BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FORMAL SYSTEMS BASED UPON AUTOMATA AND GRAMMARS

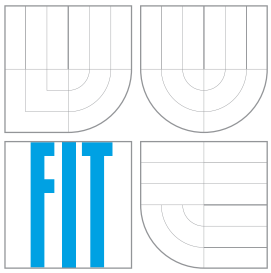
PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR
AUTOR PRÁCE

MARTIN ČERMÁK

BRNO 2012

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FORMAL SYSTEMS BASED UPON AUTOMATA AND GRAMMARS

FORMÁLNÍ SYSTÉMY AUTOMATŮ A GRAMATIK

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR
AUTOR PRÁCE

MARTIN ČERMÁK

SUPERVISOR
VEDOUCÍ PRÁCE

prof. RNDr. ALEXANDER MEDUNA, CSc.

BRNO 2012

Abstrakt

Tyto teze navazují na studium gramatických a automatových systémů. Na začátku, práce pojednává o regulárně řízených CD gramatických systémech využívající frázově strukturované gramatiky jako komponenty. Do systémů jsou zavedena tři nová omezení na derivacích a je studován jejich vliv na vyjadřovací sílu těchto systémů. Poté, tato práce definuje dva automatové protějšky ke kanonickým multi-generatiním nonterminálem a pravidlově synchronizovyným gramatickým systemům, generujících vektory řetězců, a ukazuje, že všechny tyto vyšetřované systemy si jsou vzájemně ekvivalentní. Dále táto práce tyto systémy zobecňuje a zakládá fundamentalní hierarchii n-jazyků (množin n-tic řetězců). V souvislosti se zavedenými systémy tyto teze zavádí automatově-gramatický převodník založený na konečném automatu a bezkontextové gramatice. Tento převodník je pak studovaný a použitý jako nástroj přímého překladu. V poslední části jsou v této práci zavedené automatové systémy jádrem pársovací metody založené na stromově řízených gramatikách s n omezenými cestami.

Abstract

The present thesis continues with study of grammar and automata systems. First of all, it deals with regularly controlled CD grammar systems with phrase-structure grammars as components. Into these systems, three new derivation restrictions are placed and their effect on the generative power of these systems are investigated. Thereafter, this thesis defines two automata counterparts of canonical multi-generative nonterminal and rule synchronized grammar systems, generating vectors of strings, and it shows that these investigated systems are equivalent. Furthermore, this thesis generalizes definitions of these systems and establishes fundamental hierarchy of n-languages (sets of n-tuples of strings). In relation with these mentioned systems, automaton-grammar translating systems based upon finite automaton and context-free grammar are introduced and investigated as a mechanism for direct translating. At the end, in this thesis introduced automata systems are used as the core of parse-method based upon n-path-restricted tree-controlled grammars.

Klíčová slova

automat, gramatika, automatový systém, gramatický systém, řízené přepisování, multi-generování, n-jazyk

Keywords

automaton, grammar, automata system, grammar system, controlled rewriting, multi-generation, n-language

Citace

Martin Čermák: Formal Systems Based upon Automata and Grammars, disertační práce, Brno, FIT VUT v Brně, 2012

Citation

Martin Čermák: Formal Systems Based upon Automata and Grammars, PhD thesis, Brno, FIT BUT, 2012

Formal Systems Based upon Automata and Grammars

Declaration

Hereby I declare that this thesis is my authorial work that have been created under supervision of prof. RNDr. Alexander Meduna, CSc. Some parts of this thesis are based on papers created in collaboration with colleagues RNDr. Tomáš Masopust, Ph.D., Ing. Jiří Koutný, or Ing. Petr Horáček. Specifically, the first two theorems in Chapter 4 are largely the work of prof. RNDr. Alexander Meduna, CSc. and RNDr. Tomáš Masopust, Ph.D. Chapter 8 and some parts of Chapter 6 are based on papers, which have been written in cooperation with Ing. Jiří Koutný. Chapter 7 is based on paper, where the part consulting applications of investigated transducer has been written in cooperation with Ing. Petr Horáček. All other results are an outcome of my own research. All sources, references, and literature used or excerpted during elaboration of this thesis are properly cited in complete reference to the due source.

.....
Martin Čermák
June 13, 2012

Acknowledgements

I would like to thank Alexander Meduna, my supervisor, for his inspiration, willingness, and friendly pedagogic and scientific leading. Without his guidance, this work would not have been possible. I also would like to thank Erzsébet Csuhaj-Varjú and György Vaszil for consultations, and my colleagues Jiří Koutný, Zbyněk Křivka, Tomáš Masopust, and Petr Horáček for their inspiration and motivation. Last, but certainly not least, I would like to thank my wife Daniela for her infinite support.

Contents

Part I

Survey of Current State of Knowledge

Chapter 1

Introduction

1.1 Motivation

In the seventh century before Christ, Egyptians believed they are the oldest nation in the world. The former king, Psantek I., wanted to confirm this assumption. The confirmation was based on the idea that children, who cannot learn to speak from adults, will use innate human language. That language was supposed to be Egyptian. For this purpose, Psantek I. took two children from a poor family and let them to grow up in care of a shepherd in an environment, where nobody was allowed to speak with these children. Although the test ultimately failed, it brings us testimony that already in old Egypt, people somehow felt the importance of languages (the whole story you can see in *The story of psychology* by *Morton Hunt*).

In 1921, *Ludwig Wittgenstein* published a philosophical work (*Logisch-philosophische Abhandlung*) containing claim that says "The limits of my language mean the limits of my world". In the computer science, this claim is doubly true. Languages are a way how people express information and ideas in terms of computer science or information technology. In essence, any task or problem, which a computer scientist is able to describe, can be described by a language. The language represents a problem and all sentences belonging into this language are its solutions.

Fact about the limitation by languages led to the birth of a new research area referred to as *theory of formal languages* studying languages from a mathematical point of view. The main initiator was linguist *Noam Chomsky*, who, in the late fifties, introduced hierarchy of formal languages given by four types of language generators. By this work, Noam Chomsky inspired many mathematicians and computer scientists so they began to extend this fundamental hierarchy by adding new models for language definition. Because the theory of formal languages examines the languages from the precise mathematical viewpoint, its results are significant for many areas in information technology. Models, which are studied by the theory, are used in compilers, mathematical linguistics, bioinformatics, especially genetics and simulation of natural biology processes, artificial intelligence, computer graphics, computer networks, and others.

The classical formal language theory uses three approaches to define formal languages.

1. Grammatical approach, where the languages are generated by *grammars*.
2. Automata approach, where the languages are recognized by *automata*.
3. Algebraic approach, where the languages are defined by some *language operations*.

To be more precise, in the grammatical approach, a grammar generates its language by application of *derivation steps* replacing sequences of symbols by other sequences according to its prescribed rules. The symbols can be *terminal* or *nonterminal*, and the sequences of these symbols are called *strings*. In a single derivation step, the grammar, by application of its rule, replaces a part of string by some other string. Any string, which contains no nonterminal symbol and which can be generated from a start nonterminal by application of a sequence of derivation steps, belongs to the language of the grammar. The language of the grammar is represented by the set of such generated strings.

While a grammar generates language, an automaton represents formal algorithm by which the automaton can recognize correctly made sequences of symbols belonging into the language the automaton defines. More specifically, an automaton has string written on its input tape. By application of prescribed rules, it processes the string symbol by symbol and changes its current state to determine whether the string belongs to the language represented by the automaton. If so, the string is accepted by the automaton. The set of all strings accepted by the automaton is the language that the automaton defines.

All models, investigated in the theory of formal languages, are designed to reflect needs of given information technology. Today, when a task distribution, parallel and cooperation process are extremely popular, the main attention is focused on controlled models and systems of models. The necessity of efficient data processing, computer networks, parallel architectures, parallel processing, and nature motivated computing devices justify studying of these approaches in terms of the theory of formal models, where the mechanisms representing these approaches are called *systems of formal models*. The main motivation for investigation of systems lies in a possibility to distribute a task into several smaller tasks, which are easier to solve and easier to describe. These tasks can be solved sequentially or in parallel, and usually, due a communication, the cooperating models are more efficient than the models themselves. The present thesis concentrates on this modern approaches and brings new, or generalized, formal mechanisms and results into the theory. More specifically, this thesis mainly deals with systems of automata and grammars and studies their properties.

This work, at first, continues with studying of sequential grammar systems, known as *cooperating distributed grammar systems* (shortly *CD grammar systems*). These were introduced in the late eighties as a model for blackboard problem solving. The main idea standing behind the CD grammar systems is in a cooperation of well-known simple grammars working on a shared string under a cooperation protocol. Unfortunately, the increased efficiency, obtained from the cooperation, is given by higher degree of ambiguity and non-determinism, what is unpleasant for a practical purpose. This thesis introduces several restrictions limiting the ambiguity or non-determinism, and investigates their effect on the systems.

The further investigation builds on the work of Roman Lukáš and Alexander Meduna, who, in 2006, introduced a new variant of parallel grammar systems named as *multi-generating grammar systems*. In contrast with classic widely studied *parallel communicating grammar systems*, where included grammars are used as supporting elements and the language of a parallel grammar system is generated by one predetermined grammar, these new systems take into account strings from all their grammars. The final strings are obtained from all generated strings by a string operation. This thesis introduces two versions of automata counterpart to these grammar systems and proves their equivalence. Thereafter, the investigated systems are generalized and a fundamental hierarchy of these systems is established. Finally, the thesis suggests systems based on mentioned approaches

as a direct translator of natural languages and parser of languages generated by a specific type of controlled grammars.

1.2 Organization

The thesis is divided into two main parts. Part I, *Survey of Current State of Knowledge*, presents the basic mathematical concepts, used terminology, survey of currently studied systems of formal models, and several known important results relevant to this thesis. Part II, *New Systems of Formal Models and Results*, is the key part of this work, which introduces and investigates generalized and new variants of systems of formal models. The contents of the individual chapters are outlined by the following list.

- Chapter 1, *Introduction*, introduces readers to the field of formal languages and motivates the study of systems of formal models. Furthermore, it describes the structure of the document.
- Chapter 2, *Notation and Basic Definitions*, provides information about notation and mathematical background needed for the understanding of the thesis. Furthermore, it defines various formalisms used later in the text.
- Chapter 3, *Systems of Formal Models*, discuss current knowledge about serial and parallel grammars and automata systems, recalls *CD grammar systems* and origin versions of *multi-generating grammar systems*, and summarizes known results important for this publication.
- Chapter 4, *Restrictions on CD Grammar Systems*, introduces three new restrictions, limiting a degree of non-determinism, and places them on derivations in CD grammar systems with *phrase-structure grammars*. The used grammars themselves can generate any language which can be somehow deterministically generated. Although the restricted CD grammar system is enhanced by a control unit, two restrictions significantly decrease the generative power of the investigated systems.
- Chapter 5, *n-Accepting Restricted Pushdown Automata Systems*, introduces and investigates two new variants of automata systems as an automata counterpart of multi-generating grammar systems introduced by Lukáš and Meduna. The introduced automata systems consist of automata working on acceptation of their own strings. During a computation, the usability of their rules are limited by other automata. In contrast with automata systems, studied in the classic theory of formal languages, the introduced systems accept vector of interdependent strings instead of ordinary strings. Therefore, this chapter brings new terms, such as *n-string* and *n-language*, into the theory.
- Chapter 6, *Classes of n-Languages: Hierarchy and Properties*, generalizes multi-accepting grammar systems and n-accepting restricted automata systems by allowing of usage different types of grammars and automata in one system. On these systems, it establishes fundamental hierarchy of n-languages and studies several closure properties.
- Chapter 7, *Rule-Restricted Automaton-Grammar Transducers*, deals with *language-translation systems*, *transducers* for short, composed of one automaton accepting input

string and one grammar generating a corresponding output string. Both these models are synchronized by their rules which can be used at the same time unit. First of all, this chapter shows how the *leftmost derivation restriction* placed on derivation in the grammar and introduction of rule-priority into the grammar effect the generative and accepting power of considered transducers. After that, it discusses application-related perspectives of the studied systems in linguistics.

- Chapter 8, *Parsing of n-Path-Restricted Tree-Controlled Languages*, suggests an abstraction of 3-accepting restricted automata system as a mechanism for parsing-method based upon a specific type of linguistically motivated controlled grammars, which are named as *n-path-restricted tree-controlled grammars*. In addition, this chapter proves that it is possible to parse these controlled grammars in a polynomial time.

- Finally, Chapter 9, *Conclusion*, summarizes all results obtained in Part II and outlines new areas for further investigation.

Chapter 2

Notation and Basic Definitions

This chapter briefly reviews the required terminology, notation, and fundamental terms from the area of mathematics and formal language theory. Nevertheless, it is assumed that the reader is familiar with elementary algebra and proof techniques (see [56], [59], [69], [80], or [101]).

2.1 Sets and Relations

A set is a collection of elements (objects). About any element, we can unambiguously declare whether it belongs to the set or it does not. The empty set (the set with no element) is denoted by $\emptyset$. $A = \{a, b\}$ means that A is set of elements a and b. The fact that object a belongs to A is denoted by $a \in A$. On the other hand, $c \notin A$ means that c does not belong to A. Note that each element can be in the set only once. In the general mathematics, there are many important sets. For example, $\mathbb{N} = \{1, 2, \ldots\}$ is the set of all natural numbers, $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ is the set of all natural numbers with zero, $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ is the set all integers, etc.

As one can notice, different sets can have different numbers of elements. In relation to the number of elements, we define a cardinality of a set A, $|A|$ for short, as follows:

- $|A| = 0$ iff $A = \emptyset$,
- $|A| = n$ iff $A = \{a_1, a_2, \ldots, a_n\}$ for $n \in \mathbb{N}$,
- $|A| = \infty$ iff number of elements in A is infinity.

If $|A| \in \mathbb{N}_0$, we say that A is finite; otherwise, we say that A is infinite. Elements of A can be specified either by their enumeration, e.g. $A = \{a_1, a_2, \ldots, a_n\}$, where $a_1, a_2, \ldots, a_n$ are the elements of A, or by symbolic notation of the form $A = \{a |\ \pi(a)\}$ where a is an element and $\pi(a)$ is a condition which has to hold, e.g. $A = \{i |\ i \in \mathbb{N} \text{ is odd}\}$ is the set of all odd natural numbers.

For two sets A and B, $A = B$ and $A \neq B$ denote that A is equal to B and A is not equal to B, respectively; $A \subseteq B$ denotes that A is a subset of B; $A \subset B$ denotes that $A \subseteq B$ and $A \neq B$, i.e. A is a proper subset of B.

Sets of sets are almost always called *classes* or *families of sets*. One of this kind of sets are power sets.

Definition 2.1 (Power set)
Let A be a set. Then power set of A, 2^A for short, is the set of all subsets of A. Formally, $2^A = \{X |\ X \subseteq A\}$.

Probably the most usual operations over sets are union, intersection, and difference, whose definition is given next.

Definition 2.2 (Union, intersection and difference of sets)
Let A and B be two sets.

- Union of A and B, denoted by $A \cup B$, is $A \cup B = \{x |\ x \in A \text{ or } x \in B\}$.
- Intersection of A and B, denoted by $A \cap B$, is $A \cap B = \{x |\ x \in A \text{ and } x \in B\}$.
- Difference of A and B, denoted by $A - B$, is $A - B = \{x |\ x \in A \text{ and } x \notin B\}$.

From elements of a set, we can make a list of elements, called an (*ordered*) *sequence*, where the sequence can contain an element more than once and the elements appear in a certain order. Elements in sequences are usually separated by a comma. The length of sequence x, denoted by $|x|$, is the number of elements in x, and if $|x| \in \mathbb{N}_0$, we say that x is finite; otherwise, we say that x is infinite. Finite sequences are also called tuples.

Definition 2.3 (Cartesian product)
Let A and B be two sets. The Cartesian product of A and B, $A \times B$, is defined as $A \times B = \{(a, b) |\ a \in A \text{ and } b \in B\}$.

Note that the order of objects in elements of a cartesian product is important.

Definition 2.4 (Binary relation)
Let A and B be two sets. Binary relation, or relation for short, ρ, from A to B is $\rho \subseteq A \times B$.

Special cases of relations are functions.

Definition 2.5 (Function)
Let A and B be two sets and $\phi \subseteq A \times B$ be a binary relation from A to B. ϕ is a function from A to B, if for all $a \in A$, $|\{b |\ b \in B, (a, b) \in \phi\}| \leq 1$.

In the context of relations, membership of a pair (a, b) in some relation ρ can be written as $(a, b) \in \rho$, $b \in \rho(a)$, or $a\rho b$. Furthermore, if ρ is a function, $b = \rho(a)$ and $\rho(a) = b$ can be used.

As one can see, the previous definitions do not prohibit conformity of sets A and B, i.e. relation $\rho \subseteq A \times A$. In this case, the relation ρ is called a relation on A and, on this relation, several properties can be studied.

Definition 2.6 (Basic properties on relations)
Let $\rho \subseteq A \times A$ be a relation. Then, ρ is

- *reflexive* iff $(a, a) \in \rho$, for all $a \in A$,
- *symmetric* iff $(a, b) \in \rho$ implies that $(b, a) \in \rho$, for all $a, b \in A$,

- *transitive* iff $(a,b),(b,c) \in \rho$ implies that $(a,c) \in \rho$, for all $a,b,c \in \rho$,
- *antisymmetric* iff $(a,b),(b,a) \in \rho$ implies that $a = b$, for all $a,b \in A$.

A reflexive, transitive, and antisymmetric relation ρ on a set A is called a *partial order* and a pair (A,ρ) is called an *ordered set*. Let $K \subset \mathbb{N}_0$ is a final set. Then, $max(K) = k$, where $k \in K$ and for all $h \in K$, $k \geq h$; and $min(K) = l$, where $l \in K$ and for all $h \in K$, $l \leq h$. Furthermore, let $(X, \geq)$ is an ordered set and $A \subseteq X$. We say that $x \in X$ is an upper and lower bound of A, if for all $a \in A$, $a \leq x$ and $x \leq a$, respectively. The least upper bound is called *supremum*, written as $sup(A)$. Conversely, the greatest lower bound is known as *infimum*, denoted $inf(A)$.

Over relation $\rho \subseteq A \times A$, we can study how many times the property of transitivity holds. For this purpose we use k-fold products and closures over the ρ.

Definition 2.7 (k-Fold product and closures)
Let $\rho \subseteq A \times A$ is a relation over A. For some $k \geq 1$, a k-fold product of ρ, ρ^k is recursively defined in the following way:

- $a\rho^0 b$ iff $a = b$
- $a\rho^1 b$ iff $a\rho b$
- $a\rho^n b$ iff $a\rho c$ and $c\rho^{n-1} b$

We define the transitive closure of ρ, ρ^+, as $a\rho^+ b$ iff $a\rho^k b$ for some $k \geq 1$; and the reflexive-transitive closure of ρ, ρ^*, as $a\rho^* b$ iff $a = b$, for $a \in A$ or $a\rho^+ b$.

2.2 Alphabets, Strings, and Languages

An *alphabet* is arbitrary finite non-empty set of elements, which are called *symbols*. A finite sequence of symbols, w, is referred to as *string*. For brevity, we simply juxtapose the symbols and omit all separating commas. If $|x| = 0$, we write $x = \varepsilon$ and call x an *empty string*. Let T be an alphabet and x, y be two strings consisting of symbols from T. We say that x and y are strings over T, and xy the *concatenation* of x and y. The equation $x\varepsilon = \varepsilon x = x$ is an immediate consequence of the definition. Over any string x, we define set of prefixes, $pref(x) = \{y |\ x = y\alpha\}$, set of suffixes, $suf(x) = \{y |\ x = \alpha y\}$, set of substrings, $sub(x) = \{y |\ x = \alpha y \beta\}$, and set of symbols used in x, $alph(x) = \{a |\ a$ is included in $x\}$.

Definition 2.8 (Occurrence of symbols)
Let Σ be an alphabet and $W \subseteq \Sigma$. Then, $occur(w, W)$ denotes the number of occurrences of symbols from W in w.

Definition 2.9 (Reversation of string)
Let $x = a_1 a_2 \dots a_{n-1} a_n$ with $a_i \in \Sigma$ for all $i = 1, \dots, n$. Reverse x, denoted by $(x)^R$, is the string $a_n a_{n-1} \dots a_2 a_1$.

Definition 2.10 (Power of string)
Let x be a string over Σ. Power of x is defined as follows:

- $x^0 = \varepsilon$,

- $x^i = x^i x^{i-1}$ for $x \geq 1$.

Let Σ be an alphabet and Σ^* be a set of all strings over Σ. Language L is defined as any subset of Σ^*, symbolically, $L \subseteq \Sigma^*$, and the *complement* of L, written as $\overline{L}$, is $\overline{L} = \Sigma^* - L$.

Definition 2.11 (Concatenation of languages)
Let L_1 and L_2 be two languages. Concatenation of L_1 and L_2 is the language $L_1 L_2 = \{xy |\ x \in L_1 \text{ and } y \in L_2\}$.

Definition 2.12 (Power of language)
Let L be a language. Power of L is defined in the following way:

- $L^0 = \{\varepsilon\}$
- $L^i = L^i L^{i-1}$ for $i \geq 1$

Definition 2.13 (Iteration and positive iteration of language)
Let L be a language over Σ. Then,

- iteration of language L is defined as $L^* = \bigcup_{i=0}^{\infty} L^i$ and
- positive iteration of language L is defined as $L^+ = \bigcup_{i=1}^{\infty} L^i$ (i.e. $L^+ = L^* - \{\varepsilon\}$)

2.3 Grammars

In this section, we define devices used for generating languages. Such devices are called grammars and play the main role in the theory of formal languages.

Definition 2.14 (Grammar)
A Grammar is a quadruple $G = (N, T, P, S)$, where

- N is an alphabet of nonterminal symbols (nonterminals),
- T is an alphabet of terminal symbols (terminals) and $N \cap T = \emptyset$,
- P is a finite set of rules of the form $\alpha \to \beta$ with $\alpha \in (T \cup N)^* N (T \cup N)^*$ and $\beta \in (T \cup N)^*$, and
- S is the start symbol.

Any string x from $(N \cup T)^*$ is called a *sentential form* of G and if x does not contain nonterminal symbols, then x is called a *sentence*. For two sentential forms $u\alpha v$ and $u\beta v$, if there is a rule $r = \alpha \to \beta \in P$, G can make a derivation step from $u\alpha v$ to $u\beta v$ by rule r, mathematically written as $u\alpha v \Rightarrow u\beta v[r]$, or $u\alpha v \Rightarrow u\beta v$ for short. A rule of the form $\alpha \to \varepsilon$ we call an *epsilon rule*.

Definition 2.15 (Sequence of derivations)
Consider grammar $G = (N, T, P, S)$ and sentential forms χ, χ'.

- $\chi \Rightarrow^0 \chi[\varepsilon]$, or shortly $\chi \Rightarrow^0 \chi$.

- $\chi \Rightarrow^n \chi'[\pi]$, or shortly $\chi \Rightarrow^n \chi'$, for $n \geq 1$, if there exist $n+1$ sentential forms, $\chi_1, \chi_2, \ldots, \chi_n, \chi_{n+1}$, such that $\chi = \chi_1$, $\chi' = \chi_{n+1}$, $\chi_i \Rightarrow \chi_{i+1}[r_i]$ for all $i = 1, 2, \ldots, n$, and $\pi = r_1 r_2 \ldots r_n$.
- $\chi \Rightarrow^* \chi'[\pi]$, or shortly $\chi \Rightarrow^* \chi'$, if $\chi \Rightarrow^n \chi'[\pi]$ for some $n \geq 0$.
- $\chi \Rightarrow^+ \chi'[\pi]$, or shortly $\chi \Rightarrow^+ \chi'$, if $\chi \Rightarrow^n \chi'[\pi]$ for some $n \geq 1$.

Mathematically, $\Rightarrow^k$, $\Rightarrow^*$, and $\Rightarrow^+$ denote k-fold product, reflexive-transitive closure, and transitive closure of $\Rightarrow$, respectively.

Definition 2.16 (Language generated by grammar G)
Let $G = (N, T, P, S)$ be a grammar. Then, the language generated by grammar G, written as $L(G)$, is the set of all sentences of G. Mathematically, $L = \{w |\ S \Rightarrow^* w\}$.

Grammars G_1 and G_2 are said to be equivalent, if and only if they generate the same language, i.e. $L(G_1) = L(G_2)$.

Chomsky Hierarchy of Languages

At the end of 50's, linguist Noam Chomsky has introduced an initial classification of grammars and the classes of languages they generate. Each of these grammars has a differently restricted set of rules (see [22]).

Definition 2.17 (Unrestricted grammar and phrase-structure grammar)
A grammar G with no restriction on rules is called an *unrestricted grammar*. An unrestricted grammar is called a *phrase-structure grammar*, PSG, if all its rules have form $\alpha \to \beta$, where $\alpha \in N^+$ and $\beta \in (N \cup T)^*$. The class of languages that both of these grammars define is referred to as the class of *recursively-enumerable* languages, **RE** for short.

Definition 2.18 (Context-sensitive grammar)
An unrestricted grammar is called a *context sensitive grammar*, CSG, if all its rules have form $\alpha \to \beta$, where $\alpha \in (T \cup N)^* N (T \cup N)^*, \beta \in (N \cup T)^*$, and either $|\alpha| \leq |\beta|$, or $\alpha = S$ and $\beta = \varepsilon$. The class of languages it describes is referred to as the *class of context-sensitive languages*, **CS** for short.

Definition 2.19 (Context-free and linear grammars)
An unrestricted grammar with all rules of the form $A \to \beta$, where $A \in N$ and $\beta \in (N \cup T)^*$ is called a *context-free grammar*, CFG, and furthermore, if $\beta \in T^* N T^*$, we say that the grammar is *linear*, LNG for short. The classes of languages, which can be generated by context-free and linear grammars, are referred to as the *class of context-free languages* (**CF**) and the *class of linear languages* (**LIN**), respectively.

Definition 2.20 (Right-linear grammar)
An unrestricted grammar, where all its rules have form $A \to aB$ with $A \in N$, $a \in T \cup \{\varepsilon\}$, and $B \in N \cup \{\varepsilon\}$, is called a *right-linear grammar*, RLNG, and the class of languages generated by RLNGs is called the class of *regular languages*, **REG** for short.

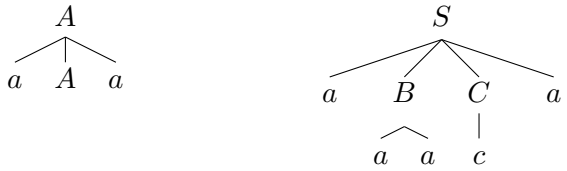


Figure 2.1: Rule tree (left) and derivation tree (right)

For the classes of languages generated by regular, linear, context-free, context-sensitive, and unrestricted grammars, respectively, the following theorem holds (see [80]).

Theorem 2.21 $\mathbf{REG} \subset \mathbf{LIN} \subset \mathbf{CF} \subset \mathbf{CS} \subset \mathbf{RE}$.

Sequences of derivation steps in a context-free grammar can be graphically expressed as a derivation tree representing rules used during a generation.

Definition 2.22 (Derivation tree and ambiguity)
Let $G = (N, T, P, S)$ be a CFG. For $r = A \to x \in P$, the rule tree that represents r is denoted as $A\langle x \rangle$. The *derivation trees* representing derivations in G are defined recursively as follows:

- One-node tree X is the derivation tree corresponding to $X \Rightarrow^0 X$ in G, where $X \in N \cup T$.
- Let d be the derivation representing $A \Rightarrow^* uBv[\rho]$ with $frontier(d) = uBv$, and let $l = B \to z \in P$. The derivation tree representing $A \Rightarrow^* uBv[\rho] \Rightarrow uzv[l]$ is obtained by replacing the $(|u|+1)$st leaf (from the left) in d, B, with the rule tree corresponding to l, $B\langle z \rangle$.
- A *derivation tree* in G is any tree t for which there is a derivation represented by t.

Set of all derivation trees in G with $frontier(x)$ is denoted by ${}_G\blacktriangle(x)$. If there exists $x \in L(G)$ such that $|{}_G\blacktriangle(x)| > 1$, we say that a grammar G is *ambiguous*; otherwise, G is *unambiguous*. Furthermore, a context-free language L is *inherently ambiguous*, if L is generated by no unambiguous grammar.

Figure 2.1 shows an example of rule tree $A\langle aAa \rangle$ (left) and derivation tree representing the derivation $S \Rightarrow aBCa \Rightarrow aaBcaa \Rightarrow aaaca$ (right).

2.4 Automata

In this section, basic devices for recognition of strings belonging to a given language are defined. The notation is based on [80].

Finite Automata

Definition 2.23 (Finite automaton)
A finite automaton, FA, is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of states,
- Σ is an input alphabet,
- R is a finite set of transition rules of the form $pa \to q$ with $p, q \in Q$ and $a \in \Sigma \cup \{\varepsilon\}$,
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is a set of final states.

Let $M = (Q, \Sigma, R, q_0, F)$ be an FA. Configuration of M is any word $w \in Q\Sigma^*$. For two configurations paw and qw, if there exists a rule $r = pa \to q \in R$, then M can make a move from paw to qw by r, written as $paw \vdash qw[r]$, or simply, $qaw \vdash qw$.

Definition 2.24 (Sequence of moves in finite automaton)
Let $M = (Q, \Sigma, R, q_0, F)$ be an FA and χ, χ' be two configuration of M.

- $\chi \vdash^0 \chi'[\varepsilon]$, or simply $\chi \vdash^0 \chi'$, iff $\chi = \chi'$.
- $\chi \vdash^i \chi'[\pi]$, or simply $\chi \vdash^i \chi'$, iff there are $i+1$ configurations, $\chi_1, \ldots, \chi_{i+1}$, such that $\chi = \chi_1$ and $\chi' = \chi_{i+1}$, for all $1 \leq j \leq i$, $\chi_j \vdash \chi_{j+1}[r_j]$, and $\pi = r_1 r_2 \ldots r_i$.
- $\chi \vdash^* \chi'[\pi]$, or simply $\chi \vdash^* \chi'$, iff there is an integer $i \geq 0$ such that $\chi \vdash^i \chi'[\pi]$.
- $\chi \vdash^* \chi'[\pi]$ or simply $\chi \vdash^* \chi'$, iff there is an integer $i \geq 1$ such that $\chi \vdash^i \chi'[\pi]$.

Mathematically, $\vdash^k$, $\vdash^+$, and $\vdash^*$ denote k-fold product, transitive closure, and reflexive-transitive closure of $\vdash$, respectively.

Definition 2.25 (Language defined by finite automaton)
Let $M = (Q, \Sigma, R, q_0, F)$ be an FA. The language of FA M is defined as $L(M) = \{w |\ w \in \Sigma^*, q_0 w \vdash^* f, \text{ and } f \in F\}$.

Pushdown Automata

Definition 2.26 (Pushdown automaton)
Pushdown automaton, PDA, is a septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$, where

- Q is a finite set of states,
- Σ is an input alphabet,
- Γ is a pushdown alphabet,
- R is a finite set of transition rules of the form $zpa \to \gamma q$ with $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $z \in \Gamma$, and $\gamma \in \Gamma^*$,
- $z_0 \in \Gamma$ is the first pushdown symbol,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is a set of final states.

Let $M = (Q, \Sigma, \Gamma, R, q_0, z_0, F)$ be a PDA. A configuration of M is any word $w \in \Gamma^* Q \Sigma^*$. For two configuration of M, $\gamma Apaw$ and γxqw, if there exists a rule $r = Apa \to xq \in R$, then M can make move from $\gamma Aqaw$ to γxqw, written as $\gamma Apaw \vdash \gamma xqw[r]$, or simply $\gamma Apaw \vdash \gamma xqw$.

As usual, we define $\vdash^k$, $\vdash^+$, and $\vdash^*$.

Definition 2.27 (Languages defined by pushdown automaton)
Let $M = (Q, \Sigma, \Gamma, R, q_0, z_0, F)$ be a PDA. The language of M

- *accepting by empty pushdown and final state* is $L(M) = \{w |\ w \in \Sigma^*, z_0 q_0 w \vdash^* f, \text{ and } f \in F\}$;
- *accepting by final state* is $L_f(M) = \{w |\ w \in \Sigma^*, z_0 q_0 w \vdash^* \gamma f, \gamma \in \Gamma^*, \text{ and } f \in F\}$;
- *accepting by empty pushdown* is $L_\varepsilon(M) = \{w |\ w \in \Sigma^*, z_0 q_0 w \vdash^* q, \text{ and } q \in Q\}$.

Definition 2.28 (k-Turn pushdown automaton)
A k-turn PDA is a PDA in which the length of the pushdown tape alternatively increases and decreases at most k-times during any sweep of the pushdown automaton.

Definition 2.29 (Stateless pushdown automaton)
Pushdown automaton M is stateless, SPDA, if it has exactly one state.

Turing Machine

Definition 2.30 (Turing machine)
Turing machine, TM, is a 6-tuple $M = (Q, \Sigma, \Gamma, R, q_0, F)$, where

- Q is a finite set of states,
- Σ is an input alphabet,
- Γ is a tape alphabet, $\square \in \Gamma$, $\Sigma \subseteq \Gamma$,
- R is a finite set of rules of the form $pa \to qbt$, where $p, q \in Q$, $a, b \in \Sigma$, $t \in \{\mathbf{S}, \mathbf{R}, \mathbf{L}\}$,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is a set of final states.

Definition 2.31 (Configuration of Turing machine)
Let $M = (Q, \Sigma, \Gamma, R, q_0, F)$ be a TM. A configuration of M is a string $\chi = xpy$, where $x \in \Gamma^*$, $p \in Q$, $y \in \Gamma^*(\Gamma - \{\square\}) \cup \{\square\}$.

Definition 2.32 (Move in Turing machine)
Let $M = (Q, \Sigma, \Gamma, R, q_0, F)$ be a TM and let χ and χ' be two configurations of M. Then,

- M makes a stationary move form χ to χ' according to r, written as $\chi \vdash_S \chi'[r]$, or simply $\chi \vdash_S \chi'$, if $\chi = xpay$, $\chi' = xqby$, $r = pa \to qb\mathbf{S} \in R$.
- M makes a right move form χ to χ' according to r, written as $\chi \vdash_R \chi'[r]$, or simply $\chi \vdash_R \chi'$, if $\chi = xpay$, $r = pa \to qb\mathbf{R} \in R$ and

(1) $\chi' = xbqy$, $y \neq \varepsilon$ or

(2) $\chi' = xbq\Box$, $y = \varepsilon$

- M makes a left move form χ to χ' according to r, written as $\chi \vdash_L \chi'[r]$, or simply $\chi \vdash_L \chi'$, if

  (1) $\chi = xcpay$, $\chi' = xqcby$, $y \neq \varepsilon$ or $b \neq \Box$, $r = pa \to qb\mathbf{L} \in R$

  (2) $\chi = xcpa$, $\chi' = xqc$, $r = pa \to q\Box\mathbf{L} \in R$

- M makes a move from χ to χ' according to r, written as $\chi \vdash \chi'[r]$, or simply $\chi \vdash \chi'$ if $\chi \vdash_X \chi'[r]$, for some $X \in \{\mathbf{S}, \mathbf{R}, \mathbf{L}\}$.

$\vdash^k$, $\vdash^+$, and $\vdash^*$ are defined as usual.

Definition 2.33 (Language accepted by Turing machine)
Let $M = (Q, \Sigma, \Gamma, R, q_0, F)$ be a TM. The language accepted by M, $L(M)$, is defined as $L(M) = \{w |\ w \in \Sigma^*, q_0 w \vdash^* xfy; x, y \in \Sigma^*, f \in F\} \cup \{\varepsilon |\ q_0\Box \vdash^* xfy; x, y \in \Gamma^*, f \in F\}$.

Definition 2.34 (Time complexity)
Let M be a Turing machine. M runs in $O(n^k)$ time if there exists some constant c such that M runs in at most cn^k moves for any input of length n, for $k, n \geq 0$.

Definition 2.35 (Linear bounded automaton)
Linear bounded automaton, LBA, is a TM that cannot extend its tape by any rule.

For the classes of languages accepted by the defined automata, the following theorem holds (see [80]).

Theorem 2.36 Let $X \in \{\text{LBA, TM, FA, PDA, PDA}_f\text{, PDA}_\varepsilon\text{, SPDA, } k\text{-turn PDA} |\ k \geq 1\}$, where PDA, PDA_f, and PDA_ε denote PDA accepting by final state and empty pusdown, PDA accepting by final state, and PDA accepting by empty pushdown, respectively; and let $\mathscr{L}(X)$ denoted the class of languages accepted by X. Then,

- **REG** $= \mathscr{L}$(FA),
- **LIN** $= \mathscr{L}$(1-turn PDA),
- **CF** $= \mathscr{L}$(PDA) $= \mathscr{L}(\text{PDA}_f) = \mathscr{L}(\text{PDA}_\varepsilon)$,
- **CS** $= \mathscr{L}$(LBA), and
- **RE** $= \mathscr{L}$(TM).

2.5 Regulated Formal Models

There is no doubt about the fact that, over its history, the most studied languages of the Chomsky hierarchy were regular and context-free languages. The main reason probably lies in the simplicity, possibility of natural graphical representation, and practical use of models defining these languages. However, for some practical applications, it was discovered that context-free languages are not sufficient. On the other hand, using of models defining whole

class of context-sensitive languages were inefficient and needlessly strong. Therefore, new ways how to describe languages of such types were looked for. In this section, we briefly recall several approaches, needed for this thesis, increasing the power of existing formal models by their regulation. Furthermore, on Figure 2.2, where $A \to B$ and $A \leftrightarrow B$ denote $A \subset B$ and $A = B$, respectively, we show a hierarchy of languages defined by these models. More details about regulation can be found in books [37, 85] and papers [4, 8, 35, 41, 45, 57, 77, 79, 78, 81, 65, 83].

Definition 2.37 (Programmed grammar)
A *programmed grammar* (see [42]) is a septuple $G = (N, T, S, P, \Lambda, \sigma, \phi)$, where

- N and T are alphabets such that $N \cap T = \emptyset$,
- $S \in N$,
- P is a finite set of productions of the form $A \to \beta$, where $A \in N$ and Λ is a finite set of labels for the productions in P.
- Λ can be interpreted as a function which outputs a production when being given a label,
- σ and ϕ are functions from Λ into the 2^{Λ}.

For $(x, r_1), (y, r_2) \in (N \cup T)^* \times \Lambda$ and $\Lambda(r_1) = (\alpha \to \beta)$, we write $(x, r_1) \Rightarrow (y, r_2)$ iff either $x = x_1 \alpha x_2$, $y = x_1 \beta x_2$ and $r_2 \in \sigma(r_1)$, or $x = y$, and rule $\alpha \to \beta$ is not applicable to x, and $r_2 \in \phi(r_1)$.

The *language of* G is denoted by $L(G)$ and defined as $L(G) = \{w |\ w \in T^*, (S, r_1) \Rightarrow^* (w, r_2)$, for some $r_1, r_2 \in \Lambda\}$. Let $\mathscr{L}(\mathrm{P}, \mathrm{ac})$ denote the class of languages generated by programmed grammars. If $\phi(r) = \emptyset$, for each $r \in \Lambda$, we are led to the class $\mathscr{L}(\mathrm{P})$.

Let G be a programmed grammar. For a derivation $D : S = w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n = w$, $w \in T^*$, of G, $ind(D, G) = max(\{occur(w_i, N) |\ 1 \leq i \leq n\})$, and for $w \in T^*$, $ind(w, G) = min(\{ind(D, G) |\ D$ is a derivation of w in $G\})$. The *index of* G is $ind(G) = sup(\{ind(w, G) |\ w \in L(G)\})$. For a language L in the class $\mathscr{L}(P)$ generated by programmed grammars, $ind(L) = inf(\{ind(G) |\ L(G) = L\}$. For the class $\mathscr{L}(P)$, $\mathscr{L}_n(P) = \{L |\ L \in \mathscr{L}(P)$ and $ind(L) \leq n$, for $n \geq 1\}$ (see [42]).

Definition 2.38 (Matrix grammar)
A *matrix grammar*, MAT, is a pair $H = (G, C)$, where $G = (N, T, P, S)$ is a context-free grammar and $C \subset P^*$ is a finite set of strings denoted as matrices. A sentential form of H is any string from $(N \cup T)^*$. Let u, v be two sentential forms. Then, we say that H makes a derivation step from u to v according to r, written as $u \Rightarrow v[m]$, or simply $u \Rightarrow v$, if $m = p_1 \ldots p_m \in C$ and there are $v_0, \ldots, v_m$, where $v_0 = u$, $v_m = v$, and $v_0 \Rightarrow v_1[p_1] \Rightarrow \ldots \Rightarrow v_m[p_m]$ in G. Let $\Rightarrow^*$ and $\Rightarrow^+$ denote reflexive-transitive and transitive closure of $\Rightarrow$. The language of H is defined as $L(H) = \{w |\ S \Rightarrow w_1[m_1] \Rightarrow \ldots \Rightarrow w_n[m_n], w_n = w, m_1, \ldots, m_n \in C, w \in T^*, n \geq 0\}$. The class of languages generated by matrix grammars is denoted by $\mathscr{L}(MAT)$.

Definition 2.39 (Matrix grammar with appearance checking)
A *matrix grammar with appearance checking*, $\mathrm{MAT}_{\mathrm{ac}}$, is a pair $H = (G, C)$, where $G =$

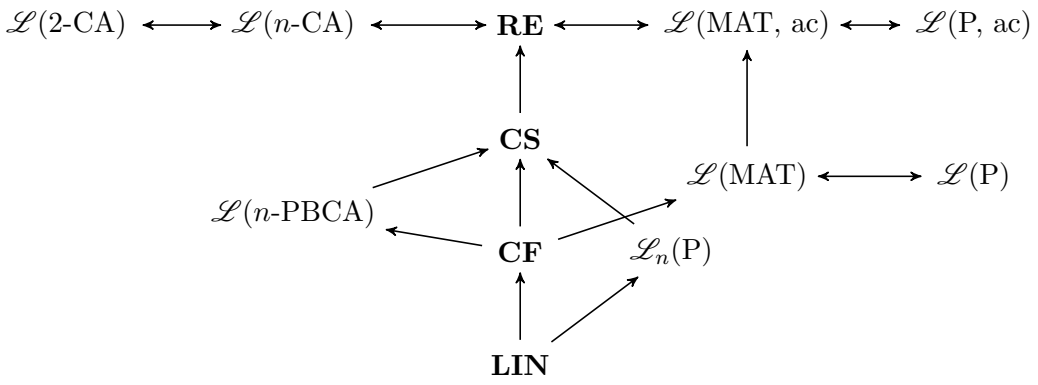


Figure 2.2: Hierarchy of languages

(N, T, P, S) is a context-free grammar and C is a finite set of strings, matrices, of pairs (p, t) with $p \in P$ and $t \in \{-, +\}$. A sentential form of H is any string from $(N \cup T)^*$. Let u, v be two sentential forms. Then, we say that H makes a derivation step from u to v according to m, written as $u \Rightarrow v[m]$, or simply $u \Rightarrow v$, if $m = (p_1, t_1) \dots (p_m, t_m) \in C$ and there are $v_0, \dots, v_m$, where $v_0 = u$, $v_m = v$, and for all $i = 0, \dots, m-1$, either $v_i \Rightarrow v_{i+1}[p_{i+1}]$ in G, or $t_{i+1} \in \{-\}$, $v_i = v_{i+1}$, and p_{i+1} is not applicable on v_i in G. Let $\Rightarrow^*$ and $\Rightarrow^+$ denote reflexive-transitive and transitive closure of $\Rightarrow$. The language of H is defined as $L(H) = \{w |\ S \Rightarrow w_1[m_1] \Rightarrow \dots w_n[m_n], w_n = w, m_1, \dots, m_n \in C, w \in T^*, n \geq 0\}$. The class of languages generated by matrix grammars with appearance checking is denoted by $\mathscr{L}(MAT_{ac})$.

Definition 2.40 (Counter automaton)
A *k-counter automaton*, k-CA, is a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ with k integers $v = (v_1, \dots, v_k)$ in $\mathbb{N}_0^k$ as an additional storage. Transition rules in δ are of the form $pa \to q(t_1, \dots, t_n)$, where $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, and $t_i \in \{-\} \cup \mathbb{Z}$. A configuration of k-CA is any string from $Q\Sigma^*\mathbb{N}_0^k$. Let $\chi_1 = paw(v_1, \dots, v_k)$ and $\chi_2 = qw(v'_1, \dots, v'_k)$ be two configurations of M and $r = pa \to q(t_1, \dots, t_k) \in \delta$, where the following holds: if $t_i \in \mathbb{Z}$, then $v'_i = v_i + t_i$; otherwise, it is satisfied that $v_i, v'_i = 0$. Then, M makes a move from configuration χ_1 to χ_2 according to r, written as $\chi_1 \Rightarrow \chi_2[r]$, or simply $\chi_1 \Rightarrow \chi_2$. $\Rightarrow^*$ and $\Rightarrow^+$ represent reflexive-transitive and transitive closure of $\Rightarrow$, respectively. The language of M is defined as $L(M) = \{w |\ w \in \Sigma^*, q_0w(0, \dots, 0) \Rightarrow^* f(0, \dots, 0), f \in F\}$.

