BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

SYNCHRONOUS FORMAL SYSTEMS BASED ON GRAMMARS AND TRANSDUCERS

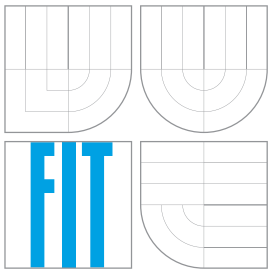
PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR
AUTOR PRÁCE

PETR HORÁČEK

BRNO 2014

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

SYNCHRONOUS FORMAL SYSTEMS BASED ON GRAMMARS AND TRANSDUCERS

SYNCHRONNÍ FORMÁLNÍ SYSTÉMY ZALOŽENÉ NA GRAMATIKÁCH A PŘEVODNÍCÍCH

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR
AUTOR PRÁCE

PETR HORÁČEK

SUPERVISOR
VEDOUCÍ PRÁCE

Prof. RNDr. ALEXANDER MEDUNA, CSc.

BRNO 2014

Abstract

This doctoral thesis studies synchronous formal systems based on grammars and transducers, investigating both theoretical properties and practical application perspectives. It introduces new concepts and definitions building upon the well-known principles of regulated rewriting and synchronization. An alternate approach to synchronization of context-free grammars is proposed, based on linked rules. This principle is extended to regulated grammars such as scattered context grammars and matrix grammars. Moreover, based on a similar principle, a new type of transducer called the rule-restricted transducer is introduced as a system consisting of a finite automaton and context-free grammar. New theoretical results regarding the generative and accepting power are presented. The last part of the thesis studies linguistically-oriented application perspectives, focusing on natural language translation. The main advantages of the new models are discussed and compared, using select case studies from Czech, English, and Japanese to illustrate.

Abstrakt

Tato disertační práce studuje synchronní formální systémy založené na gramatikách a převodnících a zkoumá jak jejich teoretické vlastnosti, tak i perspektivy praktických aplikací. Práce představuje nové koncepty a definice vycházející ze známých principů řízeného přepisování a synchronizace. Navrhuje alternativní způsob synchronizace bezkontextových gramatik, založený na propojení pravidel. Tento princip rozšiřuje také na řízené gramatiky, konkrétně gramatiky s rozptýleným kontextem a maticové gramatiky. Dále je představen na podobném principu založený nový druh převodníku, tzv. pravidlově omezený převodník. Jedná se o systém složený z konečného automatu a bezkontextové gramatiky. Práce prezentuje nové teoretické výsledky ohledně generativní a přijímající síly. Poslední část práce zkoumá možnosti lingvisticky orientovaných aplikací se zaměřením na překlad přirozeného jazyka. Diskutuje a srovnává hlavní výhody nových modelů s využitím vybraných případových studií z českého, anglického a japonského jazyka pro ilustraci.

Keywords

formal systems, grammars, transducers, regulated rewriting, synchronization, natural language syntax, natural language translation

Klíčová slova

formální systémy, gramatiky, převodníky, řízené přepisování, synchronizace, syntaxe přirozeného jazyka, překlad přirozeného jazyka

Citation

Petr Horáček: Synchronous Formal Systems Based on Grammars and Transducers, PhD thesis, Brno, FIT BUT, 2014

Citace

Petr Horáček: Synchronous Formal Systems Based on Grammars and Transducers, disertační práce, Brno, FIT VUT v Brně, 2014

Synchronous Formal Systems Based on Grammars and Transducers

Declaration

I hereby declare that this doctoral thesis is my own work that has been created under the supervision of prof. RNDr. Alexander Meduna, CSc. Some parts of this thesis are based on papers created in collaboration with colleagues. Specifically, Section 3.1, which provides an overview of formal models important to the area of natural language processing, is partially based on joint work with Ing. Eva Zámečníková and in Chapter 5, the new definitions and theorems for rule-restricted transducers are the result of joint work with Ing. Martin Čermák, PhD. All other results were established in cooperation with my supervisor only.

.....
Petr Horáček
March 26, 2014

Acknowledgments

I would like to thank Alexander Meduna for his pedagogic and scientific guidance, inspiration, and support throughout my doctoral study. I am also very grateful to my colleagues, particularly Zbyněk Křivka, Eva Zámečníková, Martin Čermák, and Jiří Koutný, as our discussions inspired and shaped many of the ideas presented in this work. Last but not least, I would like to thank my family.

This work was partially supported by the FR97/2011/G1, CZ.1.05/1.1.00/02.0070, FIT-S-10-2, and FIT-S-11-2 grant projects and by the MSM0021630528 research plan.

Contents

Part I

Introduction

Chapter 1

Motivation and Organization

Formal language theory is an essential part of theoretical computer science. It defines and studies languages as sets of strings (words, sentences), which are finite sequences of symbols. This definition covers natural languages (e.g. Czech, English, or Japanese) as well as artificial languages (such as programming languages).

To describe languages mathematically, formal language theory studies models which define them. Many of these models are based on rewriting systems—that is, formal systems which gradually change strings by rewriting some of their symbols in each step, according to a given set of rules. Most rewriting systems fall into one of the two basic categories: generative language models (generally known as grammars), and accepting language models (generally known as automata). A generative model defines a language by generating all strings of this language. In other words, a string belongs to this language if and only if it can be generated by the model. An accepting model analyzes a string and either accepts, or rejects it. The language defined by the accepting model is the set of all strings which the model accepts.

The applications of formal language theory are found in many scientific disciplines. It provides mathematical background primarily in areas that deal with languages themselves (linguistics, programming language theory etc.) but there are also other topics that can be formalized as languages (e.g. DNA and RNA sequences in biology).

Of particular interest to our work is the area of computational linguistics. Specifically, we focus on formal description of natural language syntax and its transformations. We study the application perspectives of known formal models and introduce new related concepts and definitions. We also study the theoretical properties of the models and present new results.

1.1 Motivation

Natural language processing is a field of theoretical informatics and linguistics and is concerned with the interactions between computers and human (natural) languages. It is defined as a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts (which means any language) at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications (according to [4]).

The history goes back to the the late 1940s, when there was an effort to understand and formally describe the syntax of natural languages. A big step forward was the publishing of

the book called Syntactic Structures, by Noam Chomsky, introducing the idea of generative grammar.

At first, computer processing of natural languages was in interest of artificial inteligence as a part of human-computer interaction. Subsequently, it split into two separate disciplines. Today, natural language processing studies many other aspects of natural languages besides their syntax (such as morphology or semantics). This discipline is focused mainly on practical applications. Some of the most frequent tasks are information retrieval, information extraction, question answering, summarization, and machine translation, and in broader scope, we can even include tasks as speech recognition and speech synthesis.

The second discipline encompasses a set of formalisms, which are, in general, known as formal language theory. Formal language theory is considered a part of theoretical computer science, and it focuses mainly on theoretical studies of various formal models and their properties. Its applications are now found in many other areas besides computational linguistics.

One of the major trends in formal language theory is regulated rewriting. This concept was introduced already in the 1960s, as the models of the now traditional Chomsky hierarchy have been found unsatisfactory for certain practical applications. For example, it has been argued that some linguistic phenomena could not be described by context-free grammars, while context-sensitive and unrestricted grammars were inefficient for practical use (because of the complexity of parsing). Because of this, ways to increase the power of context-free grammars—while retaining their practical applicability—were investigated.

Regulated rewriting essentially means that we take a certain known formal model (usually a context-free grammar, for reasons mentioned above) and in some way regulate (hence the name) the way in which it generates (or, in the case of automata, accepts) sentences. This can be done by adding some mathematically simple mechanism that controls the use of rules (such as in programmed grammars), or by changing the form of rules themselves (as, for example, in scattered context grammars). Thus, the expressive power is increased by limiting available derivations (or computations).

The purpose of our work is twofold. From a theoretical point of view, we contribute to the study of formal language theory by introducing new formal models and investigating their properties. Rather than trying to create completely new formalisms from scratch, we establish the new models as generalizations, extensions, or modifications of well-known and well-studied formal models (such as context-free grammars and finite automata) and principles (such as regulated rewriting and synchronization).

In [40], we have presented an alternate approach to synchronization, based on linking rules instead of nonterminals. In this fashion, we have extended the principle to models with regulated rewriting, specifically matrix grammars and scattered context grammars. We have continued with further theoretical study of synchronous grammars based on linked rules, and particularly of synchronous versions of regulated grammars, in [38] and [41].

In [10], we have introduced a new type of transducer, the rule-restricted automaton-grammar transducer, as a system consisting of a finite automaton, which is used to read an input string, and a context-free grammar, which simultaneously produces a corresponding output string. Also in [10], we have investigated the theoretical properties—namely, the generative and accepting power—of this new system and its variants.

For an overview of our new results, see the following Section 1.2, specifically the parts describing Chapter 4 and Chapter 5 for results concerning synchronous grammars and transducers, respectively.

Meanwhile, from a more practical viewpoint, we investigate how some of the well-

known and well-studied models from formal language theory can be adapted or extended for applications in natural language processing. In other words, the ideas and concepts behind the new formal models mentioned above are motivated by the possibility of their linguistic applications.

Inspired by such works as [63], where the authors discuss linguistically-oriented applications of scattered context grammars (using examples from the English language), we explore similar application perspectives of other regulated formal models as well. In [36], we have discussed potential applications of matrix grammars in the description of the Japanese syntax. Subsequently, we have been focusing on translation of natural languages.

Machine translation is one of the major tasks in natural language processing. With increasing availability of large corpora, corpus-based systems became favoured over rule-based, using statistical methods and machine-learning techniques. They mostly rely on formal models that represent local information only, such as n-gram models. However, recently, there have been attempts to improve results by incorporating syntactic information into such systems (see [50], [86], or [6]).

To do so, we need formal models that can describe syntactic structures and their transformations. Based on the principles of synchronous grammars (see [13]), we have proposed synchronous versions of some regulated grammars, such as matrix grammars (see [21]) and scattered context grammars (see [63]). We first introduced the idea in [37], and further elaborated upon it in [40]. Revised definitions, a study of theoretical properties, and a further discussion of linguistically-oriented application perspectives can be found in [41]; applications in particular are also investigated in [39].

Other type of models we can use are transducers (see [2]). Unlike synchronous grammars, which generate a pair of sentences in one derivation and thus define translation, transducers take a given input sentence and transform it into a corresponding output sentence. 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 [30] or [69]). In [10], we have introduced the rule-restricted automaton-grammar transducer and its variants, and discussed its advantages for natural language translation, illustrated by examples from Czech, English, and Japanese.

1.2 Organization

This doctoral thesis is divided into three parts and seven chapters, organized as follows.

1.2.1 Introduction

The first chapter introduces the topic of our work and presents the motivation behind it. It also describes the structure of this document and provides an overview of its contents.

Following this introductory chapter, Chapter 2 provides the mathematical background required for understanding of the topics discussed in this work. First, we summarize the well-known essential concepts and definitions from set theory, such as sets and relations. Subsequently, we use these notions to present an introduction to formal language theory. We give formal definitons of concepts such as alphabet, string, and language. We also introduce formal models that define languages, namely grammars and automata. We define different types of languages and present the resulting hierarchy of the respective language classes. Finally, we describe and formally define several models related to the concept of regulated rewriting.

In Chapter 3, we present a brief introduction to computational linguistics. The first section of this chapter provides an overview of select formal models related to natural language processing. We discuss both models of historical and practical importance. Transformational grammars, augmented transition networks and generalized phrase structure grammars are examples of the former category, while the latter includes head-driven phrase structure grammars, lexical functional grammars, and lexicalized tree-adjoining grammars. We also mention probabilistic context-free grammar as an example of a formal model used in statistical natural language processing.

We present the basic concept of dependency grammars as well. Although for the most part our work does not deal with dependency grammars, they certainly deserve a mention as an important alternative to phrase structure grammars, which is is also often used in practice. Moreover, we sometimes use the notion of dependency (and some related notions, particularly nonprojectivity, the crossing of dependencies) when discussing application perspectives of our formal models (Chapter 6).

Finally, we consider the application of some traditional models from formal language theory as an alternative. We focus on models with regulated rewriting. In particular, we discuss linguistically-oriented application perspectives of scattered context grammars and their variant, transformational scattered context grammars.

The second section of Chapter 3 introduces the area of machine translation, which is one of major tasks of natural language processing. First, we briefly review the historical development and classification of translation systems. Subsequently, we provide a summary of recent trends in this area, and we show how our work relates to them.

1.2.2 Synchronous Formal Systems

The second part of this thesis consists of two theoretically oriented chapters. These chapters contain both informal explanations and formal definitions of the new models, and present the related theoretical results that we have established.

More specifically, Chapter 4 deals with synchronous grammars. First, we briefly recall the well-known synchronous context-free grammars. We then introduce the notion of new synchronous grammars as systems consisting of two context-free grammars with linked rules instead of linked nonterminals. This allows us to naturally extend the principle of synchronization beyond context-free grammars. We present synchronous versions of some regulated grammars, namely scattered-context grammars and matrix grammars.

Further, we study theoretical properties of these grammars. Specifically, we investigate their generative power and achieve the following three main results. First, if we synchronize context-free grammars by linking rules as proposed and defined in this chapter, we obtain generative power coinciding with the power of matrix grammars. Consequently, we significantly increase the power in this way because the traditional synchronous CFGs only generate the family of context-free languages. Second, perhaps unsurprisingly, the class of languages defined by synchronous scattered context grammars equals the class of recursively enumerable languages. Finally, we show that if we synchronize matrix grammars by linking matrices, we obtain no increase in power. That is, synchronous matrix grammars have the same generative power as matrix grammars.

Chapter 5 introduces a new type of transducer, referred to as rule-restricted automaton-grammar transducer, based upon a finite automaton and a context-free grammar. A restriction set controls the computation. It defines which rules can be simultaneously used by the automaton and by the grammar. We discuss the power of this system working in

an ordinary way as well as in a leftmost way (more precisely, the context-free grammar is restricted to leftmost derivation). In addition, we introduce an appearance checking, which allows us to check whether some symbols are present in the rewritten string, and we investigate its effect on the power.

We achieve the following main results. First, we show that the generative power of rule-restricted transducers is equal to the generative power of matrix grammars. Second, the accepting power coincides with the power of partially blind multi-counter automata. 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 system can accept and generate all recursively enumerable languages.

1.2.3 Application Perspectives and Final Remarks

In the final part of this thesis, we consider the newly introduced models from a more practical viewpoint. Specifically, Chapter 6 explores their application perspectives with particular focus on natural language translation. We discuss and compare their main advantages illustrating them by examples from Czech, English, and Japanese.

One of the main advantages of both types of presented models is their power, as they are able to describe even some non-context-free structures. The new synchronous grammars also provide high flexibility, allowing for elegant and efficient description of language features. On the other hand, rule-restricted transducers are based on a simple, straightforward principle, which can be an advantage for practical implementation.

The concluding Chapter 7 summarizes all achieved results. In particular, using a graphical representation, we show how our new results relate to a known hierarchy of language classes. We also discuss further research prospects, both in theoretical and practical direction.

Chapter 2

Mathematical Background

This chapter recalls the fundemental terminology from mathematics and formal language theory that is of key importantance to the topics discussed in this work.

We assume that the reader is familiar with basic mathematical terms, concepts, and notations. In particular, knowledge and understanding of elementary algebra, logic, and common proof techniques is assumed (see [8], [45], and [53]).

2.1 Set Theory

Before we can talk about formal languages and the models that define them, we first need to introduce several notions from set theory (see [23], [33], or [42]).

2.1.1 Sets and Sequences

In mathematics, there are many well-known, important sets, such as the set of all integers, $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, natural numbers, $\mathbb{N} = \{1, 2, \ldots\}$, or natural numbers extended with zero, $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$. Intuitively, the notion of set as a collection of certain elements is easy to imagine. Now let us define it formally.

Definition 2.1 (Set)**.** A *set* is a collection of elements, which are taken from some prespecified universe. If an element a from this universe is contained in some set Σ, we say that a is a member of Σ, denoted by $a \in \Sigma$. Otherwise, we say that a is not a member of Σ, denoted by $a \notin \Sigma$. The set that has no members is called the *empty set* and it is denoted by $\emptyset$.

Definition 2.2 (Cardinality of a set)**.** Let Σ be a set. The *cardinality* of Σ, denoted by $\mathrm{card}(\Sigma)$, is the number of members of Σ. Note that $\mathrm{card}(\emptyset) = 0$.

Definition 2.3 (Finite and infinite set)**.** Let Σ be a set. If Σ has a finite number of members, that is, $\mathrm{card}(\Sigma) \in \mathbb{N}_0$, we say that Σ is a *finite set*, otherwise, if $\mathrm{card}(\Sigma) = \infty$, we say that Σ is an *infinite set*.

We usually specify a finite set Σ by listing its members—that is, $\Sigma = \{a_1, a_2, \ldots, a_n\}$, where $a_i \in \Sigma$ for all $1 \leq i \leq n$. For example, the finite set $A = \{1, 2\}$ contains two elements, 1 and 2, and $\mathrm{card}(A) = 2$. In contrast, an infinite set Ω is usually specified by a common property π. Then, a given element is a member of Ω if and only if it satisfies this property. We write this specification as $\Omega = \{a \colon \pi(a)\}$.

We can also compare sets. We can decide whether two given sets are equal, or whether one set is included in another.

Definition 2.4 (Set equivalence)**.** Let Σ and Ω be sets. We say that Σ and Ω are *equal*, denoted by $\Sigma = \Omega$, if and only if for all $a \in \Sigma$, $a \in \Omega$ holds, and for all $b \in \Omega$, $b \in \Sigma$ holds. Otherwise $\Sigma \neq \Omega$.

Definition 2.5 (Subset)**.** Let Σ and Ω be sets. If for all $a \in \Sigma$, $a \in \Omega$ holds, we say that Σ is a *subset* of Ω, denoted by $\Sigma \subseteq \Omega$, otherwise $\Sigma \nsubseteq \Omega$. Further, if $\Sigma \subseteq \Omega$ and $\Sigma \neq \Omega$, we say that Σ is a *proper subset* of Ω, denoted by $\Sigma \subset \Omega$, otherwise $\Sigma \not\subset \Omega$.

For example, for the sets of numbers mentioned above, $\mathbb{N} \subset \mathbb{N}_0 \subset \mathbb{Z}$ holds. Also note that for any two sets $\Sigma = \Omega$ if and only if $\Sigma \subseteq \Omega$ and $\Omega \subseteq \Sigma$.

There are many operations that we can define over sets. For our purposes, we only need to introduce three of the most basic ones, namely intersection, union, and difference.

Definition 2.6 (Set operations)**.** Let Σ and Ω be sets. We define the following operations:

- The *intersection* of Σ and Ω, denoted by $\Sigma \cap \Omega$, as $\Sigma \cap \Omega = \{a \colon a \in \Sigma \wedge a \in \Omega\}$; if $\Sigma \cap \Omega = \emptyset$, we say that Σ and Ω are *disjoint*.
- The *union* of Σ and Ω, denoted by $\Sigma \cup \Omega$, as $\Sigma \cup \Omega = \{a \colon a \in \Sigma \vee a \in \Omega\}$.
- The *difference* of Σ and Ω, denoted by $\Sigma - \Omega$, as $\Sigma - \Omega = \{a \colon a \in \Sigma \wedge a \notin \Omega\}$.

Besides sets, we will also need sequences. This is a similar notion to set, with two important differences. First, a sequence may contain an element more than once, and second, the elements appear in certain order.

Definition 2.7 (Sequence)**.** Let $a_1, a_2, \ldots, a_n$ for some $n \in \mathbb{N}$ be elements taken from some pre-specified universe. Then, $(a_1, a_2, \ldots, a_n)$ denotes the (finite) *sequence* consisting of elements $a_0, a_1, \ldots, a_n$, in that order. Further, let $x = (a_1, a_2, \ldots, a_k)$ and $y = (b_1, b_2, \ldots, b_l)$ for some $k, l \in \mathbb{N}$ be sequences. Then, $x = y$ if and only if $k = l$ and for all $1 \leq i \leq k$, $a_i = b_i$.

For example, $\{1, 2\} = \{2, 1\}$ but $(1, 2) \neq (2, 1)$. Also note that, while we only deal with finite sequences in this work, it is possible—and indeed usual—to define infinite sequences (analogous to inifinite sets) as well.

Definition 2.8 (Length of a sequence)**.** Let $x = (a_1, a_2, \ldots, a_n)$ for some $n \in \mathbb{N}$ be a sequence. Then, the *length* of x, denoted by $|x|$, is defined as $|x| = n$.

A finite sequence of length n is also called an n-tuple. Further, a 2-tuple is also called an (ordered) pair, and for $3 \leq n \leq 7$, the respective n-tuples are called a triple, quadruple, quintuple, sextuple, and septuple (we could continue in this fashion but it becomes increasingly uncommon).

2.1.2 Relations

Relations are another important notion of set theory. To introduce them formally, we first need to define one more operation over sets, the Cartesian product.

Definition 2.9 (Cartesian product)**.** Let Σ and Ω be sets. The *Cartesian product* of Σ and Ω, denoted by $\Sigma \times \Omega$, is defined as $\Sigma \times \Omega = \{(a, b) \colon a \in \Sigma, b \in \Omega\}$.

Definition 2.10 (Relation)**.** Let Σ and Ω be sets. A (binary) *relation* ρ from Σ to Ω is any set ρ such that $\rho \subseteq \Sigma \times \Omega$.

Instead of $(a, b) \in \rho$, we usually write $a \in \rho(b)$ or $a\rho b$.

Definition 2.11 (k-fold product)**.** Let ρ be a relation from Σ to Σ and $k \in \mathbb{N}_0$. The *k-fold product* of ρ, denoted by ρ^k, is recursively defined as follows:

1. $a\rho^0 b$ if and only if $a = b$,
2. $a\rho^1 b$ if and only if $a\rho b$,
3. $a\rho^n b$ if and only if there is a $c \in \Sigma$ such that $a\rho^{n-1}c$ and $c\rho b$.

Definition 2.12 (Closure)**.** Let ρ be a relation from Σ to Σ. The *transitive closure* of ρ, denoted by ρ^+, is defined as follows: $a\rho^+ b$ if and only if for some $k \in \mathbb{N}$, $a\rho^k b$. Further, the *transitive and reflexive closure* of ρ, denoted by ρ^*, is defined as follows: $a\rho^* b$ if and only if $a\rho^+ b$ or $a = b$.

Definition 2.13 (Function)**.** Let Σ and Ω be sets and let ϕ be a relation from Σ to Ω such that for all $a \in \Sigma$, $\mathrm{card}(\{b\colon b \in \Omega, (a, b) \in \phi\}) \leq 1$. Then, we call ϕ a *function* from Σ to Ω.

