is used to remove marker symbols, which are not part of limiting or output lanugages.

The formating function γ is used in order to pick a production rule. Deterministic application of a rule blocks any other rules by intersecting limiting language of possible locations with complements of limiting lanugages of all non used rules. If inherently non-deterministic production rule would be applied deterministically, then the other rule with the same limiting language would block it, because intersection of limiting lanugage with its complement is empty.

6.2.1 Leftmost Modification

From definition of determinism-paused mode, we may see, that in many cases, we are standing between non-deterministic choice. Take a simple example of a sentential form, which contains two or more nonterminal symbols. When the already acquired conditions are not specific enough, we may successfully derive both of them. This results in undeterministic decision, which has to be writen into the control languge. This is unwanted, when we want to shorten a string of control language.

The situation changes rapidly, when we apply the regulated leftmost modification 3.4, on top of the determinism-paused modification. We can always limit the derived nonterminal down to a single one. We only need to choose appropriate rule.

We can also modify the formating function, to force a different definition of an inherently nondeterministic rule. As an example, we can choose the Leftmost 1 limitation 6.4.

The effect of this compound $LL(1)$ modification can be seen in example 6.4.

6.3 Other Regulations

This modification is intendet to be more general, than modify only regular=controlled grammars. We can outline few propositions for these modifications. There exists manz tzpes of regulated grammars. This list is not neither complete, nor does it contain complete rigorous definitions.

6.3.1 Random Context Grammars

This model can use similar modification to the one used for the regular-controlled grammars. It can attempt to find a production rule by first ignoring its permitting and forbidding sets. If a single rule is found this way, it can be applied. In other cases, the original model is used.

6.3.2 Tree-Regulated Grammars

Ordinary tree-regulated grammars are checking whole derivation tree or some subset of paths or cuts [12]. Determinism-paused modification of these grammars then must also take into account the whole derivation tree. Possible path can be seen, using two additional mechanisms.

The first mechanism can be used to annotate nodes in drivation tree with one of two possible marks, carrying information, whetever it was rewriten deterministically or not. When a derivation tree is to be checked, the second mechanism can be imployed to create filtered image of an original tree. this image could contain only nondetermistically created symbols, while retaining path relations between them, only cumulating multiple edges into single one. Then the original model can perform checks on this filtered image.

6.4 Examples

We will investigate a few simple examples of languages, generated by determinism-paused modification of regulated grammars.

Example 6.6 ($a^nb^nc^n$)
Consider the regular-lanugage regulated grammar H_1 and its determinism-paused variant H_1^{DP}.

$$\begin{aligned}
H_1 &= (G_1, \Xi_1) && (6.15)\\
G_1 &= (N, T, \Psi, P, S) && (6.16)\\
N &= \{S, A, B, C\} && (6.17)\\
T &= \{a, b, c\} && (6.18)\\
P &= \{p_0 : S \rightarrow ABC,\\
&\quad p_1 : A \rightarrow aA, p_2 : B \rightarrow bB, p_3 : C \rightarrow cC,\\
&\quad p_4 : A \rightarrow a, p_5 : B \rightarrow b, p_6 : C \rightarrow c\} && (6.19)\\
\Xi_1 &= p_0(p_1p_2p_3)^*p_4p_5p_6 && (6.20)\\
L_1 &= \{a^nb^nc^n | n \geq 1\} && (6.21)
\end{aligned}$$

The grammar H_1 is generating canonical language L_1. The control language can be divided into phrases of three rules, where every phrase injects exactly one terminal of each kind.

$$\begin{aligned}
S &\Rightarrow ABC[p_0]\\
&\Rightarrow aABC[p_1]\\
&\Rightarrow aAbBC[p_2]\\
&\Rightarrow aAbBcC[p_3]\\
&\Rightarrow aabBcC[p_4]\\
&\Rightarrow aabbcC[p_5]\\
&\Rightarrow aabbcc[p_6] && (6.22)
\end{aligned}$$

The determinism-paused version does differ in the control lanugage. This difference

$$\begin{aligned}
H_1^{DP} &= (G_{1D}, \Xi_1^{DP}) && (6.23)\\
G_{1D} &= (N, T, \Psi, P, S) && (6.24)\\
N &= \{S, A, B, C, X_1, X_2\} && (6.25)\\
T &= \{a, b, c\} && (6.26)\\
P &= \{p_0 : S \rightarrow ABC,\\
&\quad p_1 : A \rightarrow aA, p_2 : B \rightarrow bB, p_3 : C \rightarrow cC,\\
&\quad p_4 : A \rightarrow a, p_5 : B \rightarrow b, p_6 : C \rightarrow cX_1,\\
&\quad p_7 : C \rightarrow cX_2, p_8 : X_1 \rightarrow \varepsilon, p_9 : X_2 \rightarrow \varepsilon\} && (6.27)\\
\Xi_1^{DP} &= (p_1p_2p_3)^*p_4p_5p_6 && (6.28)
\end{aligned}$$