Definition 2.41 (Partially blind k-counter automaton)
A *partially blind k-counter automaton*, k-PBCA, is a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ with k integers $v = (v_1, \dots, v_k)$ in $\mathbb{N}_0^k$ as an additional storage. Transition rules in δ are of the form $pa \to qt$, where $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, and $t \in \mathbb{Z}^k$. As a configuration of k-PBCA we understand any string from $Q\Sigma^*\mathbb{N}_0^k$. Let $\chi_1 = paw(v_1, \dots, v_k)$ and $\chi_2 = qw(v'_1, \dots, v'_k)$ be two configurations of M and $r = pa \to q(t_1, \dots, t_k) \in \delta$, where $(v_1 + t_1, \dots, v_k + t_k) = (v'_1, \dots, v'_k)$. Then, M makes a move from configuration χ_1 to χ_2 according to r, written as $\chi_1 \Rightarrow \chi_2[r]$, or simply $\chi_1 \Rightarrow \chi_2$. $\Rightarrow^*$ and $\Rightarrow^+$ represent reflexive-transitive and transitive closure of $\Rightarrow$, respectively. The language of M is defined as $L(M) = \{w |\ w \in \Sigma^*, q_0w(0, \dots, 0) \Rightarrow^* f(0, \dots, 0), f \in F\}$.

Chapter 3

Systems of Formal Models

Unlike the classic formal languages and automata theory, which studies models accepting or generating language by one automaton or grammar, a modern computer science aims to distribute this computation. The main reasons follow from necessities and possibilities of computer networks, distributed databases, parallel processors, etc., which give us new terms such as *distribution*, *communication*, *concurrency*, and *parallelism*.

A formal system is defined as a set of formal models working together under a specified protocol. Such systems have many advantages. For example, the generative or accepting power of used models usually increases, the (descriptional) complexity of a language decreases, there is a possibility of parallel cooperation, etc.

The main role in the theory of formal systems is played by cooperation protocols and used formal models. This chapter considers four basic classes of systems of formal models: *sequential grammar systems*, *parallel grammar systems*, *sequential automata systems*, and *parallel automata systems*.

3.1 Cooperating Distributed Grammar System

A *cooperating distributed grammar system*, *CD grammar system* for short, was first introduced in [86] related to two-level grammars. Several years later, by investigation of this system in relation with multi-agent systems and blackboard problem solving architectures in [23], studies of CD grammar systems became an intense research area.

A CD grammar system consists of finite number of grammars, called *components*. These symbolize agents. The common sentential form, which the agents sequentially modify according to a mode given by a certain protocol, represents the current state of the problem to be solved. The authors of [23] considered five modes under which agents work: $*$*-mode* – the active agent works as long as it wants; t*-mode* – the active agent works as long as it is able to work; and, $\geq k, \leq k$, and $= k$ *modes* correspond to a time limitation of agents activity, when the active agent has to make i steps for $i \geq k, i \leq k$, and $i = k$, respectively. If a terminal string is generated, the problem is solved (see definitions 3.1 through 3.4 specifying CD grammar systems in terms of formal languages).

Definition 3.1 (Cooperating distributed grammar system)
A *cooperating distributed grammar system*, a *CD grammar system* for short, is an $(n+3)$-tuple $\Gamma = (N, T, S, P_1, \ldots, P_n)$, where N, T are alphabets such that $N \cap T = \emptyset$, $V = N \cup T$, $S \in N$, and $G_i = (N, T, P_i, S), 1 \leq i \leq n$, is a context-free grammar.

Definition 3.2 (Mode of derivation in CD grammar systems)
Let $\Gamma = (N, T, S, P_1, \ldots, P_n)$ be a CD grammar system.

- For every $i = 1, 2, \ldots, n$, *terminating derivation* by ith component, written as $\Rightarrow^t_{P_i}$, is defined as

$$x \Rightarrow^t_{P_i} y \text{ iff } x \Rightarrow^*_{P_i} y \text{ and there is no } z \in \Sigma^* \text{ such that } y \Rightarrow_{P_i} z.$$

- For every $i = 1, 2, \ldots, n$, *k-steps derivation* by ith component, written as $\Rightarrow^{=k}_{P_i}$, is defined as

$$x \Rightarrow^{=k}_{P_i} y \text{ iff there are } x_1, \ldots, x_{k+1} \text{ and for every } j = 1, \ldots, k,\ x_j \Rightarrow_{P_i} x_{j+1}.$$

- For every $i = 1, 2, \ldots, n$, *at most k-steps derivation* by ith component, written as $\Rightarrow^{\leq k}_{P_i}$, is defined as

$$x \Rightarrow^{\leq k}_{P_i} y \text{ iff } x \Rightarrow^{=k'}_{P_i} y \text{ for some } k' \leq k.$$

- For every $i = 1, 2, \ldots, n$, *at least k-steps derivation* by ith component, written as $\Rightarrow^{\geq k}_{P_i}$, is defined as

$$x \Rightarrow^{\geq k}_{P_i} y \text{ iff } x \Rightarrow^{=k'}_{P_i} y \text{ for some } k' \geq k.$$

Definition 3.3 (Language generated by a CD grammar system)
Let $\Gamma = (N, T, S, P_1, \ldots, P_n)$ be a CD grammar system and $f \in D$ be a mode of derivation, where $D = \{*, t\} \cup \{\leq k, = k, \geq k |\ k \in \mathbb{N}\}$. Then, the *language generated by* Γ, $L_f(\Gamma)$, is

$$L_f(\Gamma) = \{\omega \in T^* |\ S \Rightarrow^f_{P_{i_1}} \omega_1 \Rightarrow^f_{P_{i_2}} \ \ldots \Rightarrow^f_{P_{i_m}} \omega_m = \omega, m \geq 1, 1 \leq j \leq m, 1 \leq i_j \leq n\}.$$

Definition 3.4 (Classes of languages generated by CD grammar systems)
The classes of languages generated by CD grammar systems we denote by $\mathscr{L}(\mathrm{CD}, n, f)$, where $f \in \{*, t\} \cup \{= k, \leq k, \geq k |\ k \in \mathbb{N}\}$, and $n \in \mathbb{N} \cup \{\infty\}$ is the number of components.

By the following theorems, we summarize selected basic results regarding the power of CD grammar systems.

Theorem 3.5 $\mathbf{CF} = \mathscr{L}(\mathrm{CD}, 1, t) = \mathscr{L}(\mathrm{CD}, 2, t) \subset \mathscr{L}(\mathrm{CD}, 3, t) = \mathscr{L}(\mathrm{CD}, \infty, t) \subset \mathbf{CS}$.

Theorem 3.6 If $f \in \{= 1, \geq 1, *\} \cup \{\leq k |\ k \geq 1\}$, then $\mathscr{L}(\mathrm{CD}, \infty, f) = \mathbf{CF}$.

Theorem 3.7 $\mathbf{CF} = \mathscr{L}(\mathrm{CD}, 1, f) \subset \mathscr{L}(\mathrm{CD}, 2, f) \subseteq \mathscr{L}(\mathrm{CD}, r, f) \subseteq \mathscr{L}(\mathrm{CD}, \infty, f) \subseteq \mathscr{L}(\mathrm{MAT})$, for all $f \in \{= k, \geq k |\ k \geq 2\}$ and $r \geq 3$.

Other Variants of CD Grammar Systems

The standard CD grammar systems, defined above, use only conditions saying when the enabled component can, or has to, stop working on a sentential form. Selection of component for work is non-deterministic. However, in [26], [24], [33], [6], etc., you can find discussions about many variants of CD grammar systems with several approaches how to select working components.

As a natural extension of CD grammar systems, Mitrana and Păun introduced a *hybrid cooperating distributed grammar systems* in [90] and [98]. In contrast with CD grammar systems, where all components work in the same mode, these systems consists of components working in different modes.

The generative power of CD grammar systems can be increased by teams. This idea was introduced and has been firstly investigated in [63]. Formally, a *CD grammar system with teams* is defined as a tuple $\Gamma = (N, T, S, P_1, \ldots, P_n, R_1, \ldots, R_m)$, where $\Gamma = (N, T, S, P_1, \ldots, P_n)$ is an ordinary CD grammar system and $R_i \subseteq \{P_1, \ldots, P_n\}$ is a team, for all $i = 1, \ldots, m$. At one moment, components from a team simultaneously rewrite corresponding part of a shared sentential form. Precisely, $x \Rightarrow_{R_i} y$ iff $x = x_1 A_1 x_2 \ldots x_s A_s x_{s+1}$, $y = x_1 y_1 x_2 \ldots x_s y_s x_{s+1}$, for all $j = 1, \ldots, s+1$, $x_j \in (N \cup T)^*$, and for all $k = 1, \ldots, s$, $A_k \to y_k \in P \in R_i$. For this one step derivation, k-steps derivation, at most k-steps derivation, at least k-steps derivation, and derivation of any number of steps are defined as usual. Only terminating derivation has three variants, where the active team stops working if the team as a whole cannot perform any further step, no component can apply any of its rules, or at least one component cannot rewrite any symbol of the current sentential form (see [63, 46, 99]).

Besides mentioned variants, many others appear in the literature from the introduction of ordinary CD grammar systems in [86] and [23] up to these days, e.g. CD grammar systems with external storage (see [38, 108, 40, 43]), CD grammar systems consisting of different components (see [110, 51, 74, 28]), hierarchical systems (see [3]), deterministic systems (see [88]), etc.

3.2 Parallel Communicating Grammar Systems

Parallel communicating grammar systems, *PC grammar systems* for short, were introduced in [100]. These systems consist of a finite number of grammars (components), which work on their own sentential form. The components are synchronized and make derivation steps concurrently. During derivation, the communication is performed through special *query* symbols. Whenever at least one component generates a query symbol, all components suspend generating and the grammar system makes a *communication step*—that is, for every component in the system, each occurrence of a query symbol in its sentential form is replaced by the sentential form of the component to which the query symbol is pointing to. One component of the system is called *master* and the language of the master is the language of PC grammar system.

Similarly as in the case of CD grammar systems, the theory of formal languages studies different variants of PC grammar systems, such as

- returning PC grammar systems, where each component that has sent its sentential form to another starts from the start nonterminal;
- centralized PC grammar systems, where only the master can generate query symbols;
- non-synchronized PC grammar systems, where all components include rules of the form $A \to A$ for every nonterminal symbol A;
- PC grammar system with communication by commands, where each component has a control language and in a certain situation all components send their current sentential form to other components owning a control language to which the sentential form belongs to;

- PC grammar systems with languages given by concatenation of all strings over terminal symbols after the end of generation;
- PC grammar systems using query strings instead of query symbols, where the communication steps is done after at least one component generates a query string pointing to another component;
- PC grammar systems, where components make different number of steps;
- and many others, see [26], [25], [60], [104], etc.

Probably the most important features of parallel communicating grammar systems are *communication protocol* and *types of used components* together with the way they work. Further important feature is *synchronization*. Habitually, the synchronization of components is done by an universal clock (each component make one derivation step in each time unit), but others synchronization mechanisms are also studied (see [26], [97], [30]). Two of the most natural variants are synchronization by rules, which can be applied simultaneously, and synchronization by nonterminals, which can be rewritten at the same time unit. Both these approaches Lukáš and Meduna used in [82] and [70], where they have investigated multi-generative grammar systems.

Multi-Generative Grammar Systems

Multi-generative grammar systems are a variant of parallel communicating grammar systems, where the communication is provided only by synchronization. This synchronization restricts either rules, which can be used for each common derivation step, or nonterminals, which can be simultaneously rewritten. For successful generation, all components have to produce sentences at the same time. Lukáš and Meduna have considered three types of languages defined by multi-generative grammar systems—languages consisting of all sentences produced by all components, languages consisting of concatenations of all sentences produced by all components, and languages consisting of sentences produced by the first component of a multi-generating grammar system.

Definition 3.8 (Multi-generative nonterminal synchronized grammar system)
A *multi-generative nonterminal synchronized grammar system*, GN, is an $(n+1)$-tuple

$$\Gamma = (G_1, \ldots, G_n, Q), \text{ where}$$

- $G_i = (N_i, T_i, P_i, S_i)$ is a context-free grammar, for all $i = 1, \ldots, n$,
- Q is a finite set of control n-tuples of the form $(A_1, \ldots, A_n)$, where $A_i \in N_i$ for all $i = 1, \ldots, n$.

Definition 3.9 (Multi-generative rule synchronized grammar system)
A *multi-generative rule synchronized grammar system*, GR, is an $(n+1)$-tuple

$$\Gamma = (G_1, \ldots, G_n, Q), \text{ where}$$

- $G_i = (N_i, T_i, P_i, S_i)$ is a context-free grammar for all $i = 1, \ldots, n$,
- Q is a finite set of control n-tuples of the form $(r_1, \ldots, r_n)$, where $r_i \in P_i$ for all $i = 1, \ldots, n$.

Definition 3.10 (Multi-sentential form)
Let $\Gamma = (G_1, \ldots, G_n, Q)$ be either GN or GR. Then a *multi-sentential form* is an n-tuple $\chi = (x_1, \ldots, x_n)$, where $x_i \in (T_i \cup N_i)^*$ for all $i = 1, \ldots, n$.

Definition 3.11 (Derivation step in GN)
Let $\Gamma = (G_1, \ldots, G_n, Q)$ be a GN, let $\chi = (u_1 A_1 v_1, \ldots, u_n A_n v_n)$, $\chi' = (u_1 x_1 v_1, \ldots, u_n x_n v_n)$, be two multi-sentential forms, where $A_i \in N_i, u_i, v_i, x_i \in (N_i \cup T_i)^*$ for all $i = 1, \ldots, n$. Let $A_i \to x_i \in P_i$ for all $i = 1, \ldots, n$, and $(A_1, \ldots, A_n) \in Q$. Then, χ *directly derives* χ', written as $\chi \Rightarrow \chi'$.

Definition 3.12 (Derivation step in GR)
Let $\Gamma = (G_1, \ldots, G_n, Q)$ be a GR, let $\chi = (u_1 A_1 v_1, \ldots, u_n A_n v_n)$, $\chi' = (u_1 x_1 v_1, \ldots, u_n x_n v_n)$, be two multi-sentential forms, where $A_i \in N_i, u_i, v_i, x_i \in (N_i \cup T_i)^*$ for all $i = 1, \ldots, n$. Let $r_i = A_i \to x_i \in P_i$ for all $i = 1, \ldots, n$, and $(r_1, \ldots, r_n) \in Q$. Then, χ *directly derives* χ', written as $\chi \Rightarrow \chi'$.

Definition 3.13 (Multi-language generated by GN and GR)
Let $\Gamma = (G_1, \ldots, G_n, Q)$ be either GN or GR. Then, the *n-language generated by* Γ, n-$L(\Gamma)$, is

$$n\text{-}L(\Gamma) = \{(w_1, \ldots, w_n) |\ (S_1, \ldots, S_n) \Rightarrow^* (w_1, \ldots, w_n),\ w_i \in T_i^* \text{ for all } i = 1, \ldots, n\}.$$

Definition 3.14 (Languages of GN and GR)
Let $\Gamma = (G_1, \ldots, G_n, Q)$ be either GN or GR. Then, we define

- *the language generated by* Γ *in union mode*, $L_\cup(\Gamma)$), as

$$L_\cup(\Gamma) = \bigcup_{i=1}^{n} \{w_i |\ (w_1, \ldots, w_n) \in n\text{-}L(\Gamma)\},$$

- *the language generated by* Γ *in concatenation mode*, $L_\bullet(\Gamma)$, as

$$L_\bullet(\Gamma) = \{w_1 \ldots w_n |\ (w_1, \ldots, w_n) \in n\text{-}L(\Gamma)\},$$

- *the language generated by* Γ *in first-component-selection mode*, $L_1(\Gamma)$), as

$$L_1(\Gamma) = \{w_1 |\ (w_1, \ldots, w_n) \in n\text{-}L(\Gamma)\}.$$

Definition 3.15 (Canonical multi-generative grammar systems)
We say that GN and GR are *canonical* if all the components of GN and GR can make only the leftmost derivations, i.e. only the leftmost nonterminal can be rewritten in each sentential form. Canonical multi-generative rule synchronized grammar systems and canonical multi-generative nonterminal synchronized grammar systems are denoted by CGR and CGN, respectively.

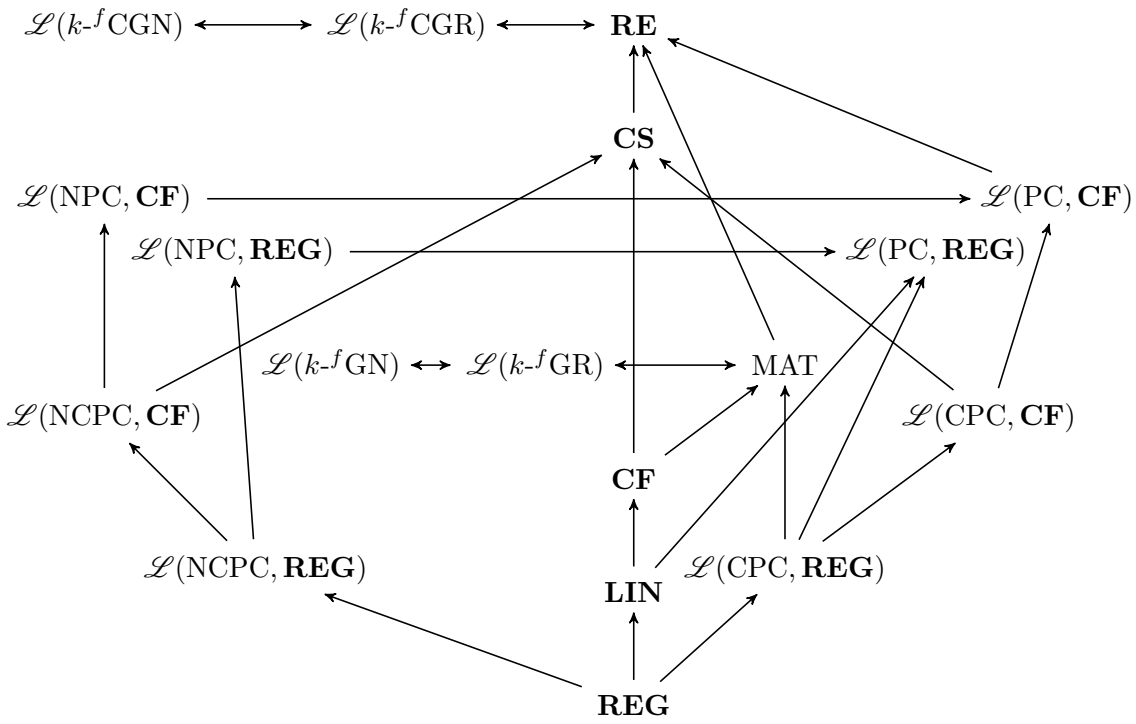


Figure 3.1: Hierarchy of languages (it is considered that $k \geq 2$ and $f \in \{\cup, \bullet, 1\}$)

Convention 3.16 If there is an attention on the number of components in a multi-generative grammar system, we use terms n-generative grammar system, n-GN, n-GR, n-CGR, n-CGN, sentential n-form, and n-language, for some positive integer n, rather than multi-generative grammar system, GN, GR, CGN, CGR, multi-sentential form, and multi-language, respectively.

Definition 3.17 (Classes of n-GN, n-CGN, n-GR, and n-CGR n-languages)
Let $X \in \{\text{GN}, \text{CGN}, \text{GR}, \text{CGR}\}$. The class of n-languages of n-X, $\mathscr{L}({}_nX)$, is defined as $\mathscr{L}({}_nX) = \{n\text{-}L |\ n\text{-}L \text{ is an } n\text{-language generated by } n\text{-}X\}$.

Definition 3.18 (Classes of n-GN, n-GR, n-CGN, and n-CGR languages)
Let $X \in \{\text{GN}, \text{CGN}, \text{GR}, \text{CGR}\}$ and $f \in \{\cup, \bullet, 1\}$. The class of languages generated by an n-X in f-mode, $\mathscr{L}(n\text{-}^fX)$, is defined as $\mathscr{L}(n\text{-}^fX) = \{L |\ L \text{ is a language generated in the } f\text{-mode by } n\text{-}X\}$.

Let's say that $\mathscr{L}(X\text{PC}, Y)$ with $X \in \{\varepsilon, \text{C}, \text{N}, \text{NC}\}$ and $Y \in \{\mathbf{REG}, \mathbf{CF}\}$ denote the classes of languages generated by XPC grammar systems with unlimited number of components, where N and C before PC say that PC grammar systems are non-returning and centralized, respectively, and furthermore, $Y = \mathbf{REG}$ and $Y = \mathbf{CF}$ mean that the components of the systems are regular grammars and context-free grammars, respectively. In Figure 3.1 one can see several important relationships between the classes of languages defined by parallel grammar systems. The results are taken from [104], [82], and [70].

3.3 Automata Systems

Automata and automata systems are used in many areas of computer science. One can find them in computer networks, formal analysis and verifications, pattern matching, parallel computers, DNA computing, artificial intelligence, etc. In this section, we briefly outline several important cooperating models in terms of theory of formal languages.

A *multiprocessor automaton* is based upon finite automata, called *processors*. These processors are coordinated by a central arbiter determining which processor is to become active or inactive (an inactive processor preserves its configuration) at a given step. The only information that the arbiter has for decision about automata activities are the current state of each automaton and number of steps proceeding by active automata (see [10]).

Similar system to the multiprocessor automata allows to share information about current states of processors. In such system, each automaton makes a move with respect of the current input symbol and states of all automata. If we reduce all these automata to one with multiple reading head, we make equivalent model called *multi-head automaton* (see [102]).

In relation to automata, [36] has firstly investigated an idea to apply strategies akin to those that cooperating distributed grammar systems use. For this purpose, Mitrana and Dassow introduced special types of *multi-stack pushdown automata*. However, they do not form the automata counterpart of CD grammar systems. This was introduced and has been studied in [29] under the name *distributed pushdown automata system*.

A *distributed pushdown automata system* contains a shared one-way input tape, one reading head, and finite number of components having their own pushdown and finite sets of states. At any moment, only one component is active. According to a cooperation protocol, the active component must perform k, at least k, at most k, for $k \geq 1$, or it must work as long as it is able to perform a move.

Parallel communicating automata systems have been investigated both with finite automata and pushdown automata as components. The first variant, *parallel communicating finite automata system*, was introduced by Martín-Vide, Mateescu, and Mitrana in [72]. Finite automata in such systems work independently but on a request, they communicate by states to each other. More precisely, the finite automata are entitled to request the current state of any other component. In [72] has been discussed several variants, where contacted automaton after communication is/is not returned to the initial state (*returning/non-returning parallel communication automata systems*), or, only one automaton has/all automata have the right to ask the current state from the others (*centralized/non-centralized parallel communication automata systems*). By application of these strategies on pushdown automata, the investigation was continued in [27], where the attention is focused especially on communication by stacks, i.e. on request an asked automaton send the content of its pushdown to requesting automata which push it on their pushdowns).

In the same way as in the case of grammar systems discussed above, you can find many other variants of automata systems in the literature (see [106, 107, 93, 87, 94, 32]). Generally, we can say that the theory of formal languages reflects the approaches used in grammar systems into automata systems and studies the accepting power of given systems in relation to component represented by automata working in many different ways.

Part II

New Systems of Formal Models and Results

Chapter 4

Restrictions on CD Grammar Systems

Formal language theory has investigated various left restrictions placed on derivations in grammars working in a context-free way. In ordinary context-free grammars, these restrictions have no effect on the generative power. In terms of regulated context-free grammars, the formal language theory has introduced a broad variety of leftmost derivation restrictions, many of which change their generative power (see [4, 8, 35, 37, 41, 44, 50, 57, 75, 76, 78, 79]). In terms of grammars working in a context-sensitive way, significantly fewer left derivation restrictions have been discussed in the language theory. Indirectly, this theory has placed some restrictions on the productions so the resulting grammars make only derivations in a left way (see [4, 8]). This theory also directly restricted derivations in the strictly leftmost way so the rewritten symbols are preceded only by terminals in the sentential form during every derivation step (see [75]). In essence, all these restrictions result in decreasing the generative power to the power of context-free grammars (see page 198 in [105]). This chapter generalizes the discussion of this topic by investigating regularly controlled cooperating distributed grammar systems (see Chapter 4 in [105]) whose components are phrase-structure grammars restricted in some new ways.

4.1 Three Restrictions on Derivations in Grammar Systems

First of all, we define the restrictions on derivations in phrase-structure grammars. In the following, we consider V as the total alphabet of $G = (N, T, P, S)$, i.e. $V = N \cup T$.

Definition 4.1 (Derivation-restriction of type I)
Let $l \in \mathbb{N}$ and let $G = (N, T, P, S)$ be a phrase-structure grammar. If there is $\alpha \to \beta \in P$, $u = x_0\alpha x_1$, and $v = x_0\beta x_1$, where $x_0 \in T^*N^*$, $x_1 \in V^*$, and $\text{occur}(x_0\alpha, N) \leq l$, then $u \; {}_l\Leftrightarrow \; v \; [\alpha \to \beta]$ in G, or simply $u \; {}_l\Leftrightarrow \; v$.

The k-fold product of ${}_l\Leftrightarrow$, where $k \geq 0$, is denoted by ${}_l\Leftrightarrow^k$. The reflexive-transitive closure and transitive closure of ${}_l\Leftrightarrow$ are denoted by ${}_l\Leftrightarrow^*$ and ${}_l\Leftrightarrow^+$, respectively.

The restriction from Definition 4.1 requires that a rule of the form $\alpha \to \beta$ can be used only if α is within the first l nonterminals for some $l \in \mathbb{N}$. For instance, if there is a sentential form $abABCDEFKL$ and $l = 4$, then $BC \to x_1$ is applicable on the sentential form while $DE \to x_2$ is not.

Definition 4.2 (Derivation-restrictions of type II and III)
Let $m, h \in \mathbb{N}$. $W(m)$ denotes the set of all strings $x \in V^*$ satisfying 1 given next. $W(m,h)$ denotes the set of all strings $x \in V^*$ satisfying 1 and 2.

1. $x \in (T^*N^*)^m T^*$;
2. $(y \in \mathrm{sub}(x)$ and $|y| > h)$ implies $\mathrm{alph}(y) \cap T \neq \emptyset$.

Let $u \in V^*N^+V^*$, $v \in V^*$, and $u \Rightarrow v$. Then, $u \,{}^{h}_{m}\!\!\Rightarrow v$ in G, if $u, v \in W(m,h)$; and if $u, v \in W(m)$, $u \,{}_{m}\!\!\Rightarrow v$ in G.

The k-fold product of ${}^{h}_{m}\!\!\Rightarrow$ and ${}_{m}\!\!\Rightarrow$ are denoted by ${}^{h}_{m}\!\!\Rightarrow^{k}$ and ${}_{m}\!\!\Rightarrow^{k}$, respectively, where $k \geq 0$. The reflexive-transitive closure and transitive closure of ${}^{h}_{m}\!\!\Rightarrow$ are denoted by ${}^{h}_{m}\!\!\Rightarrow^{*}$ and ${}^{h}_{m}\!\!\Rightarrow^{+}$, respectively; and the reflexive-transitive closure and transitive closure of ${}^{h}_{m}\!\!\Rightarrow$ and ${}_{m}\!\!\Rightarrow$ are denoted by ${}_{m}\!\!\Rightarrow^{*}$ and ${}_{m}\!\!\Rightarrow^{+}$, respectively.

Informally, the second restriction permits only such derivations, where all sentential forms have m or less blocks of nonterminals, for $m \in \mathbb{N}$, e.g. if $m = 2$, $abABCaDEFKL \Rightarrow abABaDEFKa[L \to a]$, but $E \to a$ is not applicable on $abABCaDEFKL$. The third restriction extends the second restriction and says how many blocks of nonterminals with limited length can be in every sentential form. That is, if $m, h = 5$, $L \to LLL$ cannot be used for the derivation step from $abABCaDEFKL$.

As already stated, these defined restrictions are investigated in terms of CD grammar systems, which are controlled by regular languages. See the following convention and definitions.

Convention 4.3 Let $\Gamma = (N, T, S, P_1, \ldots, P_n)$ be a CD grammar system with phrase-structure grammars as its components and $V = N \cup T$ be the total alphabet of Γ. Furthermore, let $u \in V^*N^+V^*$, $v \in V^*$, $k \geq 0$. Then, we write $u \,{}_{l}\!\!\Rightarrow^{k}_{P_i} v$, $u \,{}^{h}_{m}\!\!\Rightarrow^{k}_{P_i} v$, and $u \,{}_{m}\!\!\Rightarrow^{k}_{P_i} v$ to denote that $u \,{}_{l}\!\!\Rightarrow^{k} v$, $u \,{}^{h}_{m}\!\!\Rightarrow^{k} v$, and $u \,{}_{m}\!\!\Rightarrow^{k} v$, respectively, was performed by P_i. Analogously, we write $u \,{}_{l}\!\!\Rightarrow^{*}_{P_i} v$, $u \,{}^{h}_{m}\!\!\Rightarrow^{*}_{P_i} v$, $u \,{}_{m}\!\!\Rightarrow^{*}_{P_i} v$, $u \,{}_{l}\!\!\Rightarrow^{+}_{P_i} v$, $u \,{}^{h}_{m}\!\!\Rightarrow^{+}_{P_i} v$, $u \,{}_{m}\!\!\Rightarrow^{+}_{P_i} v$, $u \,{}^{h}_{m}\!\!\Rightarrow^{t}_{P_i} v$, and $u \,{}_{m}\!\!\Rightarrow^{t}_{P_i} v$.

Definition 4.4 (Languages of restricted and controlled CD grammar system based upon phrase-structure grammars)
Let $\Gamma = (N, T, S, P_1, \ldots, P_n)$ be a CD grammar system with phrase-structure grammars as its component and C be a control language. Then,

$$
\begin{aligned}
{}_{\mathrm{l}}L^{C}(\Gamma) &= \{w \in T^* |\ S \,{}_{l}\!\!\Rightarrow^{t}_{P_{i_1}} w_1 \,{}_{l}\!\!\Rightarrow^{t}_{P_{i_2}} \cdots {}_{l}\!\!\Rightarrow^{t}_{P_{i_p}} w_p = w, \\
&\quad p \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq p,\ i_1 i_2 \ldots i_p \in C\} \\
{}_{\mathrm{N}}L^{C}(\Gamma, m, h) &= \{w \in T^* |\ S \,{}^{h}_{m}\!\!\Rightarrow^{t}_{P_{i_1}} w_1 \,{}^{h}_{m}\!\!\Rightarrow^{t}_{P_{i_2}} \cdots {}^{h}_{m}\!\!\Rightarrow^{t}_{P_{i_p}} w_p = w, \\
&\quad p \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq p,\ i_1 i_2 \ldots i_p \in C\} \\
{}_{\mathrm{N}}L^{C}(\Gamma, m) &= \{w \in T^* |\ S \,{}_{m}\!\!\Rightarrow^{t}_{P_{i_1}} w_1 \,{}_{m}\!\!\Rightarrow^{t}_{P_{i_2}} \cdots {}_{m}\!\!\Rightarrow^{t}_{P_{i_p}} w_p = w, \\
&\quad p \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq p,\ i_1 i_2 \ldots i_p \in C\}.
\end{aligned}
$$

Definition 4.5 (Classes of languages)
Let $l, m, h \in \mathbb{N}$ and let $\Gamma = (N, T, S, P_1, \ldots, P_n)$ be a CD grammar system with phrase-structure grammars. We define the following classes of languages.

$$\mathscr{L}({}_{\mathrm{l}}\mathrm{CD}^{\mathbf{REG}}) = \{{}_{\mathrm{l}}L^{C}(\Gamma) |\ C \in \mathbf{REG}\}$$

$$\mathscr{L}({}_{\mathrm{N}}\mathrm{CD}^{\mathbf{REG}}(m,h)) = \{{}_{\mathrm{N}}L^{C}(\Gamma,m,h)|\ C \in \mathbf{REG}\}$$
$$\mathscr{L}({}_{\mathrm{N}}\mathrm{CD}^{\mathbf{REG}}(m)) = \{{}_{\mathrm{N}}L^{C}(\Gamma,m)|\ C \in \mathbf{REG}\}$$

4.2 Generative Power of Restricted Grammar Systems

At the beginning of this section, we show that for any language L from $\mathscr{L}({}_{\mathrm{l}}\mathrm{CD}^{\mathbf{REG}})$ there exists a pushdown automaton M such that $L = L(M)$ and for every pushdown automaton M', the language $L(M')$ is in $\mathscr{L}({}_{\mathrm{l}}\mathrm{CD}^{\mathbf{REG}})$.

Lemma 4.6 For every CD grammar system $\Gamma = (N, T, S, P_1, \ldots, P_n)$, every finite automaton $\bar{M}$ and every $l \in \mathbb{N}$, there is a pushdown automaton M such that $L(M) = {}_{\mathrm{l}}L^{L(\bar{M})}(\Gamma)$.

Proof of Lemma 4.6. Let $\Gamma = (N, T, S, P_1, \ldots, P_n)$, $\bar{M} = (\bar{Q}, \bar{\Sigma}, \bar{\delta}, \bar{s_0}, \bar{F})$, $l \geq 1$, and let $N_{left}(P) = \{\alpha|\ \alpha \to \beta \in P\}$. Consider the following pushdown automaton $M = (\{s_0, f\} \cup \{\langle \gamma, s, \bar{s}, i\rangle|\ \gamma \in N^*, |\gamma| \leq l, s \in \{q, r, e\}, \bar{s} \in \bar{Q}, i \in \{1, \ldots, n\}\}, T, T \cup N \cup \{Z\}, \delta, s_0, Z, \{f\})$, where $Z \notin T \cup N$ and δ contains rules of the following form.

1.	$s_0 \to \langle S, q, \bar{s_0}, i\rangle$	$i \in \{1, \ldots, n\}$
2.	$\langle \gamma, q, s, i\rangle \to (\gamma')^R\langle \varepsilon, r, s, i\rangle$	if $\gamma \in N^*, \lvert\gamma\rvert \leq l$ s.t. $\gamma\ {}_l\Leftrightarrow^1_{P_i}\ \gamma'$
3.	$a\langle \varepsilon, r, s, i\rangle a \to \langle \varepsilon, r, s, i\rangle$	$i \in \{1, \ldots, n\}$
4.	$Z\langle \varepsilon, r, s, i\rangle \to f$	if $si \to s' \in \bar{\delta}$ for some $s' \in \bar{F}$
5.	$A\langle A_1 \ldots A_o, r, s, i\rangle \to \langle A_1 \ldots A_o A, r, s, i\rangle$	if $A \in N, o < l$
6.	$\langle A_1 \ldots A_l, r, s, i\rangle \to \langle A_1 \ldots A_l, e, s, i\rangle$	$i \in \{1, \ldots, n\}$
7.	$a\langle A_1 \ldots A_o, r, s, i\rangle \to a\langle A_1 \ldots A_o, e, s, i\rangle$	if $o < l, a \in T$
8.	$Z\langle A_1 \ldots A_o, r, s, i\rangle \to Z\langle A_1 \ldots A_o, e, s, i\rangle$	if $o < l$
9.	$\langle \gamma, e, s, i\rangle \to \langle \gamma, q, s', i'\rangle$	if $sub(\gamma) \cap N_{\mathrm{left}}(P_i) = \emptyset$, $si \to s' \in \bar{\delta}$, $i' \in \{1, \ldots, n\}$
10.	$\langle \gamma, e, s, i\rangle \to \langle \gamma, q, s, i\rangle$	if $sub(\gamma) \cap N_{\mathrm{left}}(P_i) \neq \emptyset$

Proof of $L(M) = {}_{\mathrm{l}}L_f^{L(\bar{M})}(\Gamma)$ is given by the following claims.

First, we prove the following claim.

Claim 4.7 If $Z(\zeta)^R\langle \gamma, q, s, i_1\rangle w \vdash^* f$ in M, then $\gamma\zeta\ {}_l\Leftrightarrow^t_{P_{i_1}}\ w_1\ {}_l\Leftrightarrow^t_{P_{i_2}}\ w_2 \ldots {}_l\Leftrightarrow^t_{P_{i_p}}\ w_p = w$, $p \geq 0$ in Γ and $i_1 \ldots i_p \in suf(L(\bar{M}))$.

Proof of Claim 4.7. By induction on the number of rules constructed in 2 used in a sequence of moves.

Basis: Only one rule constructed in 2 is used. Then,

$$Z(\zeta)^R\langle \gamma, q, s, i_0\rangle w \vdash Z(\gamma'\zeta)^R\langle \varepsilon, r, s, i_0\rangle w \vdash^{|\gamma'\zeta|} Z\langle \varepsilon, r, s, i_0\rangle \vdash f,$$

where $\gamma = \gamma_0\alpha\gamma_1$, $\gamma' = \gamma_0\beta\gamma_1$, $\alpha \to \beta \in P_{i_0}$, $\gamma \in N^+, \gamma'\zeta \in T^*$. Therefore, $\gamma_0 = \gamma_1 = \varepsilon$, $\gamma'\zeta = w$. Then,

$$\gamma\zeta\ {}_l\Leftrightarrow_{P_{i_0}}\ w.$$

By a rule constructed in 4 $i_0 \in suf(L(\bar{M}))$ and the basis holds.

Induction hypothesis: Suppose that the claim holds for all sequences of moves containing no more than j rules constructed in 2.

Induction step: Consider a sequence of moves containing $j+1$ rules constructed in 2:

$$\begin{array}{lll} & Z(\zeta)^R\langle\gamma,q,s,i_0\rangle w & \\ \vdash & Z(\gamma'\zeta)^R\langle\varepsilon,r,s,i_0\rangle w & \text{(by a prod. constructed in 2)} \\ \vdash^* & Z(\zeta')^R\langle\varepsilon,r,s,i_0\rangle w' & \text{(by prod. constructed in 3)} \\ \vdash^* & Z(\zeta'')^R\langle\gamma'',r,s,i_0\rangle w' & \text{(by prod. constructed in 5)} \\ \vdash & Z(\zeta'))^R\langle\gamma'',e,s,i_0\rangle w' & \text{(by a prod. constructed in 6, 7 or 8)} \\ \vdash & Z(\zeta'')^R\langle\gamma'',q,s',i_1\rangle w' & \text{(by a prod. constructed in 9 or 10)} \\ \vdash^* & f, & \end{array}$$

where $\gamma=\gamma_0\alpha\gamma_1$, $\gamma'=\gamma_0\beta\gamma_1$, $\alpha\to\beta\in P_{i_0}$, $\zeta'\in NV^*\cup\{\varepsilon\}$, $v\in T^*$, $\gamma'\zeta=v\zeta'$, $vw'=w$, $\zeta'=\gamma''\zeta''$, either $si\to s'$ or $s=s'$, and one of the following holds:

- $|\gamma''|=l$, or
- $|\gamma''|<l$ and $\zeta''\in TV^*\cup\{\varepsilon\}$.

Then, by the rule $\alpha\to\beta$,

$$\gamma_0\alpha\gamma_1\zeta\ {}_l\Leftrightarrow_{P_{i_0}}\ \gamma_0\beta\gamma_1\zeta,$$

where $|\gamma_0\alpha\gamma_1|\le l$, $\gamma_0\beta\gamma_1\zeta=v\zeta'=v\gamma''\zeta''$ and, by the induction hypothesis,

$$v\gamma''\zeta''\ {}_l\Leftrightarrow^t_{P_{i_1}}\ vw_1\ {}_l\Leftrightarrow^t_{P_{i_2}}\ vw_2\ldots\ {}_l\Leftrightarrow^t_{P_{i_p}}\ vw_p=vw \text{ and}$$

$$i_1\ldots i_p\in suf(L(\bar{M})),$$

where $p\ge 0$.

If a rule constructed in 9 was used, $\gamma_0\alpha\gamma_1\zeta\ {}_l\Leftrightarrow^t_{P_{i_0}}\ \gamma_0\beta\gamma_1\zeta$ is a t-mode derivation, $i_0i_1i_2\ldots i_p\in suf(L(\bar{M}))$ and the claim holds.

If a rule constructed in 10 was used, $i_0=i_1$, $\gamma_0\alpha\gamma_1\zeta\ {}_l\Leftrightarrow^t_{P_{i_1}}\ vw_1$, $i_1i_2\ldots i_p\in suf(L(\bar{M}))$ and the claim holds. □

Let $Zs_0w\vdash Z\langle S,q,\bar{s_0},i_1\rangle w$, by a rule constructed in 1. By Claim 4.7, $Z\langle S,q,\bar{s_0},i_1\rangle w\vdash^* f$ implies $S\ {}_l\Leftrightarrow^t_{P_{i_1}}\ w_1\ {}_l\Leftrightarrow^t_{P_{i_2}}\ w_2\ldots\ {}_l\Leftrightarrow^t_{P_{i_p}}\ w_p=w, p\ge 0$ in Γ and $i_1\ldots i_p\in suf(L(\bar{M}))$.

Claim 4.8 If $\tau_0x_0\ {}_l\Leftrightarrow^t_{P_{i_1}}\ w_1\ {}_l\Leftrightarrow^t_{P_{i_2}}\ w_2\ldots\ {}_l\Leftrightarrow^t_{P_{i_p}}\ w_p=w$ in Γ, where $p\ge 0$, $\tau_0\in N^+$, $x_0\in TV^*\cup\{\varepsilon\}$, $w_i\in V^*$, $i\in\{1,\ldots,p-1\}$, $w_p\in T^*$ and $i_1\ldots i_p\in suf(L(\bar{M}))$, then $Z(\tau_0^2x_0)^R\langle\tau_0^1,q,s,i_1\rangle w\vdash^* f$, for some $s\in\bar{Q}$, where $\tau_0=\tau_0^1\tau_0^2$, $|\tau_0|\le l$ implies $\tau_0^1=\tau_0$, and $|\tau_0|>l$ implies $|\tau_0^1|=l$.

Proof of Claim 4.8. By induction on the length of derivations.

Basis: Let $\tau_0x_0\ {}_l\Leftrightarrow_{P_{i_0}}\ \tau_0'x_0=w$, where $\tau_0^1=\gamma_0\alpha\gamma_1$, $\tau_0'=\gamma_0\beta\gamma_1\tau_0^2$, $\alpha\to\beta\in P_{i_0}$, $\tau_0'x_0\in {}_1L_f^{L(\bar{M})}(\Gamma)$. Therefore, $\gamma_0=\gamma_1=\tau_0^2=\varepsilon$ and for some $s\in\bar{Q}$, $si_0\to s'\in\bar{\delta}$, where $s'\in\bar{F}$. M simulates this derivation step in the following way:

$$\begin{array}{lll} & Z(\tau_0^2x_0)^R\langle\tau_0^1,q,s,i_0\rangle w & \\ \vdash & Z(\tau_0'x_0)^R\langle\varepsilon,r,s,i_0\rangle w & \text{(by a prod. constructed in 2)} \\ \vdash^{|\tau_0'x_0|} & Z\langle\varepsilon,r,s,i_0\rangle & \text{(by prod. constructed in 3)} \\ \vdash & f & \text{(by a prod. constructed in 4).} \end{array}$$

Therefore, the basis holds.

Induction hypothesis: Suppose that the claim holds for all derivations of length j or less.