By definition, for a given function ϕ and a given element a, there may be at most one b satisfying $b \in \phi(a)$. Therefore, in case of functions we usually write $b = \phi(a)$ instead of $b \in \phi(a)$.

Let max be a function from $\mathbb{Z} \times \mathbb{Z}$ to $\mathbb{Z}$ defined as follows: for $x, y \in \mathbb{Z}$, $\max(x, y) = x$ if and only if $x \geq y$, otherwise $\max(x, y) = y$. That is, $\max(x, y)$ returns the greater value from x and y. Note that, for brevity, we write $\max(x, y)$ instead of $\max((x, y))$.

2.2 Formal Language Theory

Using the notions from Section 2.1, we can now introduce the fundamental terms and concepts of formal language theory (see [34], [43], [55], [60], [74], or [75]).

2.2.1 Formalization of Languages

Informally, we consider languages as sets of sentences or sets of words. Indeed, the formal notion of language is similar. A formal language is a set of strings over some given alphabet.

Definition 2.14 (Alphabet and symbols)**.** An *alphabet* is a finite, nonempty set. Its members are called *symbols*.

Definition 2.15 (String)**.** Let Σ be an alphabet. A *string* over Σ is recursively defined as follows:

1. ε is a string over Σ. ε denotes the string with no symbols, and we call it the *empty string*.
2. If x is a string over Σ and $a \in \Sigma$, then xa is a string over Σ.

Further, let Σ^* denote the set of all strings over Σ.

Definition 2.16 (Language)**.** Let Σ be an alphabet. A *language* over Σ is any set L such that $L \subseteq \Sigma^*$.

The following definitions presents select operations over strings.

Definition 2.17 (String operations)**.** Let x and y be strings over an alphabet Σ. Then, we define the following operations:

- The *concatenation* of x and y as xy. For any string x, let $x\varepsilon = \varepsilon x = x$.
- The *length* of x, denoted by $|x|$, as follows:
  1. If $x = \varepsilon$, then $|x| = 0$.
  2. If $x = a_1 a_2 \dots a_n$ for some $n \geq 1$, where $a_i \in \Sigma$ for all $1 \leq i \leq n$, then $|x| = n$.
- The *reversal* of x, denoted by $(x)^R$, as follows:
  1. If $x = \varepsilon$, then $(x)^R = \varepsilon$.
  2. If $x = a_1 a_2 \dots a_{n-1} a_n$ for some $n \geq 1$, where $a_i \in \Sigma$ for all $1 \leq i \leq n$, then $(x)^R = a_n a_{n-1} \dots a_2 a_1$.
- The (i-th) *power* of x, denoted by x^i, as follows:
  1. $x^0 = \varepsilon$,
  2. $x^i = x x^{i-1}$ for $i \geq 1$.

Definition 2.18 (Substring)**.** Let x and y be strings over Σ. We say that x is a *substring* of y, if there are strings $u, v \in \Sigma^*$ such that $y = uxv$.

Let x be a string over Σ and let $x_1, x_2, \dots, x_n$ for some $n \in \mathbb{N}$ be substrings of x such that $x = x_1 x_2 \dots x_n$. Then, we call $x_1 x_2 \dots x_n$ a factorization of x.

Definition 2.19 (Occurence of symbols)**.** Let x be a string over an alphabet Σ and let $\Omega \subseteq \Sigma$. Then, $\mathrm{occur}(x, \Omega)$ denotes the number of occurrences of symbols from Ω in x.

Since languages are sets, we can apply all set operations (union, intersection etc.) on languages as well. We also introduce operations specific to languages, such as concatenation, where the sentences of the resulting language are formed by concatenation of sentences of the original languages, formally defined below.

Definition 2.20 (Concatenation of languages)**.** Let L_1 and L_2 be languages. We define the *concatenation* of L_1 and L_2, denoted by $L_1 \cdot L_2$ or simply $L_1 L_2$, as $L_1 \cdot L_2 = \{xy \colon x \in L_1, y \in L_2\}$.

Definition 2.21 (Language class)**.** Any set of languages is called a *language class*.

2.2.2 Grammars and Language Classes

Generally, there are two basic types of language-defining models, namely grammars and automata. The former are formal devices that generate strings, while the latter accept strings. Informally, a grammar is based on the following basic principle. We start from some abstract symbol, and according to some given rules, we rewrite it to some string. Then, we take some symbols from this string and again rewrite them according to some rules, thus obtaining a new string, and then continue in the same fashion. Now we only we need to know when to terminate this process. To that end, we designate some of the symbols as terminal, and we finish when we obtain a string consisting solely of these terminal symbols.

Definition 2.22 (Grammar)**.** An (unrestricted generative) *grammar* G is a quadruple $G = (N, T, P, S)$, where

- N is a *nonterminal* alphabet,
- T is a *terminal* alphabet, $N \cap T = \emptyset$,
- P is a finite relation from $(N \cup T)^+$ to $(N \cup T)^*$, represented as a set of *derivation rules* of the form
$$x \to y,$$
where $x \in (N \cup T)^+$, $y \in (N \cup T)^*$, and
- $S \in N$ is the *start symbol*.

Any string $w \in (N \cup T)^*$ is called a *sentential form* of G. Further, if $w \in T^*$, we call w a *sentence*. Let uxv and uyv be two sentential forms and let $p = x \to y \in P$. Then, we say that uxv *directly derives* uyv in G according to rule p, written as $uxv \Rightarrow_G uyv\,[p]$ or simply $uxv \Rightarrow uyv$. Alternatively, we say that G makes a *derivation step* from uxv to uyv (according to rule p), and any sequence of derivation steps starting by rewriting S is called a *derivation*. As with any relation, $\Rightarrow^k$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the k-fold product, the transitive closure, and the transitive and reflexive closure of $\Rightarrow$, respectively.

The *language generated by* G, denoted by $L(G)$, is defined as $L(G) = \{w \colon S \Rightarrow^* w\}$.

Let $G = (N, T, P, S)$ be a grammar. For a rule $p = x \to y \in P$, we say that x is the left-hand side of p and y is the right-hand side of p. Further, let w be a sentential form of G. We say that $p = x \to y$ is applicable to w if and only if x is a substring of w.

We say that a derivation $S \Rightarrow^* w$ is successful if and only if it produces a sentence ($w \in T^*$). Conversely, we say that a derivation $S \Rightarrow^* w$ is unsuccessful if and only if $\mathrm{occur}(w, N) \geq 1$ and there is no rule applicable to w (informally, the derivation cannot continue but there are still some nonterminals left).

Chomsky Hierarchy of Languages

In the late 1950s, the linguist Noam Chomsky introduced an initial classification of grammars and their respective language classes [16], now famous as the Chomsky hierarchy of languages. This classification is based on different restriction placed on the form of rules.

Definition 2.23 (Context-sensitive grammar)**.** Let $G = (N, T, P, S)$ be a grammar. We say that G is a *context-sensitive grammar* (CSG for short), if and only if all the rules in P are of the form
$$uAv \to uxv,$$
where $A \in N$ and $x, u, v \in (N \cup T)^*$.

Definition 2.24 (Context-free grammar)**.** Let $G = (N, T, P, S)$ be a CSG. We say that G is a *context-free grammar* (CFG for short), if and only if all the rules in P are of the form
$$A \to x,$$
where $A \in N$ and $x \in (N \cup T)^*$.

Definition 2.25 (Regular grammar)**.** Let $G = (N, T, P, S)$ be a CFG. We say that G is a (right) *regular grammar* (RG for short), if and only if all the rules in P are of the form

$$A \to aB,$$

where $A \in N$, $a \in T \cup \{\varepsilon\}$, and $B \in N \cup \{\varepsilon\}$.

Based on the above types of grammars, we define the respective types of languages and language classes.

Definition 2.26 (Language types and classes)**.** Let L be a language. We say that L is a:

- *recursively enumerable language*, if and only if there is a grammar G such that $L(G) = L$;
- *context-sensitive language*, if and only if there is a CSG G such that $L(G) = L$;
- *context-free language*, if and only if there is a CFG G such that $L(G) = L$;
- *regular language*, if and only if there is a RG G such that $L(G) = L$.

Further, let **RE**, **CS**, **CF**, and **REG** denote the class of all recursively enumerable, context-sensitive, context-free, and regular languages, respectively.

For the above language classes, it holds that **REG** $\subset$ **CF** $\subset$ **CS** $\subset$ **RE** (see [60]).

2.2.3 Automata

While grammars generate languages, automata are language acceptors. That is, an automaton reads a string and decides whether or not this string belongs to a certain language. Informally, an automaton is a formal system that has different states, and it moves between these states based on its input, according to some given rules. Generally, some of the states are designated as final, and an automaton accepts a string if and only if it reads all its symbols and in doing so, reaches a final state. In formal language theory, there are many well-known types of automata. Here, we only define two of the most fundamental ones, namely (nondeterministic) finite automaton and pushdown automaton.

Definition 2.27 (Finite automaton)**.** A *finite automaton* (FA for short) M is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of *states*,
- Σ is an *input* alphabet,
- δ is a finite relation from $Q \times (\Sigma \cup \{\varepsilon\})$ to Q, represented as a set of *transition rules* of the form

  $$pa \to q,$$

  where $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$,
- $q_0 \in Q$ is the *start state*, and
- $F \subseteq Q$ is a set of *final states*.

Any string $\chi \in Q\Sigma^*$ is called a *configuration* of M. Let pax and qx be two configurations of M and let $r = pa \to q \in \delta$. Then, we say that M makes a *move* (or *computation step*) from pax to qx according to rule r, written as $pax \Rightarrow_M qx\,[r]$ or simply $pax \Rightarrow qx$. As with any relation, $\Rightarrow^k$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the k-fold product, the transitive closure, and the transitive and reflexive closure of $\Rightarrow$, respectively.

The *language accepted by* M, denoted by $L(M)$, is defined as $L(M) = \{w\colon q_0 w \Rightarrow^* f, f \in F\}$.

Further, let $\mathscr{L}(\text{FA})$ denote the class of all languages accepted by FAs.

Informally, a pushdown automaton is an extension of FA with the added ability to "remember" some symbols by storing them in the pushdown. At any given time, only the top of the pushdwon is accessible—that is, symbols can only be placed on top of the pushdown and only the topmost symbol can be read and removed.

Definition 2.28 (Pushdown automaton)**.** A *pushdown automaton* (PDA) M is a septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, S, F)$, where

- Q is a finite set of *states*,
- Σ is an *input* alphabet,
- Γ is a *pushdown* alphabet,
- δ is a finite relation from $\Gamma \times Q \times (\Sigma \cup \{\varepsilon\})$ to $\Gamma^* \times Q$, represented as a set of *transition rules* of the form
$$Apa \to wq,$$
where $A \in \Gamma$, $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $w \in \Gamma^*$,
- $q_0 \in Q$ is the *start state*,
- $S \in \Gamma$ is the *initial pushdown symbol*, and
- $F \subseteq Q$ is a set of *final states*.

Any string $\chi \in \Gamma^* Q \Sigma^*$ is called a *configuration* of M. Let $xApay$ and $xwqy$ be two configurations of M and let $r = Apa \to wq \in \delta$. Then, we say that M makes a *move* (or *computation step*) from $xApay$ to $xwqy$ according to rule r, written as $xApay \Rightarrow_M xwqy\,[r]$ or simply $xApay \Rightarrow xwqy$. As with any relation, $\Rightarrow^k$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the k-fold product, the transitive closure, and the transitive and reflexive closure of $\Rightarrow$, respectively.

The *language accepted by* M, denoted by $L(M)$, is defined as $L(M) = \{w\colon Sq_0 w \Rightarrow^* f, f \in F\}$.

Further, let $\mathscr{L}(\text{PDA})$ denote the class of all languages accepted by PDAs.

For both FA and PDA, we call any sequence of computation steps starting from the start state a *computation*.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an FA. For a rule $r = pa \to q \in \delta$, we say that pa is the left-hand side of p and q is the right-hand side of p. Further, let χ be a sentential form of M. We say that $r = pa \to q$ is applicable to χ if and only if $\chi = pax$ for some $x \in \Sigma^*$.

We say that a computation $q_0 x \Rightarrow^* py$ is successful if and only if $p \in F$ and $y = \varepsilon$ (informally, we have read the whole input string and reached an end state). A succesful computation $q_0 w \Rightarrow^* f$ is called an acceptance of w. Conversely, we say that a computation $q_0 x \Rightarrow^* py$ is unsuccessful if and only if there is no rule applicable to py and $|y| \geq 1$ or $p \notin F$

(informally, the computation cannot continue but the conditions for succesful acceptance are not met).

The above notions are analogous for PDA.

As for the respective language classes, it is known that $\mathscr{L}(\text{FA}) = \mathbf{REG}$ and $\mathscr{L}(\text{PDA}) = \mathbf{CF}$ (see [60]).

2.2.4 Regulated Rewriting

Here, we present several formal models based on the principle of regulated rewriting (see [20], [21], [25], [52], [58], [62], or [64]). In this work, we primarily deal with matrix grammars (see [1] and [21]) and scattered context grammars (see [29] and [63]).

Grammars

One of the earliest examples of regulated grammars is matrix grammar, introduced already in 1965 [1]. In essence, it is a system consisting of a CFG and a controlling set, which restricts available derivations in the CFG by specifying sequences of rules that must be applied directly after each other.

Definition 2.29 (Matrix grammar)**.** A *matrix grammar* (MAT for short) H is a pair $H = (G, M)$, where

- $G = (N, T, P, S)$ is a CFG and
- M is a finite language over P ($M \subset P^*$); members of M are called *matrices*.

Any string $w \in (N \cup T)^*$ is called a *sentential form* of H. Further, if $w \in T^*$, we call w a *sentence*. Let u and v be two sentential forms. We say that u *directly derives* v in H according to matrix m, written as $u \Rightarrow_H v\,[m]$ or simply $u \Rightarrow v$, if and only if $m = p_1 \dots p_n \in M$ and there are strings $x_0, \dots, x_n$ such that $x_0 = u$, $x_n = v$, and for all $0 \leq i < n$, $x_i \Rightarrow x_{i+1}\,[p_{i+1}]$ in G. Note that this makes for one derivation step in H (during which an arbitrary number of derivation steps in G may be performed, or even none at all, if $m = \varepsilon$). As with any relation, $\Rightarrow^k$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the k-fold product, the transitive closure, and the transitive and reflexive closure of $\Rightarrow$, respectively.

The *language generated by* H, denoted by $L(H)$, is defined as $L(H) = \{w \colon S \Rightarrow^* w\}$.

We can extend matrix grammar with the possibility to skip select rules in a matrix if they are not applicable. This is an example of the principle known as appearance checking (recall that a given rule is applicable to a sentential form if and only if its left-hand side appears in the sentential form).

Definition 2.30 (Matrix grammar with appearance checking)**.** A *matrix grammar with appearance checking* (MAT_{ac}) H is a pair $H = (G, M)$, where

- $G = (N, T, P, S)$ is a CFG and
- M is a finite set of strings of pairs (p, t), where $p \in P$ and $t \in \{-, +\}$; members of M are called *matrices*.

Any string $w \in (N \cup T)^*$ is called a *sentential form* of H. Further, if $w \in T^*$, we call w a *sentence*. Let u and v be two sentential forms. We say that u *directly derives* v in H according to matrix m, written as $u \Rightarrow_H v\,[m]$ or simply $u \Rightarrow v$, if and only if $m = (p_1, t_1) \dots (p_n, t_n) \in M$ and there are strings $x_0, \dots, x_n$ such that $x_0 = u$, $x_n = v$, and for all $0 \leq i < n$, either

- $x_i \Rightarrow x_{i+1}\,[p_{i+1}]$ in G, or
- $t_{i+1} \in \{-\}$, $x_i = x_{i+1}$, and p_{i+1} is not applicable on x_i in G.

As with any relation, $\Rightarrow^k$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the k-fold product, the transitive closure, and the transitive and reflexive closure of $\Rightarrow$, respectively.

The *language generated by* H, denoted by $L(H)$, is defined as $L(H) = \{w \colon S \Rightarrow^* w\}$.

Another well-known example is scattered context grammar, first introduced in 1969 [29]. Informally, it is a modification of CFG where, instead of rewriting one nonterminal to one string, we simultaneously rewrite n nonterminals to n strings in one derivation step.

Definition 2.31 (Scattered context grammar)**.** A *scattered context grammar* (SCG) G is a quadruple $G = (N, T, P, S)$, where

- N is a *nonterminal* alphabet,
- T is a *terminal* alphabet, $N \cap T = \emptyset$,
- P is a finite set of *rules* of the form

$$(A_1, \ldots, A_n) \to (x_1, \ldots, x_n),$$

  where $n \geq 1$ and for all $1 \leq i \leq n$, $A_i \in N$, $x_i \in (N \cup T)^*$, and
- $S \in N$ is the *start symbol*.

Any string $w \in (N \cup T)^*$ is called a *sentential form* of G. Further, if $w \in T^*$, we call w a *sentence*. Let u and v be two sentential forms. We say that u *directly derives* v in G according to rule p, written as $u \Rightarrow_G v\,[p]$ or simply $u \Rightarrow v$, if and only if there is a factorization of $u = u_1 A_1 \ldots u_n A_n u_{n+1}$ and $v = u_1 x_1 \ldots u_n x_n u_{n+1}$ where for all $1 \leq i \leq n+1$, $u_i \in (N \cup T)^*$, such that $p = (A_1, \ldots, A_n) \to (x_1, \ldots, x_n) \in P$. As with any relation, $\Rightarrow^k$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the k-fold product, the transitive closure, and the transitive and reflexive closure of $\Rightarrow$, respectively.

The *language generated by* G, denoted by $L(G)$, is defined as $L(G) = \{w \colon S \Rightarrow^* w\}$.

The further notions regarding rules (sides, applicability) and derivations (successful, unsuccessful) for MAT, MAT$_{ac}$, and SCG are analogous with unrestricted grammars (see Definition 2.22).

Further, let $\mathscr{L}(\text{MAT})$, $\mathscr{L}(\text{MAT}_{ac})$, and $\mathscr{L}(\text{SCG})$ denote the class of all languages generated by matrix grammars, matrix grammars with appearance checking, and scattered context grammars, respectively. It is known that $\mathbf{CF} \subset \mathscr{L}(\text{MAT}) \subset \mathbf{CS}$ [21], $\mathscr{L}(\text{MAT}_{ac}) = \mathbf{RE}$ [21], and $\mathscr{L}(\text{SCG}) = \mathbf{RE}$ [59].

Automata

Although the term regulated rewriting is primarily associated with grammars, there are also some well-known automata closely related to this concept, such as counter automata (see [27], [28], [35], or [81]).

Definition 2.32 (k-counter automaton)**.** A *k-counter automaton* (k-CA) M is an FA $M = (Q, \Sigma, \delta, q_0, F)$ with k integers $v = (v_1, \ldots, v_k)$ in $\mathbb{N}_0^k$ as an additional storage. *Transition rules* in δ are of the form $pa \to q(t_1, \ldots, t_n)$, where $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, and $t_i \in \{-\} \cup \mathbb{Z}$.

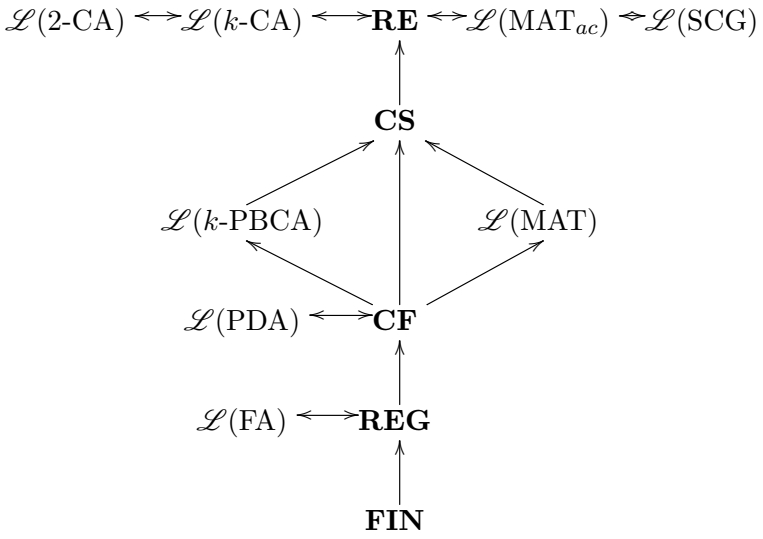


Figure 2.1: Hierarchy of select language classes (for $k \geq 2$). $\mathscr{L}_1 \to \mathscr{L}_2$ denotes $\mathscr{L}_1 \subset \mathscr{L}_2$ and $\mathscr{L}_1 \leftrightarrow \mathscr{L}_2$ denotes $\mathscr{L}_1 = \mathscr{L}_2$

A *configuration* of k-CA is any string from $Q\Sigma^*\mathbb{N}_0^k$. Let $\chi_1 = paw(v_1, \ldots, v_k)$ and $\chi_2 = qw(v'_1, \ldots, v'_k)$ be two configuration of M and $r = pa \to q(t_1, \ldots, t_k) \in \delta$, 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* (or *computation step*) from configuration χ_1 to χ_2 according to r, written as $\chi_1 \Rightarrow \chi_2 \, [r]$, or simply $\chi_1 \Rightarrow \chi_2$. As with any relation, $\Rightarrow^k$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the k-fold product, the transitive closure, and the transitive and reflexive closure of $\Rightarrow$, respectively.

The *language accepted by* M, denoted by $L(M)$, is defined as $L(M) = \{w \colon w \in \Sigma^*, q_0 w(0, \ldots, 0) \Rightarrow^* f(0, \ldots, 0), f \in F\}$.

Definition 2.33 (Partially blind k-counter automaton)**.** A *partially blind k-counter automaton* (k-PBCA) M is an FA $M = (Q, \Sigma, \delta, q_0, F)$ with k integers $v = (v_1, \ldots, v_k)$ 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 $\chi_1 = paw(v_1, \ldots, v_k)$ and $\chi_2 = qw(v'_1, \ldots, v'_k)$ be two configurations of M and $r = pa \to q(t_1, \ldots, t_k) \in \delta$, where $(v_1 + t_1, \ldots, v_k + t_k) = (v'_1, \ldots, v'_k)$. Then, M makes a *move* (or *computation step*) from configuration χ_1 to χ_2 according to r, written as $\chi_1 \Rightarrow \chi_2 \, [r]$, or simply $\chi_1 \Rightarrow \chi_2$. As with any relation, $\Rightarrow^k$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the k-fold product, the transitive closure, and the transitive and reflexive closure of $\Rightarrow$, respectively.