Most of the derivations are done in nondeterministic manner. This is a simple result of multiple nonterminals available, even when there exists only a single applicable rule for each of the nonterminals. It is an intrinsic property of grammars without limitation of used nonterminals.

$$
\begin{aligned}
S \; &\Rightarrow ABC[(p_0, 0)] \\
&\Rightarrow aABC[(p_1, 1)] \\
&\Rightarrow aAbBC[(p_2, 1)] \\
&\Rightarrow aAbBcC[(p_3, 1)] \\
&\Rightarrow aabBcC[(p_4, 1)] \\
&\Rightarrow aabbcC[(p_5, 1)] \\
&\Rightarrow aabbccX_1[(p_6, 1)] \\
&\Rightarrow aabbcc[(p_8, 0)] \qquad (6.29)
\end{aligned}
$$

In this example, we can see, that reasoning about RC^{DP} grammars can be hard in situations, when the grammar transitiones from using mutliple possible nonterminals to single one. We have to be avare of this possibility and create rules acordingly. In the first case, we are not forced to use nondeterministic rules, while still using them nondeterministically. In the second case, when we are limited to a single nonterminal, we have to create artificial nondeterminism, when it is not already there, in order to control the production with control language.

When we would used set of rules from original grammar H_1, we would be able to produce symbol s in any number $\geq n$.

Example 6.7 ($LL(1)$)
Lets look at another example. Consider the regular-lanugage regulated grammar H_2 and its $LL(1)$ determinism-paused variant $H_2^{DP,LL1}$.

$$
\begin{aligned}
H_2 &= (G_2, \Xi_2) && (6.30) \\
G_2 &= (N, T, \Psi, P, S) && (6.31) \\
N &= \{S, A, B, C\} && (6.32) \\
T &= \{a, b, c\} && (6.33) \\
P &= \{p_0 : S \; \rightarrow XtYt, \\
&\qquad p_1 : X \rightarrow aX, p_2 : X \rightarrow cdX, p_3 : X \rightarrow ceX, p_4 : X \rightarrow \varepsilon, \\
&\qquad p_5 : Y \rightarrow cdY, p_6 : Y \rightarrow ceY, p_7 : Y \rightarrow \varepsilon\} && (6.34) \\
\Xi_2 &= p_0(p_1^*(p_2p_5|p_3p_6))^*p_4p_7 && (6.35) \\
L_2 &= \{uv | u = (a^*(cd|ce))^n t \wedge v = u \text{ bez } a \wedge n \geq 0\} && (6.36)
\end{aligned}
$$

The deterministic version differs only in used control lanugage.

$$
\begin{aligned}
H_2^{DP,LL1} &= (G_2, \Xi_{2D}) && (6.37) \\
\Xi_{2D} &= (p_2p_5|p_3p_6)^* && (6.38)
\end{aligned}
$$

Only the leftmost derivable nonterminal are taken into consideration. We are able to simplify control language to the bare minimum, needed for contex-sensitive parts of the language. Most of the derivations are deterministical.

$$\begin{aligned}S &\Rightarrow XtYt[(p_0,0)]\\ &\Rightarrow aXtYt[(p_1,0)]\\ &\Rightarrow acdXtYt[(p_2,1)]\\ &\Rightarrow acdaXtYt[(p_1,0)]\\ &\Rightarrow acdaXtcdYt[(p_5,1)]\\ &\Rightarrow acdaceXtcdYt[(p_3,1)]\\ &\Rightarrow acdaceaXtcdYt[(p_1,0)]\\ &\Rightarrow acdaceatcdYt[(p_4,0)]\\ &\Rightarrow acdaceatcdceYt[(p_6,1)]\\ &\Rightarrow acdaceatcdcet[(p_7,0)]\end{aligned} \tag{6.39}$$

In the example , we can see an interesting phenomena. We have statically restricted ourselves in selection of both nonterminal and terminal symbols. As a response, we have acquired a simpler control model, then we would have without these limitations. The control has effectively shifted between these two restrictions. Or rather it is not copied in both of them, since both can complement each other.