Induction step: Consider a derivation of length $j+1$:

$$\tau_0 x_0 \ {}_l\Leftrightarrow_{P_{i_0}} \ \tau_0' x_0 = v_1\tau_1 x_1 \ {}_l\Leftrightarrow^t_{P_{i_1}} \ v_1 w_1 \ {}_l\Leftrightarrow^t_{P_{i_2}} \ w_2 \dots \ {}_l\Leftrightarrow^t_{P_{i_p}} \ w_p = w = v_1 w',$$

where $p \geq 0$, $v_1 \in T^*$, $\tau_0, \tau_1 \in N^+$, $\tau_0' \in V^*$, $x_0, x_1 \in TV^* \cup \{\varepsilon\}$, $w_i \in V^*, i \in \{1, \dots, p-1\}$, $w_p, w' \in T^*$. Then, M simulates this derivation as follows:

	$Z(\tau_0^2 x_0)^R\langle \tau_0^1, q, s, i_0\rangle w$	
$\vdash$	$Z(\tau_0' x_0)^R\langle \varepsilon, r, s, i_0\rangle w$	(by a prod. constructed in 2)
$=$	$Z(v_1\tau_1 x_1)^R\langle \varepsilon, r, s, i_0\rangle v_1 w'$	
$\vdash^{\lvert v_1\rvert}$	$Z(\tau_1 x_1)^R\langle \varepsilon, r, s, i_0\rangle w'$	(by prod. constructed in 3)
$\vdash^{\lvert \tau_1^1\rvert}$	$Z(\tau_1^2 x_1)^R\langle \tau_1^1, r, s, i_0\rangle w'$	(by prod. constructed in 5)
$\vdash$	$Z(\tau_1^2 x_1)^R\langle \tau_1^1, e, s, i_0\rangle w'$	(by a prod. constructed in 6, 7, or 8)
$\vdash$	$Z(\tau_1^2 x_1)^R\langle \tau_1^1, q, s', i_1\rangle w'$	(by a prod. constructed in 9 or 10)
$\vdash^*$	f	(by the induction hypothesis)

If $\tau_0 x_0 \ {}_l\Leftrightarrow_{P_{i_0}} \ \tau_0' x_0$ is a t-mode derivation, a rule of type 9 is used during the simulation. Otherwise, a rule of type 10 is used (and therefore $i_0 = i_1$). Hence, the claim holds. □

Let $S \ {}_l\Leftrightarrow_{P_{i_0}} \ u\tau_0 x_0 \ {}_l\Leftrightarrow^t_{P_{i_1}} \ \dots \ {}_l\Leftrightarrow^t_{P_{i_p}} \ uw$, where $p \geq 0$, $u, w \in T^*$, $\tau_0 \in N^+ \cup \{\varepsilon\}$, $x_0 \in TV^* \cup \{\varepsilon\}$ and $i_1 \dots i_p \in L(\bar{M})$. If $u\tau_0 x_0 \notin T^*$, M simulates this derivation in the following way:

	$Z s_0 uw$	
$\vdash$	$Z\langle S, q, s, i_0\rangle$	(by a prod. constructed in 1)
$\vdash$	$Z(u\tau_0 x_0)^R\langle \varepsilon, r, s, i_0\rangle uw$	(by a prod. constructed in 2)
$\vdash^{\lvert u\rvert}$	$Z(\tau_0 x_0)^R\langle \varepsilon, r, s, i_0\rangle w$	(by prod. constructed in 3)
$\vdash^{\lvert \tau_0^1\rvert}$	$Z(\tau_0^2 x_0)^R\langle \tau_0^1, r, s, i_0\rangle w$	(by prod. constructed in 5)
$\vdash$	$Z(\tau_0^2 x_0)^R\langle \tau_0^1, e, s, i_0\rangle w$	(by a prod. constructed in 6, 7, or 8)
$\vdash$	$Z(\tau_0^2 x_0)^R\langle \tau_0^1, q, s', i_1\rangle w$	(by a prod. constructed in 9 or 10)
$\vdash^*$	f	(by the previous claim)

If $u\tau_0 x_0 \in T^*$, M simulates this derivation in the following way:

	$Z s_0 uw$	
$\vdash$	$Z\langle S, q, s, i_0\rangle$	(by a prod. constructed in 1)
$\vdash$	$Z(u\tau_0 x_0)^R\langle \varepsilon, r, s, i_0\rangle uw$	(by a prod. constructed in 2)
$\vdash^{\lvert u\tau_0 x_0\rvert}$	$Z\langle \varepsilon, r, s, i_0\rangle w$	(by prod. constructed in 3)
$\vdash$	f	(by a prod. constructed in 4)

Hence and from the previous claims, it follows that the lemma holds. ■

By Lemma 4.6, we have the following result.

Theorem 4.9 Let $l \in \mathbb{N}$. Then, $\mathbf{CF} = \mathscr{L}({}_l\mathrm{CD}^{\mathbf{REG}})$.

Proof of Theorem 4.9. One inclusion is clear, the other follows from Lemma 4.6. ■

Theorem 4.9 says that grammar systems under the first restriction are much weaker than grammar systems without this restriction. Now, we prove that the second restriction in this chapter has no effect on the generative power.

Theorem 4.10 $\mathbf{RE} = \mathscr{L}({}_{\mathrm{N}}\mathrm{CD}^{\mathbf{REG}}(1))$.

Proof of Theorem 4.10. It is well-known (see [47] or [48]) that any recursively enumerable language L is generated by a grammar G in the Geffert normal form, i.e., by a grammar of the form

$$G = (\{S, A, B, C\}, T, P \cup \{ABC \to \varepsilon\}, S)\,,$$

where P contains context-free productions of the form

$$\begin{aligned} S &\to uSa \\ S &\to uSv \\ S &\to uv \end{aligned}$$

where $u \in \{A, AB\}^*$, $v \in \{BC, C\}^*$, and $a \in T$. In addition, any terminal derivation in G is of the form $S \vdash^* w_1 w_2 w$ by productions from P, where $w_1 \in \{A, AB\}^*$, $w_2 \in \{BC, C\}^*$, $w \in T^*$, and $w_1 w_2 w \vdash^* w$ is derived by $ABC \to \varepsilon$.

Clearly, G is a CD grammar system with only one component. Set the control language to be $\{1\}^*$. Then, the theorem holds. ■

The last theorem of this chapter says that generative power of grammar systems under the third restriction is less than generative power of grammar systems without any restriction.

Theorem 4.11 For any m, $h \geq 1$, $\mathscr{L}_m(\mathrm{P}) = \mathscr{L}({}_{\mathrm{N}}\mathrm{CD}^{\mathbf{REG}}(m, h))$.

Proof of Theorem 4.11. All strings in the derivation contain no more than m blocks of nonterminals and these blocks are also of length no more than h. Hence, it is possible to represent each possible block by a single nonterminal and create an equivalent grammar system, which contains only context-free productions. From this and from Theorem 7.10 in [42], Theorem 4.11 holds. ■

Chapter 5

n-Accepting Restricted Pushdown Automata Systems

In this chapter, we introduce two versions of n-accepting restricted pushdown automata systems that represent automata counterpart of multi-generative grammar systems (see Section 3.2). First, we define *n-accepting state-restricted pushdown automata systems*. By using prescribed n-state sequences, the restriction of these systems determines which of the components perform a move and which of them do not. Second, we define *n-accepting move-restricted pushdown automata systems*, where the restriction precisely determines which transition rule can be used in each of the n components.

As the main results, we demonstrate that both restricted systems under discussion are equally powerful in the sense that an n-language is accepted by a state-restricted system if and only if the same n-language is accepted by a move-restricted system. In fact, this crucial result is proven effectively. Indeed, we give a transformation algorithm that converts any state-restricted system to a move-restricted system so the above equivalence holds. Furthermore, it is shown that both variants of n-accepting restricted pushdown automata systems define the same class of n-languages as n-CGR and n-CGN do.

Note. In the rest of the publication we assume that $n \in \mathbb{N}-\{1\}$ and by PDA we understand pushdown automaton accepting by empty pushdown unless stated otherwise.

5.1 n-Accepting State-Restricted Automata Systems

n-Accepting state-restricted automata system consists of set of switch rules and n PDAs (see Definition 5.1), where each PDA computes on its own input string. During computation, some automata can be suspended and after some computation steps resumed. In this way, the system works on the n-string composed of these strings, and the system accepts the n-string if and only if all PDAs accept their input.

Definition 5.1 (n-Accepting state-restricted automata system)
Let $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, s_i, z_{i,0}, \emptyset)$ be a PDA, for all $i = 1, \ldots, n$. Then, an n-accepting state-restricted automata system, n-SAS, is defined as $\vartheta = (M_1, \ldots, M_n, \Psi)$, where Ψ is a finite set of switch rules of the form $(q_1, \ldots, q_n) \to (\hbar_1, \ldots, \hbar_n)$ with $q_i \in Q_i$ and $\hbar_i \in \{e, d\}$, for all $i = 1, \ldots, n$. Symbols e and d are referred to as an enable and disable component of the automata system, respectively.

Definition 5.2 (n-Configuration of n-SAS)
Let $\vartheta = (M_1, \ldots, M_n, \Psi)$ be an n-SAS and $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, s_i, z_{i,0}, \emptyset)$ for all $i = 1, \ldots, n$. An n-configuration is defined as an n-tuple $\chi = ((x_1)^{\hbar_1}, \ldots, (x_n)^{\hbar_n})$, where for all $i = 1, \ldots, n$, $x_i \in \Gamma_i^* Q_i \Sigma^*$ is a configuration of M_i and $\hbar_i \in \{d, e\}$.

Definition 5.3 (Computation step in n-SAS)
Let $n \in \mathbb{N}$, $\vartheta = (M_1, \ldots, M_n, \Psi)$ be an n-SAS and $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, s_i, z_{i,0}, \emptyset)$, for all $i = 1, \ldots, n$. Let

- $\chi = ((\gamma_1 q_1 \omega_1)^{\hbar_1}, \ldots, (\gamma_n q_n \omega_n)^{\hbar_n})$ and
- $\chi' = ((\gamma'_1 q'_1 \omega'_1)^{\hbar'_1}, \ldots, (\gamma'_n q'_n \omega'_n)^{\hbar'_n})$

be two n-configurations of the n-SAS with $\gamma_i, \gamma'_i \in \Gamma_i^*$, $q_i, q'_i \in Q_i$ and $\omega_i, \omega'_i \in \Sigma^*$. ϑ makes a computation step from n-configuration χ to χ', denoted $\chi \vdash \chi'$, where for every $i = 1, \ldots, n$, the following holds:

- if $\hbar_i = d$, then $\gamma_i q_i \omega_i = \gamma'_i q'_i \omega'_i$
- if $\hbar_i = e$, then $\gamma_i q_i \omega_i \vdash \gamma'_i q'_i \omega'_i$ in M_i
- if $(q'_1, \ldots, q'_n) \to (\hbar_1, \ldots, \hbar_n) \in \Psi$, then $\hbar'_i = \hbar_i$
- if $(q'_1, \ldots, q'_n) \to (\hbar_1, \ldots, \hbar_n) \notin \Psi$, for all $(\hbar_1, \ldots, \hbar_n) \in \{e, d\}_1 \times \ldots \times \{e, d\}_n$, then $\hbar'_i = \hbar_i$

In the standard way, $\vdash^*$ and $\vdash^+$ denote reflexive-transitive and transitive closure of $\vdash$, respectively.

Informally, during single computation step in n-SAS, all enable PDAs make a move with respect to their transition rules, while the other automata are frozen and do nothing. Thereafter, the system checks whether there is a member, for the current combination of states, in its set of switch rules. If so, activities of the PDAs in the systems are changed; otherwise, the activities remain unchanged.

Definition 5.4 (n-Language of n-SAS)
Let $\vartheta = (M_1, \ldots, M_n, \Psi)$ be an n-SAS and for every $i = 1, \ldots, n$, $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, s_i, z_{i,0}, \emptyset)$ be a PDA accepting input strings by empty pushdown. Let

- $\chi_0 = ((z_{1,0} s_1 \omega_1)^e, \ldots, (z_{n,0} s_n \omega_n)^e)$ be the start n-configuration and
- $\chi_f = ((q_1)^{\hbar_1}, \ldots, (q_n)^{\hbar_n})$ be a final n-configuration of n-SAS,

where for all $i = 1, \ldots, n$, $q_i \in Q_i$, $\hbar_i, \in \{d, e\}$, $\omega_i \in \Sigma^*$. The n-language of n-SAS is defined as $n\text{-}L(\vartheta) = \{(\omega_1, \ldots, \omega_n) |\ \chi_0 \vdash^* \chi_f\}$.

Example 5.5
Consider a 2-SAS $\vartheta = (M_1, M_2, \Psi)$ given by next.

- $M_1 = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{\triangleright, a\}, \delta_1,\ q_0, \triangleright, \emptyset)$,
- $M_2 = (\{q_0, q_1\}, \{a, b\}, \{\triangleright, a\}, \delta_2,\ q_0, \triangleright, \emptyset)$,

1.PDA	2.PDA	Rule(s)	Ψ
$(\triangleright q_0 aabbcc)^e$	$(\triangleright q_0 aabb)^e$		
$(\triangleright a q_0 abbcc)^e$	$(\triangleright a q_0 abb)^e$	(r_1, r_1)	
$(\triangleright aa q_0 bbcc)^e$	$(\triangleright aa q_0 bb)^e$	(r_2, r_2)	
$(\triangleright a q_1 bcc)^e$	$(\triangleright a q_1 b)^d$	(r_3, r_3)	$(q_1, q_1) \to (e, d) \in \Psi$
$(\triangleright q_1 cc)^e$	$(\triangleright a q_1 b)^d$	$(r_4, -)$	
$(\triangleright q_2 c)^e$	$(\triangleright a q_1 b)^e$	$(r_5, -)$	$(q_2, q_1) \to (e, e) \in \Psi$
$(\triangleright q_2)^e$	$(\triangleright q_1)^e$	(r_6, r_4)	
$(q_2)^e$	$(q_1)^e$	(r_7, r_5)	

Table 5.1: Acceptance of $(aabbcc, aabb)$ by 2-SAS from Example 5.5

- $\delta_1 = \{ \quad r_1 = \triangleright q_0 a \to \triangleright a q_0,$
  $r_2 = a q_0 a \to aa q_0,$
  $r_3 = a q_0 b \to q_1,$
  $r_4 = a q_1 b \to q_1,$
  $r_5 = \triangleright q_1 c \to \triangleright q_2,$
  $r_6 = \triangleright q_2 c \to \triangleright q_2,$
  $r_7 = \triangleright q_2 \to q_2$
  $\}$

  $\delta_2 = \{ \quad r_1 = \triangleright q_0 a \to \triangleright a q_0,$
  $r_2 = a q_0 a \to aa q_0,$
  $r_3 = a q_0 b \to q_1,$
  $r_4 = a q_1 b \to q_1,$
  $r_5 = \triangleright q_1 \to q_1$
  $\}$

- $\Psi = \{(q_0, q_1) \to (d, d), (q_1, q_0) \to (d, d), (q_1, q_1) \to (e, d), (q_2, q_1) \to (e, e)\}$

Both PDAs start with pushing as on their pushdowns. As soon as bs are the first input symbols in both automata, the PDAs remove as from the pushdowns, move to q_1, and by Ψ, the system makes M_2 disable—that is, only M_1 continues. Note that the PDAs have to move to the state q_1 at the same time. Otherwise, the system makes both automata frozen and the inputs are not accepted. In the state q_1, M_1 compares number of as and bs by removing as from the pushdown and reading bs from its input. After that, if the number of as and bs coincides, M_1 moves to q_2 and M_2 becomes enabled. By using rules r_6 and r_4 in M_1 and M_2, respectively, M_1 reads cs from the input of M_1 while M_2 compares number of as and bs. One can see that the last step of the successful computation must be done by rules r_7 in M_1 and r_5 in M_2 at the same time. In this way, the 2-SAS accepts 2-strings from the 2-language 2-$L(\vartheta) = \{(a^m b^m c^m, a^m b^m) |\ m \in \mathbb{N}\}$. Example of acceptance of the 2-string $(aabbcc, aabb)$ is shown in Table 5.1.

For any n-CGN, we can construct an n-SAS that defines the same n-language.

Basic Idea. With regard to the organization, for every CFG G in the n-CGN, the n-SAS has a PDA which simulates top-down parsing based on G (see [80]). The automaton contains one state for each nonterminal, where the nonterminal can be expanded; in addition, it contains two other states for comparisons and acceptation. By the states of the automata, the n-SAS is able to control what its automata do. If some automata compare terminal symbols on their pushdowns with input symbols, the other automata have to wait until all automata are ready to the next expansion or until they accept their inputs. The automata in the n-SAS make the expansions at the same time, and the n-SAS allows the expansions if and only if the nonterminals on the tops of the pushdowns can be derived at the same time in the n-CGN. If an expansion is not allowed, the n-SAS suspends all its automata.

Algorithm 5.6
Construction of n-SAS from n-CGN

Input: n-CGN $\widehat{\Gamma} = (G_1, \ldots, G_n, \widehat{Q})$, where $G_i = (\widehat{N}_i, \widehat{T}_i, \widehat{P}_i, \widehat{S}_i)$ be a context-free grammar, for $i = 1, \ldots, n$.

Output: n-MAS $\vartheta = (M_1, \ldots, M_n, \Psi)$, where for every $i = 1, \ldots, n$, $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, s_{i,0}, z_{i,0}, \emptyset)$ is a pushdown automaton accepting by empty pushdown, and $n\text{-}L(\vartheta) = n\text{-}L(\widehat{\Gamma})$.

Method:

- For every $i = 1, \ldots, n$, set $Q_i = \{\langle A\rangle |\ A \in \widehat{N}_i\} \cup \{r_i, f_i\}$, $\Sigma = \bigcup_{i=1}^{n} \widehat{T}_i$, $\Gamma_i = \Sigma \cup \widehat{N}_i \cup \{A' |\ A \in \widehat{N}_i\} \cup \{\Delta, \Delta'\}$, $s_{i,0} = \langle \widehat{S}_i\rangle$ and $z_{i,0} = \Delta'$.
- For all $i = 1, \ldots, n$, δ_i is constructed as follows:
  0. set $\delta_i = \emptyset$,
  1. add $\Delta'\langle \widehat{S}_i\rangle \to \Delta \widehat{S}_i \widehat{S}_i' \langle \widehat{S}_i\rangle$ to δ_i,
  2. add $a\langle A\rangle \to a r_i$ to δ_i for every $a \in \widehat{T}_i$ and every $A \in \widehat{N}_i$,
  3. add $A'\langle B\rangle \to A' r_i$ to δ_i for every $A, B \in \widehat{N}_i$,
  4. add $a r_i a \to r_i$ to δ_i for every $a \in \widehat{T}_i$,
  5. add $A' r_i \to \langle A\rangle$ to δ_i for every $A \in \widehat{N}_i$,
  6. add $\Delta\langle A\rangle \to f_i$ to δ_i for every $A \in \widehat{N}_i$,
  7. add $\Delta r_i \to f_i$ to δ_i,
  8. for all $(A \to \alpha) \in \widehat{P}_i$, add $A\langle A\rangle \to \theta(\alpha)\langle A\rangle$ to δ_i, where θ is a projection from $(\widehat{N}_i \cup \widehat{T}_i)^*$ to $(\widehat{N}_i\{A' |\ A \in \widehat{N}_i\} \cup \widehat{T}_i)^*$ such that $\theta(\omega) = \omega'$, where ω' is made from $(\omega)^R$ by replacing each $A \in \widehat{N}_i$ in $(\omega)^R$ by AA'. For example, if $a, b \in T_i$ and $A \in N_i$, $\theta(aAb) = bAA'a$.
- Ψ is constructed in the following way:
  0. set $\Psi = \emptyset$
  1. for all $(A_1, \ldots, A_n) \in \widehat{Q}$, add $(\langle A_1\rangle, \ldots, \langle A_n\rangle) \to (e, \ldots, e)$ to Ψ
  2. add $(f_1, \ldots, f_n) \to (e, \ldots, e) \in \Psi$,
  3. for all $(q_1, \ldots, q_n) \in Q_1 \times \ldots \times Q_n$, where $\{r_1, \ldots, r_n\} \cap \{q_1, \ldots, q_n\} \neq \emptyset$, add $(q_1, \ldots, q_n) \to (l_1, \ldots, l_n)$ such that for all $o = 1, \ldots, n$, $q_o \in \{r_o\} \Leftrightarrow l_o = e$, to Ψ,
  4. for other $(q_1, \ldots, g_n) \in Q_1 \times \ldots \times Q_n$, add $(q_1, \ldots, q_n) \to (d, \ldots, d)$ to Ψ.

Lemma 5.7 Algorithm 5.6 is correct.

Proof of Lemma 5.7. We start to prove the following claims.

Claim 5.8 Let $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^* (u_1 A_1 v_1, \ldots, u_n A_n v_n)$ in $\widehat{\Gamma}$, where $A_i \in \widehat{N}_i$, $v_i \in (\widehat{N}_i \cup \widehat{T}_i)^*$, $u_i \omega_i \in \widehat{T}_i^*$, for all $i = 1, \ldots, n$, and $(u_1 A_1 v_1, \ldots, u_n A_n v_n) \Rightarrow^* (u_1\omega_1, \ldots, u_n\omega_n)$, then

$$\begin{aligned} & ((\Delta'\langle \widehat{S}_1\rangle u_1\omega_1)^e, \ldots, (\Delta'\langle \widehat{S}_n\rangle u_n\omega_n)^e) \\ \vdash^* \quad & ((\Delta\theta(v_1)A_1\langle \widehat{A}_1\rangle\omega_1)^{\hbar_1}, \ \ldots, \ (\Delta\theta(v_n)A_n\langle \widehat{A}_n\rangle\omega_n)^{\hbar_n}) \end{aligned}$$

in ϑ.

Proof of Claim 5.8. By induction hypothesis on the length of derivation.

Basis:
Let $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^0 (\widehat{S}_1, \ldots, \widehat{S}_n)$, where $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^* (\omega_1, \ldots, \omega_n)$ and $\omega_i \in \widehat{T}_i^*$ for every $i = 1, \ldots, n$. Then,

$$\begin{array}{ll} & ((\Delta'\langle\widehat{S}_1\rangle\omega_1)^e, \ldots, (\Delta'\langle\widehat{S}_n\rangle\omega_n)^e) \\ \vdash & ((\Delta\widehat{S}_1\widehat{S}_1'\langle\widehat{S}_1\rangle\omega_1)^e,\ \ldots, ((\Delta\widehat{S}_n\widehat{S}_n'\langle\widehat{S}_n\rangle\omega_n)^e) \qquad \text{[by 1. of } \delta_i \text{ and 1. of } \Psi] \\ \vdash & ((\Delta\widehat{S}_1\widehat{S}_1' r_1\omega_1)^e,\ \ldots, ((\Delta\widehat{S}_n\widehat{S}_n' r_n\omega_n)^e) \qquad \text{[by 3. of } \delta_i \text{ and 3. of } \Psi] \\ \vdash & ((\Delta\widehat{S}_1\langle\widehat{S}_1\rangle\omega_1)^{\hbar_1},\ \ldots, ((\Delta\widehat{S}_n\langle\widehat{S}_n\rangle\omega_n)^{\hbar_n}) \qquad \text{[by 5. of } \delta_i] \end{array}$$

for $\hbar_1, \ldots, \hbar_n \in \{e, d\}$.

Induction Hypothesis:
Suppose that Claim 5.8 holds for j or fewer derivation steps, where j is a non-negative integer.

Induction Step:
Consider any derivation $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^{j+1} (u_1x_1v_1, \ldots, u_nx_nv_n)$, where for all $i = 1, \ldots, n$, $u_ix_iv_i = u_iu_i'B_iv_i'$, $u_i, u_i' \in \widehat{T}_i^*$, $B_i \in \widehat{N}_i\}$, $v_i, v_i', x_i \in (\widehat{N}_i \cup \widehat{T}_i)^*$, for $u_i\omega_i \in \widehat{T}_i^*$, $(u_1u_1'B_iv_1'v_1, \ldots, u_nu_n'B_nv_n'v_n) \Rightarrow^* (u_1\omega_1, \ldots, u_n\omega_n)$. We can express this derivation as $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^j (u_1A_1v_1, \ldots, u_nA_nv_n) \Rightarrow (u_1x_1v_1, \ldots, u_nx_nv_n)$, where for all $i = 1, \ldots, n$, $A_i \in \widehat{N}_i$. By induction hypothesis,

$$\begin{array}{ll} & ((\Delta'\langle\widehat{S}_1\rangle u_1\omega_1)^e, \ldots, (\Delta'\langle\widehat{S}_n\rangle u_n\omega_n)^e) \\ \vdash^* & ((\Delta\theta(v_1)A_1\langle A_1\rangle\omega_1)^{\hbar_1},\ \ldots, ((\Delta\theta(v_n)A_n\langle A_n\rangle\omega_n)^{\hbar_n}). \end{array}$$

Obviously, $(A_1, \ldots, A_n) \in \widehat{Q}$. From Algorithm 5.6, $(\langle A_1\rangle, \ldots, \langle A_n\rangle) \to (e, \ldots, e) \in \Psi$ and for every $A_i \to x_i \in \widehat{P}_i$, there is $A_i\langle A_i\rangle \to \theta(x_i)$. Therefore, $\hbar_1, \ldots, \hbar_n = e$ and

$$\begin{array}{ll} & ((\Delta\theta(v_1)A_1\langle A_1\rangle\omega_1)^e,\ \ldots, ((\Delta\theta(v_n)A_n\langle A_n\rangle\omega_n)^e) \\ \vdash & ((\Delta\theta(x_1v_1)\langle A_1\rangle\omega_1)^e,\ \ldots, ((\Delta\theta(x_nv_n)\langle A_n\rangle\omega_n)^e) \\ = & ((\Delta\theta(v_1')(u_1'B_1'B_1)^R\langle A_1\rangle\omega_1)^e,\ \ldots, ((\Delta\theta(v_n')(u_n'B_n'B_n)^R\langle A_n\rangle\omega_n)^e). \end{array}$$

There is $a_i \in \Gamma_i - (\widehat{N}_i \cup \{\Delta, \Delta'\})$ on the top of the pushdown in each automaton. Therefore, from rules of the form 2. and 3. in δ_i and from 3. in Ψ, it follows

$$\begin{array}{ll} & ((\Delta\theta(v_1')(u_1'B_1'B_1)^R\langle A_1\rangle\omega_1)^e,\ \ldots, ((\Delta\theta(v_n')((u_n'B_n'B_n)^R\langle A_n\rangle\omega_n)^e) \\ \vdash & ((\Delta\theta(v_1')(u_1'B_1'B_1)^R r_1\omega_1)^e,\ \ldots, ((\Delta\theta(v_n')(u_n'B_n'B_n)^R r_n\omega_n)^e). \end{array}$$

Because $u_iu_i'B_iv_i' \Rightarrow^* u_i\omega_i$, $u_i\omega_i = u_iu_i'\omega_i'$. For every component M_i of ϑ, there is a sequence of computation steps $\Delta\theta(v_i')B_iB_i'(u_i')^R r_iu_i'\omega_i' \vdash^* \Delta\theta(v_i')B_iB_i'r_i\omega_i'$ (by 4. of δ_i) and $\Delta\theta(v_i')B_iB_i'r_i\omega_i' \vdash \Delta\theta(v_i')B_i\langle B_i\rangle u_i'\omega_i'$ (by 5. of δ_i). From 3. in Ψ, each automaton M_i which is in state $\langle B_i\rangle$ is suspended until no automaton M_j is in state r_j. Hence,

$$\begin{array}{ll} & ((\Delta\theta(v_1')(u_1'B_1'B_1)^R r_1u_1'\omega_1')^e,\ \ldots, ((\Delta\theta(v_n')(u_n'B_n'B_n)^R r_nu_n'\omega_n')^e) \\ \vdash^* & ((\Delta\theta(v_1')B_1\langle B_1\rangle\omega_1')^{\ell_1},\ \ldots, ((\Delta\theta(v_n')B_n\langle B_n\rangle\omega_n')^{\ell_n}) \end{array}$$

for $\ell_1 \ldots \ell_n \in \{e, d\}$. Claim 5.8 holds. □

Claim 5.9 If $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^* (\omega_1, \ldots, \omega_n)$ in $\widehat{\Gamma}$, where for every $i = 1, \ldots, n$, $\omega_i \in \widehat{T}_i^*$, there is a sequence of moves $(\Delta'\langle\widehat{S}_1\rangle\omega_1)^e, \ldots, (\Delta'\langle\widehat{S}_n\rangle\omega_n)^e) \vdash^* ((f_1)^e, \ldots, (f_n)^e)$ in ϑ.

Proof of Claim 5.9. Consider any successful derivation $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^* (\omega_1, \ldots, \omega_n)$. There must be an n-form of the form $(u_1A_1v_1, \ldots, u_nA_nv_n)$ such that $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^* (u_1A_1v_1, \ldots, u_nA_nv_n) \Rightarrow (u_1x_1v_1, \ldots, u_nx_nv_n) = (\omega_1, \ldots, \omega_n)$. By Claim 5.8,

$$\begin{array}{rl} & ((\Delta'\langle\widehat{S}_1\rangle\omega_i)^e, \ldots, (\Delta'\langle\widehat{S}_n\rangle\omega_n)^e) \\ \vdash^* & ((\Delta\theta(v_1)A_1\langle A_1\rangle\omega_1')^{\hbar_1}, \ldots, (\Delta\theta(v_n)A_n\langle A_n\rangle\omega_n')^{\hbar_n}) \end{array}$$

and for all $i = 1, \ldots, n$, $\omega_i = u_i\omega_i'$. Because $(A_1, \ldots, A_n) \in \widehat{Q}$, $(\langle A_1\rangle, \ldots, \langle A_n\rangle) \rightarrow (e, \ldots, e) \in \Psi$, for every $i = 1, \ldots, n$, $\hbar_i = e$ and ϑ moves to $(\Delta\theta(x_1v_1)\langle A_1\rangle\omega_1')^e, \ldots, (\Delta\theta(x_nv_n)\langle A_n\rangle\omega_n')^e$. For every $i = 1, \ldots, n$, $\omega_i' = x_iv_i$, because $\omega_i = u_ix_iv_i = u_i\omega_i'$. Therefore, there are these two possibilities of the top symbol on the pushdown in each automaton of ϑ:

a) The top symbol is Δ, then $x_iv_i = \varepsilon$ and $\omega_i' = \varepsilon$—that is, M_i moves to f_i. Furthermore, if any other automaton M_j is in the state r_j, automaton M_i is blocked.

b) The top symbol is $a_i \in \widehat{T}_i$, than $\Delta(x_iv_i)^R\langle A_i\rangle x_iv_i \vdash \Delta(x_iv_i)^R r_i x_i v_i$ (by rule of the form 2. of δ_i) and $\Delta(x_iv_i)^R r_i x_i v_i \vdash^* \Delta r_i$ (by rules type 4. of δ_i), $\Delta r_i \vdash f_i$ (by 7) and automaton M_i is blocked until no other automaton M_j of ϑ is in state r_j.

In this way, ϑ moves to n-configuration $((f_1)^e, \ldots, (f_n)^e)$ and Claim 5.9 holds. □

Claim 5.10 If $(\Delta'\langle\widehat{S}_1\rangle\omega_1)^e, \ldots, (\Delta'\langle\widehat{S}_n\rangle\omega_n)^e) \vdash^* ((f_1)^e, \ldots, (f_n)^e)$ in ϑ, there is a sequence of derivation steps $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^* (\omega_1, \ldots, \omega_n)$.

Proof of Claim 5.10. Consider any successful acceptance:

$$(\Delta'\langle\widehat{S}_1\rangle\omega_1)^e, \ldots, (\Delta'\langle\widehat{S}_n\rangle\omega_n)^e) \vdash^* ((f_1)^e, \ldots, (f_n)^e) \qquad (I)$$

in ϑ. From Algorithm 5.6 (by 1. of δ_i), the first step of ϑ must be

$$\begin{array}{rl} & (\Delta'\langle\widehat{S}_1\rangle\omega_1)^e, \ldots, (\Delta'\langle\widehat{S}_n\rangle\omega_n)^e) \\ \vdash & ((\Delta\widehat{S}_1\widehat{S}_1'\langle\widehat{S}_1\rangle\omega_1)^{l_1}, \ \ldots, ((\Delta\widehat{S}_n\widehat{S}_n'\langle\widehat{S}_n\rangle\omega_n)^{l_n}), \end{array}$$

and from construction of Ψ, there are two possibilities of activities of automata in ϑ:

- $(\langle\widehat{S}_1\rangle, \ldots, \langle\widehat{S}_n\rangle) \rightarrow (d, \ldots, d) \in \Psi$, but for every $i = 1, \ldots, n$, $l_i = d$ and ϑ is not successful for any input.
- Therefore, $(\langle\widehat{S}_1\rangle, \ldots, \langle\widehat{S}_n\rangle) \rightarrow (e, \ldots, e) \in \Psi$ and for every $i = 1, \ldots, n$, $l_i = e$.

From rules of the form 3. and 5. in δ_i, ϑ must make two following computation steps:

$$\begin{array}{rl} & ((\Delta\widehat{S}_1\widehat{S}_1'\langle\widehat{S}_1\rangle\omega_1)^e, \ \ldots, ((\Delta\widehat{S}_n\widehat{S}_n'\langle\widehat{S}_n\rangle\omega_n)^e) \\ \vdash & ((\Delta\widehat{S}_1\widehat{S}_1'r_1\omega_1)^e, \ \ldots, ((\Delta\widehat{S}_n\widehat{S}_n'r_n\omega_n)^e) \\ \vdash & ((\Delta\widehat{S}_1\langle\widehat{S}_1\rangle\omega_1)^e, \ \ldots, ((\Delta\widehat{S}_n\langle\widehat{S}_n\rangle\omega_n)^e). \end{array}$$

Consider any n-configuration of the form $((\Delta A_1\langle A_1\rangle\omega_1')^e, \ \ldots, ((\Delta A_n\langle A_n\rangle\omega_n')^e)$, where $((\Delta A_1\langle A_1\rangle\omega_1')^e, \ \ldots, ((\Delta A_n\langle A_n\rangle\omega_n')^e) \vdash^* ((f_1)^e, \ldots, (f_n)^e)$. ϑ makes these computation steps:

$$\begin{array}{rl} & ((\Delta A_1\langle A_1\rangle\omega_1')^e, \ \ldots, ((\Delta A_n\langle A_n\rangle\omega_n')^e) \\ \vdash & ((\Delta\gamma_1\langle A_1\rangle\omega_1')^e, \ \ldots, ((\Delta\gamma_n\langle A_n\rangle\omega_n')^e) \\ \vdash & ((\gamma_1'q_1\omega_1')^{l_1}, \ \ldots, ((\gamma_n'q_n\omega_n')^{l_n}), \end{array}$$

and (from 2., 3. and 6. of δ_i and from Ψ) for every $i = 1, \ldots, n$, it holds:

- if $\gamma_i = \varepsilon$, then $\gamma_i' = \omega_i' = \varepsilon$, $q_i = f_i$ and if $\gamma_1, \ldots, \gamma_n = \varepsilon$ (i.e. for every $j = 1, \ldots, n$, $q_j = f_j$), then $\ell_i = e$; otherwise, $\ell_i = d$,
- else $\gamma_i' = \Delta\gamma_i$, $q_i = r_i$ and $\ell_i = e$.

Hence, there are only three applicable types of moves over each automaton M_i, which is in state r_i:

- $ar_ia \to r_i$ where $a \in \widehat{T}_i$ and M_i is still active in the next computation step;
- $B_i'r_i \to \langle B_i \rangle$ where $B_i \in \widehat{N}_i$, than the next top symbol on the pushdown must be B_i and M_i is blocked until any automaton M_j of ϑ is not in state r_j;
- $\Delta r_i \to f_i$ and M_i is blocked until any automaton M_j of ϑ is in state r_j.

These three types of steps are repeatedly applied on active components of ϑ. Hence, and from the construction of Ψ, the following configuration of ϑ must be either

i. $((f_1)^e, \ldots, (f_n)^e)$ (n-string is accepted), or

ii. $((\Delta\gamma_1' B_1 \langle B_1 \rangle \omega_1'')^e$, ..., $((\Delta\gamma_n' B_n \langle B_n \rangle \omega_n'')^e)$, where for every $i = 1, \ldots, n$, $B_i \in \widehat{N}_i$ and $(\langle B_1 \rangle, \ldots, \langle B_n \rangle) \to (e, \ldots, e) \in \Psi$.

For others n-tuples of states, ϑ blocked all automata (by 4. of Ψ) and the n-string is not accepted. It is obvious that we can express (I) as

$$
\begin{array}{ll}
 & (\Delta' \langle \widehat{S}_1 \rangle \omega_1)^e, \ldots, (\Delta' \langle \widehat{S}_n \rangle \omega_n)^e) \\
\vdash^3 & ((\Delta \widehat{S}_1 \langle \widehat{S}_1 \rangle \omega_1)^e, \ \ldots, ((\Delta \widehat{S}_n \langle \widehat{S}_n \rangle \omega_n)^e) \\
\vdash^{m_1} & ((\Delta\gamma_1^{(1)} A_1^{(1)} \langle A_1^{(1)} \rangle \omega_1^{(1)})^e, \ \ldots, ((\Delta\gamma_n^{(1)} A_n^{(1)} \langle A_n^{(1)} \rangle \omega_n^{(1)})^e) \\
\vdash^{m_2} & ((\Delta\gamma_1^{(2)} A_1^{(2)} \langle A_1^{(2)} \rangle \omega_1^{(2)})^e, \ \ldots, ((\Delta\gamma_n^{(2)} A_n^{(2)} \langle A_n^{(2)} \rangle \omega_n^{(2)})^e) \\
\vdots & \\
\vdash^{m_k} & ((\Delta\gamma_1^{(k)} A_1^{(k)} \langle A_1^{(k)} \rangle \omega_1^{(k)})^e, \ \ldots, ((\Delta\gamma_n^{(k)} A_n^{(k)} \langle A_n^{(k)} \rangle \omega_n^{(k)})^e) \\
\vdash^{m_{k+1}} & ((f_1)^e, \ldots, (f_n)^e)
\end{array}
$$

for all $i = 1, \ldots k+1$, $m_i \geq 1$. The computation of ϑ can be simulated by $\widehat{\Gamma}$ as

$$
\begin{array}{ll}
 & (\widehat{S}_1, \ldots, \widehat{S}_n) \\
\Rightarrow & (u_1^{(1)} A_1^{(1)} v_1^{(1)}, \ldots, u_n^{(1)} A_n^{(1)} v_n^{(1)}) \\
\Rightarrow & (u_1^{(2)} A_1^{(2)} v_1^{(2)}, \ldots, u_n^{(2)} A_n^{(2)} v_n^{(2)}) \\
\vdots & \\
\Rightarrow & (u_1^{(k)} A_1^{(k)} v_1^{(k)}, \ldots, u_n^{(k)} A_n^{(k)} v_n^{(k)}) \\
\Rightarrow & (u_1^{(k+1)}, \ldots, u_n^{(k+1)}) = (\omega_1, \ldots, \omega_n),
\end{array}
$$

where for every $i = 1, \ldots, n$ and for every $j = 1, \ldots, k+1$, $u_i^{(j)} \in \widehat{T}_i^*$, $A_i^{(j)} \in \widehat{N}_i$, $v_i^{(j)} \in (\widehat{T}_i \cup \widehat{N}_i)^*$, and $u_i^{(j)} \omega_i^{(j)} = \omega_i$. Hence, Claim 5.10 holds. □

From Claim 5.8, 5.9, and 5.10, $n\text{-}L(\vartheta) = n\text{-}L(\widehat{\Gamma})$. Therefore, Lemma 5.7 holds. ■

5.2 n-Accepting Move-Restricted Automata Systems

The second variant of n-accepting automata systems studied in this publication are n-accepting move-restricted automata systems. In contrast to n-accepting state-restricted automata systems, all PDAs in the n-accepting move-restricted automata systems work during whole computation. The restriction is based on limitation of transition rules that PDAs can select for a common computation step.

Definition 5.11 (n-Accepting move-restricted automata system)
Let $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, s_i, z_{i,0}, \emptyset)$ be a PDA for all $i = 1, \ldots, n$. Then, an n-accepting move-restricted automata system, n-MAS for short, is defined as $\vartheta = (M_1, \ldots, M_n, \Psi)$, where Ψ is a finite set of n-tuples of the form $(r_1, \ldots, r_n)$ and for each $j = 1, \ldots, n$, $r_j \in \delta_i$.

Definition 5.12 (n-Configuration of n-MAS)
Let $\vartheta = (M_1, \ldots, M_n, \Psi)$ be an n-MAS and $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, s_i, z_{i,0}, \emptyset)$ for all $i = 1, \ldots, n$. Then, n-configuration is defined as an n-tuple $\chi = (x_1, \ldots, x_n)$, where for all $i = 1, \ldots, n$, $x_i \in \Gamma_i^* Q_i \Sigma^*$ is a configuration of M_i.

Definition 5.13 (Computation step in n-MAS)
Let $\vartheta = (M_1, \ldots, M_n, \Psi)$ be an n-MAS and for each $i = 1, \ldots, n$, $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, s_i, z_{i,0}, \emptyset)$. Let

- $\chi = (z_1\gamma_1 q_1 a_1 \omega_1, \ldots, z_n\gamma_n q_n a_n \omega_n)$ and
- $\chi' = (z_1\gamma_1' q_1' \omega_1, \ldots, z_n\gamma_n' q_n' \omega_n)$,

be two n-configurations and for all $i = 1, \ldots, n$, $q_i, q_i' \in Q_i$, $\gamma_i', z_i \in \Gamma_i^*$, $\gamma_i \in \Gamma$, $\omega_i \in \Sigma^*$, $a_i \in \Sigma \cup \{\varepsilon\}$, $r_i = \gamma_i q_i a_i \to \gamma_i' q_i' \in \delta_i$ and $(r_1, \ldots, r_n) \in \Psi$. Then, ϑ makes a computation step from n-configuration χ to χ', denoted $\chi \vdash \chi'$, and in the standard way, $\vdash^*$ and $\vdash^+$ denote the reflexive-transitive and the transitive closure of $\vdash$, respectively.

Informally, all PDAs in n-MAS have to make moves at the same time and combination of transition rules concurrently applied by PDAs have to be permitted by Ψ.

Definition 5.14 (n-Language of n-MAS)
Let $\vartheta = (M_1, \ldots, M_n, \Psi)$ be an n-MAS and for every $i = 1, \ldots, n$, $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, s_i, z_{i,0}, \emptyset)$ be a pushdown automaton accepting input strings by empty pushdown. Let

- $\chi_0 = (z_{1,0} s_1 \omega_1, \ldots, z_{n,0} s_n \omega_n)$ be the start n-configuration and
- $\chi_f = (q_1, \ldots, q_n)$ be a final n-configuration of n-MAS,

where for all $i = 1, \ldots, n$, $q_i \in Q_i$, $\omega_i \in \Sigma^*$. The n-language of n-MAS is defined as $n\text{-}L(\vartheta) = \{(\omega_1, \ldots, \omega_n) |\ \chi_0 \vdash^* \chi_f\}$.

Example 5.15
Consider a 2-MAS $\vartheta = (M_1, M_2, \Psi)$, where

- $M_1 = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{\triangleright, a\}, \delta_1,\ q_0, \triangleright, \emptyset)$,
- $M_2 = (\{q_0\}, \{a, b\}, \{\triangleright, a\}, \delta_2,\ q_0, \triangleright, \emptyset)$,

1.PDA	2.PDA	Rule(s)
$\triangleright q_0 aabbcc$	$\triangleright q_0 aabb$	
$\triangleright a q_0 abbcc$	$\triangleright a q_0 abb$	(r_1, r_1)
$\triangleright aa q_0 bbcc$	$\triangleright aa q_0 bb$	(r_2, r_2)
$\triangleright a q_1 bcc$	$\triangleright a q_0 b$	(r_3, r_3)
$\triangleright q_1 cc$	$\triangleright a q_0 b$	(r_4, r_4)
$\triangleright q_2 c$	$\triangleright a q_0 b$	(r_5, r_4)
$\triangleright q_2$	$\triangleright q_0$	(r_6, r_3)
q_2	q_0	(r_7, r_7)

Table 5.2: Acceptance of $(aabbcc, aabb)$ by 2-MAS from Example 5.15

- $\delta_1 = \{ \quad r_1 = \triangleright q_0 a \to \triangleright a q_0,$
  $r_2 = a q_0 a \to a a q_0,$
  $r_3 = a q_0 b \to q_1,$
  $r_4 = a q_1 b \to q_1,$
  $r_5 = \triangleright q_1 c \to \triangleright q_2,$
  $r_6 = \triangleright q_2 c \to \triangleright q_2,$
  $r_7 = \triangleright q_2 \to q_2$
  $\}$

  $\delta_2 = \{ \quad r_1 = \triangleright q_0 a \to \triangleright a q_0,$
  $r_2 = a q_0 a \to a a q_0,$
  $r_3 = a q_0 b \to q_0,$
  $r_4 = a q_0 \to a q_0,$
  $r_5 = \triangleright q_0 \to \triangleright q_0$
  $r_6 = a q_0 b \to q_0,$
  $r_7 = \triangleright q_0 \to q_0$
  $\}$

- $\Psi = \{(r_1, r_1), (r_2, r_2), (r_3, r_3), (r_4, r_4), (r_4, r_5), (r_5, r_4), (r_5, r_4), (r_6, r_3), (r_7, r_7)\}$

The automata system works in a similar way like the n-SAS in Example 5.5 does. Both PDAs start with pushing as on their pushdowns. As soon as bs are the first input symbols in both automata, the PDAs remove as from the pushdowns and M_1 moves to q_1. In the state q_1, M_1 compares number of as and bs while M_2 is cycling by transition rule r_4 or r_5. After that, if the number of as and bs coincides, M_1 moves to q_2 with reading c. By using rules r_6 and r_3 in M_1 and M_2, respectively, M_1 reads cs from the input of M_1 while M_2 is comparing number of as and bs. The last step of the successful computation is done by rule r_7 in M_1 and M_2 at the same time. The 2-language that ϑ accepts is 2-$L(\vartheta) = \{(a^m b^m c^m, a^m b^m) |\ m \in \mathbb{N}\}$. Example of acceptance of the 2-string $(aabbcc, aabb)$ is shown in Table 5.2.