The *language accepted by* M, denoted by $L(M)$, is defined as $L(M) = \{w \colon w \in \Sigma^*, q_0 w(0, \ldots, 0) \Rightarrow^* f(0, \ldots, 0), f \in F\}$.

For $k \in \mathbb{N}$, let $\mathscr{L}(k\text{-CA})$ and $\mathscr{L}(k\text{-PBCA})$ denote the class of all languages accepted by k-CAs and k-PBCAs, respectively. It is known that for any $k \geq 2$, $\mathscr{L}(k\text{-CA}) = \mathbf{RE}$ [35], and $\mathbf{CF} \subset \mathscr{L}(k\text{-PBCA}) \subset \mathbf{CS}$ [27, 28].

2.2.5 Hierarchy of Languages

Throughout this chapter, we have introduced a number of language-defining models and their respective language classes. For a summary of their relations, see the graphical representation in Figure 2.1. **FIN** denotes the class of all finite languages (finite sets).

Chapter 3

Computational Linguistics: An Overview

This two-part chapter serves as an introduction to the field of computational linguistics (see [4], [9], [56], [68], or [70]). More specifically, in the first part, we discuss different kinds of formal models applied in natural language processing, both from historical and practical point of view. In the second part of this chapter, we briefly introduce the task of machine translation and some of its recent trends.

3.1 Formal Models in Natural Language Processing

This section provides an overview of various formalisms that have been used in computational linguistics and particularly in the description of natural language syntax. Note that we only recall the core principles and concepts that are relevant to our work. Detailed information about particular formal models can be found in referenced literature.

The content of this section is partially based on results of the MŠMT FRVŠ grant project FR97/2011/G1 and on [85].

3.1.1 Dependency Grammars

In 1959, Lucien Tesnière presented the theory of structural syntax [79], which is now considered the starting point of modern dependency grammar theory. Since then, the term dependency grammars has grown to encompass many particular formalisms, such as word grammar [44], functional generative description [77], meaning-text theory [66], or extensible dependency grammar [22]. Here, we review the common core principles behind these formal models.

The fundamental notion of *dependency* is based on the idea that the syntactic structure of a sentence consists of binary asymmetrical relations between words. That is, one word is the head of a phrase, and other words modify it. For example, in the phrase *pink elephants jumped*, there is a relation between the words *elephants* and *jumped* such that the noun *elephants* modifies the verb *jumped* by providing further specification (who jumped). The noun is then itself modified by the adjective *pink*.

Words in dependency relation are often called *parent* and *child*, although, depending on the particular formalism, other terms may be used as well (such as governor and dependent or head and modifier). In this work, we choose the terms parent and child, as they are perhaps the most general.

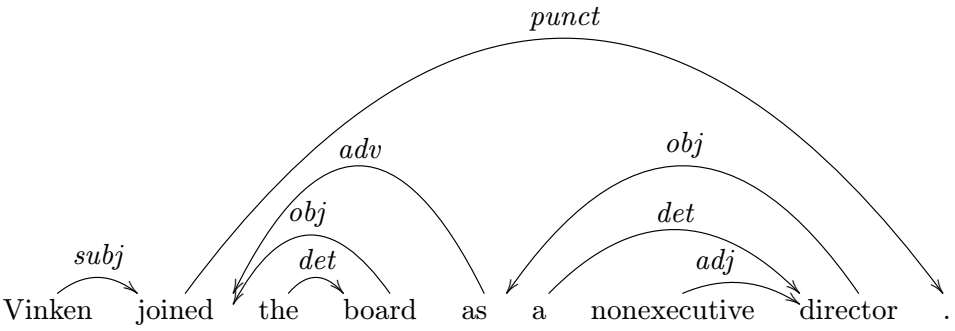


Figure 3.1: Example of dependency tree (with labels)

If w is the child of v (thus, v is the parent of w), we write $w \to v$.[1] As usual, $\to^*$ denotes the transitive and reflexive closure of $\to$. We say that there is a *path* from w to v if and only if $w \to^* v$.

Let $s = w_1 w_2 \dots w_n$ be a sentence. A dependency tree has the following properties:

- *Single head* – each word has one and only one parent (except for the root node, which has none).
- *Connected* – all words form a connected graph. That is, for all $1 \leq i \leq n$, $1 \leq j \leq n$, there is a path from w_i to w_j or a path from w_j to w_i.
- *Acyclic* – the graph does not contain cycles. That is, for all $1 \leq i \leq n$, $1 \leq j \leq n$, the following holds: if $w_i \to w_j$, then $w_j \to^* w_i$ never holds.
- *Projective* – there is no crossing between dependencies. That is, for all $1 \leq i \leq n$, $1 \leq j \leq n$, the following holds: if $w_i \to w_j$, then for all w_k such that $i < k < j$, there is either a path from w_k to w_i, or a path from w_k to w_j.

In general, the first three conditions must always be satisfied. On the other hand, while some dependency formalisms assume projectivity, other allow non-projective dependency trees as well.

An example of a projective dependency tree can be seen in Figure 3.1. Note that we can assign labels to the branches (dependencies) to provide further information about the way in which a child modifies its parent word. For instance, here we can see that *board* is an object of the verb *joined*. Figure 3.2 shows an example of non-projectivity, as there is a crossing between branches (*yesterday* $\to$ *ate* and *was* $\to$ *cake*).

Dependency grammars are often used in practice because of their relative simplicity, robustness, and portability. Their principles are easy to understand, they allow for faster manual annotation of sentences in corpora (in phrase structure grammars discussed below, the trees are generally much more complicated, and we need some base set of grammar rules), and also for efficient parsing, which is a very important factor in practical implementations. They are uniformly applicable to many different languages, and, in general, they can parse any sentence. Permutations of words are possible without affecting syntactic structure, which can be an advantage when dealing with free-word-order languages (such as Czech).

[1] Note that in some dependency formalisms, the arrows are drawn in the opposite direction (from parent to child) instead.

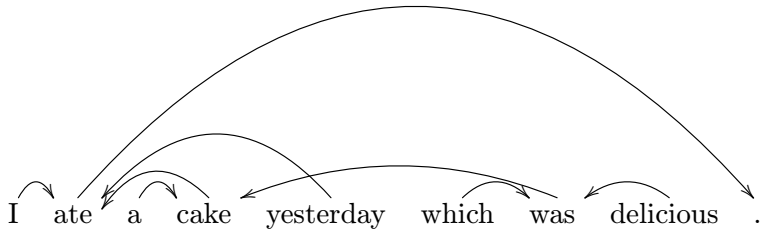


Figure 3.2: Non-projective dependency tree

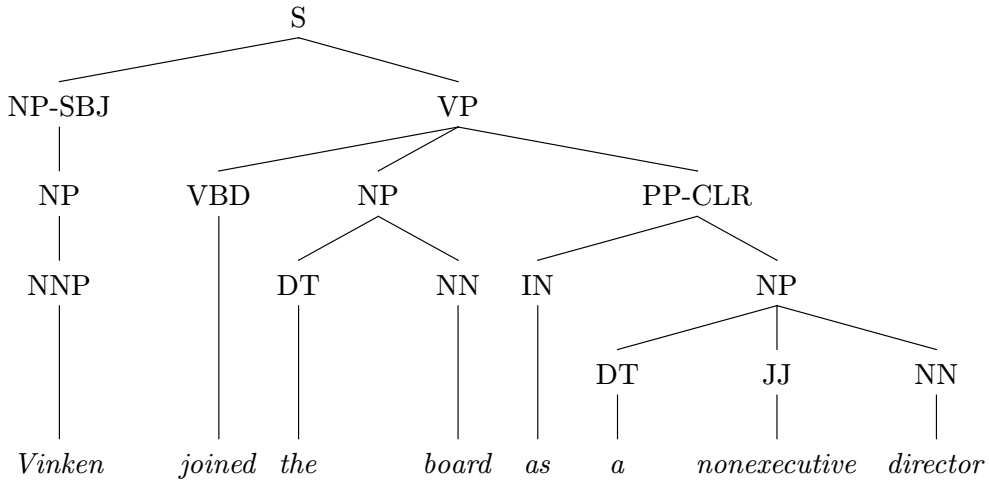


Figure 3.3: Example of phrase structure tree

3.1.2 Phrase Structure Grammars

Traditionally, the term *phrase structure grammars* refers to the grammars of the Chomsky hierarchy. Indeed, RGs, CFGs, and CSGs are all examples of phrase structure grammars. However, particularly in the field of natural language processing, the term has been used to denote any formal grammars that are based on the *constituency* relation (in contrast to the dependecy relation in dependency grammars). Throughout this text, we assume the latter interpretation.

Phrase structure grammars create hierarchical structures over sentences, based on the idea that a sentence is composed of several *constituents* (words or phrases), and each constituent itself consists of one or more constituents. The lowest-level constituents are (usually) words. A group of elements forms a constituent whenever they have been introduced by the application of the same rule.

We usually present the structure of a sentence in the form of *phrase structure tree* (or *syntax tree*), which recapitulates the process by which a sentence is generated by the rules of a grammar. An example of a syntax tree (adapted from Penn Treebank) is given in Figure 3.3. For instance, the node NP represents a noun phrase, a constituent formed by, for example, a determiner (DT) and a noun (NN). Also compare the dependency tree in Figure 3.1, which describes the same sentence.

Note that while the dependency relation is a one-to-one relation (for every word in a sentence, there is exactly one corresponding node in the syntactic structure), the con-

stituency relation is a one-to-one-or-more correspondence—that is, there is at least one node corresponding to each word (but there can be more).

Phrase structure grammars are generally considered to be more suitable for languages with fixed word order and clear constituency structures. A major advantage over dependency grammars lies in the fact that we have explicit information about the constituents of a sentence. This is also the main reason why we choose to focus on phrase structure grammars rather than dependency grammars in our work.

In this section, we will take a closer look at transformational grammar [70], generalized phrase structure grammar [26], and head-driven phrase structure grammar [72]. Other examples of phrase structure grammars used in computational linguistics include lexical functional grammar [24], (lexicalized) tree-adjoining grammar [49], or combinatory categorial grammar [7].

Transformational Grammars

Developed by Noam Chomsky in 1957 [17], transformational grammars are among the oldest formal models in computational linguistics. They are based on the idea that a sentences in a natural language has two levels of representation: a *deep structure* and a *surface structure*. The former represents the core semantic relations of a sentence, while the latter closely follows its phonological form. Transformations realize the mapping of deep structures onto surface structures.

A *transformational grammar* H consists of three components: a phrase structure grammar (usually a CFG) G, called the *base* of H, a set of *transformations* T, and a set of *restrictions* on these transformations R (specifying that some transformations in T are obligatory). Then, deep structures are derivation trees generated by G, and surface structures are trees which can be obtained from deep structures by successively applying transformations from T (according to restrictions in R). The language generated by a transformational grammar H, denoted by $L(H)$, is the set of strings that we may read off the surface structures generated by H.

Transformational grammars (with context-free bases) are strictly stronger than CFGs, as we can use transformations to obtain the intersection of two context-free languages.

Generalized Phrase Structure Grammar

Generalized phrase structure grammar (GPSG for short) was created in an attempt to show that it is possible to describe natural languages in a context-free framework, without using transformations. To recreate the effects of transformations (from transformational grammars), GPSG introduces *features* and *metarules*.

There are two types of features: *atom-valued* and *category-valued*. Atom-valued features have Boolean values (denoted by + and −), and are represented by symbols such as $[-INF]$ (finite, an inflected verb, e.g. *eats*), $[+INF]$ (infinitival, e.g. *to eat*), or $[-INV]$ (inverted, indicating subject-auxiliary inversion, as in *Is John sick?*). Category-valued features essentially have nonterminal symbols as their values. For example, consider a transitive verb phrase VP. VP$[SLASH$ = NP] (or VP/NP for short) represents this verb phrase with a missing noun phrase (object), for example in *wh*-questions (e.g. *Who did John hit?*).

Two important operations in GPSG are feature extension and feature unification. The former essentially means adding a new feature to a feature specification (a set of features). For example, for a feature specification $\{[+N], [+V]\}$ (adjective, which is a category with

both noun- and verb-like qualities), a possible extension is $\{[+N], [+V], [+PRED]\}$ (adjective in a predicative position). Feature unification combines two feature specifications into one. It is similar to the set union with the exception that, if the original specifications contain contradicting features (such as $+N$ and $-N$), unification is undefined.

Metarules allow generalizations. In essence, they are rules that generate grammar rules. That is, instead of specifying all context-free rules directly, we only give some select ones, and use metarules to describe how to obtain related rules from them. This allows for a more economical and efficient description. Formally, a metarule is a function from grammar rules to grammar rules.

Ultimately, GPSGs have not been used in many practical applications, but their principles inspired some of the much more successful models, such as head-driven phrase structure grammars.

Head-driven Phrase Structure Grammar

Head driven phrase structure grammar (HPSG for short) is a generative grammar in the sense that it can generate strings and, consequently, languages, but it is unification-based rather than derivational. All HPSG components (i.e. grammar principles, grammar rules, and lexical entries) are formalized as typed feature structures.

HPSG is sometimes considered a direct successor to GPSG, but there is influence from other formalisms as well (such as lexical functional grammar). For example, HPSG categories are more complex than those in GPSG and HPSG makes more specific claims about universals and variation. HPSG is more suitable for computer implementation because of its emphasis on precise mathematical modeling of linguistic entities, uniform representation, and modularity, and as such it is often used in practice in natural language processing.

An important concept in HPSG is the *sign*, which is the type of feature structure that represents a constituent. It is a collection of information, including phonological, syntactic, and semantic constrains, and it is usually represented as an *attribute-value matrix* (AVM for short). AVMs encode feature structures where each attribute (feature) has a type is paired with a value. One of the common forms of AVM notation follows.

$$\begin{bmatrix} \text{type} & \\ \text{attribute} & \text{value} \\ \text{attribute} & \text{value} \\ \vdots & \end{bmatrix}$$

Signs receive either the subtype *word*, or *phrase*, depending on their phrasal status. These subtypes differ in that they conform to different constraints, but both contain attributes such as phonology (PHON) and syntax/semantics (SYNSEM). PHON has as its value a list of phonological descriptions. An example of an AVM (for the English word *walks*) is shown in Figure 3.4. Note that a value can be either atomic (such as singular for number), or complex (such as SYNSEM), in which case it can itself be represented by another (nested) AVM.

3.1.3 Automata as Natural Language Models

Besides formal grammars, automata have been also used in computational linguistics. In fact, augmented transition networks (see [70]) are among the oldest natural language models. The main advantage of automata is their closeness to practical implementation.

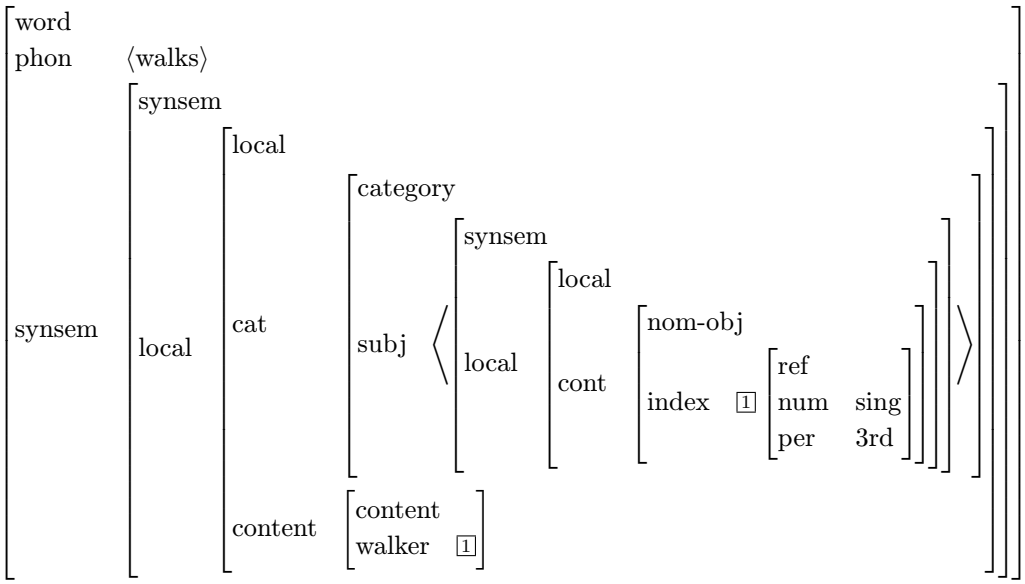


Figure 3.4: AVM example

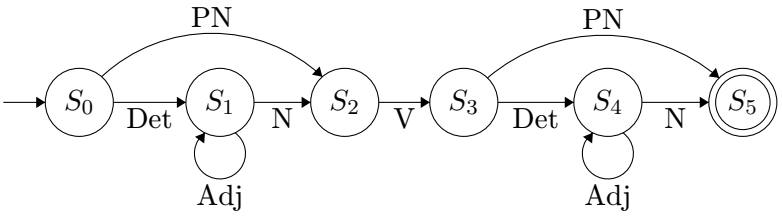


Figure 3.5: NDA for a simple fragment of English

Augmented Transition Network

Augmented transition networks have been created as a generalization and extension of nondeterministic finite state acceptors, which are in turn based on FAs.

Definition 3.1. A *nondeterministic finite state acceptor* (NDA for short) M is a FA $M = (Q, C, \delta, q_0, F)$ extended with an alphabet X, which is called the *input vocabulary*, and a function CAT from Σ to C such that $CAT(x)$ is the defined *category* of x. C is called the *category vocabulary*.

The input vocabulary X consists of words of a natural language, while the category vocabulary C contains grammatical categories of the language (such as NP for noun phrase, VP for verb phrase, or V for verb). The function CAT assigns to each word its appropriate category (for instance, in English, $CAT(dog) = \text{N}$). Transition rules are defined over categories (rather than words). An example of a NDA (for a simple fragment of English) given by its graphical representation is shown in Figure 3.5.

The NDA model in itself has been found unsatisfying because the transitions do not capture the effects of constituency. For instance, in Figure 3.5, we can intuitively see that the path from S_0 to S_2 represents a noun phrase, but there is no way to formally describe this in the NDA. NDA fails to capture an important generalization in the structure of a language.

To solve this problem, *recusive transition network* (RTN for short) has been introduced.

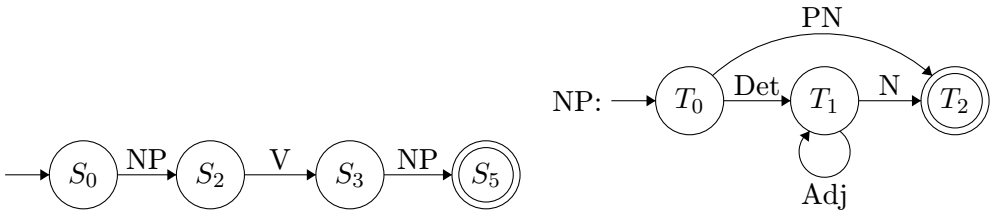


Figure 3.6: RTN example (restructuralizaton of NDA in Figure 3.5)

In RTNs, we allow state transitions to refer to nonterminals (phrase structures) as well as terminals. That is, we allow a transition between two consecutive states to be made by reading a string x instead of a single symbol (a single word). The transition can be made only if x belongs to the appropriate category (noun phrase in the above example), and this holds if x itself has been accepted by another RTN (for that category). As the RTN's name implies, this can be done recursively (e.g. for recognizing A, we call B, which in turns calls A again). An example of a RTN is given in Figure 3.6. Note that it is a restructuralization of the NDA in Figure 3.5.

RTNs allow generalizations, but they are still not without drawbacks. For example, there is no way to relate a *wh*-question with its indicative form. Moreover, some non-local dependencies (such as subject-verb agreement) are impossible to express. RTN has been further extended by adding *registers*, which may hold arbitrary information about input vocabulary (for example, whether a word is in singular or plural). The resulting model is called *augmented transition network* (ATN for short). It is known that ATNs are Turing-complete, that is, they can accept any recursively enumerable language (see [70]).

3.1.4 Statistical Natural Language Processing

With the increasing availability of large corpora (both annotated and unannotated) for various languages, corpus-based statistical approaches have become prominent in modern natural language processing. They abandon or at least limit the use of hand-crafted rules in favour of machine-learning techniques, which allow for automated extraction of generalized information from a training corpus. A comprehensive overview of statistical natural language processing can be found in [56].

In statistical natural language processing, we often use relatively simple language models, such as *n-gram models*. Essentially, n-grams are n-tuples of words that appear next to each other in text (preserving the order). In practical use, trigrams (triples of words) are perhaps the most common version.

Unlike grammars and automata, n-gram models do not create any structures over texts, sentences, or phrases, and this fact is both their main advantage and disadvantage. It allows for simple and very efficient implementation, which is an important factor for practical applications. On the other hand, it means that we can only work with local information. That is, we only know the immediate context of each word. We have no information about the syntactic structure of a phrase or sentence as a whole, and any potential long-distance dependencies are not captured. However, despite this limitation, n-gram models have been successfully applied in many areas of natural language processing.

Grammars and automata are also used in statistical natural language processing. In fact, many of the formal models discussed above can be—and indeed have been—adapted for statistical methods. Often, we can simply assign probabilities (or weights) to rules, as

it is done for example in probabilistic CFG (see [48] or [56]).

Probabilistic Context-Free Grammar

Informally, a probabilistic CFG (also known as stochastic CFG) is a CFG where a certain fixed probability is assigned to each rule. In this way, some derivations become more likely than other.

Definition 3.2 (Probabilistic context-free grammar)**.** A *probabilistic context-free grammar* (PCFG for short) G is a quintuple $G = (N, T, R, S, P)$, where

- $N = \{A_1, A_2 \ldots, A_n\}$ is a nonterminal alphabet,
- $T = \{w_1, w_2 \ldots, w_m\}$ is a terminal alphabet,
- $R = \{A_i \to \zeta_j \colon \zeta_j \in (N \cup T)^*, 1 \leq i \leq n\}$ is a set of rules,
- $S = A_1$ is the start symbol, and
- P is a corresponding set of *probabilities* on rules such that for all $1 \leq i \leq n$,

$$\sum_j P(A_i \to \zeta_j) = 1.$$

Further notions such as sentential form, sentence, derivation, and generated language are defined by analogy with CFGs (see Definition 2.24).

In a natural way, the probability of a derivation Δ in a PCFG G, denoted by $P(\Delta|G)$, is given by the product of probabilities of all applied rules. Subsequently, the sentence probability of a sentence s in G, denoted by $P(s|G)$, is the sum of probabilities of all possible derivations in G that result in s.