6.5 Generation

Grammars are, in oposition to its coresponding automata, intended to be primarily generative tools for specified languages. From this reason alone, we should

With rule-based regulation applied to grammar, our job of generating a valid sentence for a given language, becomes harder. We need to adhere to the specified order of rule applications. This will limit the set of rules, we are able to use at any given time. Thi limitations can change between every derivation. But when we adhere to this limits, we can still use the same algorithm, we have used for the uncontrolled grammars.

Even the switch-paused mode of operation does not change this basic concept. It only adds another level of regulation to the way, we use to acquire a limitation for every derivation step.

This simple concept changes, when we use a determinism-paused modification of controlled grammars. We need to introduce another concept to the process of generation of strings of the target language. It is a concept of determinism, which is usually used in terms of parsing.

We start the genetion in the same manner as with unregulated grammars, by initializing working sentence to starting symbol. Then we employ regulating model to filter set of rules to a subset of all rules. We are then able to use only those rules.

With determinism-paused modification, we need to predict deterministic choices that would ocur in parsing of any given string. this will be

6.6 Parsing

For a newly introduced model, a parsing technique was crated. This is based upon a transformations of the basic model for regular-language regulation.

6.6.1 Application of Limitations

We have defined a grammar, which is using sets of limitations on subsentences generated from a particular subtree. We have to reflect this design in a parsing routine.

Every application of a rule adds its limitation to a certain substring of a generated sentence. In generation of a string, we start with universal lanuguage and limit it by using a rules. The generated sentence is then an intersection of all already applied limitations. We are applying another rules, until the intersection does not contain exactly one sentence and no more limitations can be added.

In parsing, we already have a single sentence, and trying to find an order of rules applications, which satisfies this single input sentence. Every intersection of a unary language with another lanugage can yield only the unary language itself, or empty language. When the intersection of applied limitations and input string is empty language, we were not successful in finding apropriate limitations.

Because the possible intersection can have only single sentence, we do not need to store all applied limitations in any other way, than a current sentential form, because all of them is already writen into an input sentence.

6.6.2 Specific Transformations

Every rule in automata, created by basic transformations, is using feasibility function $c(p)$, which is implemented as a search for every possible location of derivation by a rule p. At every state, small single rule loop for every production rule q is added. This loop starts with a single rule on a smaller priority and applies a single production rule. This rule is using feasibility function $o(q)$, which is possitive only when the rule q and no other is applicable.

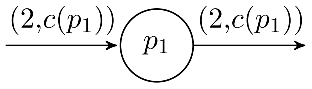

Figure 6.1: Original state

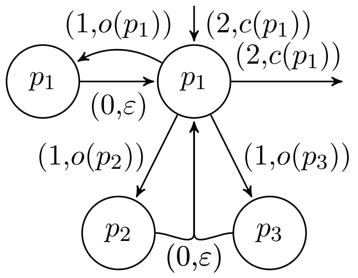

Figure 6.2: Transformed state

6.7 Properties

We will discuss a properties of the newly introduced determinism-paused grammars. We will compare them with its original version.

6.7.1 Original to Determinism-paused

Every regular-controlled grammar can be converted to determinism-paused version, that generates the same language. The algorithm 6.3 is using inherently nondeterministic rules, to achieve this transformation. For every rule of original grammar, we create at least one other rule, which is in the same format. These two rules becomes inherently nondeterministic.

We cannot create second identical rule, because the rules are not organised in multiset. But we can introduce new nonterminal symbols, to create semanticaly identical rule. Every nonterminal symbol is doubled, to obtain two symbols with identical semantic function. For every rule, there is created a set of inherently nondeterministic rules, created by differentiation in used nonterminal symbols. By this system, we cannot duplicate the rules containing no nonterminal symbols. These rules can be multiplied by adding a nonterminal symbols, which are immediately deleted.

At last, new controll language, which uses new rules, has to be created. In place of every original rule, any of new rules, created from it, could be used.