By Algorithm 5.16, it is shown that for any n-SAS, we are able to construct an n-MAS that define the same n-language as the origin n-SAS does. Pushdown automata of the n-MAS are defined in the same way as in n-SAS, but each of these automata, in addition, includes empty loops for all states. By the set Ψ, n-MAS determines that each automaton has to use this rule if the automaton is suspended in the n-SAS.

Algorithm 5.16
Construction of n-MAS from n-SAS

Input: n-MAS $\vartheta = (M_1, \ldots, M_n, \Psi)$, where for all $i = 1, \ldots, n$, $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, s_i, z_i, \emptyset)$ is a PDA accepting by empty pushdown.

Output: n-SAS $\widehat{\vartheta} = (\widehat{M}_1, \ldots, \widehat{M}_n, \widehat{\Psi})$, where for all $i = 1, \ldots, n$, $\widehat{M}_i = (\widehat{Q}_i, \Sigma, \widehat{\Gamma}_i, \widehat{\delta}_i, \widehat{s}_i, \#, \emptyset)$ is a PDA accepting by empty pushdown and n-$L(\vartheta) = n$-$L(\widehat{\vartheta})$.

Method:

- For all $i = 1, \ldots, n$, set $\widehat{Q}_i = Q_i \cup \{\widehat{s}_i\}$ and $\widehat{\Gamma}_i = \Gamma_i \cup \{\#\}$.
- The sets $\widehat{\delta}_i$ is constructed in the following way:
  1. set $\widehat{\delta}_i = \emptyset$,
  2. add $\#\widehat{s}_i \to \#z_i s_i$ to $\widehat{\delta}_i$, where z_i is the initial pushdown symbol of pushdown automaton M_i,
  3. for every $r \in \delta_i$, add r to $\widehat{\delta}_i$,
  4. for every $q \in Q_i$ and for every $a \in \widehat{\Gamma}_i$, add $r_{aq} = aq \to aq$ to $\widehat{\delta}_i$,
  5. for every $q \in Q_i$, add $\#q \to q$ to $\widehat{\delta}_i$.
- $\widehat{\Psi} =$

$$
\begin{aligned}
&\{(p_1, \ldots, p_n) | \\
&\quad * \ ((z_1 s_1 \omega_1)^e, \ldots, (z_n s_n \omega_n)^e) \vdash^* ((\gamma_1 q_1)^{\hbar_1}, \ldots, (\gamma_n q_n)^{\hbar_n}) \text{ in } \vartheta, \\
&\quad * \text{ for all } i = 1, \ldots, n, \ \#y_i q_i a_i \vdash \#\alpha_1 q_1'[p_i] \text{ in } \widehat{M}_i, \text{ where } p_i = y_i q_i \to y_i q_i \text{ and} \\
&\qquad a_i = \varepsilon \text{ if } \hbar_i = d; \text{ otherwise } p_i = y_i q_i a_i \to \alpha_i q_i', \\
&\quad \text{with } \omega_i \in \Sigma^*, \ q_i, q_i' \in Q_i, \ y_i \in \widehat{\Gamma}, \ \alpha_i \in \Gamma^*, \ a_i \in \Sigma \cup \{\varepsilon\}\} \\
\cup\ &\{(\#q_1 \to q_1, \ldots, \#q_n \to q_n) | \ \text{ for all } q_i \in Q_i\} \\
\cup\ &\{(\#\widehat{s}_1 \to \#z_1 s_1, \ldots, \#\widehat{s}_n \to \#z_n s_n)\}.
\end{aligned}
$$

Lemma 5.17 Algorithm 5.16 is correct.

Proof of Lemma 5.17. For the proof, we establish the following claims.

Claim 5.18 Let $((z_1 s_1 \omega_1)^{\hbar_1}, \ldots, (z_n s_n \omega_n)^{\hbar_n}) \vdash^* ((\gamma_1 q_1 \omega_1')^{\hbar_1'}, \ldots, (\gamma_n q_n \omega_n')^{\hbar_n'})$ in ϑ, then $(\#z_1 s_1 \omega_1, \ldots, \#z_n s_n \omega_n) \vdash^* (\#\gamma_1 q_1 \omega_1', \ldots, \#\gamma_n q_n \omega_n')$ in $\widehat{\vartheta}$.

Proof of Claim 5.18. By induction on the number of computation steps.

Basis:
Let $((z_1 s_1 \omega_1)^{\hbar_1}, \ldots, (z_n s_n \omega_n)^{\hbar_n}) \vdash^0 ((\gamma_1 q_1 \omega_1')^{\hbar_1'}, \ldots, (\gamma_n q_n \omega_n')^{\hbar_n'})$ in ϑ. Visibly, $((z_1 s_1 \omega_1)^{\hbar_1}, \ldots, (z_n s_n \omega_n)^{\hbar_n}) = ((\gamma_1 q_1 \omega_1')^{\hbar_1'}, \ldots, (\gamma_n q_n \omega_n')^{\hbar_n'})$ and $(\#\gamma_1 q_1 \omega_1', \ldots, \#\gamma_n q_n \omega_n') \vdash^0 (\#\gamma_1 q_1 \omega_1', \ldots, \#\gamma_n q_n \omega_n')$ in $\widehat{\vartheta}$.

Induction hypothesis:
Suppose that Claim 5.18 holds for j or fewer computation steps.

Induction step:
Let

$$
\begin{aligned}
& ((z_1 s_1 \omega_1)^{\hbar_1}, \ldots, (z_n s_n \omega_n)^{\hbar_n}) \vdash^j ((\gamma_1 q_1 \omega_1')^{\hbar_1'}, \ldots, (\gamma_n q_n \omega_n')^{\hbar_1'}) \\
\vdash\ & ((\gamma_1' q_1' \omega_1'')^{\hbar_1''}, \ldots, (\gamma_n' q_n' \omega_n'')^{\hbar_1''})
\end{aligned}
$$

in ϑ. By the induction hypothesis

$$(\#z_1 s_1 \omega_1, \ldots, \#z_n s_n \omega_n) \vdash^* (\#\gamma_1 q_1 \omega_1', \ldots, \#\gamma_n q_n \omega_n')$$

in $\widehat{\vartheta}$. For every $i = 1, \ldots, n$ such that:

- $h_i' = d$, $(\gamma_i q_i \omega_i')^{\hbar_i'} = (\gamma_i' q_i' \omega_i'')^{\hbar_i''}$. By 4. of $\widehat{\delta}_i$, there is a rule $r_i \in \widehat{\delta}_i$, where $\#\gamma_i q_i \omega_i' \vdash \#\gamma_i q_i \omega_i'[r_i]$ in $\widehat{M}_i$.

- $h'_i = e$, $\gamma_i q_i \omega'_i \vdash \gamma'_i q_i \omega''_i[r_i]$ in M_i. However, if $r_i \in \delta_i$, then $r_i \in \widehat{\delta}_i$ and $\#\gamma_i q_i \omega'_i \vdash \#\gamma'_i q'_i \omega''_i[r_i]$ in $\widehat{M}_i$.

Furthermore, $(r_1, \ldots, r_n) \in \widehat{\Psi}$. Therefore, from $\widehat{\delta}_i$ and $\widehat{\Psi}$, $(\#\gamma_1 q_1 \omega'_1, \ \ldots, \#\gamma_n q_n \omega'_n) \vdash (\#\gamma'_1 q'_1 \omega''_1, \ldots, \#\gamma'_n q'_n \omega''_n)$ in $\widehat{\vartheta}$. Claim 5.18 holds. □

Claim 5.19 Let $(\#z_1 s_1 \omega_1, \ldots, \#z_n s_n \omega_n) \vdash^* (\#\gamma_1 q_1 \omega'_1, \ldots, \#\gamma_n q_n \omega'_n)$ in n-MAS $\widehat{\vartheta}$, then $((z_1 s_1 \omega_1)^{\hbar_1}, \ldots, (z_n s_n \omega_n)^{\hbar_n}) \vdash^* ((\gamma_1 q_1 \omega'_1)^{\hbar'_1}, \ldots, (\gamma_n q_n \omega'_n)^{\hbar'_n})$ in ϑ.

Proof of Claim 5.19. By induction on the number of computation steps.

Basis:
Let $(\#z_1 s_1 \omega_1, \ldots, \#z_n q_n \omega_n) \vdash^0 (\#\gamma_1 q_1 \omega'_1, \ldots, \#\gamma_n q_n \omega'_n)$ in $\widehat{\vartheta}$. Surely, $(z_1 s_1 \omega_1, \ldots, z_n q_n \omega_n) = (\gamma_1 q_1 \omega'_1, \ldots, \gamma_n q_n \omega'_n)$, and consequently, $((z q_1 \omega'_1)^{\hbar_1}, \ldots, (z_n q_n \omega'_n)^{\hbar_n}) \vdash^0 ((\gamma_1 q_1 \omega'_1)^{\hbar'_1}, \ldots, (\gamma_n q_n \omega'_n)^{\hbar'_n})$ in ϑ and $\hbar_i = \hbar'_i$ for every $i = 1, \ldots, n$.

Induction hypothesis:
Suppose that Claim 5.19 holds for j or fewer computation steps.

Induction step:
Let $(\#z_1 s_1 \omega_1, \ldots, \#z_n s_n \omega_n) \vdash^j (\#\gamma_1 q_1 \omega'_1, \ldots, \#\gamma_n q_n \omega'_n) \vdash (\#\gamma'_1 q'_1 \omega''_1, \ldots, \#\gamma'_n q'_n \omega''_n)$ in $\widehat{\vartheta}$. By the induction hypothesis $((z_1 s_1 \omega_1)^{\hbar_1}, \ldots, (z_n s_n \omega_n)^{\hbar_n}) \vdash^j ((\gamma_1 q_1 \omega'_1)^{\hbar'_1}, \ldots, (\gamma_n q_n \omega'_n)^{\hbar'_n})$ in ϑ. For every $i = 1, \ldots, n$ such that:

- $\#\gamma_i q_i \omega'_i \neq \#\gamma'_i q'_i \omega''_i$, there is a rule $r_i \in \widehat{\delta}_i$ constructed by 1 of $\widehat{\delta}_i$ such that $\gamma_i q_i \omega'_i \vdash \gamma'_i q'_i \omega''_i[r_i]$ in $\widehat{M}_i$. Therefore, $r_i \in \delta_i$ and from $\widehat{\Psi}$, $h'_i = e$. That is, $\gamma_i q_i \omega'_i \vdash \gamma'_i q'_i \omega''_i[r_i]$ in M_i.

- $\#\gamma_i q_i \omega'_i = \#\gamma'_i q'_i \omega''_i$, there must be a rule $r_i \in \widehat{\delta}_i$ such that $\#\gamma_i q_i \omega'_i \vdash \#\gamma'_i q'_i \omega''_i[r_i]$ in $\widehat{M}_i$. From the construction of $\widehat{\delta}_i$,$r_i \in \delta_i \Leftrightarrow \hbar'_i = e$. Therefore, either $\gamma_i q_i \omega'_i \vdash \gamma_i q_i \omega'_i[r_i]$ in M_i, or $\hbar_i = d$ and automaton M_i is blocked.

In this way, $((\gamma_1 q_1 \omega'_1)^{\hbar'_1}, \ldots, (\gamma_n q_n \omega'_n)^{\hbar'_n}) \vdash ((\gamma'_1 q'_1 \omega''_1)^{\hbar''_1}, \ldots, (\gamma'_n q'_n \omega''_n)^{\hbar''_n})$ in $\widehat{\vartheta}$. That is, Claim 5.19 holds. □

By 2. of $\widehat{\delta}_i$ and because $(\#\widehat{s}_1 \rightarrow \#z_1 s_1, \ldots, \#\widehat{s}_n \rightarrow \#z_n s_n) \in \widehat{\Psi}$, the first step of $\widehat{\vartheta}$ is $(\#\widehat{s}_1 \omega_1, \ldots, \#\widehat{s}_n \omega_n) \vdash (\#z_1 s_1 \omega_1, \ldots, \#z_n s_n \omega_n)$. Similarly, the last step, for acceptance, is step of the form $(\#q_1, \ldots, q_n) \vdash (\#q_1, \ldots, q_n)$ because of 1 in construction of $\widehat{\delta}_i$ and $(\#q_1 \rightarrow q_1, \ldots, \#q_n \rightarrow q_n) \in \widehat{\Psi}$. Hence, from Claims 5.18 and 5.19, Lemma 5.17 holds. ■

It is well-known that for every PDA M, there is a context-free grammar G such that $L(M) = L(G)$. The conversion of a PDA into CFG is usually performed by Algorithm 5.20.

Algorithm 5.20
Construction of CFG from PDA

Input: A PDA $M = (Q_M, \Sigma_M, \Gamma_M, \delta_M, s_0, Z_0, \emptyset)$

Output: A CFG $G = (N_G, T_G, S_G, P_G)$ such that $L(M) = L(G)$

Method:

- Set $N_G = \{\langle qAp \rangle |\ p \in Q_M, A \in \Sigma_M\} \cup \{S\}$, $P_G = \{S \rightarrow \langle aAp \rangle |\ p \in Q_M\}$, $T_G = \Sigma_M$ and $i = 1$.

- For each $r = Aq_0a \to B_n \dots B_1 q_1 \in \delta_M$, where $a \in \Sigma_M \cup \{\varepsilon\}$, $q_0, q_1 \in Q_M$, $A, B_1, \dots, B_n \in \Sigma_M$, for some $n \geq 0$ do:
  - $\rho(r) = \{\langle q_0 A q_{n+1}\rangle \to a\langle q_1 B_1 q_2\rangle \dots \langle q_n B_n q_{n+1}\rangle |\ \ q_2, \dots, q_{n+1} \in Q_M\}$
  - Set $P_G = P_G \cup \rho(r)$ and $i = i + 1$.

Lemma 5.21 For any n-MAS ϑ, there is an n-CGR Γ such that $L(\vartheta) = L(\Gamma)$.

Proof of Lemma 5.21. Consider an n-MAS $\vartheta = (M_1, \dots, M_n, \Psi)$. It is indisputable that Algorithm 5.20 can be applied on each component of automata system and in this way, the components of n-CGR $\Gamma = (G_1, \dots, G_n, Q)$ can be constructed. Observe that there is a set of context-free rules $\rho_i(r)$ constructed for every transition rule r and each PDA M_i of ϑ, and for every $i = 1, \dots, n$, $L(M_i) = L(G_i)$. Furthermore, for every $i = 1, \dots, n$, G_i simulates M_i step by step (see p.486–491 in [80]). It is easy to see that $L(\Gamma) = L(\vartheta)$ for $\Gamma = (G_1, \dots, G_n, Q)$ where $Q = \{(r_1, \dots r_n) |\ (p_1, \dots, p_n) \in \Psi$ and for all $i = 1, \dots, n$, $r_i \in \rho_i(p_i)\}$. ■

This chapter is closed by the following theorem saying that the investigated systems define the same class of n-languages—in other words, the systems are equivalent.

Theorem 5.22 The classes of n-languages given by n-CGN, n-CGR, n-SAS, and n-MAS coincide.

Proof of Theorem 5.22. From [82], we know that the classes of n-languages generated by n-CGN and n-CGR coincide. Hence, from Lemma 5.7, Lemma 5.17, and from Lemma 5.21, Theorem 5.22 holds. ■

Chapter 6

Classes of n-Languages: Hierarchy and Properties

In the previous chapter, we have shown that multi-generative grammar systems have their twins in terms of automata, which can accept such n-languages that the grammar systems generate. However, only context-free grammars and pushdown automata as the components of systems have been considered. In the following sections, we generalize the theory of n-languages and discuss *hybrid canonical rule-synchronized n-generative grammar systems* and *hybrid n-accepting move-restricted automata systems*, where components with different generative and accepting power can be used in one grammar and automata system, respectively.

More specifically, in Section 6.1, we introduce grammar systems, which combine right-linear grammars, linear grammars, and context-free grammars; and automata systems, which combine finite automata, 1-turn pushdown automata, and pushdown automata in one instance. After that, in Section 6.2, the class hierarchy based on the mentioned models is established. In addition, several closure properties are discussed.

6.1 Definitions

Similarly to n-CGR, the hybrid canonical rule-synchronized n-generative grammar system consists of n generative components (grammars) and a control set of n-tuples of rules. Each grammar generates its own string in the leftmost way, while the control set determines which grammar rules can be used at the same derivation step in all components. In this work, we study combination of right-linear, linear, and context-free grammars (see Definition 6.1).

Definition 6.1 (Hybrid canonical rule-synchronized n-generative grammar system)
A *hybrid canonical rule-synchronized n-generative grammar system*, shortly $\mathrm{HCGR}^{(t_1,\ldots,t_n)}$, is an $n+1$-tuple $\Gamma = (G_1, \ldots, G_n, Q)$, where

- $G_i = (N_i, T_i, P_i, S_i)$ is a right-linear, linear, or context-free grammar for every $i = 1, \ldots, n$,
- Q is a finite set of n-tuples of the form $(r_1, \ldots, r_n)$, where $r_i \in P_i$ for every $i = 1, \ldots, n$, and
- for all $i = 1, \ldots, n$, $t_i \in \{\mathrm{RLNG}, \mathrm{LNG}, \mathrm{CFG}\}$ denotes type of ith component.

A *sentential n-form* of HCGR$^{(t_1,\ldots,t_n)}$ is an n-tuple $\chi = (x_1, \ldots, x_n)$, where $x_i \in (N_i \cup T_i)^*$ for all $i = 1, \ldots, n$.

Consider two sentential n-forms, $\chi = (u_1A_1v_1, \ldots, u_nA_nv_n)$ and $\chi' = (u_1x_1v_1, \ldots, u_nx_nv_n)$ with

- $A_i \in N_i$,
- $u_i \in T^*$,
- $v_i, x_i \in (N \cup T)^*$,
- $r_i = A_i \to x_i \in P_i$, for all $i = 1, \ldots, n$, and
- $(r_1, \ldots, r_n) \in Q$.

Then, $\chi \Rightarrow \chi'$, and $\Rightarrow^*$ and $\Rightarrow^+$ are its reflexive-transitive and transitive closure, respectively.

The *n-language of* Γ is defined as $n\text{-}L(\Gamma) = \{(w_1, \ldots, w_n) |\ (S_1, \ldots, S_n) \Rightarrow^* (w_1, \ldots, w_n), w_i \in T_i^*, \text{ for all } i = 1, \ldots, n\}$.

The generalized version of n-MAS are hybrid n-accepting move-restricted automata systems. In accordance with n-MAS (see Section 5.2), a hybrid n-accepting move-restricted automata system is a vector of automata working on their own input, together with a set of n-tuples of transition rules. This set restricts moves that the components can make in the same computation step. If and only if all components accept their input, the automata system accepts the n-tuple of these input strings. The generalized version allows combination of different types of automata in one automata system.

Definition 6.2 (Hybrid n-accepting move-restricted automata system)
A hybrid *n-accepting move-restricted automata system*, denoted by HMAS$^{(t_1,\ldots,t_n)}$, is defined as an $n+1$-tuple $\vartheta = (M_1 \ldots, M_n, \Psi)$ with M_i as a finite or (1-turn) pushdown automaton, for all $i = 1, \ldots, n$, and with Ψ as a finite set of n-tuples of the form $(r_1, \ldots, r_n)$, where for every $j = 1, \ldots, n$, $r_j \in \delta_j$ in M_j. Furthermore, for all $i = 1, \ldots, n$, $t_i \in$ {FA, 1-turn PDA, PDA} indicates the type of ith automaton.

An n-configuration is defined as an n-tuple $\chi = (x_1, \ldots, x_n)$, where for all $i = 1, \ldots, n$, x_i is a configuration of M_i. Let $\chi = (x_1, \ldots, x_n)$ and $\chi' = (x_1', \ldots, x_n')$ be two n-configurations, where for all $i = 1, \ldots, n$, $x_i \vdash x_i'\,[r_i]$ in M_i, and $(r_1, \ldots, r_n) \in \Psi$, then ϑ makes a computation step from χ to χ', denoted by $\chi \Rightarrow \chi'$, and in the standard way, $\vdash^*$ and $\vdash^+$ denote the reflexive-transitive and the transitive closure of $\vdash$, respectively.

Let $\chi_0 = (x_1\omega_1, \ldots, x_n\omega_n)$ be the start and $\chi_f = (q_1, \ldots, q_n)$ be a final n-configuration of HMAS$^{(t_1,\ldots,t_n)}$, where for all $i = 1, \ldots, n$, ω_i is the input string of M_i and q_i is state of M_i. The n-language of HMAS$^{(t_1,\ldots,t_n)}$ is defined as $n\text{-}L(\vartheta) = \{(\omega_1, \ldots, \omega_n) |\ \chi_0 \vdash^* \chi_f$ and for every $i = 1, \ldots, n$, M_i accepts$\}$.

Convention 6.3 In a special case, where all components are of type X, we write ${}_nX$ instead of $(X, \ldots, X)$. For example, a hybrid n-accepting move-restricted automata system, where all components are PDAs, is denoted by HMAS$^{n\,PDA}$. If there is no attention on the number and type of components, we write HMAS and HCGR rather than HMAS$^{(t_1,\ldots,t_n)}$ and HCGR$^{(t_1,\ldots,t_n)}$, respectively.

Definition 6.4 (Classes of n-Languages)
– $\mathscr{L}(\text{HMAS}^{(t_1,\ldots,t_n)})$ is the class of n-languages accepted by $\text{HMAS}^{(t_1,\ldots,t_n)}$
– $\mathscr{L}(\text{HCGR}^{(t_1,\ldots,t_n)})$ is the class of n-languages generated by $\text{HCGR}^{(t_1,\ldots,t_n)}$

6.2 Class Hierarchy and Closure Properties

We start this section by the following lemma, which says that for every $\text{HCGR}^{(t_1,\ldots,t_n)}$ containing RLNGs and CFGs, we can construct an $\text{HMAS}^{(t'_1,\ldots,t'_n)}$, including FAs and PDAs, that defines the same n-language as the $\text{HCGR}^{(t_1,\ldots,t_n)}$ does.

Lemma 6.5 For every $\text{HCGR}^{(t_1,\ldots,t_n)}$ $\widehat{\Gamma} = (\widehat{G}_1, \ldots, \widehat{G}_n, \widehat{Q})$, where $\widehat{G}_i$ is either RLNG or CFG for all $i = 1, \ldots, n$, there is an $\text{HMAS}^{(t'_1,\ldots,t'_n)}$ $\vartheta = (M_1, \ldots, M_n, \Psi)$ such that $n\text{-}L(\vartheta) = n\text{-}L(\widehat{\Gamma})$ and for all $i = 1, \ldots, n$, M_i is either FA or PDA.

Proof of Lemma 6.5. Consider an $\text{HCGR}^{(t_1,\ldots,t_n)}$ $\widehat{\Gamma} = (\widehat{G}_1, \ldots, \widehat{G}_n, \widehat{Q})$ and, by the following algorithm, construct $\text{HMAS}^{(t'_1,\ldots,t'_n)}$ $\vartheta = (M_1, \ldots, M_n, \Psi)$.

For every $i = 1, \ldots, n$,

- if $\widehat{G}_i = (\widehat{N}_i, \widehat{T}_i, \widehat{P}_i, \widehat{S}_i)$ is an RLNG, then construct FA $M_i = (Q_i, \Sigma_i, \delta_i, \langle \widehat{S}_i \rangle, F_i)$ in the following way:
  - set $Q_i = \{\langle A \rangle |\ A \in \widehat{N}_i\}$;
  - set $\Sigma_i = \widehat{T}_i$;
  - set $\delta_i = \{\langle A \rangle a \to \langle B \rangle |\ A \to aB \in \widehat{P}_i\} \cup \{\langle A \rangle \to \langle A \rangle |\ \langle A \rangle \in Q_i\} \cup \{\langle A \rangle a \to f_i |\ A \to a \in \widehat{P}_i\}$;
  - set $F_i = \{f_i\}$;
- if $\widehat{G}_i = (\widehat{N}_i, \widehat{T}_i, \widehat{P}_i, \widehat{S}_i)$ is a CFG, construct PDA $M_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, q'_i, \triangleright, \emptyset)$ as follows:
  - set $Q_i = \{q'_i, q_i\}$;
  - set $\Sigma_i = \widehat{T}_i$;
  - set $\Gamma_i = \widehat{T}_i \cup \widehat{N}_i$;
  - set $\delta_i = \{\triangleright q'_i \to \triangleright \widehat{S}_i q_i\} \cup \{Aq_i \to xq_i |\ A \to (x)^R \in \widehat{P}_i\} \cup \{Aq_i \to Aq_i |\ A \in \widehat{N}_i \cup \{\triangleright\}\} \cup \{aq_i a \to q_i |\ a \in \widehat{T}_i\} \cup \{\triangleright q_i \to q_i\}$;

The set Ψ is constructed in this way: set $\Psi = \{(\Diamond p_1 \to \Diamond t_1 p'_1, \ldots, \Diamond p_n \to \Diamond t_n p'_n)\} \cup \{(\Diamond \widehat{p}_1 \to \widehat{p}'_1, \ldots, \Diamond \widehat{p}_n \to \widehat{p}'_n\}$, where

- $\Diamond t_i = \varepsilon$, $p_i, p'_i = \langle \widehat{S}_i \rangle$, $\widehat{p}_i \in Q_i - \{f_i\}$, and $\widehat{p}'_i = f_i$ iff M_i is an FA, and $p_i, \widehat{p}_i, \widehat{p}'_i = q'_i$, $p'_i = q_i$, and $\Diamond t_i = \triangleright \widehat{S}_i$;
- otherwise, for all $r_i \in \delta_i$ such that r_i is given from some $r'_i \in \widehat{P}_i$ and $(r'_1, \ldots, r'_n) \in \widehat{Q}$, include $(r_1, \ldots, r_n)$ into Ψ;
- for all $r_i \in \delta_i$ such that r_i is not given from any $r'_i \in \widehat{P}_i$ and for at least one $i = 1, \ldots, n$, r_i is of the form $aq_i a \to q_i$ with $a \in \widehat{T}_i$, include $(r_1, \ldots, r_n)$ into Ψ;

Now, we can prove the following claims.

Claim 6.6 Let ϑ is automata system given from grammar system $\widehat{\Gamma}$ by the previous algorithm. If $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^* (u_1A_1v_1, \ldots, u_nA_nv_n)$ in $\widehat{\Gamma}$ with $A_i \in \widehat{N}_i$, for every $i = 1, \ldots, n$, and $(u_1A_1v_1, \ldots, u_nA_nv_n) \Rightarrow^* (\omega_1, \ldots, \omega_n)$, then $(x_1\omega_1, \ldots, x_n\omega_n) \vdash^* (\Diamond(v_1)^R t_1 p_1 \omega_1', \ldots, \Diamond(v_n)^R t_n p_n \omega_n')$ in ϑ, where for all $1 \leq i \leq n$, $\omega_i = u_i\omega_i'$, and if M_i is PDA, $x_i = \triangleright q_i'$, $t_i = A_i$, $p_i = q_i$, and $\Diamond = \triangleright$; otherwise, $x_i = \langle \widehat{S}_i \rangle$, $\Diamond t_i = \varepsilon$, and $p_i = \langle A_i \rangle$.

Proof of Claim 6.6. By induction hypothesis on the number of derivation steps.

Basis. Let $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^0 (\widehat{S}_1, \ldots, \widehat{S}_n)$ in $\widehat{\Gamma}$. Then, $(x_1\omega_1, \ldots, x_n\omega_n) = (\Diamond p_1\omega_1, \ldots, \Diamond p_n\omega_n) \vdash (\Diamond t_1 p_1 \omega_1, \ldots, \Diamond t_n p_n \omega_n)$ with $\Diamond t_i = \varepsilon$ if M_i is an FA; otherwise, $\Diamond t_i p_i = \triangleright \widehat{S}_i q_i$ and $x_i = \triangleright q_i'$. The claim holds.

Induction Hypothesis. Suppose that Claim 6.6 holds for j or fewer derivation steps, where j is a non-negative integer.

Induction Step. Consider derivation of the form $(\widehat{S}_1, \ldots, S_n) \Rightarrow^{j+1} (u_1'A_1v_1', \ldots, u_n'A_nv_n')$ in $\widehat{\Gamma}$, where $(u_1'A_1v_1', \ldots, u_n'A_nv_n') \Rightarrow^* (\omega_1, \ldots, \omega_n)$. We can express this derivation as $(\widehat{S}_1, \ldots, S_n) \Rightarrow^j (u_1B_1v_1, \ldots, u_nB_nv_n) \Rightarrow (u_1'A_1v_1', \ldots, u_n'A_nv_n')$. By the induction hypothesis, $(x_1\omega_1, \ldots, x_n\omega_n) \vdash^* (\Diamond(v_1)^R t_1 p_1 \omega_1', \ldots, \Diamond(v_n)^R t_n p_n \omega_n')$, where for all $i = 1, \ldots, n$, $\omega_i = u_i\omega_i'$, and if M_i is PDA, $x_i = \triangleright \widehat{S}_i q_i$, $t_i = B_i$, and $p_i = q_i$; otherwise, $x_i = \langle \widehat{B}_i \rangle$, $t_i = \varepsilon$, and $p_i = \langle B_i \rangle$. Because $(u_1B_1v_1, \ldots, u_nB_nv_n) \Rightarrow^1 (u_1'A_1v_1', \ldots, u_n'A_nv_n')$, there have to be n rules, $r_1', \ldots, r_n'$, such that $u_iB_iv_i \Rightarrow u_i'A_iv_i'[r_i']$ in $\widehat{G}_i$. From the construction of ϑ, there are n rules, $r_1, \ldots, r_n$, created from r_i' in δ_i. These rules are of the form $r_i = \langle B_i \rangle a \to \langle A_i \rangle$ if M_i is FA, or $r_i = B_iq_i \to x_iq_i$, otherwise. As $(u_1B_1v_1, \ldots, u_nB_nv_n) \Rightarrow^1 (u_1'A_1v_1', \ldots, u_n'A_nv_n')$ in $\widehat{\Gamma}$, $(r_1, \ldots, r_n) \in \widehat{Q}$ and $(r_1', \ldots, r_n') \in \Psi$. Hence, for all $i = 1, \ldots, n$ and $\omega_i' = a_i\omega_i''$ with $a_i \in \Sigma_i \cup \{\varepsilon\}$, if M_i is PDA, $\triangleright(v_i)^R B_i q_i \omega_i' \vdash \triangleright(x_i'v_i')^R q_i \omega_i'' = \triangleright(v_i')^R A_i (u_i')^R q_i \omega_i''$; otherwise, $\langle B_i \rangle a_i \omega_i'' \vdash \langle A_i \rangle$. After this step, some PDAs can have a terminal symbol as the topmost pushdown symbol. In this case, from the construction of Ψ, these PDAs have to compare topmost pushdown symbols with their inputs until no terminal symbol is on the top of their pushdown. The other automata loop and wait until no PDA is reading its input. Therefore, $(\Diamond(v_1)^R t_1 p_1 \omega_1', \ldots, \Diamond(v_n)^R t_n p_n \omega_n') \vdash^* (\Diamond(v_1')^R t_1' p_1' \omega_1'', \ldots, \Diamond(v_n')^R t_n' p_n' \omega_n'')$ in ϑ, where $t_i' = A_i$ and $p_i' = q_i$ in case of PDA; otherwise, $t_i' = \varepsilon$ and $p_i' = \langle A_i \rangle$. Claim 6.6 holds. □

Claim 6.7 Let ϑ is automata system given from grammar system $\widehat{\Gamma}$ by the previous algorithm. If $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^* (\omega_1, \ldots, \omega_n)$ in $\widehat{\Gamma}$ with $\omega_i \in \widehat{T}_i$ for every $i = 1, \ldots, n$, then $(x_1\omega_1, \ldots, x_n\omega_n) \vdash^* (\Diamond p_1, \ldots, p_n)$ in ϑ, where for all $i = 1, \ldots, n$, if M_i is PDA, $p_i = q_i$; otherwise, $p_i = f_i$.

Proof of Claim 6.7. Consider that $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^* (u_1A_1v_1, \ldots, u_nA_nv_n) \Rightarrow (u_1x_1v_1, \ldots, u_nx_nv_n)$ in $\widehat{\Gamma}_i$. Then, $u_ix_iv_i \in \widehat{T}_i^*$ for all $i = 1, \ldots, n$. By Claim 6.6, $(x_1\omega_1, \ldots, x_n\omega_n) \vdash^* (\Diamond(v_1)^R t_1 p_1 \omega_1', \ldots, \Diamond(v_n)^R t_n p_n \omega_n')$ in ϑ, where for all $i = 1, \ldots, n$, $\omega_i = u_i\omega_i'$, and if M_i is PDA, $x_i = \triangleright q_i'$, $t_i = A_i$, $p_i = q_i$, and $\Diamond = \triangleright$; otherwise, $x_i = \langle \widehat{S}_i \rangle$, $\Diamond t_i = \varepsilon$, and $p_i = \langle A_i \rangle$. Because $r_i = A_i \to x_i \in P_i$ for all $i = 1, \ldots, n$, and $(r_1, \ldots, r_n) \in \widehat{Q}$, there is $r_i' = A_iq_i \to (x_i)^R q_i \in \delta_i$ for every pushdown automaton, M_i, $r_i' = \langle A_i \rangle x_i \to f_i$ for every FA, M_i, and $(r_1', \ldots, r_n') \in \Psi$. Therefore, the next configuration of each PDA and FA, M_i, is of the form $\triangleright(x_iv_i)^R q_i x_i v_i$ and f_i, respectively. By the definition of Ψ, each PDA with terminal symbol on the pushdown's top makes a move by $aq_ia \to q_i$ until $\triangleright$ is the first pushdown symbol. The others loop and read no symbol. As all PDAs have $\triangleright$ on the pushdown top, $\triangleright$s are removed and ϑ accepts. Hence, Claim 6.7 holds. □

Claim 6.8 Let ϑ is automata system given from grammar system $\widehat{\Gamma}$ by the previous algorithm. If $(x_1\omega_1, \ldots, x_n\omega_n) \vdash^* (\Diamond(v_1)^R t_1 p_1, \ldots, \Diamond(v_n)^R t_n p_n)$ in ϑ, where for all $i = 1, \ldots, n$, $\omega_i = u_i\omega_i'$, and if M_i is PDA, $x_i = \triangleright\widehat{S}_i q_i$, $t_i = A_i$, $p_i = q_i$, and $\Diamond = \triangleright$; otherwise, $x_i = \langle\widehat{S}_i\rangle$, $\Diamond, t_i = \varepsilon$, and $p_i = \langle A_i\rangle$, then $(\widehat{S}_1, \ldots, \widehat{S}_n) \Rightarrow^* (\omega_1, \ldots, \omega_n)$ in $\widehat{\Gamma}$ with $u_i \in T^*, A_i \in \widehat{N}_i, v_i \in (\widehat{T}_i \cup \widehat{N}_i)^*$, for every $i = 1, \ldots, n$.

Proof of Claim 6.8. Consider any successful acceptance:

$$(x_1\omega_1, \ldots, x_n\omega_n) \vdash^* (\Diamond p_1, \ldots, \Diamond p_n), \tag{II}$$

where for all $i = 1, \ldots, n$, and if M_i is PDA, $x_i = \triangleright q_i'$, $p_i = q_i$, and $\Diamond = \triangleright$; otherwise, $x_i = \langle\widehat{S}_i\rangle$, $\Diamond = \varepsilon$, $p_i = \langle A_i\rangle$, and $p_i \in \widehat{F}_i$. The automata system starts with initialization of PDAs, i.e. after the first computation step, each PDA, M_i, must have configuration $\triangleright\widehat{S}_i q_i \omega_i$, while the configuration of all FAs are unchanged. By the construction of Ψ, the PDAs have to rewrite their nonterminals on the top of their pushdowns at the same time and it can do it only if every PDA has a nonterminal symbol on the pushdown top. Otherwise, all PDAs having terminal symbols on the pushdown tops compare inputs with the terminal symbols, while the others are looping with reading no symbol. The same holds for $\triangleright$ that ϑ can remove at the same time and only if the FAs are in final states. Therefore, (II) can be expressed as
$(x_1\omega_1^{(0)}, \ldots, x_n\omega_n^{(0)}) = (\Diamond\gamma_1^{(1)} t_1^{(1)} p_1^{(1)} \omega_1^{(1)}, \ldots, \Diamond\gamma_n^{(1)} t_n^{(1)} p_n^{(1)} \omega_n^{(1)})$
$\vdash^* (\Diamond\gamma_1^{(2)} t_1^{(2)} p_1^{(2)} \omega_1^{(2)}, \ldots, \Diamond\gamma_n^{(2)} t_n^{(2)} p_n^{(2)} \omega_n^{(2)})$
$\vdash^* \ldots$
$\vdash^* (\Diamond\gamma_1^{(k-1)} t_1^{(k-1)} p_1^{(k-1)} \omega_1^{(k-1)}, \ldots, \Diamond\gamma_n^{(k-1)} t_n^{(k-1)} p_n^{(k-1)} \omega_n^{(k-1)})$
$\vdash^* (\Diamond p_1^{(k)}, \ldots, \Diamond p_n^{(k)}) \vdash (p_1^{(k)}, \ldots, p_n^{(k)})$, where for all $i = 1, \ldots, n$, and for all $j = 1, \ldots, k$, $\omega_i^{(j-1)} = u_i^j \omega_i^{(j)}$ and: if M_i is a PDA, then $x_i = \triangleright q_i'$, $\Diamond\gamma_i^{(j)} t_i^{(j)} \in \{\triangleright\}(\widehat{N}_i \cup \widehat{T}_i)^* \widehat{N}_i$, and $p_i^{(j)} = q_i$; otherwise, $\Diamond\gamma_i^{(j)} = \varepsilon$, $p_i^{(j)} = \langle A_i^{(j)}\rangle$. The computation of ϑ can be simulated by $\widehat{\Gamma}$ as:

$$\begin{aligned}
& (\widehat{S}_1, \ldots, \widehat{S}_n) \\
\Rightarrow\ & (u_1^{(1)} A_1^{(1)} v_1^{(1)}, \ldots, u_n^{(1)} A_n^{(1)} v_n^{(1)}) \\
\Rightarrow\ & (u_1^{(2)} A_1^{(2)} v_1^{(2)}, \ldots, u_n^{(2)} A_n^{(2)} v_n^{(2)}) \\
& \vdots \\
\Rightarrow\ & (u_1^{(k)} A_1^{(k-1)} v_1^{(k-1)}, \ldots, u_n^{(k-1)} A_n^{(k)} v_n^{(k-1)}) \\
\Rightarrow\ & (u_1^{(k)}, \ldots, u_n^{(k)}) = (\omega_1, \ldots, \omega_n),
\end{aligned}$$

where for every $i = 1, \ldots, n$ and every $j = 1, \ldots, k$, $u_i^{(j)} \in \widehat{T}_i^*$, $A_i^{(j)} \in \widehat{N}_i$, $v_i^{(j)} \in (\widehat{T}_i \cup \widehat{N}_i)^*$ and $u_i^{(j)}\omega_i^{(j)} = \omega_i$. Hence, Claim 6.8 holds. □

From Claim 6.6, Claim 6.7, and Claim 6.8, Lemma 6.5 holds. ■

To prove that $\mathrm{HCGR}^{(t_1,\ldots,t_n)}$, based on RLNGs and CFGs, and $\mathrm{HMAS}^{(t_1',\ldots,t_n')}$, based on FAs and PDAs, are equivalent, Lemma 6.9 is needed.

Lemma 6.9 For every $\mathrm{HMAS}^{(t_1',\ldots,t_n')}$ $\vartheta = (M_1, \ldots, M_n, \Psi)$, where components are FAs and PDAs, there is an $\mathrm{HCGR}^{(t_1,\ldots,t_n)}$ $\widehat{\Gamma} = (\widehat{G}_1, \ldots, \widehat{G}_n, \widehat{Q})$ such that $n\text{-}L(\vartheta) = n\text{-}L(\widehat{\Gamma})$ and for all $i = 1, \ldots, n$, $\widehat{G}_i$ is either RLNG or CFG.

Proof of Lemma 6.9. It is well-known that for every PDA and FA M, there is a CFG and RLNG G such that $L(M) = L(G)$, respectively. The algorithm, that we use for construction of CFG $G = (N_G, T_G, P_G, S_G)$ from a PDA $M = (Q_M, \Sigma_M, \Gamma_M, \delta_M, s_0, Z_0, \emptyset)$ is defined in the following way:

- set $N_G = \{\langle qAp\rangle |\ p, q \in Q_M, A \in \Gamma_M\} \cup \{S\}$, $P_G = \{S \to \langle aAp\rangle |\ p \in Q_M\}$, and $T_G = \Sigma_M$
- for each $r = Aq_0a \to B_n \dots B_1 q_1 \in \delta_M$, where $a \in \Sigma_M \cup \{\varepsilon\}$, $q_0, q_1 \in Q_M$, $A, B_1, \dots, B_n \in \Gamma_M$, for some $n \geq 0$ do:
  - set $\rho(r) = \{\langle q_0 A q_{n+1}\rangle \to a\langle q_1 B_1 q_2\rangle \dots \langle q_n B_n q_{n+1}\rangle |\ \ q_2, \dots, q_{n+1} \in Q_M\}$;
  - set $P_G = P_G \cup \rho(r)$;

Similarly, RLNG $G = (N_G, T_G, P_G, S_G)$ constructed from FA $M = (Q_M, \Sigma_M, \delta_M, s_0, F_M)$ has form:

- set $N_G = \{A |\ A \in Q_M\}$, $S_G = q_0$, $P_G = \emptyset$, and $T_G = \Sigma_M$;
- for each $r = Aa \to B \in \delta_M$, where $a \in \Sigma_M \cup \{\varepsilon\}$, $A, B \in Q_M$ do:
  - set $\rho(r) = \{A \to aB\}$;
  - if $B \in F_M$, then $\rho(r) = \rho(r) \cup \{A \to a\}$;
  - set $P_G = P_G \cup \rho(r)$;

Consider an HMAS$^{(t'_1,\dots,t'_n)}$ $\vartheta = (M_1, \dots, M_n, \Psi)$. It is clear that the algorithms can be applied on each component of automata system, and in this way, components of n-generative grammar system $\Gamma = (G_1, \dots, G_n, Q)$ can be constructed. Note, there is a set of grammar rules $\rho_i(r)$ constructed for every transition rule r and each automaton M_i of the automata system, and for every $i = 1, \dots, n$, $L(M_i) = L(G_i)$. Furthermore, for every $i = 1, \dots, n$, G_i simulates M_i step by step (see p.486–491 in [80]). It is easy to see that $L(\Gamma) = L(\vartheta)$ for $\Gamma = (G_1, \dots, G_n, Q)$ where $Q = \{(r_1, \dots r_n) |\ (p_1, \dots, p_n) \in \Psi$ and for all $i = 1, \dots, n$, $r_i \in \rho_i(p_i)\}$. Lemma 6.9 holds. ■

Finally, Theorem 6.10 can be established.

Theorem 6.10 $\mathscr{L}(\text{HMAS}^{(t_1,\dots,t_n)}) = \mathscr{L}(\text{HCGR}^{(t'_1,\dots,t'_n)})$, where $t_i \in \{\text{FA, PDA}\}$ and $t'_i \in \{\text{CFG, RLNG}\}$ for all $i = 1, \dots, n$.

Proof of Theorem 6.10. It directly follows from Lemma 6.5 and Lemma 6.9. ■

In contrast to ordinary FAs that are closed over union, intersection, complementation, and concatenation, the class of n-languages accepted by HMASs consisting of finite automata is closed only over union and concatenation.

Theorem 6.11 If L_1 and $L_2 \in \mathscr{L}(\text{HMAS}^n\text{FA})$, then $L_1 \cup L_2 \in \mathscr{L}(\text{HMAS}^n\text{FA})$.