PCFGs are typically used to answer the following three questions:

1. What is the probability of a sentence s according to a PCFG G?

$$P(s|G) = ?$$

2. What is the most likely parse Δ of a sentence s according to a PCFG G?

$$\arg\max_{\Delta} P(\Delta|s, G) = ?$$

3. How do we set the rule probabilites of G to maximize the probability of a sentence s?

$$\arg\max_{G} P(s|G) = ?$$

Note that the last question is essential for machine learning. The goal is to set the rule probabilities in such a way that resulting the sentence probabilities reflect their frequency in the training corpus.

3.1.5 Regulated Rewriting as an Alternative

One of the major trends in formal language theory is regulated rewriting. The basic idea is to increase the generative power of a given formal model (usually a CFG) by adding some mathematically simple mechanism that controls (or, regulates) the sentence generation. Such models can be useful in NLP as well, because they allow us to capture even features of natural languages that are difficult or impossible to describe with only context-free rules.

In particular, our work is heavily inspired by [63], which proposes the use of SCGs and transformational SCGs in the description of the English language.

Transformational Scattered Context Grammar

Informally, a transformational SCG is a modification of SCG where derivations start with a string (taken from a given language) instead of a single start symbol.

Definition 3.3 (Transformational scattered context grammar)**.** A *transformational scattered context grammar* (TSCG for short) G is a quadruple $G = (N, T, P, I)$, where

- N is a *nonterminal* alphabet,
- T is a *terminal* alphabet called the *output vocabulary*, $N \cap T = \emptyset$,
- P is a finite set of *rules* of the form

$$(A_1, \ldots, A_n) \to (x_1, \ldots, x_n),$$

  where $n \geq 1$ and for all $1 \leq i \leq n$, $A_i \in N$, $x_i \in (N \cup T)^*$, and
- $I \subseteq N \cup T$ is the *input vocabulary*.

Further notions such as sentential form, sentence, and derivation are defined by analogy with SCGs (see Definition 2.31). The *transformation* that G defines from $K \subseteq I^*$, denoted by $T(G, K)$, is defined as $T(G, K) = \{(x, y) \colon x \Rightarrow_G^* y, x \in K, y \in T^*\}$.

To demonstrate scattered context in English syntax, several case studies are discussed in [63]. To illustrate, here we recall a simple example dealing with *neither-nor* clauses. Specifically, we want to negate the clause.

Neither Thomas nor his wife went to the party.
⇒ Both Thomas and his wife went to the party.

Example 3.1. Let T be the set of all English words (including all their inflectional forms). Let $G = (N, T, P, I)$ be a TSCG, where $N = I = \{\langle x \rangle \colon x \in T\}$ and $P = \{(\langle \text{neither} \rangle, \langle \text{nor} \rangle) \to (\langle \text{both} \rangle, \langle \text{and} \rangle)\} \cup \{(\langle x \rangle) \to (x) \colon x \in T - \{\text{neither}, \text{nor}\}\}$.

A derivation in G performing the negation of the example sentence may proceed as follows:

$\langle \text{neither} \rangle \langle \text{thomas} \rangle \langle \text{nor} \rangle \langle \text{his} \rangle \langle \text{wife} \rangle \langle \text{went} \rangle \langle \text{to} \rangle \langle \text{the} \rangle \langle \text{party} \rangle$
$\Rightarrow$ both $\langle \text{thomas} \rangle$ and $\langle \text{his} \rangle \langle \text{wife} \rangle \langle \text{went} \rangle \langle \text{to} \rangle \langle \text{the} \rangle \langle \text{party} \rangle$
$\Rightarrow$ both thomas and $\langle \text{his} \rangle \langle \text{wife} \rangle \langle \text{went} \rangle \langle \text{to} \rangle \langle \text{the} \rangle \langle \text{party} \rangle$
$\Rightarrow$ both thomas and his $\langle \text{wife} \rangle \langle \text{went} \rangle \langle \text{to} \rangle \langle \text{the} \rangle \langle \text{party} \rangle$
$\Rightarrow^5$ both thomas and his wife went to the party

Note that we assume here that the set of all English words, T, is finite and fixed. Such assumption is reasonable in practice, as we all commonly use a finite and fixed vocabulary in everyday English. However, in [63], the authors also rise an interesting point that, at least from a theoretical point of view, the set of all well-formed English words is infinite.

For example, consider the following sentences:

Your grandparents are all your grandfathers and all your grandmothers.
Your great-grandparents are all your great-grandfathers and all your great-grandmothers.
Your great-great-grandparents are all your great-great-grandfathers and all your great-great-grandmothers.
⋮

We can continue in this fashion indefinitely—even if the resulting sentences become less and less common—which gives us the infinite language $L = \{\text{your } \{\text{great-}\}^i\text{grandparents are all your } \{\text{great-}\}^i\text{grandfathers and all your } \{\text{great-}\}^i\text{grandmothers} : i \geq 0\}$. Note that L is not context-free.

Example 3.2. Let $G = (N, T, P, S)$ be a SCG, where $T = \{\text{all}, \text{and}, \text{are}, \text{grandfathers}, \text{grandmothers}, \text{grandparents}, \text{great-}, \text{your}\}$, $N = \{S, \#\}$, and P consists of the following three rules:

$$(S) \to (\text{your \#grandparents are all your \#grandfathers and all your \#grandmothers}),$$
$$(\#, \#, \#) \to (\text{\#great-,\#great-,\#great- }),$$
$$(\#, \#, \#) \to (\varepsilon, \varepsilon, \varepsilon)$$

An example of a derivation follows.

S ⇒ your #grandparents are all your #grandfathers
and all your #grandmothers
⇒ your #great-grandparents are all your #great-grandfathers
and all your #great-grandmothers
⇒ your great-grandparents are all your great-grandfathers
and all your great-grandmothers

3.2 Introduction to Machine Translation

Machine translation (see [51], [68], and [83]) is one the main parts of the motivation behind our work. In this section, we briefly review its history, core principles, and recent trends.

3.2.1 History and Classification

Machine translation is among the oldest tasks in the field of natural language processing. According to [68], the first documented idea of using computers for translation of natural languages is from 1947, by Warren Weaver. Subsequently, an extensive research in the area began.

The practical results, however, were generally disappointing, as the task was proving much more difficult than expected. In the 1960s, there still were only few working translation systems, and even in their case, the output quality was mostly unsatisfactory. This led to a decline in interest. The research was greatly slowed down, and it was only fully

restarted in the 1970s. There has been major development since then, with a number of practically applied systems.

Historically, machine translation systems can be divided into several categories based on various criteria. Depending on the number of languages, we distinguish between *bilingual* systems, which are designed specifically for a translation between a given pair of languages (for example Czech and English), and *multilingual* systems, which are able to process several different languages.

Furthermore, machine translation systems can be either *unidirectional* (performing translation in one direction only, e.g. from Czech to English, but not vice-versa), or *bidirectional* (able to translate in both directions, e.g. both from Czech to English and from English to Czech). Multilingual systems are usually bidirectional, while bilingual systems may often be unidirectional only.

There have been three classic approaches to the design of machine translation systems:

- *Direct translation* – this is the oldest approach. From the start, the system is designed for a given pair of languages only (therefore it is by definition always a bilingual system). The translation is realized directly from the source language to the target language. The syntax and semantic analysis of the source text is limited to the necessary minimum (for example to resolve ambiguities). These systems usually contain a large bilingual dictionary as their core and only a simple program for the analysis and generation of text.

- *Interlingual translation* – with this approach, we assume that it is possible to convert the source texts to some abstract lexical-semantic representation called interlingua, which is language-independent (or at least common for several different languages). The translation then consists of two main phases. First, we analyse the text in the source language and obtain its representation in interlingua. Subsequently, we can generate corresponding texts in various target languages from this representation.

- *Transfer* – this approach is similar to interlingual translation as it also uses an internal abstract representation. However, we use different representations for different languages. The translation is perfomed in three steps. As with interlingual translation, we first create an abstract representation of the source text. Then, we convert this internal representation into another abstract representation, which corresponds to the target language. From this representation we can generate the target text.

Direct translation is based on intuitive approach. Besides its simplicity, an important advatange of direct translation lies in the fact that since we only deal with two select languages, we can focus on their specific aspects and features. Unfortunately, this also means that direct translation systems are not general. That is, they cannot be easily modified for different languages than the ones that they are originally designed for.

In contrast, the main motivation behind interlingual translation is generality. Ideally, if the interlingua is indeed completely language-independent, adding a new language to the system would only require a specification of conversion between the language and the interlingua, without any need for changes in the rest of the system. In practice, however, there are various key differences between natural languages, which makes it very difficult to design an universal representation.

The last approach, transfer, is an effort to combine the advantages of both direct and interlingual translation. Because we have a different abstract representation for each language, we can study and describe the key features of each particular natural language

separately. On the other hand, the system as a whole is still general enough to allow for, for example, the addition of new languages.

3.2.2 Recent Trends

Following a similar general trend in natural language processing, statistical machine translation quickly came to the center of attention, with a shift from traditional the rule-based systems to corpus-based ones (as multilingual corpora also became more available). These systems mostly rely on formal models that represent local information only, such a n-gram models mentioned above.

However, in the recent years, there have been attempts to improve the translation quality by incorporating non-local, syntactic information (usually within the scope of a sentence). These approaches are generally called syntax-based, syntax-aided, or syntax augmented translation (see [50] or [86]). The formal background (namely synchronous tree substitution grammars) of one such syntax-based translation system (from the EuroMatrix project) is described in-depth in [6].

In order to use syntactic information, we need formal models that can capture it, such as grammars and automata. Furthermore, for translation, we also need to formally describe transformations of syntactic structures. In particular, our work is inspired by synchronous grammars, (see [13] or [15]).

Synchronous Grammars

Informally, a synchronous grammar is a grammar that generates pairs of sentences, and in this way, it can define translations. More specifically, there is a generalization of CFG called the synchronous context-free grammar (SCFG for short), where, in essence, every rule has two right-hand sides. The principles of synchronous grammars are discussed in more detail in Chapter 4.

Relatively recently, SCFGs have been successfully applied in statistical machine translation (see [12], [80], and [84]). There is also a modification called weighted SCFG (see [14]), where we assign fixed weights to rules (similarly to probabilities in PCFG).

Besides SCFG, a synchronous version of tree-adjoining grammar called synchronous tree-adjoining grammar has also been introduced and explored in the context of natural language processing (see [78]).

Part II

Synchronous Formal Systems

Chapter 4

Synchronous Systems Based on Grammars

In essence, synchronous grammars are grammars or grammar systems that generate pairs of sentences in one derivation, instead of single sentences (as for example in CFGs). In this way, they allow us to describe translations. That is, in each pair, the first string is a sentence of the source language, and the second string is a corresponding sentence of the target language.

Although the term synchronous context-free grammar (SCFG for short) is relatively recent, the essential principle was introduced already in the late 1960s in syntax-directed translation schemata [3] and syntax-directed transduction grammars [54]. These models were originally developed as formal background for compilers of programming languages. Subsequently, synchronous grammars have been succesfully used natural language processing as well, particularly in machine translation (for more details, see Section 3.2).

Informally, we can see SCFG (see [13] or [15]) as a modification of CFG where every rule has two right-hand sides, the first of which is applied to the input sentential form (source), and the second to the output sentential form (target). Nonterminals are linked, which means that in each derivation step, we rewrite both the selected nonterminal symbol in the input sentential form and its appropriate counterpart in the output sentential form.

Example 4.1. The following two rules are a fragment of a synchronous CFG which transforms arithmetic expressions from infix notation (e.g. $3 \times 5 + 4$) to postfix notation (e.g. $3\,5 \times 4\,+$). E, F, and T are nonterminals, $+$ and $\times$ are terminals, E is the start symbol.

$$1 : \ \mathrm{E} \rightarrow \mathrm{E}_{\boxed{1}} + \mathrm{T}_{\boxed{2}},\ \mathrm{E}_{\boxed{1}}\ \mathrm{T}_{\boxed{2}}\ +$$
$$2 : \ \mathrm{T} \rightarrow \mathrm{T}_{\boxed{1}} \times \mathrm{F}_{\boxed{2}},\ \mathrm{T}_{\boxed{1}}\ \mathrm{F}_{\boxed{2}}\ \times$$

A derivation using these rules may look like this:

$$\begin{aligned}
(\mathrm{E}_{\boxed{42}},\ \mathrm{E}_{\boxed{42}}) &\Rightarrow (\mathrm{E}_{\boxed{43}} + \mathrm{T}_{\boxed{44}},\ \mathrm{E}_{\boxed{43}}\ \mathrm{T}_{\boxed{44}}\ +)\ [1] \\
&\Rightarrow (\mathrm{E}_{\boxed{45}} + \mathrm{T}_{\boxed{46}} + \mathrm{T}_{\boxed{47}},\ \mathrm{E}_{\boxed{45}}\ \mathrm{T}_{\boxed{46}} + \mathrm{T}_{\boxed{47}}\ +)\ [1] \\
&\Rightarrow (\mathrm{E}_{\boxed{45}} + \mathrm{T}_{\boxed{48}} \times \mathrm{F}_{\boxed{49}} + \mathrm{T}_{\boxed{47}},\ \mathrm{E}_{\boxed{45}}\ \mathrm{T}_{\boxed{48}}\ \mathrm{F}_{\boxed{49}}\ \times + \mathrm{T}_{\boxed{47}}\ +)\ [2]
\end{aligned}$$

The boxed numbers are used to denote linked nonterminals. That is, two nonterminals are linked if they have the same number (e.g. $\boxed{1}$ and $\boxed{1}$). Every derivation in a SCFG starts with a pair of linked nonterminals, such as $(\mathrm{E}_{\boxed{42}}, \mathrm{E}_{\boxed{42}})$ here (the starting number 42 is chosen arbitrarily). Whenever we make a derivation step, we assign new, unique numbers to each newly introduced pair of linked nonterminals, as seen in the derivation example above.

In each derivation step, we can only rewrite linked nonterminals (nonterminals sharing the same boxed number). Note that when applying rule 2 above, we rewrite the first occurence of T in both sentential forms, which is allowed, as it is $\mathrm{T}_{\boxed{46}}$ in both cases. We could also choose to rewrite the second occurence in both sentential forms ($\mathrm{T}_{\boxed{47}}$). However, we cannot choose the first T in one sentential form and the second T in the other, because the assigned numbers do not correspond ($\mathrm{T}_{\boxed{46}}$ and $\mathrm{T}_{\boxed{47}}$), and thus the two nonterminals are not considered linked.

The original ideas, concepts, definitions, and theoretical results presented in this chapter were first published in [40] and [41].

4.1 Rule-Synchronized Context-Free Grammar

In [40] and [41], we have proposed synchronization based on linked rules instead of nonterminals. Informally, such synchronous grammar is a system of two grammars, G_I and G_O, in which the corresponding rules share labels. For example, if we apply rule labelled 1 in the input grammar G_I, we also have to apply rule labelled 1 in the output grammar G_O, and this makes for a single derivation step in the synchronous grammar. In other words, the input and output sentence have the same parse (a sequence of rules applied in a derivation, denoted by their labels).

Example 4.2. Rules (G_I on the left, G_O on the right):

$$\begin{array}{ll} 1: \ \mathrm{E} \to \mathrm{E} + \mathrm{T} & 1: \ \mathrm{E} \to \mathrm{E}\ \mathrm{T} + \\ 2: \ \mathrm{T} \to \mathrm{T} \times \mathrm{F} & 2: \ \mathrm{T} \to \mathrm{T}\ \mathrm{F} \times \end{array}$$

An example of a derivation using these rules in G_I follows.

$$\mathrm{E} \Rightarrow \mathrm{E} + \mathrm{T}\ [1] \Rightarrow \mathrm{E} + \mathrm{T} \times \mathrm{F}\ [2]$$

A corresponding derivation in G_O is:

$$\mathrm{E} \Rightarrow \mathrm{E}\ \mathrm{T} +\ [1] \Rightarrow \mathrm{E}\ \mathrm{T}\ \mathrm{F} \times +\ [2]$$

The parse is $(1, 2)$.

However, note that we place no restriction on the linked rules. For instance, unlike in synchronous CFGs, we do not have to rewrite the same nonterminal in both sentential forms in one derivation step. Both the right-hand sides and the left-hand sides of linked rules may be completely different, for example:

$$3: \ \mathrm{A} \to \mathrm{B}\ \mathrm{a}\ \mathrm{C} \qquad 3: \ \mathrm{P} \to \mathrm{Q}\ \mathrm{B}\ \mathrm{R}\ \mathrm{b}\ \mathrm{d}$$

In other words, rule-synchronized CFGs can be seen as a generalization of the traditional synchronous CFGs, as the latter can be defined as special case of rule-synchronized CFGs, where each two linked rules have the same left-hand side (that is, they rewrite the same nonterminal).

Formally, we define a rule-synchronized CFG as follows.

Definition 4.1 (Rule-synchronized CFG)**.** A *rule-synchronized CFG* (RSCFG for short) H is a quintuple $H = (G_I, G_O, \Psi, \varphi_I, \varphi_O)$, where

- $G_I = (N_I, T_I, P_I, S_I)$ and $G_O = (N_O, T_O, P_O, S_O)$ are CFGs,
- Ψ is a set of *rule labels*, and
- φ_I is a function from Ψ to P_I and φ_O is a function from Ψ to P_O.

We say that two rules $p_I \in P_I$ and $p_O \in P_O$ are linked, if and only if there is some label $p \in \Psi$ such that $\varphi_I(p) = p_I$ and $\varphi_O(p) = p_O$. That is, each two linked rules share the same label.

We use the following notation (presented for input grammar G_I, analogous for output grammar G_O):

$p : A_I \to x_I$ where $p \in \Psi, A_I \to x_I \in P_I$	$\varphi_I(p) = A_I \to x_I$
$x_I \Rightarrow_{G_I} y_I\,[p]$ where $x_I, y_I \in (N \cup T)^*, p \in \Psi$	derivation step in G_I applying rule $\varphi_I(p)$
$x_I \Rightarrow^n_{G_I} y_I\,[p_1 \dots p_n]$ where $x_I, y_I \in (N \cup T)^*, p_i \in \Psi$ for $1 \leq i \leq n$	derivation in G_I applying rules $\varphi_I(p_1) \dots \varphi_I(p_n)$

Definition 4.2 (Translation in RSCFG)**.** Let $H = (G_I, G_O, \Psi, \varphi_I, \varphi_O)$ be an RSCFG. The *translation defined by* H, denoted by $T(H)$, is the set of pairs of sentences, which is defined as

$$T(H) = \{(w_I, w_O) \quad : \quad w_I \in T_I^*, w_O \in T_O^*, \\ S_I \Rightarrow^*_{G_I} w_I\,[\alpha], S_O \Rightarrow^*_{G_O} w_O\,[\alpha] \text{ for some } \alpha \in \Psi^*\}.$$

Originally [40], we considered RSCFG only as a variant of synchronous CFG. However, there is in fact a significant difference. While the latter does not increase the generative power over CFG, RSCFG does, as is shown in the next subsection.

4.1.1 Generative Power

Synchronous grammars define translations—that is, sets of pairs of sentences. To be able to compare their generative power with well-known models such as CFGs, which define languages, we can consider their input and output language separately.

Definition 4.3 (Input and output language)**.** Let H be an RSCFG. Then, we define

- the *input language* of H, denoted by $L_I(H)$, as $L_I(H) = \{w_I \colon (w_I, w_O) \in T(H)\}$, and
- the *output language* of H, denoted by $L_O(H)$, as $L_O(H) = \{w_O \colon (w_I, w_O) \in T(H)\}$.

Example 4.3. Consider an RSCFG $H = (G_I, G_O, \Psi, \varphi_I, \varphi_O)$ with the following rules (nonterminals are in capitals, linked rules share the same label, S_I and S_O are the start symbols of G_I and G_O, respectively):

G_I				G_O			
1 :	S_I	$\to$	ABC	1 :	S_O	$\to$	A
2 :	A	$\to$	aA	2 :	A	$\to$	B
3 :	B	$\to$	bB	3 :	B	$\to$	C
4 :	C	$\to$	cC	4 :	C	$\to$	A
5 :	A	$\to$	ε	5 :	A	$\to$	B'
6 :	B	$\to$	ε	6 :	B'	$\to$	C'
7 :	C	$\to$	ε	7 :	C'	$\to$	ε

An example of a derivation follows.

$$
\begin{array}{lcll}
S_I & \Rightarrow & ABC & [1] \\
 & \Rightarrow & aABC & [2] \\
 & \Rightarrow & aAbBC & [3] \\
 & \Rightarrow & aAbBcC & [4] \\
 & \Rightarrow & aaAbBcC & [2] \\
 & \Rightarrow & aaAbbBcC & [3] \\
 & \Rightarrow & aaAbbBccC & [4] \\
 & \Rightarrow & aabbBccC & [5] \\
 & \Rightarrow & aabbccC & [6] \\
 & \Rightarrow & aabbcc & [7]
\end{array}
\qquad
\begin{array}{lcll}
S_O & \Rightarrow & A & [1] \\
 & \Rightarrow & B & [2] \\
 & \Rightarrow & C & [3] \\
 & \Rightarrow & A & [4] \\
 & \Rightarrow & B & [2] \\
 & \Rightarrow & C & [3] \\
 & \Rightarrow & A & [4] \\
 & \Rightarrow & B' & [5] \\
 & \Rightarrow & C' & [6] \\
 & \Rightarrow & \varepsilon & [7]
\end{array}
$$

We can easily see that $L_I(H) = \{a^n b^n c^n \colon n \geq 0\}$, which is well known not to be a context-free language. This shows that RSCFGs are stronger than (synchronous) CFGs.[1] Where exactly do synchronous grammars with linked rules stand in terms of generative power?

Let $\mathscr{L}$(RSCFG) denote the class of languages generated by RSCFGs as their input language. Note that the results presented below would be the same if we considered the output language instead.

In some of the proofs below, we use a function that removes all terminals from a sentential form, formally defined as follows.

Definition 4.4. Let $G = (N, T, P, S)$ be a CFG. Then, we define the function θ over $(N \cup T)^*$ as follows:

1. For all $w \in T^*$, $\theta(w) = \varepsilon$.
2. For all $w = x_0 A_1 x_2 A_2 \dots x_{n-1} A_n x_n$ for some $n \geq 1$, where $x_i \in T^*$ for all $0 \leq i \leq n$ and $A_j \in N$ for all $1 \leq j \leq n$, $\theta(w) = A_1 A_2 \dots A_n$.

The idea here is that if we consider only context-free rules, the applicability of rules to a given sentential form only depends on nonterminals. Therefore, we can remove terminals without affecting computational control.

For every RSCFG, we can construct an equivalent MAT, using matrices to simulate the principle of linked rules.