Algorithm 6.3 Transformation of RC grammar to RC^{DP} grammar

```
Input: $RC$ grammar $H_I = (G_I, \Xi_I)$, where $G_I = (N_I, T, \Psi_I, P_I, S)$
Output: $RC^{DP}$ grammar $H_O = (G_O, \Xi_O)$, where $G_O = (N_O, T, \Psi_O, P_O, S)$
 1: $N_O \leftarrow \{S\} \cup (\bigcup_{n \in N} m_n = \{n_1, n_2\})$
 2: $N_O \leftarrow N_0 \cup (\bigcup\{\{X_{p,1}, X_{p,2}\} | (p : l \to r) \in P_I \wedge r \in T^*\})$
 3: $P_O \leftarrow \{p_{S1} : S \to S_1, p_{S2} : S \to S_2\}$
 4: $P_O \leftarrow P_O \cup \bigcup_{p \in \Psi_I} z_p = \left\{ \begin{array}{r} \{(l_m \to r_0 r_1 \dots r_k x), (x \to \varepsilon)\} \mid l_m \in m_l \wedge r_i \in T \\ \wedge x \in \{X_{p,1}, X_{p,2}\} \wedge r_o r_1 \dots r_k \in T^* \wedge \\ 0 \le i \le k \wedge (p : l \to r_o r_1 \dots r_k) \wedge k \ge 0\} \end{array} \right\}$
 5: $P_O \leftarrow P_O \cup \bigcup_{p \in \Psi_I} z_p = \left\{ \begin{array}{l} (l_m \to s_0 s_1 \dots s_k) \mid l_m \in m_l \wedge s_i = r_i \text{ if } r_i \in T \text{ else } s_i \in m_{r_i} \wedge \\ \quad 0 \le i \le k \wedge (p : l \to r_o r_1 \dots r_k) \wedge k \ge 1 \wedge \\ \quad r_o r_1 \dots r_k \notin T^* \end{array} \right\}$
 6: $\Xi_O \leftarrow \{(p_{S1} | p_{S2}) y_0 y_1 \dots y_n | x_0 x_1 \dots x_n \in \Xi_I, x_i \in \Psi_0, y_i \in z_{x_i}, 0 \le i \le n, n \ge 0\}$
```

Lemma 6.1 (Correct transformation) Transformation algorithm 6.3 is correct.

Proof 6.1
Let us investigate properties of the transformation.

- Every rule is inherently nondeteministic.
- Every production has to take a part in a control lanugage.
- For every rule, there exists a group of one or two rules, which are a direct translation of it.
- All translations are regular.
- Regular languages are closed under transduction.

This translation has also disadvantages. It introduces an erasing rules into a grammar, hence the output grammar is not propagating. The nonterminal can be rewriten to terminals or erased. For a rule rewriting into terminals, the erasing rule is introduced.

6.7.2 Expressive Strength

Theorem 6.1
$\mathcal{L}(RC_L) \subseteq \mathcal{L}(RC_L^{DP})$, control language $L \in \{REG, CF, CS, RE\}$

Proof 6.2
We will construct a proof by providing a transformation.

- There exists a transformation 6.3 of any RC_L grammar into RC_L^{DP} grammar, defining the same language.

Open problem 6.1 (Equality)
$\mathcal{L}(RC_L) = \mathcal{L}(RC_L^{DP})$

This problem is caused by forced deterministic steps. If they would be only allowed, not forced, we could simply insert group of any not inherently nondeterministic rules between any two recorded productions. Next part is our unability to record the deterministic derivation in control language.

Theorem 6.2
$\mathcal{L}(CF) \subset \mathcal{L}(RC_L) \subseteq \mathcal{L}(RC^{DP}) \subset \mathcal{L}(RE)$. See [14][Theorem 5.1.6] and theorem 6.1

Theorem 6.3 (Empty sentence control language)
$\mathcal{L}(RC_L^{DP}) = \mathcal{L}(DCFL)$, control language $L = \{\varepsilon\}$

Proof 6.3
The grammar with empty string as a control lanugage can work only deterministically.

- Control language of every grammar in RC_L^{DP} is equal to $\{\varepsilon\}$.
- The grammar can use production rules only in deterministic mode. Every nondeterministic usage of rule would result in a non-empty string of control language.
- Every deterministic usage of rules is allowed.
- $DCFL$ denotes the class of deterministic context-free languages, which are created by deterministic context-free grammars, which use rules only deterministcally.
- Both models are using context-free rules in deterministic manner.

Theorem 6.4
$\mathcal{L}(RC_L) \subset \mathcal{L}(RC_L^{DP})$, control language $L \in \{FIN\}$

Proof 6.4
We will prove superiority of a determinism-paused model for finite control languages.

- Length of a right-side of any rule is finite.
- Number of sentences of control languages in FIN is finite.
- Hence $RC_L = FIN$.
- Consider a determinism-paused finite-controlled grammar $H^{DP} = (G, \Xi)$, where $G = (N, T, \Psi, P, S)$.
- An unlimited number of deterministic derivations can be performed.
- With an empty control sentence, the lanugage is in $\mathcal{L}(DCFL)$, see theorem 6.3.

By employing the determinism-paused regulation, we can preserve the expressive strength of a regular-lanugage regulated grammars. With limitations to finite control language, we are able to increase expressive power of a new introduced model.

Every regular-lanugage regulated grammar can be directly translated into a detrminism-paused version. The grammar, constructed by this translation, is valid determinism-paused regulated grammar. A grammar constructed in this way does not take advatage of the free deterministic steps.