Proof of Theorem 6.11. Consider two n-languages, $L_1, L_2 \in \mathscr{L}(\text{HMAS}^n\text{FA})$. Then there are n-HMAS^{n}FAs, $\vartheta_1 = (M_{1,1}, \dots, M_{1,n}, \Psi_1)$ and $\vartheta_2 = (M_{2,1}, \dots, M_{2,n}, \Psi_2)$, such that $L_1 = L(\vartheta_1)$, $L_2 = L(\vartheta_2)$, and for all $i = 1, 2$ and $j = 1, \dots, n$, $M_{i,j} = (Q_{i,j}, \Sigma_{i,j}, \delta_{i,j}, s_{i,j}, F_{i,j})$

is a component of ϑ_i. For these two automata systems, we can construct $\vartheta_{12} = (M_{12,1}, \ldots, M_{12,n}, \Psi_{12})$, with $M_{12,j} = (Q_{12,j}, \Sigma_{12,j}, \delta_{12,j}, s_{12,j}, F_{12,j})$, in the following way: for all $i = 1, 2$ and $j = 1, \ldots, n$,

- $Q_{12,j} = Q_{1,j} \cup Q_{2,j} \cup \{s_{12,j}\}$, where $s_{12,j}$ is the new start state of jth automaton,
- $\delta_{12,j} = \delta_{1,j} \cup \delta_{2,j} \cup \{p_{1,j} = s_{12,j} \to s_{1,j}, p_{2,j} = s_{12,j} \to s_{2,j}\}$,
- $\Sigma_{12,j} = \Sigma_{1,j} \cup \Sigma_{2,j}$,
- $F_{12,j} = F_{1,j} \cup F_{2,j}$, and
- $\Psi_{12} = \Psi_1 \cup \Psi_2 \cup \{(p_{1,1}, \ldots, p_{1,n}), (p_{2,1}, \ldots, (p_{2,n})\}$.

From set Ψ_{12} it follows that the first computation step has to be $(s_{12,1}\omega_1, \ldots, s_{12,n}\omega_n) \vdash (s_{i,1}\omega_1, \ldots, s_{i,n}\omega_n)$ for $i = 1, 2$, and for $\omega_j \in \Sigma_{12,j}$ with $j = 1, \ldots, n$. Therefore, $(\omega_1, \ldots, \omega_n)$ is in $L(\vartheta_{12})$ iff $(\omega_1, \ldots, \omega_n) \in L(\vartheta_1)$ or $(\omega_1, \ldots, \omega_n) \in L(\vartheta_2)$—that is, $(\omega_1, \ldots, \omega_n) \in L(\vartheta_{12})$ iff $(\omega_1, \ldots, \omega_n) \in L_1 \cup L_2$. ■

Lemma 6.12 For $n \geq 2$, an n-language, n-L, defined as n-$L = \{(a^i b^j, a^j b^i, \varepsilon, \ldots, \varepsilon) |\ i, j \in \mathbb{N}_0\}$ is not in $\mathscr{L}(\text{HMAS}^n\text{FA})$.

Proof of Lemma 6.12. From the definition of n-accepting move-restricted finite automata system, it can be seen that only possibility to compare symbols through components is read them step by step at the same time (or in a quasi parallel way). Hence, for comparing as in the first component and bs in the second one, the second component has to skip all as, and then the system can compare as and bs. After this, there is no way how to compare as in the second component with bs in the first component because finite automata cannot be returned on the start position. Similar problem arises when the system starts with comparing bs in the first component and as in the second one. The other components cannot help, because they have finite number of states. Hence, n-L does not belong to $\mathscr{L}(\text{HMAS}^n\text{FA})$. ■

Theorem 6.13 $\mathscr{L}(\text{HMAS}^n\text{FA})$ for all $n \geq 2$, is not closed under intersection.

Proof of Theorem 6.13. Let $L_1 = \{(a^i b^j, a^j b^k) |\ i, j, k \geq 0\}$, $L_2 = \{(a^i b^j, a^k b^i) |\ i, j, k \geq 0\}$. Both of them belong to $\mathscr{L}(\text{HMAS}^n\text{FA})$, because we can construct HMAS^nFAs $\vartheta_1 = (M_1, M_2, \Psi_1)$ and $\vartheta_2 = (M_1, M_2, \Psi_2)$ such that $L(\vartheta_1) = L_1$ and $L(\vartheta_2) = L_2$. All four automata are given by the definition $M = (\{q_1, q_2\}, \{a, b\}, \{r_1 = q_1 a \to q_1, r_2 = q_1 \to q_1, r_3 = q_1 b \to q_2, r_4 = q_2 b \to q_2, r_5 = q_2 \to q_2\}, s_i, \{q_1,\ q_2\})$, and $\Psi_1 = \{(r_1, r_2), (r_3, r_1), (r_4, r_1), (r_5, r_3), (r_5, r_4)\}$ and $\Psi_2 = \{(p, q) |\ (q, p) \in \Psi_1\}$. The intersection of $L(\vartheta_1)$ and $L(\vartheta_2)$ is 2-language $L_3 = \{(a^i b^j, a^j b^i) |\ i, j = \mathbb{N}_0\}$. Lemma 6.12 says that $L_3 \notin \mathscr{L}(\text{HMAS}^n\text{FA})$, and therefore, $\mathscr{L}(\text{HMAS}^2\text{FA})$ is not closed under intersection.

In general, for $n \geq 2$, consider n-languages $K_1 = \{(a^i b^j, a^j b^k(, \varepsilon)^{n-2}) |\ i, j, k \geq 0\}$ and $K_2 = \{(a^i b^j,\ a^k b^i(, \varepsilon)^{n-2}) |\ i, j, k \geq 0\}$. From Lemma 6.12, $K_1 \cap K_2 \notin \mathscr{L}(\text{HMAS}^n\text{FA})$. ■

Consider n-language $Y \subseteq \Sigma_1^* \times \ldots \times \Sigma_n^*$ and suppose that complementation of Y is defined as $\overline{Y} = (\Sigma_1^* \times \ldots \times \Sigma_n^*) - Y$. Then, we can establish the following corollary.

Theorem 6.14 $\mathscr{L}(\text{HMAS}^n\text{FA})$ for all $n \geq 2$ is not closed under complementation.

Proof of Theorem 6.14. By contradiction. Suppose that $\mathscr{L}(\text{HMAS}^{n\text{FA}})$ for all $n \geq 2$ is closed under complementation. Let $L_1, L_2 \in \mathscr{L}(\text{HMAS}^{n\text{FA}})$. From Theorem 6.11 it follows that $L_1 \cup L_2 \in \mathscr{L}(\text{HMAS}^{n\text{FA}})$, and by the supposition, $\overline{(L_1 \cup L_2)} \in \mathscr{L}(\text{HMAS}^{n\text{FA}})$ as well. From De Morgan's law, $\overline{(L_1 \cup L_2)} = (\overline{L_1} \cap \overline{L_2})$, but it is contradiction, because $\mathscr{L}(\text{HMAS}^{n\text{FA}})$ for all $n \geq 2$, is not closed under intersection. ■

Theorem 6.15 If L_1 and $L_2 \in \mathscr{L}(\text{HMAS}^{n\text{FA}})$, then $L_1 \centerdot L_2 \in \mathscr{L}(\text{HMAS}^{n\text{FA}})$, where $L_1 \centerdot L_2 = \{(w_1w'_1, \ldots, w_nw'_n) |\ (w_1, \ldots, w_n) \in L_1 \text{ and } (w'_1, \ldots, w'_n) \in L_2\}$.

Proof of Theorem 6.15. Consider two n-languages, L_1, L_2. If L_1 and $L_2 \in \mathscr{L}(\text{HMAS}^{n\text{FA}})$, then there are $\text{HMAS}^{n\text{FA}}$s, $\vartheta_1 = (M_{1,1}, \ldots, M_{1,n}, \Psi_1)$ and $\vartheta_2 = (M_{2,1}, \ldots, M_{2,n}, \Psi_2)$, such that $L_1 = L(\vartheta_1)$, $L_2 = L(\vartheta_2)$ and for all $i = 1, 2$ and $j = 1, \ldots, n$, $M_{i,j} = (Q_{i,j}, \Sigma_{i,j}, \delta_{i,j}, s_{i,j}, F_{i,j})$ is a component of ϑ_i. For these two automata systems, we can construct $\vartheta_{12} = (M_{12,1}, \ldots, M_{12,n}, \Psi_{12})$ with $M_{12,j} = (Q_{12,j}, \Sigma_{12,j}, \delta_{12,j}, s_{1,j}, F_{2,j})$ in the following way: for every $i = 1, 2$ and $j = 1, \ldots, n$, $Q_{12,j} = Q_{1,j} \cup Q_{2,j}$, $\delta_{12,j} = \delta_{1,j} \cup \delta_{2,j} \cup \{p_j = f_j \to s_{2,j} |\ f_j \in F_{1,j}\}$, $\Psi_{12} = \Psi_1 \cup \Psi_2 \cup \{(p_1, \ldots, p_n)\}$, and $\Sigma_{12,j} = \Sigma_{1,j} \cup \Sigma_{2,j}$. From Definition 6.2, $(w_1, \ldots, w_n) \in L_1$ iff $(s_{1,1}w_1, \ldots, s_{1,n}w_n) \vdash^* (f_1, \ldots, f_n)$, where for all $i = 1, \ldots, n$, $f_i \in F_{1,i}$, in ϑ_1. Clearly, $(s_{1,1}w_1, \ldots, s_{1,n}w_n) \vdash^* (f_1, \ldots, f_n)$ in ϑ_{12} as well, and $(s_{1,1}w_1w'_1, \ldots, s_{1,n}w_nw'_n) \vdash^* (f_1w'_1, \ldots, f_nw'_n)$. As $(w'_1, \ldots, w'_n) \in L_2$ and because $(f_1 \to s_{2,1}, \ldots, f_n \to s_{2,n}) \in \Psi_{12}$, $(f_1w'_1, \ldots, f_nw'_n) \vdash (s_{2,1}w'_1, \ldots, s_{2,n}w'_n)$. Naturally, $(s_{2,1}w'_1, \ldots, s_{2,n}w'_n) \vdash^* (f'_1, \ldots, f'_n)$ with $f'_i \in F_{2,i}$ in ϑ_2. Hence, $(s_{1,1}w_1w'_1, \ldots, s_{1,n}w_nw'_n) \vdash^* (f'_1, \ldots, f'_n)$ in ϑ_{12}. The theorem holds. ■

HMAS with at least one PDA are stronger than HMAS containing only FAs.

Theorem 6.16 $\mathscr{L}(\text{HMAS}^{n\text{FA}}) \subset \mathscr{L}(\text{HMAS}^{(t_1,\ldots,t_n)})$ where $t_i \in \{\text{FA, PDA}\}$ for all $i = 1, \ldots, n$, and at least one component is a PDA.

Proof of Theorem 6.16. $\mathscr{L}(\text{HMAS}^{n\text{FA}}) \subseteq \mathscr{L}(\text{HMAS}^{(t_1,\ldots,t_n)})$, with at least one $i = 1, \ldots, n$ such that $t_i = \text{PDA}$, directly follows from the definitions. It remains to prove that $\mathscr{L}(\text{HMAS}^{n\text{FA}}) \neq \mathscr{L}(\text{HMAS}^{(t_1,\ldots,t_n)})$.

Consider $\text{HMAS}^{(\text{PDA,FA})}$, $\vartheta = (M_1, M_2, \Psi_1)$ with $\Psi = \{(r_1, r_1), (r_2, r_1), (r_3, r_1), (r_4, r_2), (r_5, r_2), (r_6, r_2), (r_7, r_2), (r_8, r_3), (r_9, r_1)\}$) and the automata defined in the following way: $M_1 = (\{s, q\}, \{a, b\}, \{\#, a\}, \{r_1 = \#s \to s, r_2 = \#sa \to \#as, r_3 = asa \to aas, r_4 = \#sb \to \#q, r_5 = asb \to \#q, r_6 = \#qb \to \#q, r_7 = aqb \to \#q, r_8 = aq \to q, r_9 = \#q \to q\}, s, \#, \emptyset)$ and $M_2 = (\{s\}, \{a, b\}, \{r_1 = s \to s, r_2 = sa \to s, r_3 = sb \to s\}, s, \{s\})$. It is not hard to see that ϑ define 2-language 2-$L = \{(a^ib^j, a^jb^i) |\ i, j \in \mathbb{N}_0\}$ and works in this way: first, automaton M_2 loops in state s and reads no symbol, while M_1 shifts all as onto the pushdown. After pushing all as from the M_1's input onto the pushdown, M_1 and M_2 read bs and as, respectively, and by reading them at the same time, automata compare their number. If there is more as in M_2's input than bs in M_1's input, then the automata system is stopped and the input is not accepted. Otherwise, M_1 skips to the other state and ϑ continues with comparing as on the M_1's pushdown and bs in the M_2's input by removing as from the pushdown in M_1 and reading bs from M_2's input. Only if the input was of the form (a^ib^j, a^jb^i) with $i, j \in \mathbb{N}_0$, the automata system removes symbol # from M_1's pushdown, and ϑ accepts. Because Lemma 6.12 says that 2-$L = \{(a^ib^j, a^jb^i) |\ i, j \in \mathbb{N}_0\}$ is not in $\mathscr{L}(\text{HMAS}^{2\text{FA}})$, $\mathscr{L}(\text{HMAS}^{2\text{FA}}) \neq \mathscr{L}(\text{HMAS}^{(\text{PDA,FA})})$. In general, for all

$n \geq 2$, there is n-language $n\text{-}L = \{(a^i b^j, a^j b^i, \varepsilon, \ldots, \varepsilon) |\ i, j \in \mathbb{N}_0\}$. The n-languages can be given by HMAS $^{(\text{PDA,FA,}\ldots\text{,FA})}$, where the first two components are defined in the same way as M_1 and M_2 was. The other components loop without reading any symbols. Hence, $n\text{-}L = \{(a^i b^j, a^j b^i, \varepsilon, \ldots, \varepsilon) |\ i, j \in \mathbb{N}_0\} \in \mathscr{L}(\text{HMAS}^{(\text{PDA,FA,}\ldots\text{,FA})})$. Therefore, $\mathscr{L}(\text{HMAS}^{n\text{FA}}) \neq \mathscr{L}(\text{HMAS}^{(t_1,\ldots,t_n)})$ with at least one PDA. ■

Lemma 6.17 Let $\widehat{\Gamma} \in \text{HCGR}^{(t_1,\ldots,t_n)}$ where $t_i \in \{\text{RLNG, CFG}\}$, for all $i = 1, \ldots, n$, and no more than one component is CFG in $\widehat{\Gamma}$. Then, $L_j = \{w_j |\ (w_1, \ldots, w_n) \in n\text{-}L(\widehat{\Gamma})\} \in \mathbf{CF}$ for every $j = 1, \ldots, n$.

Proof of Lemma 6.17. Consider $\widehat{\Gamma} \in \text{HMAS}^{(t_1,\ldots,t_n)}$ with exactly one CFG. Because we only want to prove that $L_j(\widehat{\Gamma}) = \{w_j |\ (w_1, w_2, \ldots, w_n) \in n\text{-}L(\widehat{\Gamma})\}$ is context-free, by replacing all terminal symbols by ε in each rule in P_i for all $i = 1, \ldots, n$, where $i \neq j$, we construct $\widehat{\Gamma}'$ such that $n\text{-}L(\widehat{\Gamma}') = \{(w_1, \ldots, w_n) |\ (w'_1, \ldots, w'_n) \in n\text{-}L(\widehat{\Gamma})$ and for all $i = 1, \ldots, n$, if $i = j$, then $w_i = w'_i$; otherwise, $w_i = \varepsilon\}$. By Lemma 6.5, there is HMAS$^{(t'_1,\ldots,t'_n)}$ ϑ with one PDA, such that $n\text{-}L(\vartheta) = n\text{-}L(\widehat{\Gamma}')$. As $\vartheta = (M_1, \ldots, M_n, \Psi)$ can be created from $\widehat{\Gamma}'$ by the algorithm used in proof of Lemma 6.5, we can construct a PDA $M = (Q, \Sigma, \Gamma, \delta, s, S, \emptyset)$, where $L(M) = L_j(\widehat{\Gamma}')$, in the following way: consider that mth component of ϑ is a PDA and set $\Sigma = \Sigma_j$, $\Gamma = \Gamma_m \cup \Sigma_j$, $Q = \{(q_1, \ldots, q_n) |$ for every $i = 1, \ldots, n$, $q_i \in Q_i\}$, $s = (s_1, \ldots, s_n)$, $S = S_m$ and $\delta = \{A(q_1, \ldots, q_n)a \to \gamma(q'_1, \ldots, q'_n) |\ r_m = Aq_m a_m \to \gamma q'_m \in \delta_m$, $r_i = q_i a_i \to q'_i \in \delta_i$ for all $i \neq m$, $a = a_j$ and $(r_1, \ldots, r_n) \in \Psi\}$. It remains to prove Claim 6.18 and Claim 6.19.

Claim 6.18 Without any loss of generality, suppose that mth component of ϑ is a PDA. If $(\alpha_1 s_1 w_1, \ldots, \alpha_n s_n w_n) \vdash^* (\beta_1 q_1 w'_1, \ldots, \beta_n q_n w'_n)$ in ϑ, where for all $i = 1, \ldots, n$, $i = m$ implies $\alpha_i = S_m, \beta_i \in \Gamma_i^*$ and $i \neq m$ implies $\alpha_i \beta_i = \varepsilon$, then $S_m(s_1, \ldots, s_n)w_j \vdash^* \beta_m(q_1, \ldots, q_n)$ in M.

Proof of Claim 6.18. By induction on the number of derivation steps.

Basis. Let $(\alpha_1 s_1 w_1, \ldots, \alpha_n s_n w_n) \vdash^0 (\alpha_1 s_1 w_1, \ldots, \alpha_n s_n w_n)$ in ϑ. Then, $S_m(s_1, \ldots, s_n)w_j \vdash^0 S_m(s_1, \ldots, s_n)w_j$ in M.

Induction Hypothesis. Suppose that Claim 6.18 holds for j or fewer derivation steps, where $j \in \mathbb{N}_0$.

Induction Step. Consider sequence of computation steps of the form $(\alpha_1 s_1 w_1, \ldots, \alpha_n s_n w_n) \vdash^{j+1} (\beta_1 q_1 w'_1, \ldots, \beta_n q_n w'_n)$ in ϑ. This sequence of computation steps, we can express as $(\alpha_1 s_1 w_1, \ldots, \alpha_n s_n w_n) \vdash^j (\alpha'_1 p_1 w''_1, \ldots, \alpha'_n p_n w''_n) \vdash (\beta_1 q_1 w'_1, \ldots, \beta_n q_n w'_n)$. By the induction hypothesis, $S_m(s_1, \ldots, s_n)w_j \vdash^j \alpha'_m(p_1, \ldots, p_n)w''_j$ in M. As ϑ moves from $(\alpha'_1 p_1 w''_1, \ldots, \alpha'_n p_n w''_n)$ to $(\beta_1 q_1 w'_1, \ldots, \beta_n q_n w'_n)$, there have to be n rules, $r_1, \ldots, r_n$, such that $\alpha'_i p_i w''_i \vdash \beta_i q_i w'_i[r_i]$ for all $i = 1, \ldots, n$, and $(r_1, \ldots, r_n) \in \Psi$. Hence and from construction of δ, $\alpha'_m(p_1, \ldots, p_n)w''_j \vdash \beta_m(q_1, \ldots, q_m)w'_j$ in M. Claim 6.18 holds. □

Claim 6.19 Consider that mth component of ϑ is a PDA. If $S_m(s_1, \ldots, s_n)w_j \vdash^* \beta_m(q_1, \ldots, q_n)$ in M, then $(\alpha_1 s_1 w_1, \ldots, \alpha_n s_n w_n) \vdash^* (\beta_1 q_1 w'_1, \ldots, \beta_n q_n w'_n)$ in ϑ, where for all $i = 1, \ldots, n$, $i = m$ implies $\alpha_i = S_m, \beta_i \in \Gamma_i^*$ and $i \neq m$ implies $\alpha_i \beta_i = \varepsilon$.

Proof of Claim 6.19. By induction on the number of computation steps.

Basis. Let $S_m(s_1, \ldots, s_n)w_j \vdash^0 S_m(s_1, \ldots, s_n)w_j$ in M. Evidently, $(\alpha_1 s_1 w_1, \ldots, \alpha_n s_n w_n) \vdash^0 (\alpha_1 s_1 w_1, \ldots, \alpha_n s_n w_n)$ in ϑ.

Induction Hypothesis. Suppose that Claim 6.19 holds for j or fewer derivation steps, where $j \in \mathbb{N}_0$.

Induction Step. Consider sequence of moves of the form $S_m(s_1, \ldots, s_n)w_j \vdash^{j+1} \gamma_m(q_1, \ldots, q_n)w'_j$ in M. We can express this sequence as $S_m(s_1, \ldots, s_n)w_j \vdash^j \beta_m(p_1, \ldots, p_n)w''_j \vdash \gamma_m(q_1, \ldots, q_n)w'_j$. By the induction hypothesis, $(\alpha_1 s_1 w_1, \ldots, \alpha_n s_n w_n) \vdash^j (\beta_1 p_1 w''_1, \ldots, \beta_n p_n w''_n)$, where for all $i = 1, \ldots, n$, if $i = m$, then $\alpha_i = S_m$; otherwise, $\alpha_i \beta_i = \varepsilon$. Since $\beta_m(p_1, \ldots, p_n)w''_j \vdash \gamma_m(q_1, \ldots, q_n)w'_j$ in M, there are n rules, $r_1, \ldots, r_n$, such that for every $i = 1, \ldots, n$, $\beta_i p_i w''_i \vdash \gamma_i q_i w'_i[r_i]$ in M_i and $(r_1, \ldots, r_n) \in \Psi$. Hence, $(\beta_1 p_1 w''_1, \ldots, \beta_n p_n w''_n) \vdash (\gamma_1 q_1 w'_1, \ldots, \gamma_n q_n w'_n)$ in ϑ. The claim holds. □

Because we can simulate generation of language of each component by a PDA, these languages have to be context-free. Lemma 6.17 holds. ■

Corollary 6.20 Let $\vartheta \in \text{HMAS}^{(t_1,\ldots,t_n)}$ with $t_1, \ldots, t_n \in \{\text{FA, PDA}\}$ and at most one t_i=PDA. Let $L_j(\vartheta) = \{w_j |\ (w_1, w_2, \ldots, w_n) \in n\text{-}L(\vartheta)\}$ for all $j = 1, \ldots, n$. Then, $L_j \in \mathbf{CF}$.

Theorem 6.21 Let $\mathscr{L}_j(\text{HMAS}^{(t_1,\ldots,t_n)}) = \{L_j |\ L_j = \{w_j |\ (w_1, \ldots, w_n) \in K\}$ and $K \in \mathscr{L}(\text{HMAS}^{(t_1,\ldots,t_n)})\}$, where $\text{HMAS}^{(t_1,\ldots,t_n)}$ is a HMAS with exactly one PDA and $n-1$ FAs. Then, $\mathscr{L}_j(\text{HMAS}^{(t_1,\ldots,t_n)}) = \mathbf{CF}$.

Proof of Theorem 6.21. Suppose that the first component of a HMAS based upon one PDA and $n-1$ FAs. The HMAS can work in the following way: at the beginning, the PDA works on its input string while FAs are cycling with reading no symbol. If the PDA accepts its input, it removes all symbols from the pushdown, inserts the first pushdown symbol, and continues with the input of the first FA. With aim of the restriction set, the FA reads its input while the PDA accordingly works with the pushdown. If the PDA accepts this input string, the HMAS continues with other string. In this way, the FAs can use pushdown of the PDA, and therefore, they can accept context-free languages. Hence, and from Corollary 6.20, Theorem 6.21 holds. ■

Two 1-turn PDAs in HMAS, where other components are FAs, are enough to recognize any language from **RE** by any of the components in the HMAS.

Theorem 6.22 Let $\mathscr{L}_j(\text{HMAS}^{(t_1,\ldots,t_n)}) = \{L_j |\ L_j = \{w_j |\ (w_1, \ldots, w_n) \in K\}$ and $K \in \mathscr{L}(\text{HMAS}^{(t_1,\ldots,t_n)})\}$, where $\text{HMAS}^{(t_1,\ldots,t_n)}$ contains two 1-turn PDAs and $n-2$ FAs. Then, $\mathscr{L}_j(\text{HMAS}^{(t_1,\ldots,t_n)}) = \mathbf{RE}$.

Proof of Theorem 6.22. It is well-known that every recursively enumerable language can be generated by a grammar G in Geffert normal form (see [47]), i.e. by a grammar $G = (\{A, B, C, D, S\}, T, S, P)$ where P contains rules only of the form $S \to uSa$, $S \to uSv$, $S \to uv$, $AB \to \varepsilon$, and $CD \to \varepsilon$, where $u \in \{A, C\}^*$, $v \in \{B, D\}^*$, $a \in T$. In addition, every sentential form of any successful derivation have to be of the form $S \Rightarrow^* w_1 w_2 w$ where $w_1 w_2 \in \{A, C\}^*\{B, D\}^*$, $w \in T^*$, and $w_1 w_2 w \Rightarrow^* w$.

Let $G_1, \ldots, G_n$ be n grammars in the Geffert normal form and let $\vartheta = (M_1, \ldots, M_n, \Psi)$ be an $\text{HMAS}^{(t_1,\ldots,t_n)}$, M_1, M_2 be two 1-turn PDAs and $M_3, \ldots, M_n$ be FAs. M_1 and M_2 can generate strings over $\{A, C, |\}$ and $\{B, D, |\} \cup \{|\ a \in \bigcup_{i=1}^{n} \Sigma_i\}$ on the M_1's and

M_2's pushdown, respectively, in the following way: the PDAs start with symbol | on their pushdowns. First, ϑ simulates derivations in $G_1, \ldots, G_n$. The system starts with G_1. If G_i applies a rule of the form $S_i \rightarrow uS_iw$, ϑ adds $(u)^R$ and w on M_1's and M_2's pushdown, respectively. If G_i makes a derivation step by rule of the form $S_i \rightarrow uw$, then ϑ adds $(u)^R|$ and $w|$ on M_1's and M_2's pushdown, respectively, and the system starts with simulation of G_{i+1}. After the generation, PDAs start to compare topmost symbols on their pushdowns. At the same computation step, the automata can remove A with B and C with D. Whenever the | is on the M_1's pushdown top and ϑ works on ith input string (the system starts with nth input string), M_2 compares symbols on its pushdown with input symbols which reads ith automaton. If the symbols coincide, M_2 removes the symbol from the pushdown. As | is topmost pushdown symbol of both PDAs, the PDAs remove the symbol and start with $(i-1)$th input string. The automata in ϑ read their input only if they compare the input with content of M_2's pushdown. If and only if all automata read all their input and the PDAs have empty pushdown, all strings are accepted—that is, ϑ accepts the input n-string.

Evidently, ϑ can simulate derivations of n grammars in Geffert normal form, which can generate any language from **RE**. Therefore, Theorem 6.22 holds. ■

Corollary 6.23 Let $\mathscr{L}_j(\mathrm{HMAS}^{(t_1,\ldots,t_n)}) = \{L_j |\ L_j = \{w_j |\ (w_1, \ldots, w_n) \in K\}$ and $K \in \mathscr{L}(\mathrm{HMAS}^{(t_1,\ldots,t_n)})\}$, where $\mathrm{HMAS}^{(t_1,\ldots,t_n)}$ contains three 1-turn SPDAs and $n-3$ FAs, for $n \geq 3$. Then, $\mathscr{L}_j(\mathrm{HMAS}^{(t_1,\ldots,t_n)}) = \mathbf{RE}$.

Corollary 6.24 Let $\mathscr{L}_j(\mathrm{HCGR}^{(t_1,\ldots,t_n)}) = \{L_j |\ L_j = \{w_j |\ (w_1, \ldots, w_n) \in K\}$ and $K \in \mathscr{L}(\mathrm{HCGR}^{(t_1,\ldots,t_n)})\}$, where $\mathrm{HCGR}^{(t_1,\ldots,t_n)}$ is an HCGR with exactly two CFGs and $n-2$ RLNGs. Then, $\mathscr{L}_j(\mathrm{HCGR}^{(t_1,\ldots,t_n)}) = \mathbf{RE}$.

Corollary 6.25 $\mathscr{L}(\mathrm{HMAS}^{(t_1,\ldots,t_n)})\}$ with two 1-turn PDAs and $n-2$ FAs is equal to $\mathscr{L}(\mathrm{HMAS}^{n\mathrm{PDA}})$.

Corollary 6.26 $\mathscr{L}(\mathrm{HCGR}^{n\mathrm{CFG}})$ is equal to $\mathscr{L}(\mathrm{HCGR}^{(t_1,\ldots,t_n)})\}$ with two CFGs and $n-2$ RLNGs.

It is well-known that linear grammars and 1-turn PDAs are equivalent. Nevertheless, HCGRs based upon linear grammars are weaker than HMASs with 1-turn PDAs as components.

Theorem 6.27 $\mathscr{L}(\mathrm{HCGR}^{n\mathrm{LNG}}) \subset \mathscr{L}(\mathrm{HMAS}^{(t_1,\ldots,t_n)})$, where $\mathrm{HMAS}^{(t_1,\ldots,t_n)}$ has exactly two 1-turn PDAs and $n-2$ FAs.

Proof of Theorem 6.27. Consider $\mathrm{HCGR}^{n\mathrm{LNG}}$ $\widehat{\Gamma} = (\widehat{G_1}, \ldots, \widehat{G_n}, \widehat{Q})$, which generates n-language $n\text{-}L(\widehat{\Gamma}) = \{(w, \varepsilon, \ldots, \varepsilon) |\ w \in \widehat{T_1}^*$ and $(S_1, \ldots, S_n) \Rightarrow^* (w, \varepsilon, \ldots, \varepsilon)\}$. Since $G_1, \ldots, G_n$ are linear, any successful derivation can be expressed as $(\widehat{S_1}, \ldots, \widehat{S_n}) \Rightarrow^* (uA_1v, A_2, \ldots, A_n) \Rightarrow (axv, \varepsilon, \ldots, \varepsilon)$, where $axv \in \widehat{T_1}^*$. Maximal length of every sentential n-form is $m+n$, where $m = |axv|$. Hence, we can construct a linear bounded automaton M with states of the form $(A_1, \ldots, A_n)$, where $A_i \in \widehat{N_i} \cup \{\varepsilon\}$ and input alphabet $\Gamma = T_1 \cup \{\underline{a} |\ a \in T_1\}$. M starts with state $(\widehat{S_1}, \ldots, \widehat{S_n})$ and w on its input tape. During computation, it underlines symbols on the input tape by the following algorithm.

Without any loss of generality, suppose that M is in state $(A_1, \ldots, A_n)$. If there are n rules, $r_1, \ldots, r_n$, where r_1 is rule of the form $A \to uB_1v$, for $i = 2, \ldots, n$, r_i is of the form $A_i \to B_i$ and $(r_1, \ldots, r_n) \in Q$, such that u is equal to the first $|u|$ not-underlined symbols on the tape and v is equal to the last $|v|$ not-underlined symbols on the tape, M underlines these symbols and moves to the state $(B_1, \ldots, B_n)$. As soon as all symbols on the tape are underlined and the automaton is in the state $(\varepsilon, \ldots, \varepsilon)$, the input is accepted. Hence, $L_j = \{w |\ (w, \varepsilon, \ldots, \varepsilon) \in n\text{-}L(\widehat{\Gamma})\}$ belongs to **CS**. Therefore, Theorem 6.27 holds. ■

Better approximation of the power of HCGRs with linear grammars as components is given by the following lemma.

Lemma 6.28 Let $\widehat{\Gamma} \in \text{HCGR}^n\text{LNG}$. Then, $L_j = \{w_j |\ (w_1, \ldots, w_n) \in n\text{-}L(\widehat{\Gamma})\} \in \textbf{LIN}$ for every $j = 1, \ldots, n$.

Proof of Lemma 6.29. Because all components are LNGs, without any loss of generality, we have to only prove that for any HCGR^nLNG, $\widehat{\Gamma} = (\widehat{G_1}, \ldots, \widehat{G_n}, \widehat{Q})$, there is an LNG, $G = (N, T, P, S)$ that generate the same language as the first component in the system does. Since n, number of nonterminals, and number of derivation rules in each component of the systems are finite, the linear grammar can remember combination of rules used in previous computation step by one nonterminal and allow only that derivation steps, which are corresponding with derivation steps in $\widehat{\Gamma}$. In more detail, the linear grammar will contain set of nonterminals $N = \{S\} \cup \{\langle r_1, \ldots, r_n\rangle |\ (r_1, \ldots, r_n) \in \widehat{Q}\}$, set T equal to the set of terminal symbol in G_1, and set

$P =$ $\{S \to u_1\langle r_1, \ldots, r_n\rangle v_1 |\ (r_1, \ldots, r_n) \in \widehat{Q}$, $r_i = S_i \to u_iX_iv_i$, and S_i is the start symbol of $\widehat{G_i}$, u_i, v_i are strings over alphabet of terminal symbols in $\widehat{G_i}$, X_i is an nonterminal symbol from $\widehat{G_i}$ or ε, for all $i = 1, \ldots, n\}$

$\cup$ $\{\langle r_1, \ldots, r_n\rangle \to u_1'\langle r_1', \ldots, r_n'\rangle v_1' |\ (r_1, \ldots, r_n), (r_1', \ldots, r_n') \in \widehat{Q}$, $r_i = A_i \to u_iB_iv_i$, $r_i' = B_i \to u_i'C_iv_i'$, and u_i, u_i', v_i, v_i' are strings over alphabet of terminal symbols in $\widehat{G_i}$, A_i, B_i, C_i are nonterminal symbols from $\widehat{G_i}$, for all $i = 1, \ldots, n\}$

$\cup$ $\{\langle r_1, \ldots, r_n\rangle \to u_1'v_1' |\ (r_1, \ldots, r_n), (r_1', \ldots, r_n') \in \widehat{Q}$, $r_i = A_i \to u_iB_iv_i$, $r_i' = B_i \to u_i'v_i'$, and u_i, u_i', v_i, v_i' are strings over alphabet of terminal symbols in $\widehat{G_i}$, A_i, B_i are nonterminal symbols from $\widehat{G_i}$, for all $i = 1, \ldots, n\}$.

For the sake of brevity, the rigorous proof of the equality $L(G) = L_1$ is left to the reader. ■

Of course, HCGRs can allow to use all combinations of rules included in its components. In this way, linear grammars contained in such HCGRs generate at least the class of linear languages. Thus, the following theorem can be established.

Theorem 6.29 Let $\widehat{\Gamma} \in \text{HCGR}^n\text{LNG}$. Then, $L_j = \{w_j |\ (w_1, \ldots, w_n) \in n\text{-}L(\widehat{\Gamma})\} = \textbf{LIN}$, for every $j = 1, \ldots, n$.

Chapter 7

Rule-Restricted Automaton-Grammar Transducers

In formal language theory, there exist two basic translation-method categories. The first category contains interprets and compilers, which first analyse an input string in the source language and, after that, they generate a corresponding output string in the target language (see [2], [89], [92], [61], or [109]). The second category is composed of language-translation systems or, more briefly, transducers. Frequently, these transducers consist of several components, including various automata and grammars, some of which read their input strings while others produce their output strings (see [5], [53], [91], and [111]).

Although transducers represent language-translation devices, language theory often views them as language-defining devices and investigates the language family resulting from them. In essence, it studies their accepting power consisting in determining the language families accepted by the transducer components that read their input strings. Alternatively, it establishes their generative power that determines the language family generated by the components that produce their strings. The present chapter contributes to this vivid investigation trend in formal language theory.

In this chapter, we introduce a new type of transducer, referred to as rule-restricted transducer, based upon a finite automaton and a context-free grammar. In addition, a restriction set controls the rules which can be simultaneously used by the automaton and by the grammar.

The present chapter discusses the power of this system working in an ordinary way as well as in a leftmost way and investigates an effect of an appearance checking placed into the system. First, we show that the generative power is equal to the generative power of matrix grammars (see [1] or [37]). Second, the accepting power coincides with the power of partially blind multi-counter automata (see [49] and [52]). Third, under the context-free-grammar leftmost restriction, the accepting and generative power of these systems coincides with the power of context-free grammars. On the other hand, when an appearance checking is introduced into these systems, the accepting and generative power coincides with the power of Turing machines.

In the last part of the chapter, we discuss application-related perspectives of the studied systems in linguistics. Particularly, we concentrate our attention on natural language translation. First, we demonstrate the basic idea in terms of simple English sentence, performing its analysis and passive transformation. Furthermore, we describe the translation of selected sentence structures between the Czech, English, and Japanese languages. We

demonstrate that while English and Czech are structurally similar languages, some aspects of Japanese differ significantly. We also show that some linguistic Czech-language features complicate this translation very much. For example, compared to English, there is very rich inflection in Czech. Other difficult-to-handle features include non-projectivity (crossing dependencies between words in a sentence), which mainly results from the fact that Czech is a free-word-order language.

7.1 Rule-restricted transducer

In this section, we define and investigate a rule-restricted transducers consisting of a finite automaton, a context-free grammar, and a control set of pairs of rules. The finite automaton reads its own input string and, simultaneously, the context-free grammar generates an output string. During the computation, the control set determines which rules can be used at the same computation step performed by both components. The computation of the system is successful if and only if the finite automaton accepts the input string and the context-free grammar successfully generates a string of terminal symbols.

Definition 7.1 (Rule-restricted transducer)
The *rule-restricted transducer*, RT for short, is a triplet $\Gamma = (M, G, \Psi)$, where $M = (Q, \Sigma, \delta, q_0, F)$ is a finite automaton, $G = (N, T, P, S)$ is a context-free grammar, and Ψ is a finite set of pairs of the form (r_1, r_2), where r_1 and r_2 are rules from δ and P, respectively.

A 2-*configuration* of RT is a pair $\chi = (x, y)$, where $x \in Q\Sigma^*$ and $y \in (N \cup T)^*$. Consider two 2-configurations, $\chi = (pav_1, uAv_2)$ and $\chi' = (qv_1, uxv_2)$ with $A \in N$, $u, v_2, x \in (N \cup T)^*$, $v_1 \in \Sigma^*$, $a \in \Sigma \cup \{\varepsilon\}$, and $p, q \in Q$. If $pav_1 \Rightarrow qv_1[r_1]$ in M, $uAv_2 \Rightarrow uxv_2[r_2]$ in G, and $(r_1, r_2) \in \Psi$, then Γ makes a computation step from χ' to χ', written as $\chi \Rightarrow \chi'[(r_1, r_2)]$, or simply $\chi \Rightarrow \chi'$. In the standard way, $\Rightarrow^*$ and $\Rightarrow^+$ are reflexive-transitive and transitive closure of $\Rightarrow$, respectively.

The 2-*language of* Γ, 2-$L(\Gamma)$, is 2-$L(\Gamma) = \{(w_1, w_2) |\ (q_0 w_1, S) \Rightarrow^* (f, w_2),\ w_1 \in \Sigma^*, w_2 \in T^*$, and $f \in F\}$. From the 2-language we can define two languages:

- $L(\Gamma)_1 = \{w_1 |\ (w_1, w_2) \in$ 2-$L(\Gamma)\}$, and
- $L(\Gamma)_2 = \{w_2 |\ (w_1, w_2) \in$ 2-$L(\Gamma)\}$.

By $\mathscr{L}(RT)$, $\mathscr{L}(RT)_1$, and $\mathscr{L}(RT)_2$, the classes of 2-languages of RTs, languages accepted by M in RTs, and languages generated by G in RTs, respectively, are understood.

It is well-known that finite automata and context-free grammars describe different classes of languages. Specifically, by the finite automata we can accept regular languages, whereas the context-free grammars define the class of context-free languages. However, in Example 7.2 is shown that by the combination of these two models, the system is able to accept and generate non-context-free languages.

Example 7.2
Consider RT $K = (M, G, \Psi)$ with

$M = (\{1, 2, 3', 3, 4, 5', 5, 6\}, \{a, b\}, \delta, 1, \{6\})$, where

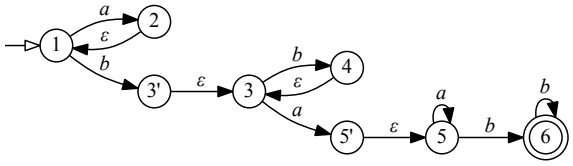


Figure 7.1: Definition of finite automaton M from Example 7.2

– $\delta = \{$
$p_1 = 1a \to 2, \quad p_2 = 2 \to 1, \quad p_3 = 1b \to 3', \quad p_4 = 3' \to 3,$
$p_5 = 3b \to 4, \quad p_6 = 4 \to 3, \quad p_7 = 3a \to 5', \quad p_8 = 5' \to 5,$
$p_9 = 5a \to 5, \quad p_{10} = 5b \to 6, \quad p_{11} = 6b \to 6\}$

(see the graphical representation of M in Figure 7.1),

$G = (\{S, A, B, C, D, D'\}, \{a, b\}, P, S)$, where

– $P = \{$
$r_1 = S \to BbD', \quad r_2 = B \to Bb, \quad r_3 = D' \to D'D,$
$r_4 = B \to aA, \quad r_5 = D' \to C, \quad r_6 = A \to aA,$
$r_7 = C \to CC, \quad r_8 = D \to b, \quad r_9 = A \to \varepsilon,$
$r_{10} = C \to a\},$

$\Psi = \{(p_1, r_1), (p_1, r_2), (p_2, r_3), (p_3, r_4), (p_4, r_5), (p_5, r_6), (p_6, r_7), (p_7, r_8), (p_8, r_9), (p_9, r_8), (p_{10}, r_{10}), (p_{11}, r_{10})\}$.

The languages of M and G are $L(M) = \{a^i b^j a^k b^l |\ j, k, l \in \mathbb{N}, i \in \mathbb{N}_0\}$ and $L(G) = \{a^i b^j a^k b^l |\ i, j, k \in \mathbb{N}, l \in \mathbb{N}_0\}$, respectively. However, 2-language of K is $L(K) = \{(a^i b^j a^i b^j, a^j b^i a^j b^i) |\ i, j \in \mathbb{N}\}$.

From the example, observe that the power of the grammar increases due to the possibility of synchronization with the automaton that can dictate sequences of usable rules in the grammar. The synchronization with the automaton enhances the generative power of the grammar up to the class of languages generated by matrix grammars.

Theorem 7.3 $\mathscr{L}(RT)_2 = \mathscr{L}(MAT)$.

Proof of Theorem 7.3. First, we prove that $\mathscr{L}(MAT) \subseteq \mathscr{L}(RT)_2$.

Consider a MAT $I = ({}_I G, {}_I C)$ and construct RT $\Gamma = ({}_\Gamma M, {}_\Gamma G, \Psi)$, such that $L(I) = L(\Gamma)_2$, as follows: Set ${}_\Gamma G = {}_I G$; construct finite automaton ${}_\Gamma M = (Q, \Sigma, \delta, s, F)$ in the following way: Set $F, Q = \{s\}$; for every $m = p_1 \dots p_k \in {}_I C$, add $k - 1$ new states, $q_1, q_2, \dots, q_{k-1}$, into Q, k new rules, $r_1 = s \to q_1, r_2 = q_1 \to q_2, \dots, r_{k-1} = q_{k-2} \to q_{k-1}, r_k = q_{k-1} \to s$, into δ, and k new pairs, $(r_1, p_1), (r_2, p_2), \dots, (r_{k-1}, p_{k-1}), (r_k, p_k)$, into Ψ.

The finite automaton simulates matrices in I by moves. That is, if $x_1 \Rightarrow x_2[p]$ in I, where $p = p_1 \dots p_i$ for some $i \in \mathbb{N}$, then there is $q_1, \dots, q_{i-1} \in Q$ such that $r_1 = s \to q_1, r_2 = q_1 \to q_2, \dots, r_{i-1} = q_{i-2} \to q_{i-1}, r_i = q_{i-1} \to s \in \delta$ and $(r_1, p_1), \dots, (r_i, p_i) \in \Psi$. Therefore, $(s, x_1) \Rightarrow^i (s, x_2)$ in Γ. Similarly, if $(s, x_1) \Rightarrow^i (s, x_2)$ in Γ, for $i \in \mathbb{N}$, and there is no $j \in \mathbb{N}$ such that $0 < j < i$ and $(s, x_1) \Rightarrow^j (s, y) \Rightarrow^* (s, x_2)$, there has to be $p \in {}_I C$ and $x_1 \Rightarrow x_2[p]$ in I. Hence, if $(s, S) \Rightarrow^* (s, w)$ in Γ, where w is a string over the set of terminal symbols in ${}_\Gamma G$, then $S \Rightarrow^* w$ in I; and, on the other hand, if $S \Rightarrow^* w$ in I for a string over the set of

terminals in ${}_IG$, then $(s, S) \Rightarrow^* (s, w)$ in Γ. The inclusion $\mathscr{L}(MAT) \subseteq \mathscr{L}(RT)_2$ has been proven.