Lemma 4.1. *For every RSCFG H, there is a MAT H' such that $L(H') = L_I(H)$.*

Proof. Let $H = (G_I, G_O, \Psi, \varphi_I, \varphi_O)$ be an RSCFG, where $G_I = (N_I, T_I, P_I, S_I)$, $G_O = (N_O, T_O, P_O, S_O)$. Without loss of generality, assume $N_I \cap N_O = \emptyset$, $S \notin N_I \cup N_O$. Construct a MAT $H' = (G, M)$, where $G = (N, T, P, S)$, as follows:

1. Set $N = N_I \cup N_O \cup \{S\}$, $T = T_I$, $P = \{S \to S_I S_O\}$, $M = \{S \to S_I S_O\}$.
2. For every label $p \in \Psi$, add rules p_I, p_O to P and add matrix $p_I p_O$ to M, where
   - $p_I = \varphi_I(p)$ and

[1] Strictly speaking, to make this claim, we also have to show that every context-free language can be generated by a RSCFG. That is however evident from the definition.

- $p_O = A \to x$ such that $\varphi_O(p) = A \to x'$, $x = \theta(x')$.[2]

Basic idea. H' simulates the principle of linked rules in H by matrices. That is, for every pair of rules $(A_I \to x_I, A_O \to x_O)$ such that $\varphi_I(p) = A_I \to x_I, \varphi_O(p) = A_O \to x_O$ for some $p \in \Psi$ in H, there is a matrix $m = A_I \to x_I A_O \to \theta(x_O)$ in H'. If, in H, $x_I \Rightarrow y_I\,[p]$ in G_I and $x_O \Rightarrow y_O\,[p]$ in G_O, then there is a derivation step $x_I\,\theta(x_O) \Rightarrow y_I\,\theta(y_O)\,[m]$ in H'. Note that since the rules are context-free, the presence (or absence) of terminals in a sentential form does not affect which rules we can apply. Furthermore, because the nonterminal sets N_I and N_O are disjoint, the sentential form in H' always consists of two distinct parts such that the first part corresponds to the derivation in G_I and the second part to the derivation in G_O.

Claim 4.2. *If $S_I \Rightarrow^* w_I\,[\alpha]$, $S_O \Rightarrow^* w_O\,[\alpha]$ in H for some $\alpha \in \Psi^*$, then $S \Rightarrow^* w_I\,\theta(w_O)$ in H'.*

Proof of Claim 4.2. By induction on the number of derivation steps in H.

Basis. Let $S_I \Rightarrow^0 S_I\,[\varepsilon]$, $S_O \Rightarrow^0 S_O\,[\varepsilon]$ in H. Then, $S \Rightarrow S_I S_O\,[p]$ in G, $p \in M$, and thus $S \Rightarrow S_I S_O\,[p]$ in H'. Claim 4.2 holds for zero derivation steps in H.

Induction hypotesis. Suppose that Claim 4.2 holds for j or fewer derivation steps in H.

Induction step. Let $S_I \Rightarrow^j w_I\,[\alpha] \Rightarrow w'_I\,[p]$, $S_O \Rightarrow^j w_O\,[\alpha] \Rightarrow w'_O\,[p]$ in H. Then, by the induction hypotesis, $S \Rightarrow^* w_I\,\theta(w_O)$ in H'. Without loss of generality, suppose that

- $w_I = u_I A_I v_I$, $w_O = u_O A_O v_O$,
- $w'_I = u_I x_I v_I$, $w'_O = u_O x_O v_O$,

where $A_I \in N_I$, $A_O \in N_O$, $u_I, v_I, x_I \in (N_I \cup T_I)^*$, and $u_O, v_O, x_O \in (N_O \cup T_O)^*$. That is, $\varphi_I(p) = A_I \to x_I$, $\varphi_O(p) = A_O \to x_O$. From the construction of H', we know that $p_I = A_I \to x_I \in P$, $p_O = A_O \to \theta(x_O) \in P$, and $p_I p_O \in M$. Therefore, in H', $S \Rightarrow^* w_I\,\theta(w_O) = u_I A_I v_I\,\theta(u_O A_O v_O) \Rightarrow u_I x_I v_I\,\theta(u_O x_O v_O)\,[p_I p_O] = w_I\,\theta(w_O)$. Claim 4.2 holds. Furthermore, if $S_I \Rightarrow^* w_I\,[\alpha]$, $S_O \Rightarrow^* w_O\,[\alpha]$ in H, where $w_I \in T_I^*$, $w_O \in T_O^*$, then $\theta(w_O) = \varepsilon$, and thus $S \Rightarrow^* w_I$ in H'. $L_I(H) \subseteq L(H')$. □

Claim 4.3. *If $S \Rightarrow^* w$ in H', then there are strings w_I, w_O such that $w = w_I\,\theta(w_O)$ and $S_I \Rightarrow^* w_I\,[\alpha]$, $S_O \Rightarrow^* w_O\,[\alpha]$ in H for some $\alpha \in \Psi^*$.*

Proof of Claim 4.3. By induction on the number of derivation steps in H'.

Basis. Consider the first derivation step in H'. Because $S \to S_I S_O$ is the only rule in P with S as its left-hand side, this must be $S \Rightarrow S_I S_O$. Then, $S_I \Rightarrow^0 S_I\,[\varepsilon]$, $S_O \Rightarrow^0 S_O\,[\varepsilon]$ in H. Claim 4.3 holds for one derivation step in H'.

Induction hypotesis. Suppose that Claim 4.3 holds for j or fewer derivation steps in H'.

Induction step. Let $S \Rightarrow S_I S_O \Rightarrow^{j-1} w \Rightarrow w'\,[m]$ in H'. Then, by the induction hypotesis, $S_I \Rightarrow^* w_I\,[\alpha]$, $S_O \Rightarrow^* w_O\,[\alpha]$ in H for some w_I, w_O such that $w = w_I\,\theta(w_O)$. From the construction of H', we know that $m = p_I p_O$, where for some $p \in \Psi$, $p_I = A_I \to x_I = \varphi_I(p)$ and $p_O = A_O \to \theta(x_O)$ such that $A_O \to x_O = \varphi_O(p)$. Therefore, there must be a factorization of w and w' such that

- $w = w_I\,\theta(w_O) = u_I A_I v_I\,\theta(u_O A_O v_O)$,

[2] This removes all terminals from the right-hand side of the rule. Note that if we leave the rule unchanged, we obtain the concatenation of the input and the output sentence. Further, if we want $L(H') = L_O(H)$ instead of $L(H') = L_I(H)$, we can simply modify p_I instead of p_O in this step.

- $w' = w'_I\,\theta(w'_O) = u_I x_I v_I\,\theta(u_O x_O v_O)$,

where $A_I \in N_I$, $A_O \in N_O$, $u_I, v_I, x_I \in (N_I \cup T_I)^*$, and $u_O, v_O, x_O \in (N_O \cup T_O)^*$. Therefore, in H,

- $S_I \Rightarrow^* w_I\,[\alpha] = u_I A_I v_I \Rightarrow u_I x_I v_I\,[p] = w'_I$ and
- $S_O \Rightarrow^* w_O\,[\alpha] = u_O A_O v_O \Rightarrow u_O x_O v_O\,[p] = w'_O$.

Claim 4.3 holds. Furthermore, if $S \Rightarrow^* w$ in H', where $w \in T^*$, then $w_I \in T_I^*$ and $\theta(w_O) = \varepsilon$, thus $w_O \in T_O^*$. $L(H') \subseteq L_I(H)$. □

By Claim 4.2, $L_I(H) \subseteq L(H')$, and by Claim 4.3, $L(H') \subseteq L_I(H)$, therefore $L(H') = L_I(H)$. Lemma 4.5 holds. □

On the other hand, for every MAT, we can construct an equivalent RSCFG. We take advantage of that fact that there is an "additonal" CFG in an RSCFG, and use it to simulate matrices.

Lemma 4.4. *For every MAT H, there is a RSCFG H' such that $L_I(H') = L(H)$.*

Proof. Let $H = (G, M)$ be a MAT, where $G = (N, T, P, S)$. Without loss of generality, assume $N \cap \{S_I, S_O, X\} = \emptyset$. Construct an RSCFG $H' = (G_I, G_O, \Psi, \varphi_I, \varphi_O)$, where $G_I = (N_I, T_I, P_I, S_I)$, $G_O = (N_O, T_O, P_O, S_O)$, as follows:

1. Set $N_I = N \cup \{S_I, X\}$, $T_I = T$, $P_I = \{S_I \to SX, X \to \varepsilon\}$, $N_O = \{S_O, X\}$, $T_O = \{\#\}$, $P_O = \{S_O \to X, X \to \#\}$, $\varphi_I = \emptyset$, $\varphi_O = \emptyset$.
2. Set $\Psi = \{0, 1\}$, $\varphi_I(0) = S_I \to SX$, $\varphi_O(0) = S_O \to X$, $\varphi_I(1) = X \to \varepsilon$, $\varphi_O(1) = X \to \#$.
3. For every matrix $m = p \in M$, where $p \in P$,
   (a) add rule p to P_I,
   (b) add rule $X \to X$ to P_O,
   (c) add new label $\langle m \rangle$ to Ψ, and
   (d) set $\varphi_I(\langle m \rangle) = p$, $\varphi_O(\langle m \rangle) = X \to X$.
4. For every matrix $m = p_1 \dots p_n \in M$, where $n > 1$ and $p_i \in P$ for all $1 \le i \le n$,
   (a) add rules $p_1, \dots, p_n$ to P_I,
   (b) add new nonterminals $\langle Xm \rangle_1, \dots, \langle Xm \rangle_{n-1}$ to N_O,
   (c) add rules $X \to \langle Xm \rangle_1$, $\langle Xm \rangle_1 \to \langle Xm \rangle_2$, $\dots$, $\langle Xm \rangle_{n-2} \to \langle Xm \rangle_{n-1}$, $\langle Xm \rangle_{n-1} \to X$ to P_O,
   (d) add new labels $\langle m \rangle_1, \dots, \langle m \rangle_n$ to Ψ, and
   (e) set φ_I and φ_O as follows:
   - $\varphi_I(\langle m \rangle_1) = p_1$, $\varphi_O(\langle m \rangle_1) = X \to \langle Xm \rangle_1$,
   - $\varphi_I(\langle m \rangle_i) = p_i$, $\varphi_O(\langle m \rangle_i) = \langle Xm \rangle_{i-1} \to \langle Xm \rangle_i$ for all $1 < i < n$, and
   - $\varphi_I(\langle m \rangle_n) = p_n$, $\varphi_O(\langle m \rangle_n) = \langle Xm \rangle_{n-1} \to X$.

Basic idea. One may notice that G_I constructed by the above algorithm is nearly identical to the original CFG G in H. Indeed, it performs essentially the same role: generating a sentence. Meanwhile, G_O restricts available derivations according to matrices from H. Each nonterminal in G_O represents a certain state of the system. For example, suppose that we have the nonterminal $\langle Xm\rangle_2$ as the current sentential form in G_O. This means that we are currently simulating the matrix m, we have succesfully applied the second rule of this matrix, and now we need to apply its next rule. The nonterminal X is a special case. It represents the state where we can either choose a new matrix to simulate, or end the derivation. It appears at the start of a derivation (along with the original start symbol from H, S) and can only appear again immediately after a successful simulation of a whole matrix (one derivation step in H).

In other words, H' simulates matrices in H by derivation in G_O. That is, if $x \Rightarrow y\,[m]$ in H, where $m = p_1 \dots p_n$ for some $n \geq 1$, then there is a sequence of derivation steps $X \Rightarrow \langle Xm\rangle_1\,[\langle m\rangle_1] \Rightarrow \langle Xm\rangle_2\,[\langle m\rangle_2] \Rightarrow \dots \Rightarrow \langle Xm\rangle_{n-2}\,[\langle m\rangle_{n-2}] \Rightarrow \langle Xm\rangle_{n-1}\,[\langle m\rangle_{n-1}] \Rightarrow X\,[\langle m\rangle_n]$ in G_O and $\varphi_I(\langle m\rangle_i) = p_i$ for $1 \leq i \leq n$. Now observe that in G_O constructed by the above algorithm, every nonterminal except X can only appear as the left-hand side of no more than one rule. This means that after rewriting X to $\langle Xm\rangle_1$, the only way for the derivation to proceed is the above sequence, and the entire matrix is simulated. Note that for matrices that only have one rule (that is, if $n = 1$), $X \Rightarrow X$ in G_O by using rule $X \to X$, and we can immediately continue with another matrix. The simulation ends by rewriting X to the only terminal $\#$ in G_O and, simultaneously, deleting X in G_I (using rule $X \to \varepsilon$). This ensures that the derivation in G_I cannot end by producing a sentence prematurely—that is, when the simulation of a matrix is incomplete—because there will always be at least one nonterminal left at that point (precisely X).

Formally, if $x \Rightarrow y\,[m]$ in H, where $m = p_1 \dots p_n$ for some n, then, in H', $x \Rightarrow^n y\,[m_1 \dots m_n]$ in G_I and $X \Rightarrow^n X\,[m_1 \dots m_n]$ in G_O. Conversely, if, in H', $x \Rightarrow^n y\,[\alpha]$ in G_I and $X \Rightarrow^n X\,[\alpha]$ in G_O for some $n \geq 1$, $\alpha \in \Psi^*$, and there is no $k < n$ such that $x \Rightarrow^k z\,[\beta] \Rightarrow^* y\,[\gamma]$ in G_I and $X \Rightarrow^k X\,[\beta] \Rightarrow^* X\,[\gamma]$ in G_O for some $\beta, \gamma \in \Psi^*$, there has to be $m \in M$ such that $x \Rightarrow y\,[m]$ in H. Therefore, if for some $\alpha \in \Psi^*$, $S_I \Rightarrow^* wX\,[\alpha] \Rightarrow w\,[1]$, $S_O \Rightarrow^* X\,[\alpha] \Rightarrow \varepsilon\,[1]$ in H', where $w \in T_I^*$, then $S \Rightarrow^* w$ in H. On the other hand, if $S \Rightarrow^* w$ in H, where $w \in T^*$, then $\alpha \in \Psi^*$, $S_I \Rightarrow^* wX\,[\alpha] \Rightarrow w\,[1]$, $S_O \Rightarrow^* X\,[\alpha] \Rightarrow f\,[1]$ in H' for some $\alpha \in \Psi^*$. Thus, $L_I(H') = L(H)$, and Lemma 4.4 holds. □

Note that G_O constructed by the above algorithm is not only context-free, but also regular.

From Lemma 4.1 and Lemma 4.4, we can establish the following theorem.

Theorem 4.5.

$$\mathscr{L}(\text{RSCFG}) = \mathscr{L}(\text{MAT})$$

Proof. From Lemma 4.1, it follows that $\mathscr{L}(\text{RSCFG}) \subseteq \mathscr{L}(\text{MAT})$. From Lemma 4.4, it follows that $\mathscr{L}(\text{MAT}) \subseteq \mathscr{L}(\text{RSCFG})$. Therefore, $\mathscr{L}(\text{RSCFG}) = \mathscr{L}(\text{MAT})$. □

4.2 Synchronous Scattered Context Grammar

The principle of synchronization based on linked rules can be naturally extended to other models beside CFGs. Indeed, the definition of synchronous SCG is analogous to Definition 4.1 for RSCFG. Essentially, we only need to replace context-free rules with scattered context rules. The notation is also analogous.

Definition 4.5 (Synchronous SCG)**.** A *synchronous SCG* (SSCG for short) H is a quintuple $H = (G_I, G_O, \Psi, \varphi_I, \varphi_O)$, where

- $G_I = (N_I, T_I, P_I, S_I)$ and $G_O = (N_O, T_O, P_O, S_O)$ are SCGs,
- Ψ is a set of *rule labels*, and
- φ_I is a function from Ψ to P_I and φ_O is a function from Ψ to P_O.

Further, the *translation defined by* H, denoted by $T(H)$, is the set of pairs of sentences, which is defined as

$$T(H) = \{(w_I, w_O) \quad : \quad w_I \in T_I^*, w_O \in T_O^*, \\ S_I \Rightarrow_{G_I}^* w_I\,[\alpha], S_O \Rightarrow_{G_O}^* w_O\,[\alpha] \text{ for some } \alpha \in \Psi^*\}.$$

We define the input and output language of SSCG by analogy with Definition 4.3 for RSCFGs. Further, let $\mathscr{L}(\text{SSCG})$ denote the class of all languages generated by SSCGs as their input language.

4.2.1 Generative Power

It is known that SCGs can generate all recursively enumerable languages (see [59]). Perhaps not surprisingly, the same is true for SSCGs. From their definition, it is easy to see that SSCGs cannot be weaker than SCGs. If we want to construct an SSCG equivalent to a given SCG, we can, for instance, essentially duplicate the original SCG and designate each two identical rules from input and output grammar as linked.

Theorem 4.6.

$$\mathscr{L}(\text{SSCG}) = \mathbf{RE}$$

Proof. Clearly, $\mathscr{L}(\text{SSCG}) \subseteq \mathbf{RE}$ must hold. From definition, it follows that $\mathscr{L}(\text{SCG}) \subseteq \mathscr{L}(\text{SSCG})$. Because $\mathscr{L}(\text{SCG}) = \mathbf{RE}$ [59], $\mathbf{RE} \subseteq \mathscr{L}(\text{SSCG})$ also holds. □

4.3 Synchronous Matrix Grammar

In the case of matrix grammars, the situation is slightly more complicated. How should we link the rules with regard to matrices? There are many options. For instance, we could strictly require that all rules in one matrix in the input grammar be linked to rules in one matrix in the output grammar, in respective order (consequently, requiring each two matrices that have their rules linked to have the same length). Alternatively, we could link only the first rule in each matrix. However, perhaps the most straightforward and intuitive approach is to link whole matrices rather than individual rules.

The notation used here is analogous to the one presented in Section 4.1 for RSCFGs, only replacing rules by matrices.

Definition 4.6 (Synchronous matrix grammar)**.** A *synchronous matrix grammar* (SMAT for short) H is a septuple $H = (G_I, M_I, G_O, M_O, \Psi, \varphi_I, \varphi_O)$, where

- (G_I, M_I) and (G_O, M_O) are MATs, where
  - $G_I = (N_I, T_I, P_I, S_I)$ and

– $G_O = (N_O, T_O, P_O, S_O)$,

- Ψ is a set of *matrix labels*, and
- φ_I is a function from Ψ to M_I and φ_O is a function from Ψ to M_O.

Further, the *translation defined by* H, denoted by $T(H)$, is the set of pairs of sentences, which is defined as

$$T(H) = \{(w_I, w_O) \quad : \quad w_I \in T_I^*, w_O \in T_O^*, \\ S_I \Rightarrow^*_{(G_I, M_I)} w_I\,[\alpha], S_O \Rightarrow^*_{(G_O, M_O)} w_O\,[\alpha] \text{ for some } \alpha \in \Psi^*\}.$$

We define the input and output language of SMAT by analogy with Definition 4.3 for RSCFGs. Further, let $\mathscr{L}(\text{SMAT})$ denote the class of all languages generated by SMATs as their input language.

4.3.1 Generative Power

Following a similar reasoning as in the case of SSCGs, we can immediately conclude that SMATs must be at least as powerful as MATs. To elaborate, to construct an SMAT equivalent to a given MAT, we can, as with SSCGs, let both input and output grammar equal the original grammar and designate the identical matrices in input and output grammar as linked.

The fact that we can also construct an equivalent MAT for every SMAT is much less immediately obvious. In essence, we can join each two linked matrices (from input and output grammar) into one matrix.

Theorem 4.7.

$$\mathscr{L}(\text{SMAT}) = \mathscr{L}(\text{MAT})$$

Proof. The inclusion $\mathscr{L}(\text{MAT}) \subseteq \mathscr{L}(\text{SMAT})$ follows from definition. It only remains to prove that $\mathscr{L}(\text{SMAT}) \subseteq \mathscr{L}(\text{MAT})$. For every SMAT $H = (G_I, M_I, G_O, M_O, \Psi, \varphi_I, \varphi_O)$, where $G_I = (N_I, T_I, P_I, S_I)$, $G_O = (N_O, T_O, P_O, S_O)$, we can construct a MAT $H' = (G, M)$, where $G = (N, T, P, S)$, such that $L(H') = L_I(H)$, as follows. Without loss of generality, assume $N_I \cap N_O = \emptyset$, $S \notin N_I \cup N_O$.

1. Set $N = N_I \cup N_O \cup \{S\}$, $T = T_I$, $P = \{S \to S_I S_O\}$, $M = \{S \to S_I S_O\}$.
2. For every label $p \in \Psi$, add rules $p_{I1}, \ldots, p_{In}$, $p_{O1}, \ldots, p_{Om}$ to P and add matrix $p_{I1} \ldots p_{In} p_{O1} \ldots p_{Om}$ to M, where
   - $p_{I1} \ldots p_{In} = \varphi_I(p)$ and
   - for $1 \leq j \leq m$, $p_{Oj} = A_j \to x_j$ such that $\varphi_O(p)[j] = A_j \to x'_j$, $x_j = \theta(x'_j)$.[3]

Basic idea. H' simulates H by combining the rules of each two linked matrices in H into a single matrix in H'. That is, for every pair of matrices (m_I, m_O) such that $m_I = \varphi_I(p), m_O = \varphi_O(p)$ for some $p \in \Psi$ in H, there is a matrix $m = m_I m'_O$ in H', where m'_O is equal to m_O with all terminals removed (formally defined above). If, in H, $x_I \Rightarrow y_I\,[p]$ in G_I and $x_O \Rightarrow y_O\,[p]$ in G_O, then there is a derivation step $x_I\,\theta(x_O) \Rightarrow y_I\,\theta(y_O)\,[m]$ in

[3] Again, this removes all terminals from the right-hand side of the rules (see Theorem 4.5). $m[j]$ denotes the j-th rule in matrix m.

H'. Note that since the rules are context-free, the presence (or absence) of terminals in a sentential form does not affect which rules we can apply. Furthermore, because the nonterminal sets N_I and N_O are disjoint, the sentential form in H' always consists of two distinct parts such that the first part corresponds to the derivation in G_I and the second part to the derivation in G_O.

Claim 4.8. *If $S_I \Rightarrow^* w_I\,[\alpha]$, $S_O \Rightarrow^* w_O\,[\alpha]$ in H for some $\alpha \in \Psi^*$, then $S \Rightarrow^* w_I\,\theta(w_O)$ in H'.*

Proof of Claim 4.8. By induction on the number of derivation steps in H.