Chapter 7

Conclusions

Formal languages are ireplaceable tool in modern theoretical informatics. Context-free languages, represented with context-free grammars, are one class of those languages. Context-free grammars have wanted properties in terms of complexity of production rules and consequently of implementation and reasoning. But they do not posses enough expressive power to solve all imaginable problems.

This work deals with the regulated grammars, which intends to increase expressive strength by combining context-free grammar with regulations into a single model. Many of these models was already introduced in the past.

In this work, a modification of grammar automata, the positioned grammar automata was introduced. This modification is adding a transfer of available positions between a checking function in rule of automata and the function applying the production rule of represented grammar.

Two new modifications of already existing regulated grammars models was introduced in this work. Both modifications are targeted on reduction of regulation of its original models, by ommiting some of productions from a control language. Both are using differrent ways to achieve this removal. Modifications are presented on regular-controlled context-free grammars. Examples of grammars using this modifications was created.

The switch-paused modification introduced additional switching sets of rules, to allow grammar to use free unregulated derivations. This could be used to ommit this parts from the control language, when the regulation is not needed.

The determinism-paused modification is using determinism, to reduce length of control sentences. Every time, a grammar is making deterministical step, it is not regulated with control lanugage. In order to specify the cases of deterministical derivation, a sets of limitations upon sentential form, applied with usage of rules, were introduced. A spectrum of these limitations was created.

A parsing methods was presented for both new modifications. This methods are working in a top-down manner, trying to simulate the derivation of an initial symbol into an input sentence. These methods was theoretically described using positioned grammar automata. An actual implementation using a programming language was not created, together with subsequent testing of this implementation.

It was proven, that none of the new modifications dimminish the expressive power of the original regular-controlled grammars model. The switch-paused modification has the exact same power of the unmodified model and the detrminism-paused modification has at least the same expressive power as the unmodified version. There exists an open problem of exact upper bound for a determinism-paused modification.

Next investigation could be directed towards solving suggested open problem. The modifications could also be defined for other regulated models of grammars. Properties would then have to be reevaluated for each of these models separately, based on details of definitions, but they should be similar for each regulation with power of regular languages.

Bibliography

[1] Aho, A. V.; aj.: *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Pearson Education, druhé vydání, 2006, ISBN 0-321-48681-1.

[2] Bednář, P.: *Syntaktická analýza založená na maticových gramatikách*. Bakalářská práce, FIT VUT v Brně, Brno, CZ, 2014.

[3] Chomsky, N.: Three models for the description of language. *Information Theory, IRE Transactions on*, ročník 2, č. 3, September 1956: s. 113–124, ISSN 0096-1000, doi:10.1109/TIT.1956.1056813.

[4] Chomsky, N.: On Certain Formal Properties of Grammars. *Information and Control*, ročník 2, č. 2, 1959: s. 137–167.

[5] Christiansen, H.: A survey of adaptable grammars. *ACM SIGPLAN Notices*, ročník 25, č. 11: s. 35–44, ISSN 0362-1340.

[6] Cormen, T. H.; aj.: *Introduction to Algorithms, Third Edition*. The MIT Press, třetí vydání, 2009, ISBN 978-0-262-03384-8.

[7] Dassow, J.; Păun, G.: *Regulated rewriting in formal language theory*. EATCS monographs on theoretical computer science, Springer-Verlag, 1989, ISBN 9783540514145.

[8] Diestel, R.: *Graph Theory (Graduate Texts in Mathematics)*. Springer, 2000, ISBN 0387950141.

[9] Grune; Jacobs: *Parsing Techniques: A Practical Guide*. Springer Publishing Company, Incorporated, druhé vydání, 2008, ISBN 978-0-387-20248-8.

[10] Harrison, M. A.: *Introduction to Formal Language Theory*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., první vydání, 1978, ISBN 0201029553.

[11] Hopcroft, J. E.; Motwani, R.; Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison Wesley, 2000, ISBN 0201441241.

[12] Koutný, J.: *Grammars with Restricted Derivation Trees*. Dizertační práce, 2012.

[13] Meduna, A.: *Automata and Languages: Theory and Applications*. Springer, 2005, ISBN 81-8128-333-3.

[14] Meduna, A.; Zemek, P.: *Regulated Grammars and Automata*. Springer, New York, 2014, ISBN 978-1-4939-0368-9.

Appendices

List of Appendices

A Content of CD **62**

Appendix A

Content of CD

Included CD contains:

- text of the technical report in PDF format;
- source code of the technical report in LaTeXformat.