For any RT $\Gamma = ({}_\Gamma M = (Q, \Sigma, \delta, s, F), {}_\Gamma G = ({}_\Gamma N, {}_\Gamma T, {}_\Gamma P, {}_\Gamma S), \Psi)$, we can construct a MAT $O = ({}_O G, {}_O C)$ such that $L(\Gamma)_2 = L(O)$ as follows: Set ${}_O G = ({}_\Gamma N \cup \{S'\}, {}_\Gamma T, {}_O P, S')$, ${}_O P = {}_\Gamma P \cup \{p_0 = S' \to \langle s \rangle {}_\Gamma S\}$, and ${}_O C = \{p_0\}$. For each pair $(p_1, p_2) \in \Psi$ with $p_1 = qa \to r$, $q, r \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $p_2 = A \to x$, $A \in {}_\Gamma N$, and $x \in ({}_\Gamma N \cup {}_\Gamma T)^*$, add $p_1 = \langle q \rangle \to \langle r \rangle$ into ${}_O P$ and $p_1 p_2$ into ${}_O C$. Furthermore, for all $q \in F$, add $p = \langle q \rangle \to \varepsilon$ into ${}_O P$ and p into ${}_O C$.

By the following claims we prove that $L(\Gamma)_2 = L(O)$.

Claim 7.4 If $(sw, {}_\Gamma S) \Rightarrow^* (qw', \omega)$ in Γ, then $S' \Rightarrow^* \langle q \rangle \omega$ in O.

Proof of Claim 7.4. By induction on the number of computation steps.

Basis. Let $(sw, {}_\Gamma S) \Rightarrow^0 (sw, {}_\Gamma S)$ in Γ. Then, $S' \Rightarrow \langle s \rangle {}_\Gamma S[p]$ in ${}_O G$ and $p \in {}_O C$. Hence, $S' \Rightarrow \langle s \rangle {}_\Gamma S[p]$ in O. Claim 7.4 holds for no steps in Γ.

Induction hypothesis. Suppose that Claim 7.4 holds for j or fewer computation steps.

Induction step. Let $(sw, {}_\Gamma S) \Rightarrow^j (qw', \omega) \Rightarrow (q'w'', \omega')$ in Γ. Then, by the induction hypothesis, $S' \Rightarrow^* \langle q \rangle \omega$ in O. Without any loss of generality suppose that $\omega = uAv$ for $u, v \in ({}_\Gamma N \cup {}_\Gamma T)^*$, $A \in {}_\Gamma N$, and $(qw', uAv) \Rightarrow (q'w'', uxv)$ with $x \in ({}_\Gamma T \cup {}_\Gamma N)^*$ and $\omega' = uxv$. From the construction of O we know that $p_1 = A \to x$ and $p_2 = \langle q \rangle \to \langle q' \rangle$ is in ${}_\Gamma P$ and $p_1 p_2 \in {}_O C$. Therefore, $S' \Rightarrow^* \langle q \rangle \omega \Rightarrow \langle q' \rangle uxv = \langle q' \rangle \omega'$ in O. Claim 7.4 holds. Furthermore, for all $f \in F$ there is a rule $p = \langle f \rangle \to \varepsilon \in {}_\Gamma P$ and $p \in {}_O C$. Hence, if $(sw, {}_\Gamma S) \Rightarrow^* (f, \omega)$, where $f \in F$ and $\omega \in {}_\Gamma T^*$ in Γ, $S' \Rightarrow^* \langle f \rangle \omega \Rightarrow \omega$ in O. That is, $L(\Gamma)_2 \subseteq L(O)$. □

It remains to prove that $L(O) \subseteq L(\Gamma)_2$.

Claim 7.5 If $S' \Rightarrow^* \langle q \rangle \omega$ in O with $\omega \in {}_\Gamma T^*$, then $(sw, {}_\Gamma S) \Rightarrow^* (f, \omega)$ in Γ for some $w \in \Sigma^*$ and $f \in F$.

Proof of Claim 7.5. Consider any successful derivation of the form

$$S' \Rightarrow \langle q_0 \rangle \omega_0 [p_0] \Rightarrow \langle q_1 \rangle \omega_1 [p_1] \Rightarrow \langle q_2 \rangle \omega_2 [p_2] \Rightarrow \ldots \Rightarrow \langle q_k \rangle \omega_k [p_k]$$

in O, where $q_0 = s$, $q_k = q$, $\omega_0 = {}_\Gamma S$, and $\omega_k = \omega$. As it follows from the construction of O, for every $i = 1, \ldots, k$, $p_i = p_i' p_i''$, where $p_i' = \langle q_{i-1} \rangle \to \langle q_i \rangle$, $\omega_{i-1} \Rightarrow \omega_i [p_i'']$ in ${}_\Gamma G$, and for $a \in \Sigma \cup \{\varepsilon\}$, $(q_{i-1} a \to q_i, p_i'') \in \Psi$. That is, $(q_{i-1} w_{i-1}, \omega_{i-1}) \Rightarrow (q_i w_i, \omega_i)$ for all $i = 1, \ldots, k$, and hence, $(sw_0, {}_\Gamma S) \Rightarrow^* (q_k w_k, \omega_k)$ with $w = w_0$. Having $\omega_k \in {}_\Gamma T^*$ and using $p = p_1 p_2$, where $p_1 = \langle q \rangle \to \varepsilon \in {}_O P$, $\omega_{k-1} \Rightarrow \omega_k [p_2]$, and $p \in {}_O C$, implies $q \in F$, $w_k = \varepsilon$, and $w \in L({}_\Gamma M)$, and therefore, $\omega_k \in L(\Gamma)_2$. $L(O) \subseteq L(\Gamma)_2$. □

Theorem 7.3 holds by claims 7.4 and 7.5. ■

On the other hand, the context-free grammar in the RT can be exploited as an additional storage space of the finite automaton to remember some non-negative integers. If the automaton uses the context-free grammar in this way, the additional storage space is akin to counters in a multi-counter machine. The following lemma says that the FAs in the RTs are able to accept every language accepted by partially blind k-counter automata.

Lemma 7.6 For every k-PBCA I, there is an RT $\Gamma = (M, G, \Psi)$ such that $L(I) = L(\Gamma)_1$.

Proof of Lemma 7.6. Let $I = ({}_IQ, \Sigma, {}_I\delta, q_0, F)$ be a k-PBCA for some $k \geq 1$ and construct RT $\Gamma = (M = ({}_MQ, \Sigma, {}_M\delta, q_0, F), G = (N, T, P, S), \Psi)$ as follows: Set $T = \{a\}$, $\Psi = \emptyset$, $N = \{S, A_1, \ldots, A_k\}$, $P = \{A \to \varepsilon |\ A \in N\}$, ${}_M\delta = \{f \to f |\ f \in F\}$, and ${}_MQ = {}_IQ$.

For each $pa \to q(t_1, \ldots, t_k)$ in ${}_I\delta$ and for $n = (\Sigma_{i=1}^{k} max(0, -t_i))$ add:

- $q_1, \ldots, q_n$ into ${}_MQ$;
- $r = S \to xS$, where $x \in (N - \{S\})^*$ and $occur(A_i, x) = max(0, t_i)$, for $i = 1, \ldots, k$, into P;
- $r_1 = q_0a \to q_1$, $r_2 = q_1 \to q_2$, ..., $r_n = q_{n-1} \to q_n$, $r_{n+1} = q_n \to q$ into ${}_M\delta$ with $q_0 = p$; and $(r_{i+1}, \alpha_i \to \varepsilon)$, where $\alpha_i = A_j$ and each A_j is erased $max(0, -t_i)$-times during the sequence, into Ψ ($n = 0$ means that only $pa \to q$, $S \to xS$ and (r_1, r) are considered);
- $(f \to f, S \to \varepsilon)$ into Ψ for all $f \in F$.

The finite automaton of the created system uses the context-free grammar as an external storage. Each counter of the I is represented by a nonterminal. Every step from p to q that modifies counters are simulated by several steps leading from p to q and during this sequence of steps the number of occurrences of each nonterminal in the grammar is modified to be equal to the corresponding counter in I. Clearly, $L(I) = L(\Gamma)_1$. ■

Lemma 7.7 states that the context-free grammar is helpful for the finite automaton in RT at most with the preservation of the non-negative integers without possibility to check their values.

Lemma 7.7 For every RT $\Gamma = (M, G, \Psi)$, there is a k-PBCA O such that $L(O) = L(\Gamma)_1$ and k is the number of nonterminals in G.

Proof of Lemma 7.7. Let $\Gamma = (M = (Q, \Sigma, {}_M\delta, q_0, F), G = (N, T, P, S), \Psi)$ be an RT. Without any loss of generality suppose that $N = \{A_1, \ldots, A_n\}$, where $S = A_1$. The partially blind $|N|$-counter automaton $O = (Q \cup \{q_0'\}, \Sigma, {}_O\delta, q_0', F)$ is created in the following way. For each $r_1 = pa \to q \in {}_M\delta$ and $r_2 = \alpha \to \beta \in P$ such that $(r_1, r_2) \in \Psi$, add $pa \to q(v_1, \ldots, v_{|N|})$, where $v_i = occur(A_i, \beta) - occur(A_i, \alpha)$, for all $i = 1, \ldots, |N|$, into ${}_O\delta$. Furthermore, add $q_0' \to q_0(1, 0, \ldots, 0)$ into ${}_O\delta$.

The constructed partially blind $|N|$-counter automaton has a counter for each nonterminal from the grammar of Γ. Whenever the automaton in Γ makes a step and a sentential form of grammar G is changed, O makes the same step and accordingly changes the number of occurrences of nonterminals in its counters. ■

From Lemma 7.6 and Lemma 7.7, we can establish the following theorem.

Theorem 7.8 $\mathscr{L}(RT)_1 = \bigcup_{k=1}^{\infty} \mathscr{L}(k\text{-}PBCA)$.

Proof of Theorem 7.8. It directly follows from Lemma 7.7 and Lemma 7.6. ■

For the better illustration of the accepting and generative power of RT, let us recall that the class of languages generated by MATs is properly included in **RE** (see [1] or [37]), and the class of languages defined by partially blind k-counter automata, with respect to the number of counters, is superset of **CF** and properly included in **CS** (see [49] and [52]).

Although the investigated system is relatively powerful, in defiance of weakness of models they are used, non-deterministic selections of nonterminals to be rewritten can be relatively problematic from the practical point of view. Therefore, we examine an effect of a restriction in the form of leftmost derivations placed on the grammar in RT.

Definition 7.9 (Leftmost restriction on derivation in RT)
Let $\Gamma = (M, G, \Psi)$ be an RT with $M = (Q, \Sigma, \delta, q_0, F)$ and $G = (N, T, P, S)$. Furthermore, let $\chi = (pav_1, uAv_2)$ and $\chi' = (qv_1, uxv_2)$ be two 2-configurations, where $A \in N$, $v_2, x \in (N \cup T)^*$, $u \in T^*$, $v_1 \in \Sigma^*$, $a \in \Sigma \cup \{\varepsilon\}$, and $p, q \in Q$. If and only if $pav_1 \Rightarrow qv_1[r_1]$ in M, $uAv_2 \Rightarrow uxv_2[r_2]$ in G, and $(r_1, r_2) \in \Psi$, Γ makes a computation step from χ to χ', written as $\chi \Rightarrow_{lm} \chi'[(r_1, r_2)]$, or simply $\chi \Rightarrow_{lm} \chi'$. In the standard way, $\Rightarrow^*_{lm}$ and $\Rightarrow^+_{lm}$ are reflexive-transitive and transitive closure of $\Rightarrow_{lm}$, respectively.

The 2-*language of* Γ with G generating in the leftmost way, denoted by 2-$L_{lm}(\Gamma)$, is defined as 2-$L_{lm}(\Gamma) = \{(w_1, w_2) |\ (q_0w_1, S) \Rightarrow^*_{lm} (f, w_2),\ w_1 \in \Sigma^*,\ w_2 \in T^*,$ and $f \in F\}$; we call Γ as *leftmost restricted RT*; and we define the languages given from 2-$L_{lm}(\Gamma)$ as $L_{lm}(\Gamma)_1 = \{w_1 |\ (w_1, w_2) \in$ 2-$L_{lm}(\Gamma)\}$ and $L_{lm}(\Gamma)_2 = \{w_2 |\ (w_1, w_2) \in$ 2-$L_{lm}(\Gamma)\}$. By $\mathscr{L}(RT_{lm})$, $\mathscr{L}(RT_{lm})_1$, and $\mathscr{L}(RT_{lm})_2$, we understand the classes of 2-languages of leftmost restricted RTs, languages accepted by M in leftmost restricted RTs, and languages generated by G in leftmost restricted RTs, respectively.

Theorem 7.10 $\mathscr{L}(RT_{lm})_2 = \mathbf{CF}$.

Proof of Theorem 7.10. The inclusion $\mathbf{CF} \subseteq \mathscr{L}(RT_{lm})_2$ is clear from the definition, because any time we can construct leftmost restricted RT, where the automaton M cycles with reading all possible symbols from the input or ε while the grammar G is generating some output string. Therefore, we only need to prove the opposite inclusion.

We know that the class of context-free languages is defined, inter alia, by pushdown automata. It is sufficient to prove that every language $L_{lm}(\Gamma)_2$ of RT can be accepted by a pushdown automaton. Consider an RT $\Gamma = (M = (Q, {}_\Gamma\Sigma, {}_\Gamma\delta, q_0, F), G = (N, T, P, S), \Psi)$ and define PDA $O = (Q, T, {}_O\Gamma, {}_O\delta, q_0, S, F)$, where ${}_O\Gamma = N \cup T$ and ${}_O\delta$ is created as follows:

- set ${}_O\delta = \emptyset$;
- for each $r_1 = A \to x \in P$ and $r_2 = pa \to q \in {}_\Gamma\delta$ such that $(r_1, r_2) \in \Psi$, add $Ap \to (x)^R q$ into ${}_O\delta$;
- for each $p \in Q$ and $a \in T$ add $apa \to p$ into ${}_O\delta$;

Now, we have to show that $L(O) = L_{lm}(\Gamma)_2$.

Claim 7.11 Let $(q_0w, S) \Rightarrow^* (pw', u\alpha v) \Rightarrow^* (f, \widehat{w})$ in RT Γ, where $u \in T^*$, $\alpha \in N$, and $v \in (N \cup T)^*$. Then, $Sq_0\widehat{w} \Rightarrow^* (v)^R \alpha p \widehat{w}'$ in PDA O, where $\widehat{w} = u\widehat{w}'$.

Proof of Claim 7.11. By the induction on the number of computation steps.

Basis. Let $(q_0w, S) \Rightarrow^0 (q_0w, S) \Rightarrow^* (f, \widehat{w})$ in Γ. Trivially, $Sq_0\widehat{w} \Rightarrow^0 Sq_0\widehat{w}$ and Claim 7.11 holds.

Induction hypothesis. Suppose that Claim 7.11 holds for j or fewer computation steps.

Induction step. Let $(q_0w, S) \Rightarrow^j (paw', u\alpha v) \Rightarrow (qw', uxv) \Rightarrow^* (f, \widehat{w})$ in Γ, where $a \in {}_\Gamma\Sigma \cup \{\varepsilon\}$, $uxv = uu'\beta v'$ and β is the new leftmost nonterminal. Then, by the induction hypothesis, $Sq_0\widehat{w} \Rightarrow^* (v)^R\alpha pa\widehat{w}'$ in O.

Since $(paw', u\alpha v) \Rightarrow (qw', uxv)$ in Γ, $paw' \Rightarrow qw'[r_1]$ in M, $u\alpha v \Rightarrow uxv[r_2]$ in G, and $(r_1, r_2) \in \Psi$. From the construction of ${}_O\delta$, O has rules $\alpha p \to (x)^Rq$ and $bqb \to q$ for all $b \in T$. Hence, $(v)^R\alpha p\widehat{w}' \Rightarrow (xv)^Rq\widehat{w}'$. Because $uxv = uu'\beta v'$, β is the leftmost nonterminal, and $(qw', uxv) \Rightarrow^* (f, \widehat{w})$, $(xv)^Rq\widehat{w}' = (u'\beta v')^Rqu'\widehat{w}''$, and obviously, $(u'\beta v')^Rqu'\widehat{w}'' \Rightarrow^* (\beta v')^Rq\widehat{w}''$ in O. Claim 7.11 holds. □

The last step of every successful computation of Γ has to be of the form $(qa, u\alpha v) \Rightarrow (f, uxv)$, with $a \in T \cup \{\varepsilon\}$, $f \in F$, $uxv \in T^*$. By Claim 7.11, suppose that O is in configuration $(\alpha v)^Rqw'$, where $uw' = uxv$. From the construction of ${}_O\delta$, $(\alpha v)^Rqw' \Rightarrow (xv)^Rfw' \Rightarrow^* f$ in O. Hence, $L_{lm}(\Gamma)_2 \subseteq L(O)$.

It remains to prove the opposite inclusion—that is, $L(O) \subseteq L_{lm}(\Gamma)_2$.

Claim 7.12 Let $Sq_0w \Rightarrow^* f$ in PDA O, where $f \in F$. Then, $(q_0\widehat{w}, S) \Rightarrow^* (f, w)$ in RT Γ.

Proof of Claim 7.12. Consider any successful acceptance:

$$Sq_0w \Rightarrow^* f \qquad (III)$$

in PDA O. We can express (III) as $\alpha_0q_0w_0 \Rightarrow v_1\alpha_1u_1q_1w_0 \Rightarrow^* v_1\alpha_1q_1w_1 \Rightarrow v_2\alpha_2u_2q_2w_1 \Rightarrow^* v_2\alpha_2q_2w_2 \Rightarrow \ldots \Rightarrow v_k\alpha_ku_kq_kw_{k-1} \Rightarrow^* v_k\alpha_kq_kw_k \Rightarrow v_ku_{k+1}fw_k \Rightarrow^* f$, where $\alpha_0 = S$ and for all $i = 1, \ldots, k$ with $k \geq 0$, $\alpha_i \in N$, $u_i, u_{k+1}, v_k \in T^*$, $v_i \in (N \cup T)^*$, $w_{i-1} = (u_i)^Rw_i$ and $w_k = (v_ku_{k+1})^R$. Openly, $(u_i)^R\alpha_i(v_i)^R \Rightarrow (u_{i+1}u_i)^R\alpha_{i+1}(v_{i+1})^R[r_i]$ in G, $q_{i-1}\widehat{w}_{i-1} \Rightarrow q_i\widehat{w}_i[r_i']$, and furthermore, $(r_i', r_i) \in \Psi$ for all $i = 0, \ldots, k$. Hence, (III) can be simulated by $(q_0\widehat{w}_0, \alpha_0) \Rightarrow (q_1\widehat{w}_1, (u_1)^R\alpha_1(v_1)^R) \Rightarrow (q_2\widehat{w}_2, (u_2u_1)^R\alpha_2(v_2)^R) \Rightarrow \ldots \Rightarrow (u_ku_{k-1}\ldots u_1)^R\alpha_k(v_k)^R \Rightarrow (f, (u_{k+1}u_ku_{k-1}\ldots u_1)^R(v_k)^R) = (f, w)$ in Γ. So, Claim 7.12 holds. □

As $L(O) \subseteq L_{lm}(\Gamma)_2$ and $L_{lm}(\Gamma)_2 \subseteq L(O)$, Theorem 7.10 holds. ■

Lemma 7.13 For every language $L \in \mathbf{CF}$, there is an RT $\Gamma = (M, G, \Psi)$ such that $L_{lm}(\Gamma)_1 = L$.

Proof of Lemma 7.13. Let $I = ({}_IN, T, {}_IP, S)$ be a context-free grammar such that $L(I) = L$. For I, we can construct context-free grammar $H = ({}_HN, T, {}_HP, S)$, where ${}_HN = {}_IN \cup \{\langle a\rangle |\ a \in T\}$ and ${}_HP = \{\langle a\rangle \to a |\ a \in T\} \cup \{A \to x |\ A \to x' \in {}_IP$ and x is created from x' by replacing all $a \in T$ in x' with $\langle a\rangle\}$. Surely, $L(I) = L(H)$ even if H replaces only the leftmost nonterminals in each derivation step. In addition, we construct finite automaton $M = (\{q_0\}, T, \delta, q_0, \{q_0\})$ with $\delta = \{q_0 \to q_0\} \cup \{q_0a \to q_0 |\ a \in T\}$, and $\Psi = \{(q_0 \to q_0, A \to x) |\ A \to x \in {}_HP, A \in {}_IN\} \cup \{(q_0a \to q_0, \langle a\rangle \to a) |\ a \in T\}$.

It is easy to see that any time when H replaces nonterminals from ${}_IN$ in its sentential form, M reads no input symbol. If and only if H replaces $\langle a\rangle$ with a, where $a \in T$, then M reads a from the input. Since H works in a leftmost way, 2-$L_{lm}(\Gamma) = \{(w, w) |\ w \in L(I)$. Hence, $L_{lm}(\Gamma)_1 = L(I)$. ■

Similarly, we show that any RT generating outputs in the leftmost way can recognize no language out of **CF**.

Lemma 7.14 Let Γ be an RT. Then, for every language $L_{lm}(\Gamma)_1$, there is a PDA O such that $L_{lm}(\Gamma)_1 = L(O)$.

Proof of Lemma 7.14. In the same way as in the proof of Theorem 7.3, we construct PDA O such that $L(O) = L_{lm}(\Gamma)_1$ for RT $\Gamma = (M = (Q, {}_\Gamma\Sigma, {}_\Gamma\delta, q_0, F), G = (N, T, P, S), \Psi)$. We define O as $O = (Q, {}_\Gamma\Sigma, N, {}_O\delta, q_0, S, F)$, where ${}_O\delta$ is created in the following way:

- set ${}_O\delta = \emptyset$;
- for each $r_1 = pa \to q \in {}_\Gamma\delta$ and $r_2 = A \to x \in P$ such that $(r_1, r_2) \in \Psi$, add $Apa \to (\theta(x))^R q$ into ${}_O\delta$, where $\theta(x)$ is a function from $(N \cup T)^*$ to N^* that replaces all terminal symbols in x with ε—that is, $\theta(x)$ is x without terminal symbols.

In the following, we demonstrate that $L(O) = L_{lm}(\Gamma)_1$.

Claim 7.15 Let $(q_0w, S) \Rightarrow^* (pw', u\alpha v)$ in RT Γ, where $u \in T^*$, $\alpha \in N$, and $v \in (N \cup T)^*$. Then, $Sq_0w \Rightarrow^* (\theta(v))^R \alpha p w'$ in PDA O.

Proof of Claim 7.15. By the induction on the number of computation steps.

Basis. Let $(q_0w, S) \Rightarrow^0 (q_0w, S)$ in Γ. Then, surely, $Sq_0w \Rightarrow^0 (\theta(S))^R q_0 w$. Claim 7.15 holds.

Induction hypothesis. Suppose that Claim 7.15 holds for j or fewer computation steps.

Induction step. Let $(q_0w, S) \Rightarrow^j (paw', u\alpha v) \Rightarrow (qw', uxv)$ in Γ, where $a \in {}_\Gamma\Sigma \cup \{\varepsilon\}$, $uxv = uu'\beta v'$ and β is the leftmost nonterminal. By the induction hypothesis, $Sq_0w \Rightarrow^* (\theta(v))^R \alpha p a w'$ in O.

Because $(paw', u\alpha v) \Rightarrow (qw', uxv)$ in Γ, $paw' \Rightarrow qw'[r_1]$ in M, $u\alpha v \Rightarrow uxv[r_2]$ in G, and $(r_1, r_2) \in \Psi$. From the construction of ${}_O\delta$, O has a rule $\alpha pa \to (\theta(x))^R q$, and $(\theta(v))^R \alpha p a w' \Rightarrow (\theta(v'))^R \beta q w'$ in O. Claim 7.15 holds. □

The last step of any successful computation in Γ is of the form $(qa, u\alpha v) \Rightarrow (f, uxv)$, where $f \in F$, $a \in {}_\Gamma\Sigma \cup \{\varepsilon\}$, $\alpha \in N$, and $uxv \in T^*$. Hence, $\alpha qa \to f \in {}_O\delta$ and $\alpha qa \Rightarrow f$ in O. So, $L_{lm}(\Gamma)_1 \subseteq L(O)$.

Claim 7.16 Let $Sq_0w \Rightarrow^* (\theta(v))^R \alpha p w'$ in PDA O. Then, $(q_0w, S) \Rightarrow^* (pw', u\alpha v)$ in RT Γ, where $u \in T^*$, $\alpha \in N$, and $v \in (N \cup T)^*$.

Proof of Claim 7.16. By the induction on the number of moves.

Basis. Let $Sq_0w \Rightarrow^0 Sq_0w$. Then, $(q_0w, S) \Rightarrow^0 (q_0w, S)$ in Γ and Claim 7.16 holds.

Induction hypothesis. Suppose that Claim 7.16 holds for j or fewer moves.

Induction step. Let $Sq_0w \Rightarrow^j (\theta(v))^R \alpha p a w' \Rightarrow (\theta(xv))^R q w'$ in O, where $a \in {}_\Gamma\Sigma \cup \{\varepsilon\}$. Then, by the induction hypothesis, $(q_0w, S) \Rightarrow^* (paw', u\alpha v)$ in Γ, where $u \in T^*$, $\alpha \in N$, and $v \in (N \cup T)^*$.

Because there is a rule $\alpha pa \to (\theta(x))^R q$ in ${}_O\delta$, from the construction of ${}_O\delta$, there are rules $r_1 = pa \to q \in {}_\Gamma\delta$ and $r_2 = \alpha \to x \in P$, and $(r_1, r_2) \in \Psi$. Therefore, $(paw', u\alpha v) \Rightarrow (qw', uxv)$ in Γ. So, Claim 7.16 holds. Furthermore, if $\theta(xv)w' = \varepsilon$ and $q \in F$, then $(paw', u\alpha v) \Rightarrow (q, uxv)$ and $L(O) \subseteq L_{lm}(\Gamma)_1$. □

Since $L(O) \subseteq L_{lm}(\Gamma)_1$ and $L_{lm}(\Gamma)_1 \subseteq L(O)$, $L(O) = L_{lm}(\Gamma)_1$. ■

Theorem 7.17 $\mathscr{L}(RT_{lm})_1 = \mathbf{CF}$.

Proof of Theorem 7.17. It directly follows from Lemma 7.13 and Lemma 7.14. ■

Unfortunately, the price for the leftmost restriction, placed on derivations in the context-free grammar, is relatively high and both accepting and generative ability of RT with the restriction decreases to **CF**.

In the following, we extend RT with possibility to prefer a rule over another—that is, the restriction sets contain triplets of rules (instead of pairs of rules), where the first rule is a rule of FA, the second rule is a main rule of CFG, and the third rule is an alternative rule of CFG, which is used only if the main rule is not applicable.

Definition 7.18 (RT with appearance checking)
The *RT with appearance checking*, $\mathrm{RT_{ac}}$ for short, is a triplet $\Gamma = (M, G, \Psi)$, where $M = (Q, \Sigma, \delta, q_0, F)$ is a finite automaton, $G = (N, T, P, S)$ is a context-free grammar, and Ψ is a finite set of triplets of the form (r_1, r_2, r_3) such that $r_1 \in \delta$ and $r_2, r_3 \in P$.

Let $\chi = (pav_1, uAv_2)$ and $\chi' = (qv_1, uxv_2)$, where $A \in N$, $v_2, x, u \in (N \cup T)^*$, $v_1 \in \Sigma^*$, $a \in \Sigma \cup \{\varepsilon\}$, and $p, q \in Q$, be two 2-configurations. Γ makes a computation step from χ to χ', written as $\chi \Rightarrow \chi'$, if and only if for some $(r_1, r_2, r_3) \in \Psi$, $pav_1 \Rightarrow qv_1[r_1]$ in M, and either

- $uAv_2 \Rightarrow uxv_2[r_2]$ in G, or
- $uAv_2 \Rightarrow uxv_2[r_3]$ in G and r_2 is not applicable on uAv_2 in G.

The 2-language 2-$L(\Gamma)$ and languages $L(\Gamma)_1, L(\Gamma)_2$ are defined in the same way as in Definition 7.1. The classes of languages defined by the first and the second component in the system are denoted by $\mathscr{L}(RT_{ac})_1$ and $\mathscr{L}(RT_{ac})_2$, respectively.

By the appearance checking both generative and accepting power of RT grow to the power of Turing machines.

Theorem 7.19 $\mathscr{L}(RT_{ac})_2 = \mathbf{RE}$.

Proof of Theorem 7.19. Because $\mathscr{L}(MAT_{ac}) = \mathbf{RE}$ (see [37]), we only need to prove that $\mathscr{L}(MAT_{ac}) \subseteq \mathscr{L}(RT_{ac})_2$.

Consider an $\mathrm{MAT_{ac}}$, $I = ({}_IG, {}_IC)$, and construct $\mathrm{RT_{ac}}$ $\Gamma = ({}_\Gamma M, {}_\Gamma G, \Psi)$, such that $L(I) = L(\Gamma)_2$, as follows: set ${}_\Gamma G = {}_IG$; add a new initial nonterminal S', nonterminal Δ, and rules $\Delta \to \Delta$, $\Delta \to \varepsilon$, $S' \to S\Delta$ into grammar ${}_\Gamma G$; and construct finite automaton ${}_\Gamma M = (Q, \Sigma, \delta, s, F)$ and Ψ in the following way: Set $F, Q = \{s\}$, $\delta = \{s \to s\}$, and $\Psi = \{(s \to s, \Delta \to \varepsilon, \Delta \to \varepsilon), (s \to s, S' \to S\Delta, S' \to S\Delta)\}$; for every $m = (p_1, t_1) \dots (p_k, t_k) \in {}_IC$, add $q_1, q_2, \dots, q_{k-1}$ into Q, $s \to q_1, q_1 \to q_2, \dots, q_{k-2} \to q_{k-1}, q_{k-1} \to s$ into δ, and $(s \to q_1, p_1, c_1), (q_1 \to q_2, p_2, c_2), \dots, (q_{k-2} \to q_{k-1}, p_{k-1}, c_{k-1})$, $(q_{k-1} \to q_s, p_k, c_k)$ into Ψ, where, for $1 \leq i \leq k$, if $t_i = -$, then $c_i = p_i$; otherwise, $c_i = \Delta \to \Delta$.

Since S' is the initial symbol, the first computation step in Γ is $(s, S') \Rightarrow (s, S\Delta)$. After this step, the finite automaton simulates matrices in I by moves. That is, if $x_1 \Rightarrow x_2[p]$ in I, where $p = p_1 \dots p_i$ for some $i \in \mathbb{N}$, then there is $q_1, \dots, q_{i-1} \in Q$ such that $r_1 = s \to q_1, r_2 =$

$q_1 \to q_2, \ldots, r_{i-1} = q_{i-2} \to q_{i-1}, r_i = q_{i-1} \to s \in \delta$ and $(r_1, p_1, c_1), \ldots, (r_i, p_i, c_i) \in \Psi$. Therefore, $(s, x_1) \Rightarrow^i (s, x_2)$ in Γ. Notice that if I can overleap some grammar rule in $m \in {}_IC$, Γ represents the fact by using $\Delta \to \Delta$ with the move in ${}_\Gamma M$. Similarly, if, for some $i \in \mathbb{N}$, $(s, x_1) \Rightarrow^i (s, x_2)$ in Γ and there is no $j < i$ such that $(s, x_1) \Rightarrow^j (s, y) \Rightarrow^* (s, x_2)$, there exists $p \in {}_IC$ such that $x_1 \Rightarrow x_2[p]$ in I. Hence, if $(s, S) \Rightarrow^* (s, w)$ in Γ, where w is a string over the set of terminals in ${}_\Gamma G$, then $S \Rightarrow^* w$ in I; and, on the other hand, if $S \Rightarrow^* w$ in I for a string over the set of terminals in ${}_IG$, then $(s, S') \Rightarrow (s, S\Delta) \Rightarrow^* (s, w\Delta) \Rightarrow (s, w)$ in Γ. ■

Theorem 7.20 $\mathscr{L}(RT_{ac})_1 = \mathbf{RE}$.

Proof of Theorem 7.20. Let $I = ({}_IQ, \Sigma, {}_I\delta, q_0, F)$ be a k-CA for some $k \geq 1$ and construct RT $\Gamma = (M = ({}_MQ, \Sigma, {}_M\delta, q_0, F), G = (N, T, P, S), \Psi)$ as follows: Set $T = \{a\}, \Psi = \emptyset, N = \{S, \Diamond, A_1, \ldots, A_k\}, P = \{A \to \varepsilon, A \to \Diamond |\ A \in N - \{\Diamond\}\} \cup \{S \to S\}$, and ${}_MQ = {}_IQ$, ${}_M\delta = \{f \to f |\ f \in F\}$. For each $pa \to q(t_1, \ldots, t_k)$ in ${}_I\delta$, $n = \Sigma_{i=1}^k \theta(t_i)$, and $m = \Sigma_{i=1}^k \widehat{\theta}(t_i)$, where if $t_i \in \mathbb{Z}$, $\theta(t_i) = max(0, -t_i)$ and $\widehat{\theta}(t_i) = max(0, t_i)$; otherwise $\theta(t_i) = 1$ and $\widehat{\theta}(t_i) = 0$, add:

- $q_1, \ldots, q_n$ into ${}_MQ$;
- $r = S \to xS$, where $x \in (N - \{S, \Diamond\})^*$ and $occur(A_i, x) = \widehat{\theta}(t_i)$, for each $i = 1, \ldots, k$, into P;
- $r_1 = q_0a \to q_1$, $r_2 = q_1 \to q_2$, ..., $r_n = q_{n-1} \to q_n$, $r_{n+1} = q_n \to q$ into ${}_M\delta$ with $q_0 = p$; and for each $i = 1, \ldots, n$, add $(r_{i+1}, \tau_i, \tau_i')$, where for each $j = 1, \ldots, k$, if $t_j \in \mathbb{N}$, for $\theta(t_j)$ *i*s, $\tau_i = \tau_i' = A_j \to \varepsilon$; otherwise, if $t_j = -$, $\tau_i = A_j \to \Diamond$ and $\tau_i' = S \to S$, into Ψ. Notice that $n = 0$ means that only $q_0a \to q$, $S \to xS$ are considered. Furthermore, add (r_1, r, r) into Ψ;
- $(f \to f, S \to \varepsilon, S \to \varepsilon)$ into Ψ for all $f \in F$.

Similarly as in the proof of Theorem 7.6, the finite automaton of the created system uses context-free grammar as an external storage, and each counter of the I is represented by a nonterminal. If I modifies some counters during a move from state p to state q, M moves from p to q in several steps during which it changes the numbers of occurrences of nonterminals correspondingly. Rules applicable only if some counters are equal to zero are simulated by using an appearance checking, where Γ tries to replace all nonterminals representing counters which have to be 0 by $\Diamond$. If it is not possible, Γ applies the rule $S \to S$ and continue with computation. Otherwise, since $\Diamond$ cannot be rewritten during the rest of computation, use of such rule leads to an unsuccessful computation. The formal proof of the equivalence of languages is left to the reader. Since, $\mathscr{L}(k\text{-}CA) = \mathbf{RE}$ for every $k \geq 2$ (see [58]), Theorem 7.20 holds. ■

7.2 Applications in Natural Language Translation

In this section, we present several examples illustrating potential applications of the formal models discussed in this paper in natural language processing (NLP), particularly in translation.

Currently, mainly because of the availability of large corpora (both unannotated and annotated) for many languages, statistical approaches with the application of machine learning are dominant in machine translation and NLP in general. Many machine translation systems are based on n-gram models and use little or no syntactic information (see [9]). However, the recent trend consists in incorporating global information of this kind (global within the scope of the sentence) into translation systems in an effort to improve the results. This approach is usually called syntax-based or syntax augmented translation (see [64], [112]). An in-depth description of the formal background of one such system can be found in [7].

In this paper, we present a new formalism that can be applied within the context of such systems to describe syntactic structures and their transformations, and we study its properties. RT provides an alternative to formal models currently used in translation systems, such as synchronous grammars (see [21]).

One of the main advantages of the RT formalism lies in its straightforward and intuitive basic principle. Indeed, we simply read the input with an FA, while generating the corresponding output using a CFG. Another important advantage is the power of RT (see section 7.1). RT has the ability to describe features of natural languages that are difficult or even impossible to capture within a purely context-free framework, such as non-projectivity, demonstrated below.

RT can be easily extended and adapted for use in statistical NLP as well. For example, similarly to probabilistic CFG (see [62]), we can assign probabilities to rules, or to pairs of rules in the control set.

First, to demonstrate the basic principles, we perform the passive transformation of a simple English sentence

The cat caught the mouse.

The passive transformation means transforming a sentence in active voice into passive, and it is a well-known principle that is common to many languages. For the above sentence, the passive form is

The mouse was caught by the cat.

Figure 7.2 shows derivation trees for the above sentences. Throughout this section, we use the following notation to represent common linguistic constituents:

AUX	auxiliary verb	P	preposition
DET	determiner	PN	pronoun
N	noun	PP	prepositional phrase
NP	noun phrase	V	verb
NP-SBJ	noun phrase in the role of subject	VP	verb phrase

Essentially, what we need to do is the following: swapping the subject and the object, adding the preposition *by* in the correct position, and changing the verb into passive, using the auxiliary verb *to be* in the appropriate form. The verb *to be* is irregular and has many different forms (paradigms) depending not only on tense, but also person and number. In most cases, we can see the tense directly from the main verb in the active form, but for the other two categories (person and number), we need to look at the subject (the object in the original sentence).

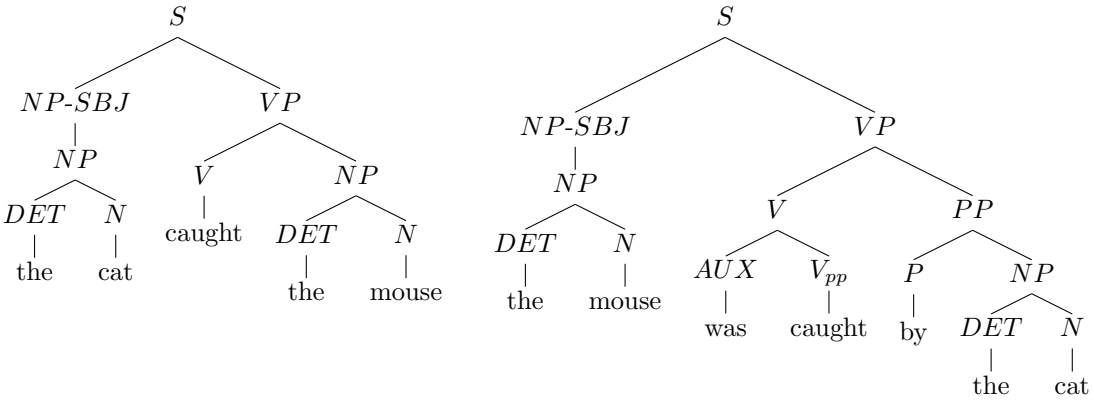


Figure 7.2: Example of the passive transformation

Example 7.21
Consider an RT $\Gamma = (M, G, \Psi)$, where $M = (Q, \Sigma, \delta, 0, F)$, $G = (N, T, P, S)$. Let $Q = \{0, 1, 2, 3, 4, 5, 6, 7, 8a, 8b, 8c, 8d, 8e, 8f, 9\}$, $\Sigma = \{\mathrm{N}_{1s}, \mathrm{N}_{2s}, \mathrm{N}_{3s}, \mathrm{N}_{1p}, \mathrm{N}_{2p}, \mathrm{N}_{3p}, \mathrm{V}_{pas}, \mathrm{V}_{ps}, \mathrm{V}_{pp}, \mathrm{DET}, \mathrm{P}, \mathrm{AUX}_{pas1s}, \mathrm{AUX}_{pas2s}, \mathrm{AUX}_{pas3s}, \mathrm{AUX}_{pas1p}, \mathrm{AUX}_{pas2p}, \mathrm{AUX}_{pas3p}, \mathrm{AUX}_{ps1s}, \mathrm{AUX}_{ps2s}, \mathrm{AUX}_{ps3s}, \mathrm{AUX}_{ps1p}, \mathrm{AUX}_{ps2p}, \mathrm{AUX}_{ps3p}\}$, $F = \{9\}$, $N = \{\mathrm{S}, \mathrm{NP\text{-}SBJ}, \mathrm{NP}, \mathrm{VP}, \mathrm{PP}, \mathrm{N}_?, \mathrm{V}_?, \mathrm{AUX}_{pas?}, \mathrm{AUX}_{ps?}\}$, $T = \Sigma$. Furthermore, let δ, P, and Ψ are given by next:

$$
\begin{array}{llllll}
\delta = \{ & r_1 = & 0 \to 1, & r_{6a} = & 3\mathrm{V}_{pas} \to 5, & r_{10a} = \quad 8a \to 9, \\
& r_2 = & 1 \to 2, & r_{6b} = & 3\mathrm{V}_{ps} \to 5, & r_{10b} = \quad 8b \to 9, \\
& r_3 = & 2 \to 3, & r_7 = & 5 \to 6, & \vdots \\
& r_4 = & 3\mathrm{DET} \to 4, & r_8 = & 6\mathrm{DET} \to 7, & \\
& r_{5a} = & 4\mathrm{N}_{1s} \to 3, & r_{9a} = & 7\mathrm{N}_{1s} \to 8a, & r_{10f} = \quad 8f \to 9, \\
& r_{5b} = & 4\mathrm{N}_{2s} \to 3, & r_{9b} = & 7\mathrm{N}_{2s} \to 8b, & \\
& \vdots & & \vdots & & \\
& r_{5f} = & 4\mathrm{N}_{3p} \to 3, & r_{9f} = & 7\mathrm{N}_{3p} \to 8f\} &
\end{array}
$$

$$
\begin{array}{llll}
P = \{ & p_1 = & \mathrm{S} \to \mathrm{NP\text{-}SBJ\ VP}, & p_{8a} = \quad \mathrm{AUX}_{pas?} \to \mathrm{AUX}_{pas1s}, \\
& p_2 = & \mathrm{NP\text{-}SBJ} \to \mathrm{NP}, & p_{8b} = \quad \mathrm{AUX}_{pas?} \to \mathrm{AUX}_{pas2s}, \\
& p_3 = & \mathrm{NP} \to \mathrm{DET\ N}_?, & \vdots \\
& p_{4a} = & \mathrm{N}_? \to \mathrm{N}_{1s}, & \\
& p_{4b} = & \mathrm{N}_? \to \mathrm{N}_{2s}, & p_{8f} = \quad \mathrm{AUX}_{pas?} \to \mathrm{AUX}_{pas3p}, \\
& & & p_{9a} = \quad \mathrm{AUX}_{ps?} \to \mathrm{AUX}_{ps1s}, \\
& \vdots & & p_{9b} = \quad \mathrm{AUX}_{ps?} \to \mathrm{AUX}_{ps2s}, \\
& p_{4f} = & \mathrm{N}_? \to \mathrm{N}_{3p}, & \vdots \\
& p_5 = & \mathrm{VP} \to \mathrm{V}_? \ \mathrm{PP}, & \\
& p_6 = & \mathrm{PP} \to \mathrm{P\ NP}, & p_{9f} = \quad \mathrm{AUX}_{ps?} \to \mathrm{AUX}_{ps3p}, \\
& p_{7a} = & \mathrm{V}_? \to \mathrm{AUX}_{pas?}\ \mathrm{V}_{pp}, & \\
& p_{7b} = & \mathrm{V}_? \to \mathrm{AUX}_{ps?}\ \mathrm{V}_{pp}\} &
\end{array}
$$

$\Psi = \{(r_1, p_1), (r_2, p_5), (r_3, p_6), (r_4, p_3), (r_{5a}, p_{4a}), (r_{5b}, p_{4b}), \ldots, (r_{5f}, p_{4f}), (r_{6a}, p_{7a}), (r_{6b}, p_{7b}), (r_7, p_2), (r_8, p_3), (r_{9a}, p_{4a}), (r_{9b}, p_{4b}), \ldots, (r_{9f}, p_{4f}), (r_{10a}, p_{8a}), (r_{10b}, p_{8b}), \ldots, (r_{10f}, p_{8f}), (r_{10a}, p_{9a}), (r_{10b}, p_{9b}), \ldots, (r_{10f}, p_{9f})\}$

One may notice that the input alphabet of the automaton, as well as the terminal alphabet of the grammar, does not contain the actual words themselves, but rather symbols repre-

senting word categories and their properties (for example, N_{3s} represents a noun in third person singular). In the examples throughout this section, we consider syntax analysis and translation on the abstract level, transforming syntactic structures in languages. That is, we assume that we already have the input sentence split into words (or possibly some other units as appropriate), and these words are tagged as, for example, a noun, pronoun, or verb.