Basis. Let $S_I \Rightarrow^0 S_I\,[\varepsilon]$, $S_O \Rightarrow^0 S_O\,[\varepsilon]$ in H. Then, $S \Rightarrow S_I S_O\,[p]$ in G, $p \in M$, and thus $S \Rightarrow S_I S_O\,[p]$ in H'. Claim 4.8 holds for zero derivation steps in H.

Induction hypotesis. Suppose that Claim 4.8 holds for j or fewer derivation steps in H.

Induction step. Let $S_I \Rightarrow^j w_I\,[\alpha] \Rightarrow w'_I\,[p]$, $S_O \Rightarrow^j w_O\,[\alpha] \Rightarrow w'_O\,[p]$ in H. Then, by the induction hypotesis, $S \Rightarrow^* w_I\,\theta(w_O)$ in H'. Furthermore, if $w_I \Rightarrow w'_I\,[p]$ in (G_I, M_I), $w_O \Rightarrow w'_O\,[p]$ in (G_O, M_O), where $\varphi_I(p) = p_{I1} \dots p_{In}$ for some n, $\varphi_O(p) = p_{O1} \dots p_{Om}$ for some m, then, in G_I, $w_I \Rightarrow w_{I1}\,[p_{I1}] \Rightarrow \dots \Rightarrow w_{In}\,[p_{In}] = w'_I$, and, in G_O, $w_O \Rightarrow w_{O1}\,[p_{O1}] \Rightarrow \dots \Rightarrow w_{On}\,[p_{Om}] = w'_O$. Without loss of generality, suppose that

- $w_I = u_{I1}A_{I1}v_{I1} \Rightarrow u_{I1}x_{I1}v_{I1}\,[p_{I1}] = u_{I2}A_{I2}v_{I2} \Rightarrow \dots \Rightarrow u_{In}x_{In}v_{In}\,[p_{In}] = w'_I$, where for $1 \leq i \leq n$, $A_{Ii} \in N_I$, $u_{Ii}, v_{Ii}, x_{Ii} \in (N_I \cup T_I)^*$, and
- $w_O = u_{O1}A_{O0}v_{O1} \Rightarrow u_{O1}x_{O1}v_{O1}\,[p_{O1}] = u_{O2}A_{O2}v_{O2} \Rightarrow \dots \Rightarrow u_{Om}x_{Om}v_{Om}\,[p_{Om}] = w'_O$, where for $1 \leq j \leq m$, $A_{Oj} \in N_O$, $u_{Oj}, v_{Oj}, x_{Oj} \in (N_O \cup T_O)^*$.

That is, $\varphi_I(p) = A_{I1} \to x_{I1} \dots A_{In} \to x_{In}$, $\varphi_O(p) = A_{O1} \to x_{O1} \dots A_{Om} \to x_{Om}$. From the construction of H', we know that for $1 \leq i \leq n$, $p_{Ii} = A_{Ii} \to x_{Ii} \in P$, for $1 \leq j \leq m$, $p'_{Oj} = A_{Oj} \to \theta(x_{Oj}) \in P$, and $t = p_{I1} \dots p_{In} p'_{O1} \dots p'_{Om} \in M$. Therefore, in G:

1. $S \Rightarrow^* w_I\,\theta(w_O) = u_{I1}A_{I1}v_{I1}\,\theta(w_O)$,
2. $u_{I1}A_{I1}v_{I1}\,\theta(w_O) \Rightarrow u_{I1}x_{I1}v_{I1}\,\theta(w_O)\,[p_{I1}] = u_{I2}A_{I2}v_{I2}\,\theta(w_O)$,
3. $u_{I2}A_{I2}v_{I2}\,\theta(w_O) \Rightarrow \dots \Rightarrow u_{In}x_{In}v_{In}\,\theta(w_O)\,[p_{In}] = w'_I\,\theta(w_O) = w'_I\,\theta(u_{O1}A_{O1}v_{O1})$,
4. $w'_I\,\theta(u_{O1}A_{O1}v_{O1}) \Rightarrow w'_I\,\theta(u_{O1}x_{O1}v_{O1})\,[p'_{O1}] = w'_I\,\theta(u_{O2}A_{O2}v_{O2})$,
5. $w'_I\,\theta(u_{O2}A_{O2}v_{O2}) \Rightarrow \dots \Rightarrow w'_I\,\theta(u_{Om}x_{Om}v_{Om})\,[p'_{Om}] = w'_I\,\theta(w'_O)$,

and thus in H', $S \Rightarrow^* w_I\,\theta\,w_O \Rightarrow w'_I\,\theta(w'_O)\,[t]$. Claim 4.8 holds. Furthermore, if $S_I \Rightarrow^* w_I\,[\alpha]$, $S_O \Rightarrow^* w_O\,[\alpha]$ in H, where $w_I \in T_I^*$, $w_O \in T_O^*$, then $\theta(w_O) = \varepsilon$, and thus $S \Rightarrow^* w_I$ in H'. $L_I(H) \subseteq L(H')$. □

Claim 4.9. *If $S \Rightarrow^* w$ in H', then there are strings w_I, w_O such that $w = w_I\,\theta(w_O)$ and $S_I \Rightarrow^* w_I\,[\alpha]$, $S_O \Rightarrow^* w_O\,[\alpha]$ in H for some $\alpha \in \Psi^*$.*

Proof of Claim 4.9. By induction on the number of derivation steps in H'.

Basis. Consider a single derivation step in H'. Because $S \to S_I S_O$ is the only rule in P with S as its left-hand side, this must be $S \Rightarrow S_I S_O$. Then, $S_I \Rightarrow^0 S_I\,[\varepsilon]$, $S_O \Rightarrow^0 S_O\,[\varepsilon]$ in H. Claim 4.9 holds for one derivation step in H'.

Induction hypotesis. Suppose that Claim 4.9 holds for j or fewer derivation steps in H'.

Induction step. Let $S \Rightarrow S_I S_O \Rightarrow^{j-1} w \Rightarrow w'\,[t]$ in H'. Then, by the induction hypotesis, $S_I \Rightarrow^* w_I\,[\alpha]$, $S_O \Rightarrow^* w_O\,[\alpha]$ in H for some w_I, w_O such that $w = w_I\,\theta(w_O)$.

From the construction of H', we know that $t = p_{I1} \dots p_{In} p'_{O1} \dots p'_{Om}$, where for some $p \in \Psi$,

- $p_{I1} = A_{I1} \to x_{I1} \dots p_{In} = A_{In} \to x_{In} = \varphi_I(p)$ and
- $p'_{O1} = A_{O1} \to \theta(x_O)_1 \dots p'_{On} = A_{Om} \to \theta(x_{Om})$ such that $p_{O1} = A_{O1} \to x_{O1} \dots p_{Om} = A_{Om} \to x_{Om} = \varphi_O(p)$.

Then, if $S \Rightarrow S_I S_O \Rightarrow^{j-1} w \Rightarrow w' [t]$ in H', $S \Rightarrow^* w = w_I \, \theta(w_O) = u_{I1} A_{I1} v_{I1} \, \theta(w_O) \Rightarrow u_{I1} x_{I1} v_{I1} \, \theta(w_O) \, [p_{I1}] = u_{I2} A_{I2} v_{I2} \, \theta(w_O) \Rightarrow \dots \Rightarrow u_{In} x_{In} v_{In} \, \theta(w_O) \, [p_{In}] = w'_I \, \theta(w_O) = w'_I \, \theta(u_{O1} A_{O1} v_{O1}) \Rightarrow w'_I \, \theta(u_{O1} x_{O1} v_{O1}) \, [p'_{O1}] = w'_I \, \theta(u_{O2} A_{O2} v_{O2}) \Rightarrow \dots \Rightarrow w'_I \, \theta(u_{Om} x_{Om} v_{Om}) \, [p'_{Om}] = w'_I \, \theta(w'_O)$ in G, where for $1 \leq i \leq n$, $A_{Ii} \in N_I$, $u_{Ii}, v_{Ii}, x_{Ii} \in (N_I \cup T_I)^*$, and for $1 \leq j \leq m$, $A_{Oj} \in N_O$, $u_{Oj}, v_{Oj}, x_{Oj} \in (N_O \cup T_O)^*$. Therefore,

- $w_I = u_{I1} A_{I1} v_{I1} \Rightarrow u_{I1} x_{I1} v_{I1} \, [p_{I1}] = u_{I2} A_{I2} v_{I2} \Rightarrow \dots \Rightarrow u_{In} x_{In} v_{In} \, [p_{In}] = w'_I$ in G_I and
- $w_O = u_{O1} A_{O0} v_{O1} \Rightarrow u_{O1} x_{O1} v_{O1} \, [p_{O1}] = u_{O2} A_{O2} v_{O2} \Rightarrow \dots \Rightarrow u_{Om} x_{Om} v_{Om} \, [p_{Om}] = w'_O$ in G_O,

and thus in H,

- $S_I \Rightarrow^* w_I \, [\alpha] = u_I A_I v_I \Rightarrow u_I x_I v_I \, [p] = w'_I$ and
- $S_O \Rightarrow^* w_O \, [\alpha] = u_O A_O v_O \Rightarrow u_O x_O v_O \, [p] = w'_O$.

Claim 4.9 holds. Furthermore, if $S \Rightarrow^* w$ in H', where $w \in T^*$, then $w_I \in T_I^*$ and $\theta(w_O) = \varepsilon$, thus $w_O \in T_O^*$. $L(H') \subseteq L_I(H)$. □

By Claim 4.8, $L_I(H) \subseteq L(H')$, and by Claim 4.9, $L(H') \subseteq L_I(H)$, therefore $L(H') = L_I(H)$. Thus, the inclusion $\mathscr{L}(\text{SMAT}) \subseteq \mathscr{L}(\text{MAT})$ has been proven, and Theorem 4.7 holds. □

Chapter 5

Synchronous Systems Based on 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, consequently, they generate a corresponding output string in the target language (see [2], [47], [67], [71], or [76]). The second category is composed of language-translation systems or, more briefly, transducers. Frequently, these trasducers consist of several components, including various automata and grammars, some of which read their input strings while others produce their output strings (see [5], [30], [69], and [82]).

Although transducers represent language-translation devices, language theory often views them as language-defining devices and investigates the language family resulting from them. That is, 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 (automaton-grammar) transducer, based upon an FA and a CFG. In addition, a restriction set controls the rules which can be simultaneously used by the automaton and by the grammar.

We discuss the power of this system working in an ordinary way as well as in a leftmost way and investigate an effect of an appearance checking placed into the system.

The original ideas, concepts, definitions, and theoretical results presented in this chapter were first published in [10].

5.1 Rule-Restricted Transducer

The rule-restricted (automaton-grammar) transducer is a hybrid system consisting of an FA and a CFG. The basic idea is straightforward: we read an input sentence with an FA while generating an appropriate output sentence with a CFG. A control set determines which rules from the FA and the CFG can be used simultaneously. The computation of the system is successful if and only if the FA accepts the input string and the CFG generates a string of terminals.

Definition 5.1 (Rule-restricted transducer)**.** The *rule-restricted transducer* (RT for short) Γ is a triple $\Gamma = (M, G, \Psi)$, where

- $M = (Q, \Sigma, \delta, q_0, F)$ is an FA,
- $G = (N, T, P, S)$ is a CFG, 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 Γ 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 $\chi \Rightarrow \chi'$. In the standard way, $\Rightarrow^*$ and $\Rightarrow^+$ are transitive-reflexive and transitive closure of $\Rightarrow$, respectively.

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

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

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

5.1.1 Generative Power

It is well-known that FAs and CFGs describe different classes of languages. Specifically, by FAs we can accept regular languages, whereas CFGs define the class of context-free languages. However, in Example 5.1 it is shown that by the combination of these two models, the system is able to accept and generate even non-context-free languages.

Example 5.1. Consider RT $K = (M, G, \Psi)$ with

- $M = (\{1, 2, 3', 3, 4, 5', 5, 6\}, \{a, b\}, \delta, 1, \{6\})$, where
  - $\delta = \left\{ \begin{array}{llll} p_1 : 1a \to 2, & p_2 : 2 \to 1, & p_3 : 1b \to 3', & p_4 : 3' \to 3, \\ p_5 : 3b \to 4, & p_6 : 4 \to 3, & p_7 : 3a \to 5', & p_8 : 5' \to 5, \\ p_9 : 5a \to 5, & p_{10} : 5b \to 6, & p_{11} : 6b \to 6 & \end{array} \right\}$

  See graphical representation of M in Figure 5.1.
- $G = (\{S, A, B, C, D, D'\}, \{a, b\}, P, S)$, where
  - $P = \left\{ \begin{array}{lll} r_1 : S \to BbD', & r_2 : B \to Bb, & r_3 : D' \to D'D, \\ r_4 : B \to aA, & r_5 : D' \to C, & r_6 : A \to aA, \\ r_7 : C \to CC, & r_8 : D \to b, & r_9 : A \to \varepsilon, \\ r_{10} : C \to a & & \end{array} \right\}$
- $\Psi = \{(p_1, r_1), (p_1, r_2), (p_2, r_3), (p_3, r_4), (p_4, r_5), (p_5, r_6), (p_6, r_7), (p_7, r_8), (p_8, r_9), (p_9, r_8), (p_{10}, r_{10}), (p_{11}, r_{10})\}$.

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

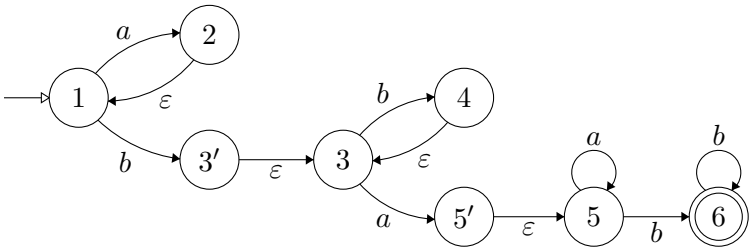


Figure 5.1: Definition of FA M from Example 5.1

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 MATs.

Theorem 5.1.

$$\mathscr{L}(\mathrm{RT})_2 = \mathscr{L}(\mathrm{MAT})$$

Proof. I. First we prove that $\mathscr{L}(\mathrm{MAT}) \subseteq \mathscr{L}(\mathrm{RT})_2$.

Consider a MAT $I = ({}_IG, {}_IC)$ and construct an RT $\Gamma = ({}_\Gamma M, {}_\Gamma G, \Psi)$, such that $L(I) = L(\Gamma)_2$, as follows. Set ${}_\Gamma G = {}_IG$. Construct ${}_\Gamma M = (Q, \Sigma, \delta, s, F)$ in the following way:

1. Set $F, Q = \{s\}$.
2. For every $m = p_1 \dots p_k \in {}_IC$, add:
   (a) $k-1$ new states, $q_1, q_2, \dots, q_{k-1}$, into Q,
   (b) 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
   (c) 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 FA ${}_\Gamma M$ simulates matrices in I by transitions. 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 $p \in {}_IC$ 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 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 $(s, S) \Rightarrow^* (s, w)$ in Γ. The inclusion $\mathscr{L}(\mathrm{MAT}) \subseteq \mathscr{L}(\mathrm{RT})_2$ has been proven.

II. 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 $O = ({}_OG, {}_OC)$ such that $L(\Gamma)_2 = L(O)$ as follows:

1. Set ${}_OG = ({}_\Gamma N \cup \{S'\}, {}_\Gamma T, {}_OP, S')$, ${}_OP = {}_\Gamma P \cup \{p_0 = S' \to \langle s \rangle {}_\Gamma S\}$, and ${}_OC = \{p_0\}$.
2. 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 ${}_OP$ and $p_1 p_2$ into ${}_OC$.
3. Furthermore, for all $q \in F$, add $p = \langle q \rangle \to \varepsilon$ into ${}_OP$ and p into ${}_OC$.

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

Claim 5.2. *If* $(sw, {}_\Gamma S) \Rightarrow^* (qw', \omega)$ *in* Γ*, then* $S' \Rightarrow^* \langle q \rangle \omega$ *in* O*.*

Proof of Claim 5.2. 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 5.2 holds for zero steps in Γ.

Induction hypothesis. Suppose that Claim 5.2 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 5.2 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 5.3. *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 5.3. 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 form 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, $(sw_0, {}_\Gamma S) \Rightarrow^* (q_k w_k, \omega_k)$. 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 $q \in F$ and $w \in L(M)$, and therefore $\omega_k \in L(\Gamma)_2$. $L(O) \subseteq L(\Gamma)_2$. □

By claims 5.2 and 5.3, $L(\Gamma)_2 = L(O)$ holds. Thus, the inclusion $\mathscr{L}(\mathrm{RT})_2 \subseteq \mathscr{L}(\mathrm{MAT})$ has been proven, and Theorem 5.1 holds. □

5.1.2 Accepting Power

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

Lemma 5.4. *For every* k*-PBCA* I*, there is an RT* $\Gamma = (M, G, \Psi)$ *such that* $L(I) = L(\Gamma)_1$*.*

Proof of Lemma 5.4. Let $I = ({}_I Q, \Sigma, {}_I \delta, q_0, F)$ be a k-PBCA for some $k \geq 1$ and construct a RT $\Gamma = (M = ({}_M Q, \Sigma, {}_M \delta, q_0, F), G = (N, T, P, S), \Psi)$ as follows:

1. Set $T = \emptyset$, $\Psi = \emptyset$, $N = \{S, A_1, \ldots, A_k\}$, $P = \{A \to \varepsilon \colon\ A \in N\}$, ${}_M \delta = \{f \to f \colon\ f \in F\}$, and ${}_M Q = {}_I Q$.

2. 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:

(a) $q_1, \ldots, q_n$ into ${}_M Q$;

(b) $r = S \to xS$, where $x \in (N - \{S\})^*$ and $\text{occur}(A_i, x) = \max(0, t_i)$, for $i = 1, \ldots, k$, into P;

(c) $r_1 = q_0 a \to q_1$, $r_2 = q_1 \to q_2$, ..., $r_n = q_{n-1} \to q_n$, $r_{n+1} = q_n \to q$ 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);

(d) $(f \to f, S \to \varepsilon)$ into Ψ for all $f \in F$.

The FA of the created system uses the CFG as an external storage. Each counter of I is represented by a nonterminal. Every step from p to q that modifies counters is 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 5.5 states that the CFG is helpful for the FA in RT at most with the preservation of the non-negative numbers without possibility to check their values.

Lemma 5.5. *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 5.5. 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 $\text{card}(N)$-counter automaton $O = (Q, \Sigma, {}_O\delta, q_0, F)$ 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 $pa \to q(v_1, \ldots, v_{\text{card}(N)})$, where $v_i = \text{occur}(A_i, \beta) - \text{occur}(A_i, \alpha)$ for all $i = 1, \ldots, \text{card}(N)$.

The constructed partially blind $\text{card}(N)$-counter automaton has a counter for each nonterminal from the grammar of Γ. Whenever the automaton in Γ makes a step and thes ential form of the grammar G is changed, O makes the same step and accordingly changes the number of occurrences of nonterminals in its counters. □

From Lemma 5.4 and Lemma 5.5, we can establish the following theorem.

Theorem 5.6.

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

Proof. It directly follows from Lemma 5.4 and Lemma 5.5. □

For 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 the class of **RE** languages [1, 21], and the class of languages defined by partially blind k-counter automata, with respect to number of counters, is superset of the class of **CF** languages and properly included in the class of **CS** languages [27, 28].

5.2 Rule-Restricted Transducer with Leftmost Restriction

Although the investigated system is relatively powerful, in defiance of weakness of models that are used, nondeterministic 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 CFG in RT.

Definition 5.2 (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 $p, q \in Q$. Γ makes a computation step from χ to χ', written as $\chi \Rightarrow_{lm} \chi'$, if and only if $pav_1 \Rightarrow qv_1\,[r_1]$ in M, $uAv_2 \Rightarrow uxv_2\,[r_2]$ in G, and $(r_1, r_2) \in \Psi$. In the standard way, $\Rightarrow^*_{lm}$ and $\Rightarrow^+_{lm}$ are transitive-reflexive 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-$L_{lm}(\Gamma) = \{(w_1, w_2)\colon\ (q_0w_1, S) \Rightarrow^*_{lm} (f, w_2),\ w_1 \in \Sigma^*,\ w_2 \in T^*,$ and $f \in F\}$; we call Γ a *leftmost restricted RT*; and we define the languages given from 2-$L_{lm}(\Gamma)$ as $L_{lm}(\Gamma)_1 = \{w_1\colon\ (w_1, w_2) \in$ 2-$L_{lm}(\Gamma)\}$ and $L_{lm}(\Gamma)_2 = \{w_2\colon\ (w_1, w_2) \in$ 2-$L_{lm}(\Gamma)\}$.

By $\mathscr{L}(\mathrm{RT}_{lm})$, $\mathscr{L}(\mathrm{RT}_{lm})_1$, and $\mathscr{L}(\mathrm{RT}_{lm})_2$, we understand the following language classes, respectively: 2-languages of leftmost restricted RTs, languages accepted by M in leftmost restricted RTs, and languages generated by G in leftmost restricted RTs.

5.2.1 Generative Power

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

Theorem 5.7.

$$\mathscr{L}(\mathrm{RT}_{lm})_2 = \mathbf{CF}$$

Proof. The inclusion $\mathbf{CF} \subseteq \mathscr{L}(\mathrm{RT}_{lm})_2$ 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 whilst 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 nondeterministic PDAs. It is therefore sufficient to prove that every language $L_{lm}(\Gamma)_2$ of RT can be accepted by a nondeterministic PDA. Consider an RT $\Gamma = ({}_\Gamma M = (Q, {}_\Gamma\Sigma, {}_\Gamma\delta, q_0, F), G = (N, T, P, S), \Psi)$ and define a 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:

1. Set ${}_O\delta = \emptyset$.
2. 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 ${}_O\delta$.
3. 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 5.8. *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 5.8. By 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 5.8 holds for zero computation steps in Γ.

Induction hypothesis. Suppose that Claim 5.8 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 5.8 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 5.8, suppose that O is in configuration $(\alpha v)^Rqw'$, where $uw' = uxv$. From 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 5.9. *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 5.9. Consider any successful acceptance:

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

in PDA O. Without any loss of generality, we can express (I) 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, (I) 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. Thus, Claim 5.9 holds. □

As $L(O) \subseteq L_{lm}(\Gamma)_2$ and $L_{lm}(\Gamma)_2 \subseteq L(O)$, Theorem 5.7 holds. □

5.2.2 Accepting Power

First, we show that any context-free language can be accepted by some leftmost restricted RT.

Lemma 5.10. *For every language $L \in$ **CF**, there is an RT $\Gamma = (M, G, \Psi)$ such that $L_{lm}(\Gamma)_1 = L$.*

Proof of Lemma 5.10. Let $I = ({}_IN, T, {}_IP, S)$ be a CFG such that $L(I) = L$. For I, we can construct a CFG $H = ({}_HN, T, {}_HP, S)$, where ${}_HN = {}_IN \cup \{\langle a\rangle\colon\ a \in T\}$ and ${}_HP = \{\langle a\rangle \to a\colon\ a \in T\} \cup \{A \to x\colon\ A \to x' \in {}_IP$ 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 an FA $M = (\{q_0\}, T, \delta, q_0, \{q_0\})$ with

$\delta = \{q_0 \to q_0\} \cup \{q_0 a \to q_0 \colon\ a \in T\}$, and $\Psi = \{(q_0 \to q_0, A \to x) \colon\ A \to x \in {}_H P, A \in {}_I N\} \cup \{(q_0 a \to q_0, \langle a \rangle \to a) \colon\ a \in T\}$.

It is easy to see that any time when H replaces nonterminals from ${}_I N$ 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-$L_{lm}(\Gamma) = \{(w, w) \colon\ w \in L(I)$. 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 5.11. *Let Γ is 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 5.11. In the same way as in the proof of Theorem 5.1, 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:

1. Set ${}_O\delta = \emptyset$.
2. 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.[1]

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

Claim 5.12. *Let $(q_0 w, S) \Rightarrow^* (pw', u\alpha v)$ in RT Γ, where $u \in T^*$, $\alpha \in N$, and $v \in (N \cup T)^*$. Then, $Sq_0 w \Rightarrow^* (\theta(v))^R \alpha p w'$ in PDA O.*

Proof of Claim 5.12. By induction on the number of computation steps.

Basis. Let $(q_0 w, S) \Rightarrow^0 (q_0 w, S)$ in Γ. Then, surely, $Sq_0 w \Rightarrow^0 (\theta(S))^R q_0 w$. Claim 5.12 holds for zero computation steps in Γ.

Induction hypothesis. Suppose that Claim 5.12 holds for j or fewer computation steps.

Induction step. Let $(q_0 w, S) \Rightarrow^j (paw', u\alpha v) \Rightarrow (qw', uxv)$ in Γ, where $a \in {}_\Gamma\Sigma \cup \{\varepsilon\}$, $uxv = uu'\beta v'$ and β is the leftmost nonterminal. By the induction hypothesis, $Sq_0 w \Rightarrow^* (\theta(v))^R \alpha p a w'$ in O.

Because $(paw', u\alpha v) \Rightarrow (qw', uxv)$ in Γ, $paw' \Rightarrow qw'\,[r_1]$ in ${}_\Gamma M$, $u\alpha v \Rightarrow uxv\,[r_2]$ 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 5.12 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 5.13. *Let $Sq_0 w \Rightarrow^* (\theta(v))^R \alpha p w'$ in PDA O. Then, $(q_0 w, S) \Rightarrow^* (pw', u\alpha v)$ in RT Γ, where $u \in T^*$, $\alpha \in N$, and $v \in (N \cup T)^*$.*

[1] See page 35 for further explanation and precise formal definition of θ (Definition 4.4).

Proof of Claim 5.13. By induction on the number of moves.

Basis. Let $Sq_0w \Rightarrow^0 Sq_0w$ in O. Then, $(q_0w, S) \Rightarrow^0 (q_0w, S)$ in Γ and Claim 5.13 holds for zero moves in O.

Induction hypothesis. Suppose that Claim 5.13 holds for j or fewer moves.

Induction step. Let $Sq_0w \Rightarrow^j (\theta(v))^R\alpha paw' \Rightarrow (\theta(xv))^R qw'$ 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 $\alpha pa \to (\theta(x))^R p$ 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 5.13 holds. Furthermore, if $\theta(xv)w' = \varepsilon$ and $q \in F$, then $(paw', u\alpha v) \Rightarrow (f, uxv)$ 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 5.14.

$$\mathscr{L}(\mathrm{RT}_{lm})_1 = \mathbf{CF}$$

Proof. It directly follows from Lemma 5.10 and Lemma 5.11. □

5.3 Rule-Restricted Transducer with Appearance Checking

We can also extend RT with the possibility to prefer a rule over another—that is, the restriction sets contain triples 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 5.3 (RT with appearance checking)**.** *RT with appearance checking* (RT_{ac} for short) Γ is a triple $\Gamma = (M, G, \Psi)$, where

- $M = (Q, \Sigma, \delta, q_0, F)$ is an FA,
- $G = (N, T, P, S)$ is a CFG, and
- Ψ is a finite set of triples 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
- $uAv_3 \Rightarrow uxv_2\,[r_3]$ 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 5.1. The classes of languages defined by the first and the second component in the system is denoted by $\mathscr{L}(\mathrm{RT}_{ac})_1$ and $\mathscr{L}(\mathrm{RT}_{ac})_2$, respectively.

5.3.1 Generative Power

By the appearance checking both generative and accepting power of RT grow to define the class of all recursively enumerable languages. To prove that the former holds, we take advantage of the known fact that matrix grammars with appearance checking can generate any language in **RE** [21], and show that, in turn, RT_{ac} can simulate MAT_{ac}.

Theorem 5.15.

$$\mathscr{L}(\mathrm{RT}_{ac})_2 = \mathbf{RE}$$

Proof. Since $\mathscr{L}(\mathrm{MAT}_{ac}) = \mathbf{RE}$ [21], we only need to prove that $\mathscr{L}(\mathrm{MAT}_{ac}) \subseteq \mathscr{L}(\mathrm{RT}_{ac})_2$.

Consider a MAT_{ac} with appearance checking $I = ({}_IG, {}_IC)$ and construct a RT $\Gamma = ({}_\Gamma M, {}_\Gamma G, \Psi)$, such that $L(I) = L(\Gamma)_2$, as follows:

1. Set ${}_\Gamma G = {}_IG$.
2. Add a new initial nonterminal S', nonterminal Δ, and rules $\Delta \to \Delta$, $\Delta \to \varepsilon$, $S' \to S\Delta$ into grammar ${}_\Gamma G$.
3. Construct an FA ${}_\Gamma M = (Q, \Sigma, \delta, s, F)$ and Ψ in the following way:
   (a) Set $F = Q = \{s\}$, $\delta = \{s \to s\}$, and $\Psi = \{(s \to s, \Delta \to \varepsilon, \Delta \to \varepsilon), (s \to s, S' \to S\Delta, S' \to S\Delta)\}$.
   (b) 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 FA simulates matrices in I by computation step. 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 $(r_1, p_1, c_1), \dots, (r_i, p_i, c_i) \in \Psi$. 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. □

5.3.2 Accepting Power

RT_{ac}'s can accept any recursively enumerable language, as evidenced by their ability to simulate k-CAs.

Theorem 5.16.

$$\mathscr{L}(\mathrm{RT}_{ac})_1 = \mathbf{RE}$$

Proof. Let $I = ({}_IQ, \Sigma, {}_I\delta, q_0, F)$ be a k-CA for some $k \geq 1$ and construct a RT $\Gamma = (M, G, \Psi)$, where $M = ({}_MQ, \Sigma, {}_M\delta, q_0, F)$, $G = (N, T, P, S)$, as follows:

1. Set $T = \{a\}, \Psi = \emptyset, P = \{A \to \varepsilon, A \to \Diamond\colon\ A \in N - \{\Diamond\}\} \cup \{S \to S\}$, ${}_MQ = {}_IQ$, ${}_M\delta = \{f \to f\colon\ f \in F\}$, and $N = \{S, \Diamond, A_1, \ldots, A_k\}$.

2. 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:

   (a) $q_1, \ldots, q_n$ into ${}_MQ$;

   (b) $r = S \to xS$, where $x \in (N - \{S, \Diamond\})^*$ and $\mathrm{occur}(A_i, x) = \widehat{\theta}(t_i)$, for each $i = 1, \ldots, k$, into P;

   (c) $r_1 = q_0a \to q_1$, $r_2 = q_1 \to q_2$, …, $r_n = q_{n-1} \to q_n$, $r_{n+1} = q_n \to q$ 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 $\theta(t_j)$ is, $\tau_i = \tau_i' = A_j \to \varepsilon$; 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;

   (d) $(f \to f, S \to \varepsilon, S \to \varepsilon)$ into Ψ for all $f \in F$.

Similarly as in the proof of Lemma 5.4, the FA of the created system uses the CFG as an external storage, and each counter of 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 continues with computation. Otherwise, since $\Diamond$ cannot be rewritten during the rest of computation, the use of such rules 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 $k \geq 2$ [35], Theorem 5.16 holds. □

Part III

Application Perspectives and Concluding Remarks

Chapter 6

Linguistic Applications: Perspectives

In this chapter, we discuss the advantages of the new formal models in regard to their potential applications in natural language processing, and particularly in translation, where they can provide an alternative to the existing models (see Chapter 3 for an overview). To illustrate, we use examples from Czech, English, and Japanese. (No prior knowledge of Czech or Japanese is required for understanding, although it can be an advantage.)

Throughout the course of this chapter, we use the following notation to represent some common linguistic constituents:

ADJ	adjective	ADV	adverb
AUX	auxiliary verb	DET	determiner
N	noun	NP	noun phrase
NP-SBJ	NP in the role of subject	NUM	numeral
P	preposition	PN	pronoun
PN-INT	interrogative pronoun	PP	prepositional phrase
PP-TMP	PP, temporal	PP-DIR	PP, directional
V	verb	VP	verb phrase

Further, note that in the example sentences presented below, we generally disregard punctuation and capitalization. For example, we consider

Where are you going?

and

where are you going

identical for the purposes of this text.

Finally, in most of the case studies presented in this chapter, we assume that we already have the input sentence split into words (or possibly some other lexical units as appropriate), and these words are classified as, for example, a noun, pronoun, or verb. Then, we consider syntax analysis and translation on an abstract level, transforming syntactic structures in languages rather than actual meanings.

Often, you will notice that the input alphabet of the automaton or the terminal alphabet of the grammar do not contain actual words themselves, but rather symbols representing word categories and properties. For example, we can use N_{3s} to denote a noun in third

person singular. While such representation is sufficient in our examples here, where, for clarity, we usually only focus on some select aspects at a time, in practice we need much more information about each word. In that case, we can, for instance, use structures resembling AVMs from HPSGs (see Section 3.1) as symbols.

6.1 Synchronous Grammars

First, we explore the application perspectives of our newly introduced synchronous grammars, or more precisely, synchronous versions of MATs and SCGs. The original results, observations, and examples presented in this section were published in [39] and [41].

To demonstrate the basic principle, consider a simple Japanese sentence

Takeshi-san wa raishuu Oosaka ni ikimasu.

We will transform this sentence (or, more precisely, the structure of this sentence) into its English counterpart

Takeshi is going to Osaka next week.

In the following examples, words in angled brackets ($\langle\rangle$) are words associated with a terminal or nonterminal symbol in a given sentence or structure. Note that this is included only to make the examples easier to follow and understand, and is not an actual part of the formalism itself.

Example 6.1. Consider a RSCFG $H = (G_I, G_O, \Psi, \varphi_I, \varphi_O)$, where $G_I = (N_I, T_I, P_I, \mathrm{S}_I)$ and $G_O = (N_O, T_O, P_O, \mathrm{S}_O)$ such that

- $N_I = \{\mathrm{S}_I, \text{NP-SBJ}, \text{VP}, \text{PP-TMP}, \text{PP-DIR}\}$,
- $T_I = \{\text{NP}, \text{V}, \text{DET}\}$,
- $N_O = \{\mathrm{S}_O, \text{NP-SBJ}, \text{VP}, \text{PP-TMP}, \text{PP-DIR}\}$,
- $T_O = \{\text{NP}, \text{V}, \text{AUX}, \text{DET}, \text{P}\}$,
- $P_I = \left\{ \begin{array}{ll} 1 : \mathrm{S}_I \to \text{NP-SBJ VP}, & 2 : \text{NP-SBJ} \to \text{NP DET}\langle wa \rangle, \\ 3 : \text{VP} \to \text{PP-TMP PP-DIR V}, & 4 : \text{PP-TMP} \to \text{NP}, \\ 4z : \text{PP-TMP} \to \varepsilon, & 5 : \text{PP-DIR} \to \text{NP DET}\langle ni \rangle, \\ 5z : \text{PP-DIR} \to \varepsilon & \end{array} \right\}$,
- $P_O = \left\{ \begin{array}{ll} 1 : \mathrm{S}_O \to \text{NP-SBJ VP}, & 2 : \text{NP-SBJ} \to \text{NP}, \\ 3 : \text{VP} \to \text{AUX V PP-DIR PP-TMP}, & 4 : \text{PP-TMP} \to \text{NP}, \\ 4z : \text{PP-TMP} \to \varepsilon, & 5 : \text{PP-DIR} \to \text{P}\langle to \rangle \text{ NP}, \\ 5z : \text{PP-DIR} \to \varepsilon & \end{array} \right\}$.

Strictly according to their definitions, synchronous grammars generate pairs of sentences. However, in practice, we usually have the input sentence in the source language, and we want to translate it into the target language. That is, we want to generate the corresponding output sentence. In that case, the translation can be divided into two steps as follows.

Form	*Paradigm*	*Person*	*Neutral*	*Negative*
Primary	Present	1st singular	*am*	*aren't*
		3rd singular	*is*	*isn't*
		Other	*are*	*aren't*
	Preterite	1st singular, 3rd singular	*was*	*wasn't*
		Other	*were*	*weren't*
	Irrealis	1st singular, 3rd singular	*were*	*weren't*
Secondary	Plain form		*be*	—
	Gerund-participle		*being*	—
	Past participle		*been*	—

Table 6.1: Paradigms of the verb *to be* in English [63]

1. First, we parse the input sentence using the input grammar. In G_I, a derivation that generates the example sentence may proceed as follows:

   $$\begin{aligned} \mathrm{S}_I &\Rightarrow \text{NP-SBJ VP [1]} \\ &\Rightarrow \text{NP}\langle \textit{Takeshi-san} \rangle\ \text{DET}\langle \textit{wa} \rangle\ \text{VP [2]} \\ &\Rightarrow \text{NP}\langle \textit{Takeshi-san} \rangle\ \text{DET}\langle \textit{wa} \rangle\ \text{PP-TMP PP-DIR V}\langle \textit{ikimasu} \rangle\ \text{[3]} \\ &\Rightarrow \text{NP}\langle \textit{Takeshi-san} \rangle\ \text{DET}\langle \textit{wa} \rangle\ \text{NP}\langle \textit{raishuu} \rangle\ \text{PP-DIR V}\langle \textit{ikimasu} \rangle\ \text{[4]} \\ &\Rightarrow \text{NP}\langle \textit{Takeshi-san} \rangle\ \text{DET}\langle \textit{wa} \rangle\ \text{NP}\langle \textit{raishuu} \rangle\ \text{NP}\langle \textit{Oosaka} \rangle\ \text{DET}\langle \textit{ni} \rangle \\ &\quad \text{V}\langle \textit{ikimasu} \rangle\ \text{[5]} \end{aligned}$$

   We have applied rules denoted by labels 1 2 3 4 5, in that order.

2. Next, we use the sequence obtained in the first step (1 2 3 4 5), and apply the corresponding rules in the output grammar. Then, the derivation in G_O proceeds as follows:

   $$\begin{aligned} \mathrm{S}_O &\Rightarrow \text{NP-SBJ VP [1]} \\ &\Rightarrow \text{NP}\langle \textit{Takeshi} \rangle\ \text{VP [2]} \\ &\Rightarrow \text{NP}\langle \textit{Takeshi} \rangle\ \text{AUX}\langle \textit{is} \rangle\ \text{V}\langle \textit{going} \rangle\ \text{PP-TMP PP-DIR [3]} \\ &\Rightarrow \text{NP}\langle \textit{Takeshi} \rangle\ \text{AUX}\langle \textit{is} \rangle\ \text{V}\langle \textit{going} \rangle\ \text{PP-DIR NP}\langle \textit{next week} \rangle\ \text{[4]} \\ &\Rightarrow \text{NP}\langle \textit{Takeshi} \rangle\ \text{AUX}\langle \textit{is} \rangle\ \text{V}\langle \textit{going} \rangle\ \text{P}\langle \textit{to} \rangle\ \text{NP}\langle \textit{Osaka} \rangle\ \text{NP}\langle \textit{next week} \rangle\ \text{[5]} \end{aligned}$$

Also note the rules $4z$ and $5z$ (in both input and outpur grammar), which can be used to erase PP-TMP and PP-DIR. This represents the fact that these constituents may be omitted.

Next, compare the following sentences in Japanese (left) and English.

Watashi wa raishuu Oosaka ni ikimasu.	*I am going to Osaka next week.*
Anata wa raishuu Oosaka ni ikimasu.	*You are going to Osaka next week.*
Takeshi-san wa raishuu Oosaka ni ikimasu.	*Takeshi is going to Osaka next week.*

In English, the form of the auxiliary verb *to be* depends on many grammatical categories such as tense, number, or, as shown in this example, person: *am* for first person (present tense, singular), *are* for second, and *is* for third (see Table 6.1).

On the other hand, note that the verb in the Japanese sentences (*ikimasu*, long form of *iku*) remains the same (out of the grammatical categories mentioned, only tense would

affect inflection). If we want to translate such sentence from Japanese to English, we need to choose the correct form of the verb. We can get the necessary information by looking at the subject.

Example 6.2. Consider a SMAT $H = (G_I, M_I, G_O, M_O, \Psi, \varphi_I, \varphi_O)$, where $G_I = (N_I, T_I, P_I, \mathrm{S}_I)$ and $G_O = (N_O, T_O, P_O, \mathrm{S}_O)$ such that

- $N_I = \{\mathrm{S}_I, \text{NP-SBJ}, \text{VP}, \text{PP-TMP}, \text{PP-DIR}\}$,
- $T_I = \{\text{NP}, \text{NP}_1, \text{NP}_2, \text{NP}_3, \text{V}, \text{DET}\}$,
- $N_O = \{\mathrm{S}_O, \text{NP-SBJ}, \text{VP}, \text{PP-TMP}, \text{PP-DIR}, \text{AUX}_x\}$,
- $T_O = \{\text{NP}, \text{NP}_1, \text{NP}_2, \text{NP}_3, \text{V}, \text{AUX}_1, \text{AUX}_2, \text{AUX}_3, \text{DET}, \text{P}\}$,
- $P_I = \left\{ \begin{array}{ll} 1 : \mathrm{S}_I \to \text{NP-SBJ VP}, & 2a : \text{NP-SBJ} \to \text{NP}_1\ \text{DET}\langle wa\rangle, \\ 2b : \text{NP-SBJ} \to \text{NP}_2\ \text{DET}\langle wa\rangle, & 2c : \text{NP-SBJ} \to \text{NP}_3\ \text{DET}\langle wa\rangle, \\ 3 : \text{VP} \to \text{PP-TMP PP-DIR V}, & 4 : \text{PP-TMP} \to \text{NP}, \\ 4z : \text{PP-TMP} \to \varepsilon, & 5 : \text{PP-DIR} \to \text{NP DET}\langle ni\rangle, \\ 5z : \text{PP-DIR} \to \varepsilon & \end{array} \right\}$,
- $P_O = \left\{ \begin{array}{ll} 1 : \mathrm{S}_O \to \text{NP-SBJ VP}, & 2a : \text{NP-SBJ} \to \text{NP}_1, \\ 2b : \text{NP-SBJ} \to \text{NP}_2, & 2c : \text{NP-SBJ} \to \text{NP}_3, \\ 3 : \text{VP} \to \text{AUX}_x\ \text{V PP-DIR PP-TMP}, & 4 : \text{PP-TMP} \to \text{NP}, \\ 4z : \text{PP-TMP} \to \varepsilon, & 5 : \text{PP-DIR} \to \text{P}\langle to\rangle\ \text{NP}, \\ 5z : \text{PP-DIR} \to \varepsilon, & 6a : \text{AUX}_x \to \text{AUX}_1, \\ 6b : \text{AUX}_x \to \text{AUX}_2, & 6c : \text{AUX}_x \to \text{AUX}_3 \end{array} \right\}$,
- $M_I = \{m_1 : 1, m_{2a} : 2a, m_{2b} : 2b, m_{2c} : 2c, m_3 : 3, m_4 : 4, m_{4z} : 4z, m_5 : 5, m_{5z} : 5z\}$, and
- $M_O = \{m_1 : 1, m_{2a} : 2a\,6a, m_{2b} : 2b\,6b, m_{2c} : 2c\,6c, m_3 : 3, m_4 : 4, m_{4z} : 4z,\ m_5 : 5, m_{5z} : 5z\}$.

An example of a derivation follows.

$$
\begin{array}{lcl}
\mathrm{S}_I & \Rightarrow & \text{NP-SBJ VP } [m_1] \\
& \Rightarrow & \text{NP-SBJ PP-TMP PP-DIR V}\langle ikimasu\rangle\ [m_3] \\
& \Rightarrow & \text{NP}_1\langle watashi\rangle\ \text{DET}\langle wa\rangle\ \text{PP-TMP PP-DIR V}\langle ikimasu\rangle\ [m_{2a}] \\
& \Rightarrow & \text{NP}_1\langle watashi\rangle\ \text{DET}\langle wa\rangle\ \text{NP}\langle raishuu\rangle\ \text{PP-DIR V}\langle ikimasu\rangle\ [m_4] \\
& \Rightarrow & \text{NP}_1\langle watashi\rangle\ \text{DET}\langle wa\rangle\ \text{NP}\langle raishuu\rangle\ \text{NP}\langle Oosaka\rangle\ \text{DET}\langle ni\rangle \\
& & \text{V}\langle ikimasu\rangle\ [m_5]
\end{array}
$$

$$
\begin{array}{lcl}
\mathrm{S}_O & \Rightarrow & \text{NP-SBJ VP } [m_1] \\
& \Rightarrow & \text{NP-SBJ AUX}_x\langle be\rangle\ \text{V}\langle going\rangle\ \text{PP-DIR PP-TMP } [m_3] \\
& \Rightarrow & \text{NP}_1\langle I\rangle\ \text{AUX}_1\langle am\rangle\ \text{V}\langle going\rangle\ \text{PP-DIR PP-TMP } [m_{2a}] \\
& \Rightarrow & \text{NP}_1\langle I\rangle\ \text{AUX}_1\langle am\rangle\ \text{V}\langle going\rangle\ \text{PP-DIR NP}\langle next\ week\rangle\ [m_4] \\
& \Rightarrow & \text{NP}_1\langle I\rangle\ \text{AUX}_1\langle am\rangle\ \text{V}\langle going\rangle\ \text{P}\langle to\rangle\ \text{NP}\langle Osaka\rangle\ \text{NP}\langle next\ week\rangle\ [m_5]
\end{array}
$$

Depending on person of the subject, we apply one of the matrices m_{2a}, m_{2b}, or m_{2c}. In the input grammar (Japanese), these matrices contain only one rule, which involves the subject itself. The verb is unaffected. In the output grammar (English), the matrices contain two rules, which ensure agreement between the subject and the (auxiliary) verb. Instead of SMAT, we can also use SSCG to the same effect, with scattered context rules such as $(\text{NP-SBJ}, \text{AUX}_x) \to (\text{NP}_1, \text{AUX}_1)$.