In the computation examples, the text in square brackets shows the words associated with the symbols for the given example sentence, but note that this is not a part of the formalism itself. This specifier is assigned to all terminals. Nonterminals are only specified by words when the relation can be established from the computation so far performed (for example, we cannot assign a word before we read the corresponding input token).

For the sentence *the cat caught the mouse* (for the purposes of this text, we disregard capitalization and punctuation) from the above example, the computation can proceed as follows:

(0 DET[*the*] N_{3s}[*cat*] V_{pas}[*caught*] DET[*the*] N_{3s}[*mouse*], $\underline{\text{S}}$)
⇒ (1 DET[*the*] N_{3s}[*cat*] V_{pas}[*caught*] DET[*the*] N_{3s}[*mouse*], NP-SBJ $\underline{\text{VP}}$) [(r_1, p_1)]
⇒ (2 DET[*the*] N_{3s}[*cat*] V_{pas}[*caught*] DET[*the*] N_{3s}[*mouse*], NP-SBJ $V_?$ $\underline{\text{PP}}$) [(r_2, p_5)]
⇒ (3 $\underline{\text{DET}}$[*the*] N_{3s}[*cat*] V_{pas}[*caught*] DET[*the*] N_{3s}[*mouse*], NP-SBJ $V_?$ P[*by*] $\underline{\text{NP}}$) [(r_3, p_6)]
⇒ (4 $\underline{N_{3s}[cat]}$ V_{pas}[*caught*] DET[*the*] N_{3s}[*mouse*], NP-SBJ $V_?$ P[*by*] DET[*the*] $\underline{N_?}$) [(r_4, p_3)]
⇒ (3 $\underline{V_{pas}[caught]}$ DET[*the*] N_{3s}[*mouse*], NP-SBJ $\underline{V_?}$ P[*by*] DET[*the*] N_{3s}[*cat*]) [(r_{5c}, p_{4c})]
⇒ (5 DET[*the*] N_{3s}[*mouse*], $\underline{\text{NP-SBJ}}$ $AUX_{pas?}$[*be*] V_{pp}[*caught*] P[*by*] DET[*the*] N_{3s}[*cat*]) [(r_{6a}, p_{7a})]
⇒ (6 $\underline{\text{DET}}$[*the*] N_{3s}[*mouse*], $\underline{\text{NP}}$ $AUX_{pas?}$[*be*] V_{pp}[*caught*] P[*by*] DET[*the*] N_{3s}[*cat*]) [(r_7, p_2)]
⇒ (7 $\underline{N_{3s}[mouse]}$, DET[*the*] $\underline{N_?}$ $AUX_{pas?}$[*be*] V_{pp}[*caught*] P[*by*] DET[*the*] N_{3s}[*cat*]) [(r_8, p_3)]
⇒ (8*c*, DET[*the*] N_{3s}[*mouse*] $\underline{AUX_{pas?}[be]}$ V_{pp}[*caught*] P[*by*] DET[*the*] N_{3s}[*cat*]) [(r_{9c}, p_{4c})]
⇒ (9, DET[*the*] N_{3s}[*mouse*] AUX_{pas3s}[*was*] V_{pp}[*caught*] P[*by*] DET[*the*] N_{3s}[*cat*]) [(r_{10c}, p_{8c})]

For clarity, in each computation step, the input symbol to be read (if any) and the nonterminal to be rewritten are underlined.

First (in states 0, 1, and 2), we generate the expected basic structure of the output sentence. Note that this is done before reading any input. In states 3 and 4, we read the subject of the original sentence, states 5 and 6 read the verb, and the rest of the states is used to process the object. When we read the verb, we generate its passive form, consisting of *to be* and the verb in past participle. However, at this point, we know the tense (in this case, past simple), but do not know the person or number yet. The missing information is represented by the question mark (?) symbol in the nonterminal $AUX_{pas?}$. Later, when we read the object of the original sentence, we rewrite $AUX_{pas?}$ to a terminal. In this case, the object is in third person singular, which gives us the terminal AUX_{pas3s} (meaning that the correct form to use here is *was*).

Next, we present examples of translation between different languages. We focus on Japanese, Czech, and English.

One problem when translating into Czech is that there is very rich inflection. The form of the words reflects many grammatical categories, such as case, gender, and number (see [54], where the author discusses this issue with regard to computational linguistics). To illustrate, compare the following sentences in Japanese, English, and Czech.

Zasshi o yondeitta onna no hito wa watashi no shiriai deshita.
Zasshi o yondeitta otoko no hito wa watashi no shiriai deshita.

The woman who was reading a magazine was an acquitance of mine.
The man who was reading a magazine was an acquitance of mine.

Žena, která četla časopis, byla moje známá.
Muž, který četl časopis, byl můj známý.

As we can see, in Czech, nearly every word is different, depending on the gender of the subject. In contrast, in both Japanese and English, the two sentences only differ in one word – *onna no hito* (*woman*) and *otoko no hito* (*man*).[1]

The above sentences also give us an example of some structural differences between Japanese and Czech. In Czech and English, the structure of the sentence is very similar, but in Japanese, there is no word that correspond directly to *který* (*which*, *who*, ...). Instead, this relation is represented by the form of the verb *yondeitta* (the dictionary form is *yomu*, meaning *to read*). Compare the derivation trees in Figure 7.3.

Example 7.22
Consider an RT $\Gamma = (M, G, \Psi)$, where $M = (Q, \Sigma, \delta, 0, F)$, $G = (N, T, P, S)$. Let $Q = \{0, m, m1, m2, f, f1, f2, n, n1, n2, 1m, 1f, 1n\}$, $\Sigma = \{\mathrm{NP}_m, \mathrm{NP}_f, \mathrm{NP}_n, \mathrm{NP}_?, \mathrm{V}, \mathrm{DET}, \#\}$, $F = \{1m, 1f, 1n\}$, $N = \{$ S, NP-SBJ, $\mathrm{NP}_?$, VP, $\mathrm{PN}_?$, $\mathrm{V}_?$, X$\}$, $T = \{\mathrm{NP}_m, \mathrm{NP}_f, \mathrm{NP}_n, \mathrm{V}_m, \mathrm{V}_f, \mathrm{V}_n, \mathrm{PN}_m, \mathrm{PN}_f, \mathrm{PN}_n\}$. Let δ, P, and Ψ are defined as follows:

$$
\begin{array}{llllll}
\delta = \{ & r_1 = & 0\mathrm{V} \to 1, & r_{m1} = & m\mathrm{V} \to m1, & r_{f1} = \quad f\mathrm{V} \to f1, \\
& r_2 = & 1 \to 0, & r_{m2} = & m1 \to m2, & r_{f2} = \quad f1 \to f2, \\
& r_3 = & 0\mathrm{NP}_? \to 0, & r_{m3} = & m2 \to m, & \\
& r_4 = & 0\mathrm{DET} \to 0, & r_{m4} = & m\mathrm{DET} \to m, & \vdots \\
& r_{5m} = & 0\mathrm{NP}_m \to m, & r_{m5} = & m\mathrm{NP}_? \to m, & r_{f7} = \quad 1f \to 1f, \\
& r_{5f} = & 0\mathrm{NP}_f \to f, & r_{m5m} = & m\mathrm{NP}_m \to m, & r_{n1} = \quad n\mathrm{V} \to n1, \\
& r_{5n} = & 0\mathrm{NP}_n \to n, & r_{m5f} = & m\mathrm{NP}_f \to m, & r_{n2} = \quad n1 \to n2, \\
& & & r_{m5n} = & m\mathrm{NP}_n \to m, & \\
& & & r_{m6} = & m\# \to 1m, & \vdots \\
& & & r_{m7} = & 1m \to 1m, & r_{n7} = \quad 1n \to 1n\}
\end{array}
$$

[1] Technically, *onna no hito* literally translates to *woman's person* or *female person*, with *onna* itself meaning *woman*, *female*. However, referring to a person only by *onna* may have negative connotations in Japanese. Similarly for *otoko no hito*.

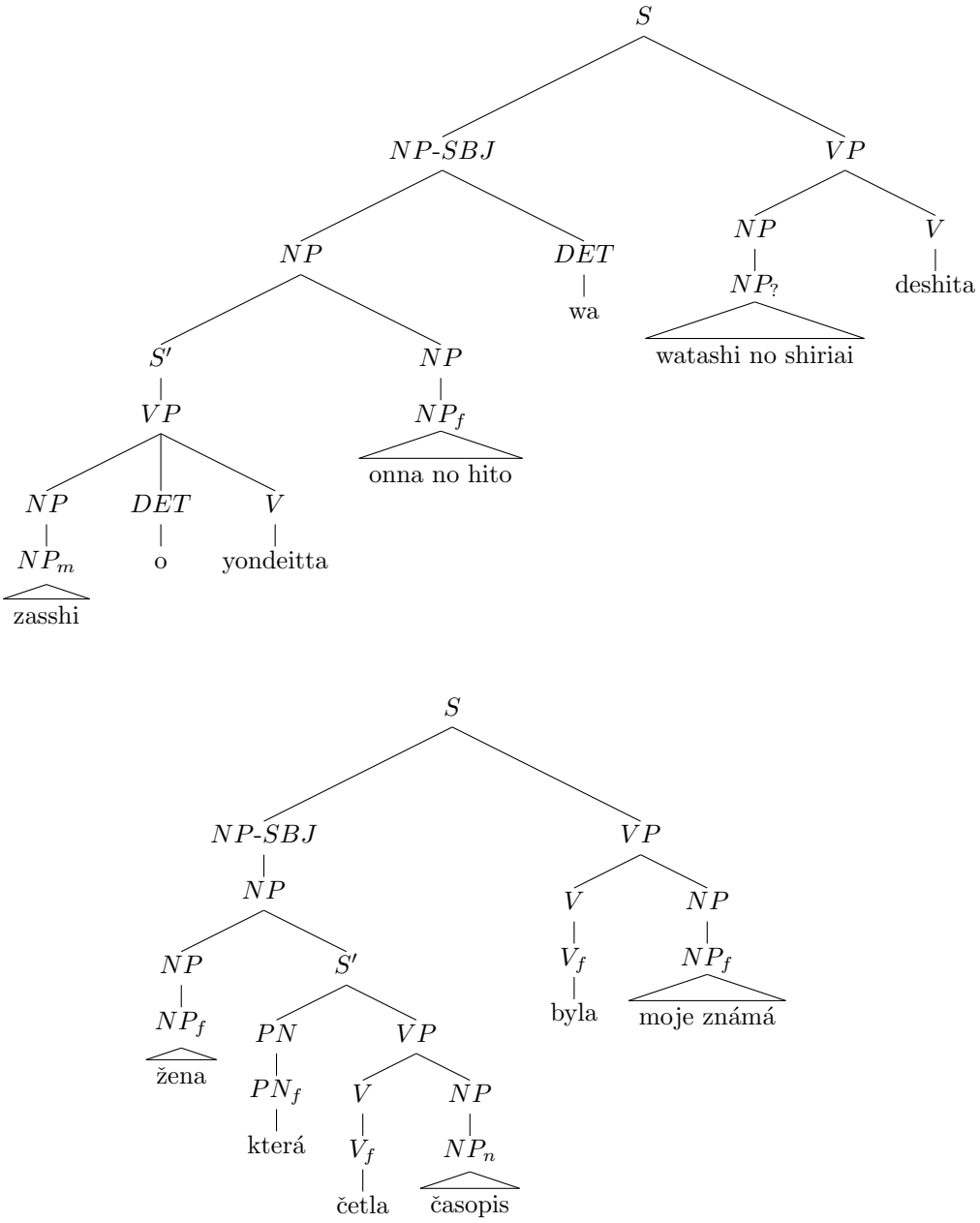


Figure 7.3: Derivation trees – Japanese (top) and Czech (bottom)

$$
\begin{aligned}
P = \{ \quad & p_1 = \text{S} \rightarrow \text{NP-SBJ VP X}, & p_{7m} &= \text{V}_? \rightarrow \text{V}_m, \\
& p_2 = \text{NP-SBJ} \rightarrow \text{NP}_?, & p_{7f} &= \text{V}_? \rightarrow \text{V}_f, \\
& p_{3m} = \text{NP}_? \rightarrow \text{NP}_m, & p_{7n} &= \text{V}_? \rightarrow \text{V}_n, \\
& p_{3f} = \text{NP}_? \rightarrow \text{NP}_f, & p_8 &= \text{S}' \rightarrow \text{PN}_? \text{ VP}, \\
& p_{3n} = \text{NP}_? \rightarrow \text{NP}_n, & p_{9m} &= \text{PN}_? \rightarrow \text{PN}_m, \\
& p_4 = \text{NP}_? \rightarrow \text{NP}_?, & p_{9f} &= \text{PN}_? \rightarrow \text{PN}_f, \\
& p_5 = \text{NP}_? \rightarrow \text{NP}_? \text{ S}', & p_{9n} &= \text{PN}_? \rightarrow \text{PN}_n, \\
& p_6 = \text{VP} \rightarrow \text{V}_? \text{ NP}_?, & p_{10} &= \text{X} \rightarrow \varepsilon \}
\end{aligned}
$$

$\Psi = \{(r_1, p_1), (r_2, p_6), (r_3, p_4), (r_4, p_2), (r_{5m}, p_4), (r_{5f}, p_4), (r_{5n}, p_4), (r_{m1}, p_5), (r_{m2}, p_8), (r_{m3}, p_6), (r_{m4}, p_4), (r_{m5}, p_4), (r_{m5m}, p_{3m}), (r_{m5f}, p_{3f}), (r_{m5n}, p_{3n}), (r_{m6}, p_{10}), (r_{m7}, p_{3m}), (r_{m7}, p_{7m}), (r_{m7}, p_{9m}), (r_{f1}, p_5), (r_{f2}, p_8), (r_{f3}, p_6), (r_{f4}, p_4), (r_{f5}, p_4), (r_{f5m}, p_{3m}), (r_{f5f}, p_{3f}), (r_{f5n}, p_{3n}), (r_{f6}, p_{10}), (r_{f7}, p_{3f}), (r_{f7}, p_{7f}), (r_{f7}, p_{9f}), (r_{n1}, p_5), (r_{n2}, p_8), (r_{n3}, p_6), (r_{n4}, p_4), (r_{n5}, p_4), (r_{n5m}, p_{3m}), (r_{n5f}, p_{3f}), (r_{n5n}, p_{3n}), (r_{n6}, p_{10}), (r_{n7}, p_{3n}), (r_{n7}, p_{7n}), (r_{n7}, p_{9n})\}$

We have added two dummy symbols: the input symbol #, which acts as the endmarker, and the nonterminal X, which we generate at the beginning of the computation and then erase when all the input has been read (including #).

In this example, we read the input sentence in reverse order (right to left). Clearly, this makes no difference from a purely theoretical point of view, but it may be more suitable in practice due to the way how Japanese sentences are organized. The computation transforming the sentence *zasshi o yondeitta onna no hito wa watashi no shiriai deshita* into *žena, která četla časopis, byla moje známá* can proceed as follows:

(0 V[*deshita*] $\text{NP}_?$[*watashi no shiriai*] DET[*wa*] NP_f[*onna no hito*] V[*yondeitta*] DET[*o*] NP_m[*zasshi*] #, S)

⇒ (1 $\text{NP}_?$[*watashi no shiriai*] DET[*wa*] NP_f[*onna no hito*] V[*yondeitta*] DET[*o*] NP_m[*zasshi*] #, NP-SBJ VP X) $[(r_1, p_1)]$

⇒ (0 $\text{NP}_?$[*watashi no shiriai*] DET[*wa*] NP_f[*onna no hito*] V[*yondeitta*] DET[*o*] NP_m[*zasshi*] #, NP-SBJ $\text{V}_?$[*byl*] $\text{NP}_?$ X) $[(r_2, p_6)]$

⇒ (0 DET[*wa*] NP_f[*onna no hito*] V[*yondeitta*] DET[*o*] NP_m[*zasshi*] #, NP-SBJ $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*] X) $[(r_3, p_4)]$

⇒ (0 NP_f[*onna no hito*] V[*yondeitta*] DET[*o*] NP_m[*zasshi*] #, $\text{NP}_?$ $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*] X) $[(r_4, p_2)]$

⇒ (*f* V[*yondeitta*] DET[*o*] NP_m[*zasshi*] #, $\text{NP}_?$[*žena*] $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*] X) $[(r_{5f}, p_4)]$

⇒ (*f*1 DET[*o*] NP_m[*zasshi*] #, $\text{NP}_?$[*žena*] S′ $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*] X) $[(r_{f1}, p_5)]$

⇒ (*f*2 DET[*o*] NP_m[*zasshi*] #, $\text{NP}_?$[*žena*] $\text{PN}_?$[*který*] VP $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*] X) $[(r_{f2}, p_8)]$

⇒ (*f* DET[*o*] NP_m[*zasshi*] #, $\text{NP}_?$[*žena*] $\text{PN}_?$[*který*] $\text{V}_?$[*četl*] $\text{NP}_?$ $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*] X) $[(r_{f3}, p_6)]$

⇒ (*f* NP_m[*zasshi*] #, $\text{NP}_?$[*žena*] $\text{PN}_?$[*který*] $\text{V}_?$[*četl*] $\text{NP}_?$ $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*] X) $[(r_{f4}, p_4)]$

⇒ (*f* #, $\text{NP}_?$[*žena*] $\text{PN}_?$[*který*] $\text{V}_?$[*četl*] NP_m[*časopis*] $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*] X) $[(r_{f5m}, p_{3m})]$

⇒ (1*f*, $\text{NP}_?$[*žena*] $\text{PN}_?$[*který*] $\text{V}_?$[*četl*] NP_m[*časopis*] $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*]) $[(r_{f6}, p_{10})]$

⇒ (1*f*, NP_f[*žena*] $\text{PN}_?$[*který*] $\text{V}_?$[*četl*] NP_m[*časopis*] $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*]) $[(r_{f7}, p_{3f})]$

⇒ (1*f*, NP_f[*žena*] PN_f[*která*] $\text{V}_?$[*četl*] NP_m[*časopis*] $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*]) $[(r_{f7}, p_{9f})]$

⇒ (1*f*, NP_f[*žena*] PN_f[*která*] V_f[*četla*] NP_m[*časopis*] $\text{V}_?$[*byl*] $\text{NP}_?$[*můj známý*]) $[(r_{f7}, p_{7f})]$

⇒ (1*f*, NP_f[*žena*] PN_f[*která*] V_f[*četla*] NP_m[*časopis*] V_f[*byla*] $\text{NP}_?$[*můj známý*]) $[(r_{f7}, p_{7f})]$

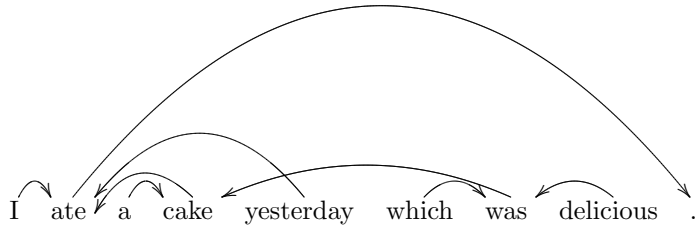


Figure 7.4: Non-projective dependency tree (English)

$$\Rightarrow \quad (1f,\ \mathrm{NP}_f[\check{z}ena]\ \ \mathrm{PN}_f[kter\acute{a}]\ \ \mathrm{V}_f[\check{c}etla]\ \ \mathrm{NP}_m[\check{c}asopis]\ \ \mathrm{V}_f[byla]\ \ \mathrm{NP}_f[moje\ zn\acute{a}m\acute{a}])\ [(r_{f7}, p_{3f})]$$

When we first read the word that determines the gender, we move to the state that represents this gender (state m, n, or f). Note that these states are functionally identical in the sense that we can read the same input symbols, while performing the same computation steps in the grammar generating the output. After we have reached the end of input, we rewrite the nonterminal symbols representing words with as of yet unknown gender to the corresponding terminal symbols, depending on the state.

Czech is considered a free-word-order language, which means that it allows for a wide range of permutations of words in a sentence without changing its syntactic structure (the meaning of the sentence may be affected). This is perhaps the main source of the relatively high amount of non-projectivity in Czech sentences.

Non-projectivity means that there are cross-dependencies. For example, consider the English sentence

I ate a cake yesterday which was delicious.

As shown in the dependency tree in Figure 7.4, there is a crossing of dependencies represented by arrows (*yesterday*, *ate*) and (*was*, *cake*). Arrows are drawn from child (modifier) to parent (head).

Arguably, the English example is somewhat artificial – even though the sentence is well-formed, in most cases it might be more natural to say simply

I ate a delicious cake yesterday.

In contrast, in Czech, a sentence such as

Nevím, jaký je mezi nimi rozdíl.
(I don't know what the difference between them is.)

is not at all unusual. The dependency tree for this sentence (see Figure 7.5) is also non-projective. For further information about the projectivity, and the issue of the non-projectivity in the Czech language in particular, see [55].

The following example illustrates how the formalism accounts for the non-projectivity.

Example 7.23
Consider an RT $\Gamma = (M, G, \Psi)$, where $M = (Q, \Sigma, \delta, 0, F)$, $G = (N, T, P, S)$. Let $Q =$

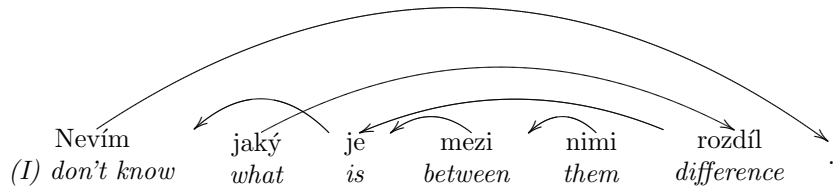


Figure 7.5: Non-projective dependency tree (Czech)

$\{0, 1, 2, 3, 4, 5, 6m, 6f, 6n\}$, $\Sigma = \{\mathrm{N}_m, \mathrm{N}_f, \mathrm{N}_n, \mathrm{V}, \mathrm{PN}, \mathrm{PN}_i, \mathrm{DET}, \mathrm{P}\}$, $F = \{0\}$, $N = \{\mathrm{S}, \text{NP-SBJ}, \mathrm{NP}, \mathrm{VP}, \mathrm{PP}, \mathrm{V}_?\}$, $T = \{\mathrm{N}_m, \mathrm{N}_f, \mathrm{N}_n, \mathrm{V}, \mathrm{PN}_m, \mathrm{PN}_f, \mathrm{PN}_n, \mathrm{P}\}$,

$$\begin{array}{llllll}
\delta = \{ & r_{1m} = & 0\mathrm{N}_m \to 6m, & r_5 = & 0\mathrm{DET} \to 0, & r_{11} = \quad 5 \to 0, \\
& r_{1f} = & 0\mathrm{N}_f \to 6f, & r_6 = & 0\mathrm{P} \to 0, & r_{12m} = \quad 6m \to 0, \\
& r_{1n} = & 0\mathrm{N}_n \to 6n, & r_7 = & 1 \to 2, & r_{12f} = \quad 6f \to 0, \\
& r_2 = & 0\mathrm{V} \to 0, & r_8 = & 2 \to 0, & r_{12n} = \quad 6n \to 0, \\
& r_3 = & 0\mathrm{PN} \to 1, & r_9 = & 3 \to 4, & r_{13} = \quad 0\mathrm{PN} \to 0, \\
& r_4 = & 0\mathrm{PN}_i \to 3, & r_{10} = & 4 \to 5\}, &
\end{array}$$

$$\begin{array}{lll}
P = \{ & p_1 = & \mathrm{S} \to \text{NP-SBJ VP}, \\
& p_2 = & \text{NP-SBJ} \to \mathrm{NP}, \\
& p_3 = & \text{NP-SBJ} \to \varepsilon, \\
& p_{4m} = & \mathrm{NP} \to \mathrm{N}_m, \\
& p_{4f} = & \mathrm{NP} \to \mathrm{N}_f, \\
& p_{4n} = & \mathrm{NP} \to \mathrm{N}_n, \\
& p_5 = & \mathrm{NP} \to \mathrm{S}, \\
& p_{6m} = & \mathrm{NP} \to \mathrm{PN}_m, \\
& p_{6f} = & \mathrm{NP} \to \mathrm{PN}_f, \\
& p_{6n} = & \mathrm{NP} \to \mathrm{PN}_n, \\
& p_7 = & \mathrm{VP} \to \mathrm{V}_? \ \mathrm{PP}\ \mathrm{NP}, \\
& p_8 = & \mathrm{V}_? \to \mathrm{V}, \\
& p_9 = & \mathrm{PP} \to \mathrm{P}\ \mathrm{NP}, \\
& p_{10} = & \mathrm{PP} \to \varepsilon\}, \text{ and}
\end{array}$$

$\Psi = \{(r_{1m}, p_{4m}), (r_{1f}, p_{4f}), (r_{1n}, p_{4n}), (r_2, p_8), (r_3, p_1), (r_4, p_{10}), (r_5, p_7), (r_6, p_9), (r_7, p_3), (r_8, p_7), (r_9, p_5), (r_{10}, p_1), (r_{11}, p_2), (r_{12m}, p_{6m}), (r_{12f}, p_{6f}), (r_{12n}, p_{6n}), (r_{13}, p_{6m})\}$.

The computation transforming the English sentence *I don't know what the difference between them is* into the (non-projective) Czech sentence *nevím, jaký je mezi nimi rozdíl* proceeds as follows:

(0 PN[*I*] V[*don't know*] PN_i[*what*] DET[*the*] N_m[*difference*] P[*between*] PN[*them*] V[*is*], S)
⇒ (1 V[*don't know*] PN_i[*what*] DET[*the*] N_m[*difference*] P[*between*] PN[*them*] V[*is*], NP-SBJ VP) $[(r_3, p_1)]$
⇒ (2 V[*don't know*] PN_i[*what*] DET[*the*] N_m[*difference*] P[*between*] PN[*them*] V[*is*], VP) $[(r_7, p_3)]$
⇒ (0 V[*don't know*] PN_i[*what*] DET[*the*] N_m[*difference*] P[*between*] PN[*them*] V[*is*], $\mathrm{V}_?$ PP NP) $[(r_8, p_7)]$
⇒ (0 PN_i[*what*] DET[*the*] N_m[*difference*] P[*between*] PN[*them*] V[*is*], V[*nevím*] PP NP) $[(r_2, p_8)]$
⇒ (3 DET[*the*] N_m[*difference*] P[*between*] PN[*them*] V[*is*], V[*nevím*] NP) $[(r_4, p_{10})]$
⇒ (4 DET[*the*] N_m[*difference*] P[*between*] PN[*them*] V[*is*], V[*nevím*] S) $[(r_9, p_5)]$
⇒ (5 DET[*the*] N_m[*difference*] P[*between*] PN[*them*] V[*is*], V[*nevím*] NP-SBJ VP) $[(r_{10}, p_1)]$

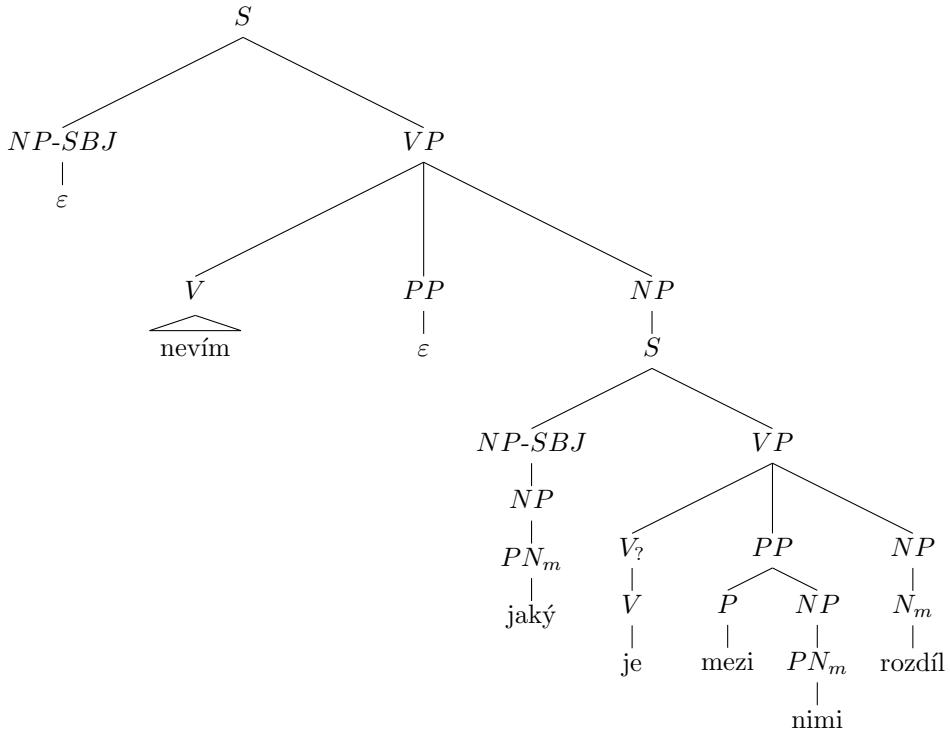


Figure 7.6: The derivation tree of G

$\Rightarrow$ (0 $\underline{\text{DET}[\textit{the}]}$ $\text{N}_m[\textit{difference}]$ $\text{P}[\textit{between}]$ $\text{PN}[\textit{them}]$ $\text{V}[\textit{is}]$, $\text{V}[\textit{nevím}]$ $\text{NP}[\textit{jaký}]$ $\underline{\text{VP}}$) $[(r_{11}, p_2)]$

$\Rightarrow$ (0 $\underline{\text{N}_m[\textit{difference}]}$ $\text{P}[\textit{between}]$ $\text{PN}[\textit{them}]$ $\text{V}[\textit{is}]$, $\text{V}[\textit{nevím}]$ $\text{NP}[\textit{jaký}]$ $\text{V}_?$ PP $\underline{\text{NP}}$) $[(r_5, p_7)]$

$\Rightarrow$ (6m $\text{P}[\textit{between}]$ $\text{PN}[\textit{them}]$ $\text{V}[\textit{is}]$, $\text{V}[\textit{nevím}]$ $\underline{\text{NP}[\textit{jaký}]}$ $\text{V}_?$ PP $\text{N}_m[\textit{rozdíl}]$) $[(r_{1m}, p_{4m})]$

$\Rightarrow$ (0 $\underline{\text{P}[\textit{between}]}$ $\text{PN}[\textit{them}]$ $\text{V}[\textit{is}]$, $\text{V}[\textit{nevím}]$ $\text{PN}_m[\textit{jaký}]$ $\text{V}_?$ $\underline{\text{PP}}$ $\text{N}_m[\textit{rozdíl}]$) $[(r_{12m}, p_{6m})]$

$\Rightarrow$ (0 $\underline{\text{PN}[\textit{them}]}$ $\text{V}[\textit{is}]$, $\text{V}[\textit{nevím}]$ $\text{PN}_m[\textit{jaký}]$ $\text{V}_?$ $\text{P}[\textit{mezi}]$ $\underline{\text{NP}}$ $\text{N}_m[\textit{rozdíl}]$) $[(r_6, p_9)]$

$\Rightarrow$ (0 $\underline{\text{V}[\textit{is}]}$, $\text{V}[\textit{nevím}]$ $\text{PN}_m[\textit{jaký}]$ $\underline{\text{V}_?}$ $\text{P}[\textit{mezi}]$ $\text{PN}_m[\textit{nimi}]$ $\text{N}_m[\textit{rozdíl}]$) $[(r_{13}, p_{6m})]$

$\Rightarrow$ (0, $\text{V}[\textit{nevím}]$ $\text{PN}_m[\textit{jaký}]$ $\text{V}[\textit{je}]$ $\text{P}[\textit{mezi}]$ $\text{PN}_m[\textit{nimi}]$ $\text{N}_m[\textit{rozdíl}]$) $[(r_2, p_8)]$

The corresponding derivation tree of G is shown in Figure 7.6.

In the examples presented in this paper, we have made two important assumptions. First, we already have the input sentence analysed on a low level – we know where every word starts and ends (which may be a non-trivial problem itself in some languages, such as Japanese) and have some basic grammatical information about it. Furthermore, we know the translation of the individual words. For practical applications in natural language translation, we need a more complex system, with at least two other components – a part-of-speech tagger, and a dictionary to translate the actual meanings of the words. Then, the component based on the discussed formal model can be used to transform the syntactic structure and also ensure that the words in the translated sentence are in the correct form.

Chapter 8

Parsing of n-Path-Restricted Tree-Controlled Languages

The previous chapter has introduced and investigated mechanisms for direct translation. In this chapter, we use an automata system for parsing based upon grammars with restricted derivation trees, which represents an important trend in todays formal language theory (see [31, 39, 67, 68, 71, 73, 95]). These grammars generate their languages just like ordinary context-free grammars do, but their derivation trees have to satisfy some simple prescribed conditions. The idea of restrictions placed on the derivation trees of context-free grammars is introduced in [31], and the resulting grammars restricted in this way are referred to as *tree-controlled grammars*. In essence, the notion of a tree-controlled grammar is defined as follows: take a context-free grammar G and a regular language R. A string w generated by G belongs to the language defined by G and R only if there is derivation tree t for w in G such that all *levels* (except the last one) of t are described by R. Based on the original definition of a tree-controlled grammar, the modifications, where many well-known types of both controlled grammars and control languages are considered, are studied in [95].

A new type of restriction in a derivation are studied in [71]. They consider a context-free grammar G and a context-free language R. A string w generated by G belongs to the language defined by G and R only if there is derivation tree t for w in G such that there exists a path of t described by R. Based on this restriction, they introduce *path-controlled grammars* and study several properties of this model. As they state in the final remarks, there are many open problems. Some of them are studied in [73] but still many modifications remain unsolved. In [73], based on the non-emptiness problem for the intersection of two languages, the authors state the polynomial recognizability. However, they say nothing about parsing methods of the languages generated by path-controlled grammars at all.

As a generalization of path-controlled grammars, the authors of [67] introduce several variants of this rewriting model where not just one, but given number of paths in derivation trees of context-free grammars have to be described by a control language. That is, [67] deals with a generalization of path-controlled grammars (G, R), where a string w generated by G belongs to the language defined by G and R only if there is derivation tree t for w in G such that there exist given n paths of t described by a linear language R, where $n \geq 1$. Such a rewriting system is referred to as n-path tree-controlled grammar. As it is demonstrated in Section 8.2, this generalized rewriting system can generate some languages not captured in the rewriting system introduced in [71]. Then, based on the common part of all restricted paths of the derivation trees of context-free grammars, the authors introduce

several modifications and state pumping properties of some of them. The possibilities of parsing methods working in polynomial time for n-path tree-controlled languages are briefly studied in [66]. This chapter continues with the investigation of these grammars and establishes a parsing method based on them. After recalling the restrictions placed on $n \geq 1$ paths in the derivation trees of context-free grammars, we introduce LL restriction (see [103]) and we present the ideas how to parse generated language class (not only how to decide membership problem), where the core of the parsing method stands on abstraction of 3-accepting restricted pushdown automata systems. Essentially, we discuss the following problem: Can we decide whether a string is recognized by n-path tree-controlled grammar in polynomial time, for $n \geq 1$?

8.1 n-Path-Restricted Tree-Controlled Grammars

In this section, as the subject of investigation in this chapter, we recall the derivation-based generalization of tree-controlled grammars based on n-path restriction as it is introduced in [67].

Let $G = (N, T, P, S)$ be a context-free grammar and $x \in T^*$. Let $t \in {}_G\triangle(x)$. A *path* of t is any sequence of the nodes with the first node equals to the root of t, the last node equals to a leaf of t, and there is an edge in t between each two consecutive nodes of the sequence. Let s be any sequence of the nodes of t. Then $word(s)$ denotes the string obtained by concatenation of all labels of the nodes of s in order from left to right.

A *tree-controlled* grammar, TC grammar for short, is a pair (G, R), where $G = (N, T, P, S)$ is a context-free grammar and $R \subseteq (N \cup T)^*$ is a control language. The *language that* (G, R) *generates under the* n*-path control by* R, $n \geq 1$, is denoted by ${}_{n\text{-}path}L(G, R)$ and it is defined by the following equivalence.

For all $x \in T^*$, $x \in {}_{n\text{-}path}L(G, R)$ if and only if there exists derivation tree $t \in {}_G\blacktriangle(x)$ such that there is a set Q_t of n paths of t such that for each $p \in Q_t$, $word(p) \in R$. Set **n-path-TC** $= \{{}_{n\text{-}path}L(G, R) |\ (G, R)$ is a TC grammar$\}$. Hereafter, TC grammars that generate the language under the n-path control are referred to as n-path TC grammars.

For each context-free grammar G, there is a regular language which describes all paths in a derivation tree of a string w in G (see Prop. 1 in [71]). Since for each context-free grammar G, there is no regular control language that increases the generative power of G with R controling the paths (see Prop 1. and Prop. 2 in [71]), we investigate TC grammar with non-regular control language. On the other hand, as it is demonstrated in Section 8.2, linear language used to control the paths in the derivation trees of context-free grammars is strong-enough mechanism to increase the generative power of context-free grammars. Thus, we study n-path TC grammars with linear control languages in the rest of this paper. Therefore, in what follows, for a TC grammar (G, R), we consider R as a linear language.

8.2 Examples

In this section, we demonstrate two non-context-free languages that belong to **n-path-TC**. The languages are first introduced for selected $n \geq 1$, and after that, they are presented in a general case. The specific examples for higher values of n tends to be excessively long and they are left to the reader.

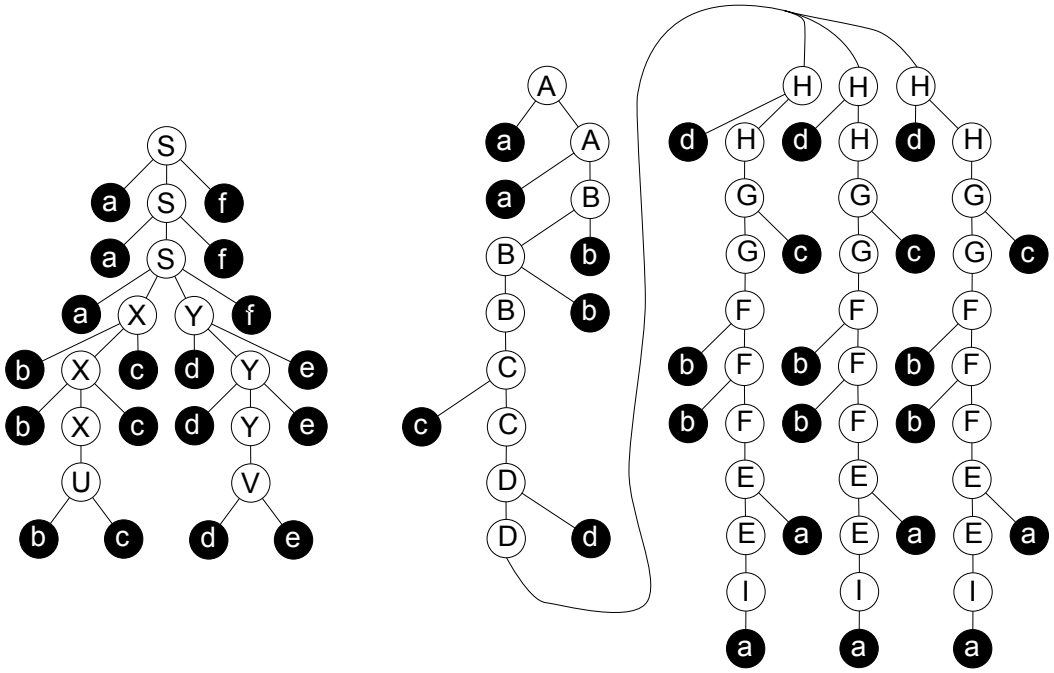


Figure 8.1: Illustration of the derivations of $a^3b^3c^3d^3e^3f^3$ with two paths of the form $S^iX^iUb \cup S^iY^iVd$, where $i \geq 1$, in Example 8.1 (left) and $(a^2cdb^2)^4$ with three paths of the form $A^rB^sC^tD^uH^uG^tF^sE^rIa$, where $r, s, t, u \geq 1$, in Example 8.3 (right).

Example 8.1
Consider the TC grammar that generates ${}_{n\text{-}path}L(G,R)$ and $n = 2$, where

$$
\begin{aligned}
&G = (\{S, X, Y, U, V\}, \{a, b, c, d, e, f\}, P, S),\\
&P = \{S \to aSf, \quad S \to aXYf, \quad X \to bXc, \quad Y \to dYe,\\
&\qquad X \to U, \quad U \to bc, \quad Y \to V, \quad V \to de\},\\
&R = \{S^iX^iUb \cup S^iY^iVd |\ i \geq 1\},\\
&{}_{2\text{-}path}L(G,R) = \{a^jb^jc^jd^je^jf^j |\ j \geq 1\}.
\end{aligned}
$$

Clearly, ${}_{2\text{-}path}L(G,R) \notin \mathbf{CF}$. The left-hand part of Figure 8.1 illustrates the derivation tree for the derivation $S \Rightarrow^* a^3b^3c^3c^3d^3e^3f^3$ in (G,R). Clearly, there are two paths described by the strings S^3X^3Ub and S^3Y^3Vd from R.

Example 8.2
Let (G,R) be a TC grammar that generates ${}_{n\text{-}path}L(G,R)$, $n \geq 1$, where

$$
\begin{aligned}
&G = (\{S\} \cup \{A_i, B_i |\ 1 \leq i \leq n\},\\
&\qquad \{a_i |\ 1 \leq i \leq 2n+2\}, P, S),\\
&P = \{S \to a_1Sa_{2n+2}, \qquad S \to a_1A_1A_2 \ldots A_na_{2n+2}\} \cup\\
&\quad \{A_{i+1} \to a_{2i+2}A_{i+1}a_{2i+3}, \quad A_{i+1} \to B_{i+1},\\
&\quad B_{i+1} \to a_{2i+2}a_{2i+3} |\ 0 \leq i \leq n-1\},\\
&R = \textstyle\bigcup_{i=1}^{n} \{S^kA_i^kB_ia_{2i} |\ k \geq 1\}.
\end{aligned}
$$

Clearly, $R \in \mathbf{LIN}$. Consider a derivation in (G,R):

$$S \Rightarrow^k a_1^kSa_{2n+2}^k$$

$\Rightarrow a_1^k a_1 A_1 A_2 \dots A_n a_{2n+2} a_{2n+2}^k$
$\Rightarrow^{n \cdot k} a_1^{k+1} a_2^k B_1 a_3^k \dots a_{2n}^k B_n a_{2n+1}^k a_{2n+2}^{k+1}$
$\Rightarrow^{n} a_1^{k+1} a_2^{k+1} a_3^{k+1} \dots a_{2n}^{k+1} a_{2n+1}^{k+1} a_{2n+2}^{k+1}$

Clearly, $n \geq 1$ paths are described by R and in this way, (G, R) generates ${}_{n\text{-}path}L(G, R) = \{a_1^k \dots a_{2n+2}^k |\ k \geq 1\} \notin \mathbf{CF}$.

Example 8.3
Consider the TC grammar (G, R) with ${}_{n\text{-}path}L(G, R)$, for $n = 3$, where

$$
\begin{aligned}
&G = (\{A, B, C, D, E, F, G, H, I\}, \{a, b, c, d\}, P, A),\\
&P = \{A \to aA, \quad A \to aB, \quad B \to Bb, \quad B \to C,\\
&\qquad C \to cC, \quad C \to D, \quad D \to Dd, \quad D \to HHH,\\
&\qquad E \to Ea, \quad E \to I, \quad F \to bF, \quad F \to E,\\
&\qquad G \to Gc, \quad G \to F, \quad H \to dH, \quad H \to G, \quad I \to a\},\\
&R = \{A^r B^s C^t D^u H^u G^t F^s E^r I a |\ r, s, t, u \geq 0\},\\
&{}_{3\text{-}path}L(G, R) = \{(a^r c^t d^u b^s)^4 |\ r > 0, s, t, u \geq 0\}.
\end{aligned}
$$

Clearly, ${}_{3\text{-}path}L(G, R) \notin \mathbf{CF}$. The right-hand part of Figure 8.1 illustrates the derivation tree for the derivation $S \Rightarrow^* (a^2 c d b^2)^4$ in (G, R). Clearly, there are three paths described by $A^2 B^3 C^2 D^2 H^2 G^2 F^3 E^2 I a$ from R.

Example 8.4
Let $m \geq 0$ with even m. Let (G, R) be a TC grammar that generates ${}_{n\text{-}path}L(G, R)$, $n \geq 1$, where

$$
\begin{aligned}
&G = (\{A_j, B_j |\ 1 \leq j \leq m\} \cup \{C\}, \{a_j |\ 1 \leq j \leq m\}, P, A_1),\\
&P = \{A_1 \to a_1 A_1, \quad A_1 \to a_1 A_2, \quad B_1 \to B_1 a_1, \quad B_1 \to C, \quad C \to a_1\} \cup\\
&\quad \{A_m \to A_m a_m, A_m \to \{B_m\}^n\} \cup\\
&\quad \{A_i \to A_i a_i, \quad A_i \to A_{i+1} |\ 2 \leq i \leq m-1 \text{ with even } i\} \cup\\
&\quad \{A_i \to a_i A_i, \quad A_i \to A_{i+1} |\ 3 \leq i \leq m-1 \text{ with odd } i\} \cup\\
&\quad \{B_i \to a_i B_i, \quad B_i \to B_{i-1} |\ 2 \leq i \leq m \text{ with even } i\} \cup\\
&\quad \{B_i \to B_i a_i, \quad B_i \to B_{i-1} |\ 3 \leq i \leq m \text{ with odd } i\},\\
&R = \{A_1^{k_1} A_2^{k_2} \dots A_m^{k_m} B_m^{k_m} B_{m-1}^{k_{m-1}} \dots B_2^{k_2} B_1^{k_1} C a_1 |\ k_i \geq 0, 1 \leq i \leq m\}.
\end{aligned}
$$

Clearly, $R \in \mathbf{LIN}$. Observe that $n \geq 1$ paths are described by R and

$$
\begin{aligned}
{}_{n\text{-}path}L(G, R) = \{&(a_1^{k_1+1} a_3^{k_3} \dots a_{m-1}^{k_{m-1}} a_m^m a_{m-2}^{k_{m-2}} a_{m-4}^{k_{m-4}} \dots a_2^{k_2})^{n+1}\\
&|\ k_i \geq 0, 1 \leq i \leq m\} \notin \mathbf{CF}.
\end{aligned}
$$

The details of a derivation in this general case is left to the reader.

8.3 Syntax analysis of n-path-TC

As it is demonstrated above, some typical non-context-free languages belong to **n-path-TC**. Thus it is natural and indisputably important for practical use to study the parsing methods possibilities for this language class. The first and most important requirement on parsing is polynomial parsability.

Theorem 8.5 For TC grammar (G, R) where G is unambiguous context-free grammar and R is linear control language, the membership $x \in {}_{n\text{-}path}L(G, R)$, $n \geq 1$, is decidable in $O(|x|^k)$, for some $k \geq 0$.

Proof of Theorem 8.5. For some $n \geq 1$, let ${}_{n\text{-}path}L(G, R)$ be the language of a TC grammar (G, R) in which G is unambiguous. We assume R is generated by some unambiguous linear grammar. Since G is unambiguous, it is well-known that we can decide whether $x \in L(G)$, or not, in $O(|x|^2)$—that is, we distinguish two cases: (1) $x \notin L(G)$ and (2) $x \in L(G)$.

- Clearly, if $x \notin L(G)$, then $x \notin {}_{n\text{-}path}L(G, R)$.
- If $x \in L(G)$, then, since G is unambiguous, we can construct unique derivation tree t of $x \in L(G)$ in $O(|x|^2)$. Since each path of t ends in a leaf, t contains $|x|$ paths. Clearly, the height of t is polynomially bounded by some $l \geq 2$ with respect to $|x|$. Thus, for any $x \in L(G)$, the length of each path p of t and therefore also $|word(p)|$ are bounded by l. Because R is unambiguous and $|word(p)| \leq l$ for each $p \in Q_t$, it is well-known that we can decide if $word(p) \in L(R)$ in polynomial time. If for at least n paths, $p_1, p_2, \ldots, p_n$, of derivation tree of x in G, $word(p_i) \in R$ holds for $i \in 1, 2, \ldots, n$, then $x \in {}_{n\text{-}path}L(G, R)$. Hence, the membership problem is decidable in polynomial time.

■

As it straightforwardly follows from above, the proposed way to solve membership problem leads to a parsing method working, in essence, in two phases:

1. construction of derivation tree t of x in G by top-down parsing method,
2. checking that at least $n \geq 1$ paths of t belong to R.

From the practical viewpoint, the situation may occur in which during the phase 1 above we already know that currently constructed derivation tree cannot contain the required number of paths described by the strings from R—informally, we do not have to wait with starting the phase 2 until the phase 1 is completely done (until t is completely constructed).

8.3.1 Top-Down Parsing of n-path-TC

For some $n \geq 1$, let ${}_{n\text{-}path}L(G, R)$ be the language of a TC grammar (G, R) where $G = (N, T, P, S)$ is an unambiguous context-free grammar and let $V = N \cup T$. We assume that R is generated by unambiguous context-free grammar $G_R = (N_R, V, P_R, S_R)$. Adjust the idea behind Theorem 8.5 as follows:

We construct labelled derivation tree with the set of labels $\Psi = \{0, 1\}$ and the following semantics. Let p be a path of derivation tree t in G and e be an edge between any two consecutive nodes of p. Then, label $0 \in \Psi$ of any e of p represents p is not described by R—that is, $word(p) \notin R$. Label $1 \in \Psi$ of all e of p represents p can potentially be described by R.

Consider that for the decision if $x \in L(G)$, we use well-known top-down parsing method to construct derivation tree t of x in G—that is, started from S, we construct derivation tree t according to the rules of G such that the frontier of t is equal to x (for more details, see [80]).

Let us suppose that a rule $r = A \to A_1 A_2 \dots A_j \in P$, $j \geq 1$, is used in the derivation step $X \Rightarrow Y$, $X, Y \in (N \cup T)^*$ in G. In addition, we need to determine the value of the labels of the edges between A and each A_j, for $j = 1, 2, \dots, n$, related to the application of r. Let t' be a derivation tree that corresponds with the derivation $S \Rightarrow^* w_1 A_1 A_2 \dots A_j w_2$ in G, for some $w_1, w_2 \in (N \cup T)^*$. Essentially, t' is a sub-tree of t. Clearly, each path of t' is the beginning part of at least one path in t. Next, we distinguish the following cases:

- If all the edges of t' are labelled, we can proceed to next derivation step in G.
- If some of the edges in t' are not labelled, we need to compute the values of missing labels.

For each unlabelled edge e of path p' in t', we check whether G_R can generate the string of the form $word(p')w$ with $w \in N_R^* T \cup \{\varepsilon\}$. Since $|word(p')|$ is finite, we can check it in polynomial time. If so, we add label $1 \in \Psi$ to edge e; otherwise, we add label $0 \in \Psi$ to edge e. Note that this phase can be optimized in such a way that we do the test whether G_R can generate $word(p')w$ with $w \in N_R^* T \cup \{\varepsilon\}$ symbol-by-symbol during the generation of $word(p')$ in G_R. The details of this optimization represents a straightforward task which is left to the reader. Next, we distinguish the following cases:

- If t' contains no leaf with input edge labelled by 1, then $x \notin {}_{n\text{-}path}L(G, R)$.
- If t' contains at least one leaf labelled by symbol of N, we proceed to next derivation step in G.
- If all the leafs of t' are labelled by the symbols of T and for at least n of the leafs of t, there is an edge labelled by 1, then $x \in {}_{n\text{-}path}L(G, R)$.

Thus, it is possible to check whether the paths of derivation tree t of LL context-free grammar can potentially be described by given unambiguous context-free language already during the building of t by LL parser. Example 8.6 explains the syntax analysis in detail.

Example 8.6
Consider ${}_{n\text{-}path}L(G, R)$ where $G = (\{S,\ A,\ B\},\ \{a, b, c, d, e, k\},\ P,\ S)$ with $P = \{1 = S \to AA,\ 2 = A \to aAd,\ 3 = A \to bBc,\ 4 = A \to e,\ 5 = B \to bBc,\ 6 = B \to k\}$ and $R = \{SA^m B^{m-1} k |\ m \geq 1\}$. Obviously, $L(G) = \{a^i(b^j k c^j + e) d^i a^s (b^t k c^t + e) d^s |\ i, j, t, s \geq 0\}$. Clearly, ${}_{1\text{-}path}L(G, R) = L(G)$ with $i = j$ or $s = t$, and ${}_{2\text{-}path}L(G, R) = L(G)$ with $i = j$ and $s = t$. The LL table for G is constructed by well-known algorithm (see [103]) and it is presented in Table 8.1.

The syntax analyser uses two pushdown automata and a list of 3-tuples containing state of the second automaton, contents of its pusdown, and a pointer to the first-automaton's pushdown. The first pushdown automaton simulates the construction of derivation tree by the LL table in the well-known way.

- If the top-most symbol on the pushdown is a nonterminal A, the first input symbol is a, and there is a rule $A \to x$ on position $[A, a]$ in the LL table, then the automaton rewrites A on the pushdown by $(x)^R$ (expansion step).
- If a on the pushdown's top is a terminal and a is the first input symbol, the automaton reads a from the input and removes a from the pushdown (comparation step).
- Other cases represent a syntax error.

	a	b	c	d	e	k	$
S	1	1			1		
A	2	3			4		
B		5				6	

Table 8.1: LL table for grammar G from Example 8.6.

1. PDA	2.PDA	Pointer	Tuples
$Sqabkcdaed$	q_0	$(q_0, \varepsilon, 1)$	$(q_0, \varepsilon, 1)$
$AAqabkcdaed$	q_0	$(q_0, \varepsilon, 1)$	$(q_1, \varepsilon, 1)$, $(q_1, \varepsilon, 2)$
$AdAaqabkcdaed$	q_1	$(q_1, \varepsilon, 2)$	$(q_1, \varepsilon, 1)$, $(q_1, A, 2)$, $(q_1, A, 3)$, $(q_1, A, 4)$
$AdAqbkcdaed$	Aq_1	$(q_1, A, 4)$	$(q_1, \varepsilon, 1)$, $(q_1, A, 2)$, $(q_1, A, 3)$
$AdcBbqbkcdaed$	Aq_1	$(q_1, A, 3)$	$(q_1, \varepsilon, 1)$, $(q_1, A, 2)$, $(q_1, AA, 3)$, $(q_1, AA, 4)$, $(q_1, AA, 5)$
$AdcBqkcdaed$	AAq_1	$(q_1, AA, 5)$	$(q_1, \varepsilon, 1)$, $(q_1, A, 2)$, $(q_1, AA, 3)$, $(q_1, AA, 4)$
$Adckqkcdaed$	AAq_1	$(q_1, AA, 4)$	$(q_1, \varepsilon, 1)$, $(q_1, A, 2)$, $(q_1, AA, 3)$, $(q_1, A, 4)$
$Adcqcdaed$	Aq_1	$(q_1, A, 4)$	$(q_1, \varepsilon, 1)$, $(q_1, A, 2)$, $(q_1, AA, 3)$, $(q_1, A, 0)$
$Adqdaed$	AAq_1	$(q_1, AA, 3)$	$(q_1, \varepsilon, 1)$, $(q_1, A, 2)$, $(q_1, A, 0)$
$Aqaed$	Aq_1	$(q_1, A, 2)$	$(q_1, \varepsilon, 1)$, $(q_1, A, 0)$
$dAaqaed$	q_1	$(q_1, \varepsilon, 1)$	$(q_1, A, 0)$, $(q_1, A, 1)$, $(q_1, A, 2)$, $(q_1, A, 3)$
$dAqed$	Aq_1	$(q_1, A, 3)$	$(q_1, A, 0)$, $(q_1, A, 1)$, $(q_1, A, 2)$
$deqed$	Aq_1	$(q_1, A, 2)$	$(q_1, A, 0)$, $(q_1, A, 1)$, $(q_1, AA, 2)$
dqd	AAq_1	$(q_1, AA, 2)$	$(q_1, A, 0)$, $(q_1, A, 1)$
q	Aq_1	$(q_1, A, 1)$	$(q_1, A, 0)$

Table 8.2: Parsing of $abkcdaed$ corresponding to TC grammar (G, R) from Example 8.6.

Let us return to the example and consider input string $abkcdaed$. In the beginning, the top-most symbol on the pushdown of the first automaton is S. Since a is the first input symbol and there is rule 1 on the position $[S, a]$ in the table, the automaton rewrites S by AA on its pushdown.

During this computation, the second automaton is parsing the potentially valid paths in the derivation tree. At the beginning, the list contains one item $(q_0, \widehat{S}, 1)$ where 1 represents the first position on the first-automaton's pushdown from the bottom and $\widehat{S}$ is the start pushdown symbol of the second automaton. If the first automaton makes a computation step with symbol a on the pushdown's top and there is a tuple (q, α, p) in the list where p is the pointer to the symbol, the second automaton places α onto its pushdown, automaton moves to q and it makes steps for a as the first input symbol until it does not need next input symbol. For example, after the expansion from S to AA, the second automaton finds list-item $(q_0, \widehat{S}, 1)$ and it moves to state q_0 and places $\widehat{S}$ onto the pushdown.

Then, it simulates step $\widehat{S}q_0S \vdash \beta q$ where $\widehat{S}q_0S \to \beta q$ is a transition rule of the second automaton. If $a = A$ was a nonterminal and first automaton made expansion by rule $A \to x$, then the syntax analyser removes the used list-item and if the second automaton did not

reject input, then there are l tuples (q, β, i) inserted into the list, for all $i = t+1, \ldots, t+l$ with $l = |x|$, topmost-symbol-position t before the expansion, and β as the current content of the pushdown. Otherwise, for a as a terminal symbol, the comparison is done. If the second automaton accepts, the pointer of the used tuple is rewritten to 0 where 0 denotes accepted path. Assuming that it does not accept with a terminal a, the tuple is removed from the list.

Consider input string *abkcdaed* again. For some definition of the second automaton, the syntax analysis proceeds as it is given in Table 8.2.

As one can see, only one item with 0 in the last component remains in the list—that is, one path belongs to the control language. If we require that $n \geq 2$ paths are described by the control language, the input string is not accepted.

Notice that the set of tuples can be represented by pushdown automata storing tuples on its pushdown, where the required tuple is always on the top. On the other hand, the needless tuples (tuples with 0 in the third component) can be remembered in states and may not occur on the pushdown. In this way, the core of the investigated parsing can stand on a 3-accepting move-restricted pushdown automata system.

8.3.2 Bottom-Up Parsing of n-path-TC

Section 8.3.1 deals in principle with top-down parsing method (LL parser) and its weakness is the assumption that a context-free grammar is unambiguous. Moreover, the method demonstrated in Example 8.6 assumes that the grammar is LL. Essentially, the same idea is applicable also on bottom-up parsing methods which can handle a larger range of the languages. Therefore, we briefly discuss the ideas of parsing methods for **n-path-TC** in terms of LR parsing.

One of the advantages of LR parser is that we do not need to require that in TC grammar (G, R), G is LL grammar. On the other hand, we have to deal with the ambiguity. However, it is well-known that the question whether a context-free grammar is or is not ambiguous is undecidable, since this problem can be reduced to the Post Correspondence Problem which is undecidable (see [96]).

It is also well-known that for some ambiguous context-free grammars, there exists equivalent context-free grammar which is unambiguous. The ambiguity of a context-free grammar can be restricted basically by removing the chain rules (i.e. rules of the form $A \to B$, $A, B \in N$). We assume that a context-free grammar contains only usable rules—that is, only those rules, which can be used during the derivation. Clearly, if $G = (N, T, P, S)$ is a context-free grammar with $r = A \to A \in P$, for some $A \in N$, then G is ambiguous since r can be used during the derivation of $x \in L(G)$ arbitrarily many times and thus generate arbitrarily many different derivation trees for x in G.

Obviously, since the chain rules generate nothing, they can be removed from a context-free grammar G without affecting $L(G)$. However, removing the chain rules from G in a TC grammar (G, R) affects the paths in the derivation trees of $x \in L(G)$. Thus the identity ${}_{n\text{-}path}L(G, R) = {}_{n\text{-}path}L(G', R)$, where G' is obtained by removing the chain rules from G, does not hold; however, the equivalence $L(G) = L(G')$ holds.

Theorem 8.7 For a TC grammar (G, R), where G is a context-free grammar and $R \in \mathbf{LIN}$, there is TC grammar (G', R') such that G' does not contain chain rules and ${}_{n\text{-}path}L(G, R) = {}_{n\text{-}path}L(G', R')$, $n \geq 1$.

Proof of Theorem 8.7. For some $n \geq 1$, let ${}_{n\text{-}path}L(G,R)$ be the language of a TC grammar (G,R) where $G = (N,T,P,S)$. Let G' be a context-free grammar obtained from G by removing the chain rules. Therefore, G' can be constructed by well-known algorithm in polynomial time (see 5.1.3.3 in [80]). We get $G' = (N,T,P',S)$ such that for all $x \in L(G')$, there is no derivation in G' of the form $B \Rightarrow^* A$, for some $A, B \in N$.

The paths in the derivation trees of G' are described by the strings of the form N^*T. Basically, we need to read such a strings and remove such symbols $A \in N$ which corresponds to the application of $B \to A \in P$ in G. This is done by gsm mapping M (see [37] for the definition) such that M reads the strings s of the form N^*T and nondeterministically removes or lets unchanged each symbol $A \in N$ with $B \to A \in P$ and BA is substring of s. Since **LIN** is closed under gsm mappings (see [34]), also $M(R) \in$ **LIN**. In this way, we get $M(R)$ such that $M(R) \neq R$. However, ${}_{n\text{-}path}L(G,R) = {}_{n\text{-}path}L(G',M(R))$. ■

Consider TC grammar (G,R) and let (G',R') be constructed as described above. As one can see, G' does not need to be unambiguous since the chain rules are not the only cause of the ambiguity. Consider, however, any $x \in {}_{n\text{-}path}L(G',R')$. Obviously, there is a derivation tree t of x in G'. Since there are no chain rules in G' and for each $x \in L(G')$, $|x|$ is finite, the height of t with respect to $|x|$ is bounded by $log\,|x|/log\,2$. Thus there is at most m, for some $m \geq 1$, derivation trees of x in G' and G is m-ambiguous.

Theorem 8.8 For TC grammar (G,R) where G is m-ambiguous LR grammar, $m \geq 1$, and an unambiguous language $R \in$ **LIN** , the membership $x \in {}_{n\text{-}path}L(G,R)$, $n \geq 1$, is decidable in $O(|x|^k)$, for some $k \geq 0$.

Proof of Theorem 8.8. For some $n \geq 1$, let ${}_{n\text{-}path}L(G,R)$ be the language of a TC grammar (G,R) in which G is m-ambiguous, for some $m \geq 1$. Thus, if $x \in L(G)$, then we can construct at most m derivation trees of $x \in L(G)$ in $O(m.|x|^2)$ by LR parser. Then, if for at least j paths, $p_1, p_2, \ldots, p_j$, of at least one derivation tree of x in G, it holds that $word(p_i) \in R$ for $i \in 1, 2, \ldots, j$, then $x \in {}_{n\text{-}path}L(G,R)$. ■

Chapter 9

Conclusion

This thesis discusses and studies formal languages and systems of formal models. Its main results are published or submitted in [19, 84, 11, 12, 13, 14, 20, 15, 17, 16, 18] and presented at the Hungarian Academy of Sciences and foreign universities in Paris, Debrecen, and Nyíregyháza. The following section summaries this results.

9.1 Summary and Further Investigation

This thesis was focused on a study of systems of formal models which plays important role in the modern information technology and computer science. Since the introduction of CD grammar systems, many other systems were studied and systems of formal models have become a vivid research area. Aim of the present thesis was to further investigate properties of the systems of formal models to their better understanding. This research can be divided into several main parts.

In the first part, we continued in studying of regularly controlled CD grammar systems, where we used phrase-structure grammars as components, and introduced three new restrictions on derivations in these systems. The first restriction requires that derivation rules could be applied within the first l nonterminals, for given $l \geq 1$. Although phrase-structured grammars define all languages from **RE**, regularly controlled CD grammar systems with phrase-structure grammars as components under such restriction generate only context-free languages. One may ask, how strong the control language must be to leave the generative power unchanged. Our assumption is that linear languages are sufficient, but a rigorous proof has not yet been done. The second restriction allows to have only limited number of undivided blocks of nonterminals in each sentential form during any successful derivation. It has been proven that this restriction has no effect on the generative power of these CD grammar systems even in the case when the restriction allows only one such block. On the other hand, the restriction limiting the maximum length and number of the blocks decreases the generative power of these systems to the classes $\mathscr{L}_m(P)$ representing infinite hierarchy, with respect of m, lying between the classes of linear and context-sensitive languages. Notice that m is maximal number of blocks and $\mathbf{CF} - \mathscr{L}_m(P) \neq \emptyset$. Question whether the stronger control language effects the generative power of CD grammar systems with phrase-structure grammars subject to the third restriction is still open.

The second part deals with parallel grammar and automata systems based upon CFGs and PDAs, respectively. More specifically, we introduced two variants of n-accepting restricted pushdown automata systems, accepting n-tuples of interdependent strings, as coun-

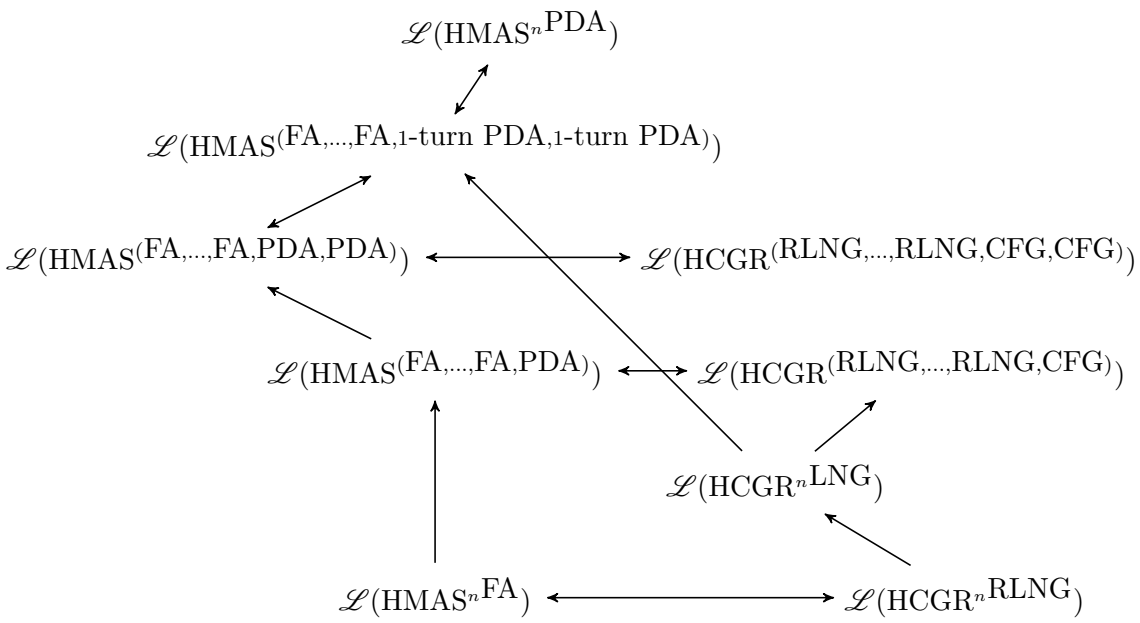


Figure 9.1: Hierarchy of n-languages for $n \geq 2$

terparts of canonical n-generating nonterminal/rule synchronized grammar systems based upon context-free grammars. Both types of the automata systems consist of n PDAs, for $n \geq 2$, and one restriction-set. In the case of n-accepting state-restricted automata systems, the restriction-set allows to suspend and resume some automata during computation in relation to combination of current states of the PDAs. In the case of n-accepting move-restricted automata systems, the restriction-set determines which combination of transition rules used in the common computation step are permitted. We have proven that these n-accepting restricted automata systems are able to accept such n-languages that the canonical n-generating grammar systems can generate and vice versa. Furthermore, we have established fundamental hierarchy of n-languages generating/accepting by these canonical multi-generating rule synchronized grammar/n-accepting rule-restricted automata systems with different types of components. First of all, we have shown that both these systems are equivalent even if we combine RLNGs with CFGs in the grammar systems and FAs with PDAs in the automata systems. After that, we have established the hierarchy given by Figure 9.1 ($\rightarrow$ and $\leftrightarrow$ mean $\subset$ and $=$, respectively), where it can be seen, inter alia, that canonical n-generating rule synchronized grammar systems based upon linear grammars are significantly weaker than n-accepting move-restricted automata systems, with two 1-turn PDAs and $n-2$ FAs as components.

The second part of this research can be continued by better approximation of power of the state/move-restricted automata systems based upon FAs (especially in relation to string-interdependences), or by investigation of restarting and/or stateless finite and pushdown automata as the components of discussed automata systems.

In the last part, we have suggested rule-restricted systems for processing of linguistically motivated languages. In this part, we introduced three variants of rule-restricted translating systems based upon finite automaton and context-free grammar. At first, we have proven that leftmost restriction placed on derivation in the context-free grammar effects both the

generative and accepting power of such systems. In addition, we introduced a rule-restricted transducer system with appearance checking, where the restriction-set Ψ is a set of 3-tuples containing one rule of the FA and two rules of the CFG. For the common computation step, the system has to use the first and second rules of a 3-tuple, if it is possible; otherwise, it can use the first and third rules from the 3-tuple. This system is able to recognize and generate any language from **RE**. Thereafter, some examples of natural language translating are given.

The investigation of processing of linguistically motivated languages continued by generalization of TC grammars that generate the language under path-based control introduced in [71]. We have considered TC grammars that generate their languages under n-path control by linear language which were introduced in [67].

We have demonstrated that for $L \in$ **n-path-TC** under assumption that L is generated by TC grammar (G, R) in which G and R are unambiguous and, furthermore, G is restricted to be LL grammar, there is parsing method working in polynomial time. This method can check whether or not the paths of the derivation tree t of $x \in L(G)$ belongs to control language R in the time of building of t. Moreover, when we consider LR parser for $L \in$ **n-path-TC** under assumption that L is generated by TC grammar (G, R) in which G has bounded ambiguity (i.e. G is unambiguous or m-ambiguous) and unambiguous language $R \in$ **LIN**, there is also a parsing method working in polynomial time.

However, the open question is whether there is polynomial time parsing method

- if G is not LL,
- if G is ambiguous.

It is also of interest to quantify the worst case of the parsing complexity more precisely.

The open investigation area is represented by the transformation of n-path TC grammars into some normal forms based on Chomsky normal form of underlying context-free grammar which would lead to possibility to use parsing methods based on transformation to Chomsky normal form.

Bibliography

[1] S. Abraham. Some questions of language theory. In *Proceedings of the 1965 conference on Computational linguistics*, COLING '65, pages 1–11, Stroudsburg, PA, USA, 1965. Association for Computational Linguistics.

[2] A.V. Aho. *Compilers: principles, techniques, & tools*. Pearson/Addison Wesley, 2007.

[3] M. Amin and R. Abd el Mouaty. Stratified grammar systems with simple and dynamically organized strata. *Computers and Artificial Intelligence*, 23(4):355–362, 2004.

[4] B. S. Baker. Context-sesitive grammars generating context-free languages. In M. Nivat, editor, *Automata, Languages and Programming*, pages 501–506. North-Holland, Amsterdam, 1972.

[5] M. Bál, O. Carton, C. Prieur, and J. Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292(1):45–63, 2003.

[6] M. H. Beek and H. C. M. Kleijn. Petri net control for grammar systems. In *Formal and Natural Computing - Essays Dedicated to Grzegorz Rozenberg [on occasion of his 60th birthday, March 14, 2002]*, pages 220–243, London, UK, UK, 2002. Springer-Verlag.

[7] O. Bojar and M Čmejrek. Mathematical model of tree transformations. In *Project Euromatrix Deliverable 3.2*. Charles University, Prague, 2007.

[8] R. V. Book. Terminal context in context-sensitive grammars. *SIAM Journal of Computing*, 1:20–30, 1972.

[9] P. F. Brown, J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, John D. Lafferty, R. L. Mercer, and P. S. Roossin. A statistical approach to machine translation. *Comput. Linguist.*, 16(2):79–85, June 1990.

[10] A. O. Buda. Multiprocessor automata. *Inf. Process. Lett.*, 25(4):257–261, June 1987.

[11] M. Čermák. Systems of formal models and their application. In *Proceedings of the 14th Conference Student EEICT 2008*, Volume 2, pages 164–166. Faculty of Electrical Engineering and Communication BUT, 2008.

[12] M. Čermák. Power decreasing derivation restriction in grammar systems. In *Proceedings of the 15th Conference and Competition STUDENT EEICT 2009 Volume 4*, pages 385–389. Faculty of Information Technology BUT, 2009.

[13] M. Čermák. Multilanguages and multiaccepting automata system. In *Proceedings of the 16th Conference and Competition STUDENT EEICT 2010 Volume 5*, pages 146–150. Faculty of Information Technology BUT, 2010.

[14] M. Čermák. Basic properties of n-languages. In *Proceedings of the 17th Conference and Competition STUDENT EEICT 2011 Volume 3*, pages 460–464. Faculty of Information Technology BUT, 2011.

[15] M. Čermák. Restrictions on derivations in n-generating grammar systems. In *Proceedings of the 18th Conference and Competition STUDENT EEICT 2012 Volume 5*, pages 371–375. Faculty of Information Technology BUT, 2012.

[16] M. Čermák, P. Horáček, and A. Meduna. Rule-restricted automaton-grammar transducers: Power and linguistic applications. *Mathematics for Applications*, 2012, in press.

[17] M. Čermák, J. Koutný, and A. Meduna. Parsing based on n-path tree-controlled grammars. *Theoretical and Applied Informatics*, 2011(23):213–228, 2012.

[18] M. Čermák, J. Koutný, and A. Meduna. On n-language classes hierarchy. *Acta Cybernetica*, 2012, sumbited.

[19] M. Čermák and A. Meduna. n-Accepting restricted pushdown automata systems. In *13th International Conference on Automata and Formal Languages*, pages 168–183. Computer and Automation Research Institute, Hungarian Academy of Sciences, 2011.

[20] M. Čermák and A. Meduna. n-Accepting restricted pushdown automata systems. In *7th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 110–110. Brno University of Technology, 2011.

[21] D. Chiang. An introduction to synchronous grammars. In *44th Annual Meeting of the Association for Computational Linguistics*, 2006.

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

[23] E. Csuhaj-Varjú and J. Dassow. On cooperating/distributed grammar systems. *Elektronische Informationsverarbeitung und Kybernetik*, 26(1/2):49–63, 1990.

[24] E. Csuhaj-Varjú, J. Dassow, and G. Păun. Dynamically controlled cooperating/distributed grammar systems. *Inf. Sci.*, 69(1-2):1–25, April 1993.

[25] E. Csuhaj-Varjú, J. Kelemen, and G. Păun. Grammar systems with wave-like communication. *Computers and Artificial Intelligence*, 15(5), 1996.

[26] E. Csuhaj-Varju, J. Kelemen, G. Păun, and J. Dassow, editors. *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach Science Publishers, Inc., Newark, NJ, USA, 1st edition, 1994.

[27] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana, and G. Vaszil. Parallel communicating pushdown automata systems. *Int. J. Found. Comput. Sci.*, 11(4):633–650, 2000.

[28] E. Csuhaj-Varjú, T. Masopust, and G. Vaszil. Cooperating distributed grammar systems with permitting grammars as components. *Romanian Journal of Information Science and Technology (ROMJIST)*, 12(2):175–189, 2009.

[29] E. Csuhaj-Varjú, V. Mitrana, and G. Vaszil. Distributed pushdown automata systems: computational power. In *Proceedings of the 7th international conference on Developments in language theory*, DLT'03, pages 218–229, Berlin, Heidelberg, 2003. Springer-Verlag.

[30] E. Csuhaj-Varjú and G. Vaszil. On context-free parallel communicating grammar systems: synchronization, communication, and normal forms. *Theor. Comput. Sci.*, 255(1-2):511–538, 2001.

[31] K. Čulik and H. A. Maurer. Tree controlled grammars. *Computing*, 19:129–139, 1977.

[32] E. Czeizler and E. Czeizler. On the power of parallel communicating watson-crick automata systems. *Theoretical Computer Science*, 358(1):142–147, 2006.

[33] J. Dassow. Cooperating/distributed grammar systems with hypothesis languages. *J. Exp. Theor. Artif. Intell.*, 3(1):11–16, 1991.

[34] J. Dassow, Gh. Păun, and A. Salomaa. Grammars with controlled derivations. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume II*, pages 101–154. Berlin: Springer, 1997.

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

[36] J. Dassow and V. Mitrana. Stack cooperation in multistack pushdown automata. *Journal of Computer and System Sciences*, 58(3):611–621, 1999.

[37] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory.* Springer, Berlin, 1989.

[38] J. Dassow and G. Păun. Cooperating/distributed grammar systems with registers: the regular case. *Found. Control Eng.*, 15:19–38, 1990.

[39] J. Dassow and B. Truthe. Subregularly tree controlled grammars and languages. In *Automata and Fromal Languages - 12th International Conference AFL 2008, Balatonfured*, pages 158–169. Computer and Automation Research Institute of the Hungarian Academy of Sciences, 2008.

[40] S. Dumitrescu. Characterization of RE using CD grammar systems with two registers and RL rules. In *New Trends in Formal Languages*, volume 1218 of *Lecture Notes in Computer Science*, pages 167–177. Springer Berlin / Heidelberg, 1997.

[41] H. Fernau. Regulated grammars under leftmost derivation. *Grammars*, 3(1):37–62, 2000.

[42] H. Fernau, M. Holzer, and R. Freund. External versus internal hybridization for cooperating distributed grammar systems, 1996.

[43] H. Fernau and R. Stiebe. On the expressive power of valences in cooperating distributed grammar systems. In *Computation, Cooperation, and Life*, volume 6610 of *Lecture Notes in Computer Science*, pages 90–106. Springer Berlin / Heidelberg, 2011.

[44] M. Frazier and C. D. Page. Prefix grammars: An alternative characterization of the regular languages. *Information Processing Letters*, 51(2):67–71, 1994.

[45] M. Frazier and C.D. Page. Prefix grammars: an alternative characterization of the regular languages. *Inf. Process. Lett.*, 51(2):67–71, 1994.

[46] R. Freund and G. Păun. A variant of team cooperation in grammar systems. *j-jucs*, 1(2):105–130, 1995.

[47] V. Geffert. Context-free-like forms for the phrase-structure grammars. In *Proceedings of the Mathematical Foundations of Computer Science 1988*, pages 309–317, New York, 1988. Springer.

[48] V. Geffert. Normal forms for phrase-structure grammars. *Informatique théorique et applications*, 25 no 5:473–496, 1991.

[49] S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages.* Elsevier Science Inc., New York, NY, USA, 1975.

[50] S. Ginsburg and S. Greibach. Mappings which preserve context-sensitive languages. *Information and Control*, 9:563–582, 1966.

[51] F. Goldefus, T. Masopust, and A. Meduna. Left-forbidding cooperating distributed grammar systems. *Theor. Comput. Sci.*, 411(40-42):3661–3667, September 2010.

[52] S. A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theor. Comput. Sci.*, 7:311–324, 1978.

[53] E. M. Gurari and O. H. Ibarra. A note on finite-valued and finitely ambiguous transducers. *Theory of Computing Systems*, 16:61–66, 1983. 10.1007/BF01744569.

[54] J. Hajič. *Disambiguation of Rich Inflection: Computational Morphology of Czech.* Wisconsin Center for Pushkin Studies. Karolinum, 2004.

[55] E. Hajičová, P. Sgall, and D. Zeman. Issues of projectivity in the prague dependency treebank. In *Prague Bulletin of Mathematical Linguistics*, 2004.

[56] M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978.

[57] T. N. Hibbard. Context-limited grammars. *Journal of the ACM*, 21:446–453, 1974.

[58] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 2000.

[59] J. Hromkovic. *Theoretical Computer Science : Introduction to Automata, Computability, Complexity, Algorithmics, Randomization, Communication, and Cryptography.* Springer, 2003.

[60] L. Ilie. Collapsing hierarchies in parallel communicating grammar systems with communication by command, 1996.

[61] O. Jirák and Z. Křivka. Design and implementation of back-end for picoblaze C compiler. In *Proceedings of the IADIS International Conference Applied Computing 2009*, pages 135–138. International Association for Development of the Information Society, 2009.

[62] M. Johnson. PCFG models of linguistic tree representations. *Comput. Linguist.*, 24(4):613–632, December 1998.

[63] L. Kari, A. Mateescu, G. Păun, and A. Salomaa. Teams in cooperating grammar systems. *Journal of Experimental & Theoretical Artificial Intelligence*, 7(4):347–359, 1995.

[64] M. Khalilov and J. A. R. Fonollosa. N-gram-based statistical machine translation versus syntax augmented machine translation: comparison and system combination. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pages 424–432, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

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

[66] J. Koutný. Syntax analysis of tree-controlled languages. In *Proceedings of the 17th Conference and Competition STUDENT EEICT 2011 Volume 3*, page 5. Faculty of Information Technology BUT, 2011.

[67] J. Koutný, Z. Křivka, and A. Meduna. Pumping properties of path-restricted tree-controlled languages. In *7th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 61–69. Brno University of Technology, 2011.

[68] J. Koutný and A. Meduna. Tree-controlled grammars with restrictions placed upon cuts and paths. *Kybernetika*, 48(1):165–175, 2012.

[69] P. Linz. *An Introduction to Formal Language and Automata.* Jones and Bartlett Publishers, Inc., USA, 2006.

[70] R. Lukáš and A. Meduna. Multigenerative grammar systems and matrix grammars. *Kybernetika*, 46(1):68–82, 2010.

[71] S. Marcus, C. Martín-Vide, V. Mitrana, and Gh. Păun. A new-old class of linguistically motivated regulated grammars. In *Walter Daelemans, Khalil Sima'an, Jorn Veenstra, Jakub Zavrel (Eds.): Computational Linguistics in the Netherlands 2000, Selected Papers from the Eleventh CLIN Meeting, Tilburg*, pages 111–125. Language and Computers - Studies in Practical Linguistics 37 Rodopi 2000, 2000.

[72] C. Martín-Vide, A. Mateescu, and V. Mitrana. Parallel finite automata systems communicating by states. *Int. J. Found. Comput. Sci.*, 13(5):733–749, 2002.

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

[74] T. Masopust. On the terminating derivation mode in cooperating distributed grammar systems with forbidding components. *International Journal of Foundations of Computer Science*, 20(2):331–340, 2009.

[75] G. Matthews. A note on symmetry in phrase structure grammars. *Information and Control*, 7:360–365, 1964.

[76] G. Matthews. Two-way languages. *Information and Control*, 10:111–119, 1967.

[77] A. Meduna. Global context conditional grammars. *Journal of Automata, Languages and Combinatorics*, 1991(27):159–165, 1991.

[78] A. Meduna. Matrix grammars under leftmost and rightmost restrictions. In Gh. Păun, editor, *Mathematical Linguistics and Related Topics*, pages 243–257. Romanian Academy of Sciences, Bucharest, 1994.

[79] A. Meduna. On the number of nonterminals in matrix grammars with leftmost derivations. In *New Trends in Formal Languages: Control, Cooperation, and Combinatorics*, pages 27–39. Springer-Verlag, 1997.

[80] A. Meduna. *Automata and Languages: Theory and Applications*. Springer, 2000.

[81] A. Meduna and D. Kolář. One-turn regulated pushdown automata and their reduction. *Fundamenta Informaticae*, 2001(21):1001–1007, 2001.

[82] A. Meduna and R. Lukáš. Multigenerative grammar systems. *Schedae Informaticae*, 2006(15):175–188, 2006.

[83] A. Meduna and T. Masopust. Self-regulating finite automata. *Acta Cybernetica*, 18(1):135–153, 2007.

[84] A. Meduna, M. Čermák, and T. Masopust. Some power-decreasing derivation restrictions in grammar systems. *Schedae Informaticae*, 2010(19):23–34, 2011.

[85] A. Meduna and M. Švec. *Grammars with Context Conditions and Their Applications*. John Wiley & Sons, 2005.

[86] R. Meersman and G. Rozenberg. *Cooperating Grammar Systems*. 1978.

[87] H. Messerschmidt and F. Otto. Strictly deterministic CD-systems of restarting automata. In Erzsébet Csuhaj-Varjú and Zoltán Ésik, editors, *Fundamentals of Computation Theory*, volume 4639 of *Lecture Notes in Computer Science*, pages 424–434. Springer Berlin / Heidelberg, 2007.

[88] V. Mihalache, V. Mitrana, T. Centre, and Computer Science. Deterministic cooperating distributed grammar systems. *Computers and AI*, 2:20, 1996.

[89] T. Mine, R. Taniguchi, and M. Amamiya. Coordinated morphological and syntactic analysis of japanese language. In *Proceedings of the 12th international joint conference on Artificial intelligence - Volume 2*, pages 1012–1017. Morgan Kaufmann Publishers Inc., 1991.

[90] V. Mitrana. Hybrid cooperating/distributed grammar systems. *Computers and artificial intelligence*, 12(1):83–88, 1993.

[91] M. Mohri. Finite-state transducers in language and speech processing. *Comput. Linguist.*, 23(2):269–311, June 1997.

[92] S. Muchnick. *Advanced compiler design and implementation.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[93] B. Nagy. On $5' \to 3'$ sensing watson-crick finite automata. In Max Garzon and Hao Yan, editors, *DNA Computing*, volume 4848 of *Lecture Notes in Computer Science*, pages 256–262. Springer Berlin / Heidelberg, 2008.

[94] F. Otto. CD-systems of restarting automata governed by explicit enable and disable conditions. In *SOFSEM 2010: Theory and Practice of Computer Science*, volume 5901 of *Lecture Notes in Computer Science*, pages 627–638. Springer Berlin / Heidelberg, 2010.

[95] Gh. Păun. On the generative capacity of tree controlled grammars. *Computing*, 21(3):213–220, 1979.

[96] E. L. Post. A variant of a recursively unsolvable problem. *Bulletion of the American Mathematical Society*, 52:264–268, 1946.

[97] G. Păun. On the synchronization in parallel communicating grammar systems. *Acta Inf.*, 30(4):351–367, 1993.

[98] G. Păun. On the generative capacity of hybrid CD grammar systems. *Elektronische Informationsverarbeitung und Kybernetik*, pages 231–244, 1994.

[99] G. Păun and G. Rozenberg. Prescribed teams of grammars. *Acta Informatica*, 31:525–537, 1994.

[100] G. Păun and L. Sântean. Parallel communicating grammar systems– the regular case. *Ann. Univ. Buc. Ser. Mat.-Inform*, 2:55–63, 1989.

[101] E. Rich. *Automata, Computability and Complexity: Theory and Applications.* Pearson Prentice Hall, 2008.

[102] A. L. Rosenberg. On multi-head finite automata. *IBM J. Res. Dev.*, 10(5):388–394, September 1966.

[103] D. J. Rosenkrantz and R. E. Stearns. Properties of deterministic top down grammars. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1969.

[104] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 2. Springer-Verlag, Berlin, 1997.

[105] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 1. Springer-Verlag, Berlin, 1997.

[106] M. H. ter Beek, E. Csuhaj-Varjú, and V. Mitrana. Teams of pushdown automata. *Int. J. Comput. Math.*, 81(2):141–156, 2004.

[107] M. H. ter Beek and J Kleijn. Team automata satisfying compositionality. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *Lecture Notes in Computer Science*, pages 381–400. Springer Berlin / Heidelberg, 2003.

[108] S. Vicolov. Cooperating/distributed grammar systems with registers: the regular case. *Computers and artificial intelligence*, 12:89–98, 1993.

[109] P. Šaloun. Parallel LR parsing. In *Proceedings of the Fifth International Scientific Conference Electronic Computers and Informatics 2002*. The University of Technology Košice, 2002.

[110] D. Wajten. On cooperating–distributed extended limited 0l systems. *International Journal of Computer Mathematics*, 63:227–244, 1997.

[111] A. Weber. On the valuedness of finite transducers. *Acta Informatica*, 27:749–780, 1990. 10.1007/BF00264285.

[112] A. Zollmann and A. Venugopal. Syntax augmented machine translation via chart parsing. In *Proceedings of the Workshop on Statistical Machine Translation*, StatMT '06, pages 138–141, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.