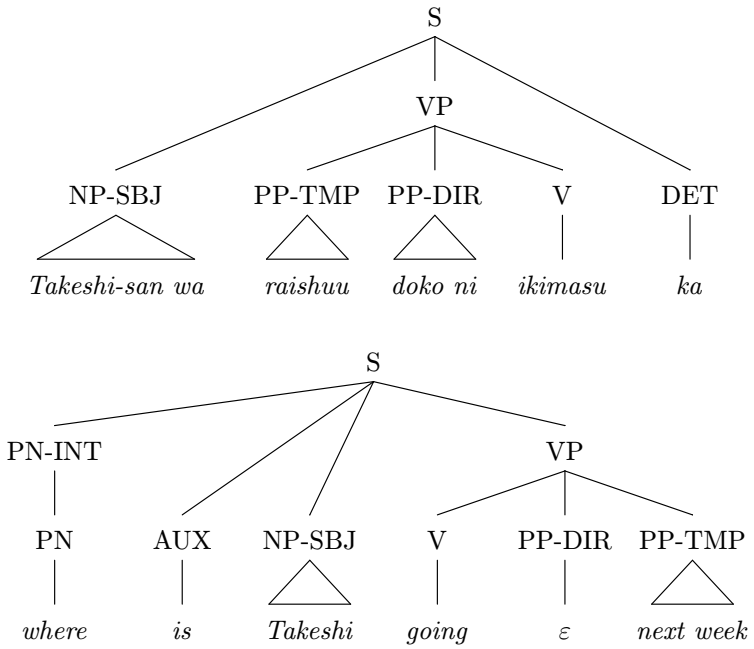


Figure 6.1: Syntax trees for Japanese (top) and English question

Let us have a look at some other syntactic structures. For instance, to form a question in Japanese, we can simply take a statement and append the particle *ka* at the end of the sentence. However, in English, we need change the order of the words, placing the auxiliary verb in front of the subject. Compare the following sentences in Japanese (left) and English.

Takeshi-san wa raishuu Oosaka ni ikimasu ka.	*Is Takeshi going to Osaka next week?*
Takeshi-san wa raishuu doko ni ikimasu ka.	*Where is Takeshi going next week?*
Takeshi-san wa itsu Oosaka ni ikimasu ka.	*When is Takeshi going to Osaka?*

Observe that in Japanese, the only difference between the three questions is the word *Oosaka* being replaced by *doko* (*where*), or *raishuu* by *itsu* (*when*). The word order and sentence structure is unaffected. (However, do note that there are also other ways the questions can be phrased in Japanese.)

On the other hand, in English, the structure of the sentence changes further, as illustrated by the syntax trees in Figure 6.1. The interrogative pronoun (*where*, *when*) is removed from its "original" position—that is, the position of the corresponding prepositional phrase—and placed at the beginning of the sentence.[1]

To reflect this in our grammar, we can add nonterminal symbol PN-INT to N_O and terminal symbol PN to both T_I and T_O. To P_I, we add rules

$$
\begin{aligned}
&1q : \text{S} \to \text{NP-SBJ VP DET}\langle ka \rangle, \\
&4q : \text{PP-TMP} \to \text{PN}\langle itsu \rangle, \\
&5q : \text{PP-DIR} \to \text{PN}\langle doko \rangle \ \text{DET}\langle ni \rangle,
\end{aligned}
$$

[1] This is a common principle known as *wh*-movement, which is present in many languages besides English.

and to P_O, rules

$$\begin{aligned} 1q &: \text{S} \to \text{AUX}_x\ \text{NP-SBJ VP}, \\ 3q &: \text{VP} \to \text{V PP-DIR PP-TMP}, \\ 1qi &: \text{S} \to \text{PN-INT AUX}_x\ \text{NP-SBJ VP}, \\ 7q &: \text{PN-INT} \to \text{PN} \end{aligned}$$

Finally, we add matrices $m_{1q}, m_{1qi}\colon 1q\,3$, $m_{4q}\colon 4q$, $m_{5q}\colon 5q$ to M_I and matrices $m_{1q}\colon 1q\,3q$, $m_{1qi}\colon 1qi\,3q$, $m_{4q}\colon 7q\,4z$, $m_{5q}\colon 7q\,5z$ to M_O (recall that the rules $4z$ and $5z$ erase PP-TMP and PP-DIR, respectively).

An example of a derivation follows.

$$\begin{aligned} \text{S}_I &\Rightarrow \text{NP-SBJ PP-TMP PP-DIR V}\langle ikimasu\rangle\ \text{DET}\langle ka\rangle\ [m_{1qi}] \\ &\Rightarrow \text{NP}_3\langle Takeshi\text{-}san\rangle\ \text{DET}\langle wa\rangle\ \text{PP-TMP PP-DIR V}\langle ikimasu\rangle\ \text{DET}\langle ka\rangle\ [m_{2c}] \\ &\Rightarrow \text{NP}_3\langle Takeshi\text{-}san\rangle\ \text{DET}\langle wa\rangle\ \text{NP}\langle raishuu\rangle\ \text{PP-DIR V}\langle ikimasu\rangle \\ &\quad \text{DET}\langle ka\rangle\ [m_4] \\ &\Rightarrow \text{NP}_3\langle Takeshi\text{-}san\rangle\ \text{DET}\langle wa\rangle\ \text{NP}\langle raishuu\rangle\ \text{PN}\langle doko\rangle\ \text{DET}\langle ni\rangle\ \text{V}\langle ikimasu\rangle \\ &\quad \text{DET}\langle ka\rangle\ [m_{5q}] \end{aligned}$$

$$\begin{aligned} \text{S}_O &\Rightarrow \text{PN-INT AUX}_x\langle be\rangle\ \text{NP-SBJ V}\langle going\rangle\ \text{PP-DIR PP-TMP}\ [m_{1qi}] \\ &\Rightarrow \text{PN-INT AUX}_3\langle is\rangle\ \text{NP}_3\langle Takeshi\rangle\ \text{V}\langle going\rangle\ \text{PP-DIR PP-TMP}\ [m_{2c}] \\ &\Rightarrow \text{PN-INT AUX}_3\langle is\rangle\ \text{NP}_3\langle Takeshi\rangle\ \text{V}\langle going\rangle\ \text{PP-DIR NP}\langle next\ week\rangle\ [m_4] \\ &\Rightarrow \text{PN}\langle where\rangle\ \text{AUX}_3\langle is\rangle\ \text{NP}_3\langle Takeshi\rangle\ \text{V}\langle going\rangle\ \text{NP}\langle next\ week\rangle\ [m_{5q}] \end{aligned}$$

Note that the matrices ensuring subject-verb agreement (see Example 6.2) work without any changes for all of the structures discussed above. Clearly, it is possible to capture the relation within a context-free grammar. In that case, however, the necessary grammatical information has to be propagated through the derivation tree. This means that we have to add separate rules covering all possibilities (person, number…) for each of the structures, even though the structures themselves are not actually affected. Using SMAT or SSCG, we are able to describe the relation more easily, with only a relatively small number of rules.

Arguably, the presented example is somewhat special in that *to be* is an irregular verb. In English, we usually only need to distinguish two cases: third person singular and anything else. In languages with rich inflection, such as Czech, this advantage becomes even more important.

Let us now consider translation between Czech and English. Czech is a relatively challenging language in terms of natural language processing. It is a free-word-order language with rich inflection (see [31]).

For example, consider the Czech sentence

Dva růžoví sloni přišli na přednášku.
(Two pink elephants came to the lecture.)

All of the following permutations of words also make for a valid sentence:

dva růžoví sloni přišli na přednášku
růžoví sloni přišli na přednášku dva
dva sloni přišli na přednášku růžoví
sloni přišli na přednášku dva růžoví
dva růžoví sloni na přednášku přišli
růžoví sloni na přednášku přišli dva
dva sloni na přednášku přišli růžoví
sloni na přednášku přišli dva růžoví

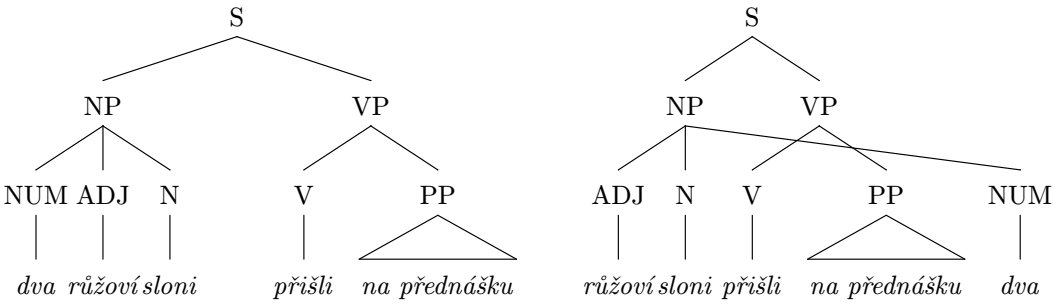


Figure 6.2: Syntax trees for example sentences in Czech

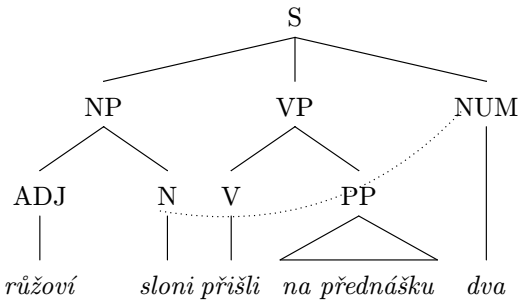


Figure 6.3: Modified syntax tree

There may be differences in meaning or emphasis, but the syntactic structure remains the same. Why is this problematic? Compare the syntax trees in Figure 6.2. Because of the crossing branches (non-projectivity), the second tree cannot be produced by any CFG. Of course, it is still possible to construct a CFG that generates the sentence *růžoví sloni přišli na přednášku dva* if we consider a different syntax tree, for example such as in Figure 6.3. However, this tree no longer captures the relation between the noun *sloni* and its modifying numeral *dva* (represented by the dotted line). We need to know this relation for instance to ensure agreement between the words (person, number, gender...), so that we can choose their appropriate forms.

As mentioned above, in a purely context-free framework, this can be complicated. The necessary information has to be propagated through the derivation tree, even if the structure is not actually affected, and this can result in a high number of rules. Recall that in GPSGs, for instance, this is countered by the introduction of metarules and features (see Section 3.1). With MATs, we can instead represent the relations using matrices.

Example 6.3. Here, we present an example of SMAT $H = (G_{cz}, M_{cz}, G_{en}, M_{en}, \Psi, \varphi_{cz}, \varphi_{en})$ that describes the translations between the English sentence *two pink elephants came to the lecture* and any of the above Czech sentences, correctly distinguishing between male and female gender in Czech (to demonstrate female gender, we also include *opice* in Czech, *monkeys* in English). Note that H is actually more general (for example allowing multiple adjectives). It is designed for easy extension to include other grammatical categories (person...) as well as different syntactic structures.

For Czech, let G_{cz} contain the following context-free rules (nonterminals are in capitals, S_{cz} is the start symbol):

s :	$\mathrm{S}_{cz} \to$ NP VP NUM ADJS,	np :	NP $\to$ NUM ADJS N,
vp :	VP $\to$ ADVS V ADVS,	num_ε :	NUM $\to \varepsilon$,
adjs :	ADJS $\to$ ADJ ADJS,	$\mathrm{adjs}_\varepsilon$:	ADJS $\to \varepsilon$,
advs :	ADVS $\to$ ADV ADVS,	$\mathrm{advs}_\varepsilon$:	ADVS $\to \varepsilon$,
n_m :	N $\to \mathrm{N}_m$,	n_f :	N $\to \mathrm{N}_f$,
n_{mm} :	$\mathrm{N}_m \to \mathrm{N}_m$,	n_{ff} :	$\mathrm{N}_f \to \mathrm{N}_f$,
v_m :	V $\to \mathrm{V}_m$,	v_f :	V $\to \mathrm{V}_f$,
adj_m :	ADJ $\to \mathrm{ADJ}_m$,	adj_f :	ADJ $\to \mathrm{ADJ}_f$,
adv :	ADV $\to$ PP,	num_m :	NUM $\to \mathrm{NUM}_m$,
num_f :	NUM $\to \mathrm{NUM}_f$,	dict_1 :	$\mathrm{N}_m \to$ *sloni*,
dict_2 :	$\mathrm{N}_f \to$ *opice*,	dict_{3m} :	$\mathrm{V}_m \to$ *přišli*,
dict_{3f} :	$\mathrm{V}_f \to$ *přišly*,	dict_{4m} :	$\mathrm{ADJ}_m \to$ *růžoví*,
dict_{4f} :	$\mathrm{ADJ}_f \to$ *růžové*,	dict_{5m} :	$\mathrm{NUM}_m \to$ *dva*,
dict_{5f} :	$\mathrm{NUM}_f \to$ *dvě*,	dict_6 :	PP $\to$ *na přednášku*

Similarly, for English, let G_{en} contain the following rules (again, nonterminals are in capitals, and S_{en} is the start symbol):

s :	$\mathrm{S}_{en} \to$ NP VP,	np :	NP $\to$ NUM ADJS N,
vp :	VP $\to$ V ADVS,	num_ε :	NUM $\to \varepsilon$,
adjs :	ADJS $\to$ ADJ ADJS,	$\mathrm{adjs}_\varepsilon$:	ADJS $\to \varepsilon$,
advs :	ADVS $\to$ ADV ADVS,	$\mathrm{advs}_\varepsilon$:	ADVS $\to \varepsilon$,
adv :	ADV $\to$ PP,	dict_1 :	N $\to$ *elephants*,
dict_2 :	N $\to$ *monkeys*,	dict_3 :	V $\to$ *came*,
dict_4 :	ADJ $\to$ *pink*,	dict_5 :	NUM $\to$ *two*,
dict_6 :	PP $\to$ *to the lecture*		

Finally, let M_{cz} and M_{en} contain the following matrices:

	M_{cz}	M_{en}		M_{cz}	M_{en}
s :	s	s	np :	np	np
vp :	vp	vp	num :	num_ε	ε
num_ε :	num_ε num_ε	num_ε	$adjs$:	adjs	adjs
$adjs_\varepsilon$:	$\mathrm{adjs}_\varepsilon$ $\mathrm{adjs}_\varepsilon$	$\mathrm{adjs}_\varepsilon$	$advs$:	advs	advs
$advs_\varepsilon$:	$\mathrm{advs}_\varepsilon$ $\mathrm{advs}_\varepsilon$	$\mathrm{advs}_\varepsilon$	n_m :	n_m	ε
n_f :	n_f	ε	v_m :	v_m n_{mm}	ε
v_f :	v_f n_{ff}	ε	adj_m :	adj_m n_{mm}	ε
adj_f :	adj_f n_{ff}	ε	adv :	adv	adv
num_m :	num_m n_{mm}	ε	num_f :	num_f n_{ff}	ε
$dict_1$:	dict_1	dict_1	$dict_2$:	dict_2	dict_2
$dict_{3m}$:	dict_{3m}	dict_3	$dict_{3f}$:	dict_{3f}	dict_3
$dict_{4m}$:	dict_{4m}	dict_4	$dict_{4f}$:	dict_{4f}	dict_4
$dict_{5m}$:	dict_{5m}	dict_5	$dict_{5f}$:	dict_{5f}	dict_5
$dict_6$:	dict_6	dict_6			

In this example, we have chosen to include the words themselves directly in the grammar rules (rather than assuming a separate dictionary) to illustrate this approach as well. For instance, consider the rule dict_{5m} in G_{cz}. This rule encodes the fact that the word *dva* (in Czech) is a numeral, of male gender (in practice, there can be much more information). We call this kind of rules dictionary rules.

Further, note for example the matrix adj_f in M_{cz}, which ensures agreement between noun and adjective (both must be in female gender). Another interesting matrix is $adjs_\varepsilon$, which terminates generation of adjectives. In the Czech sentence in this example, we have two positions where adjectives can be placed (directly within the noun phrase or at the end of the sentence). In English, there is only one possible position (within the noun phrase). This is why the rule ADJS $\to \varepsilon$ is used twice in Czech, but only once in English.

Also observe that the linked matrices (sharing the same label) in M_{cz} and M_{en} may contain completely different rules and they can even be empty (ε), in which case the corresponding grammar does not change its sentential form in that step. The definitions of MAT and SMAT allow for this kind of flexibility when describing both individual languages and their translations.

Example of a derivation in Czech follows.

$$
\begin{aligned}
\text{S}_{cz} &\Rightarrow \text{NP VP NUM ADJS } [s] \\
&\Rightarrow \text{NUM ADJS N VP NUM ADJS } [np] \\
&\Rightarrow \text{NUM ADJS N ADVS V ADVS NUM ADJS } [vp] \\
&\Rightarrow \text{ADJS N ADVS V ADVS NUM ADJS } [num] \\
&\Rightarrow \text{ADJ ADJS N ADVS V ADVS NUM ADJS } [adjs] \\
&\Rightarrow \text{ADJ N ADVS V ADVS NUM } [adjs_\varepsilon] \\
&\Rightarrow \text{ADJ N ADVS V ADV ADVS NUM } [advs] \\
&\Rightarrow \text{ADJ N V ADV NUM } [advs_\varepsilon] \\
&\Rightarrow \text{ADJ N}_m \text{ V ADV NUM } [n_m] \\
&\Rightarrow \text{ADJ N}_m \text{ V}_m \text{ ADV NUM } [v_m] \\
&\Rightarrow \text{ADJ}_m \text{ N}_m \text{ V}_m \text{ ADV NUM } [adj_m] \\
&\Rightarrow \text{ADJ}_m \text{ N}_m \text{ V}_m \text{ PP NUM } [adv] \\
&\Rightarrow \text{ADJ}_m \text{ N}_m \text{ V}_m \text{ PP NUM}_m \ [num_m] \\
&\Rightarrow \text{ADJ}_m \textit{ sloni } \text{V}_m \text{ PP NUM}_m \ [dict_1] \\
&\Rightarrow \text{ADJ}_m \textit{ sloni přišli } \text{PP NUM}_m \ [dict_{3m}] \\
&\Rightarrow \textit{růžoví sloni přišli } \text{PP NUM}_m \ [dict_{4m}] \\
&\Rightarrow \textit{růžoví sloni přišli na přednášku } \text{NUM}_m \ [dict_5] \\
&\Rightarrow \textit{růžoví sloni přišli na přednášku dva } [dict_{6m}]
\end{aligned}
$$

The corresponding derivation in English may look like this:

$$
\begin{aligned}
\text{S}_{en} &\Rightarrow \text{NP VP } [s] \\
&\Rightarrow \text{NUM ADJS N VP } [np] \\
&\Rightarrow \text{NUM ADJS N V ADVS } [vp] \\
&\Rightarrow \text{NUM ADJS N V ADVS } [num] \\
&\Rightarrow \text{NUM ADJ ADJS N V ADVS } [adjs] \\
&\Rightarrow \text{NUM ADJ N V ADVS } [adjs_\varepsilon] \\
&\Rightarrow \text{NUM ADJ N V ADV ADVS } [advs] \\
&\Rightarrow \text{NUM ADJ N V ADV } [advs_\varepsilon] \\
&\Rightarrow \text{NUM ADJ N V ADV } [n_m] \\
&\Rightarrow \text{NUM ADJ N V ADV } [v_m]
\end{aligned}
$$

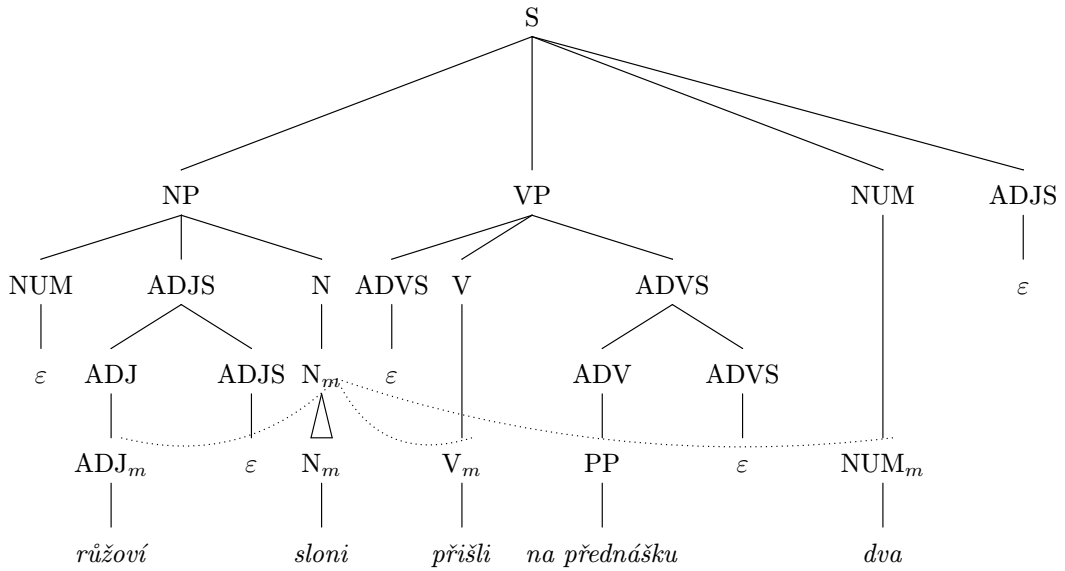


Figure 6.4: Derivation tree of G_{cz}

$\Rightarrow$ NUM ADJ N V ADV $[adj_m]$
$\Rightarrow$ NUM ADJ N V PP $[adv]$
$\Rightarrow$ NUM ADJ N V PP $[num_m]$
$\Rightarrow$ NUM ADJ *elephants* V_m PP $[dict_1]$
$\Rightarrow$ NUM ADJ *elephants came* PP $[dict_{3m}]$
$\Rightarrow$ NUM *pink elephants came* PP $[dict_{4m}]$
$\Rightarrow$ NUM *pink elephants came to the lecture* $[dict_5]$
$\Rightarrow$ *two pink elephants came to the lecture* $[dict_{6m}]$

The entire derivation tree for the Czech sentence is shown in Figure 6.4. The dotted lines represent relations described by matrices. The triangle from N_m to N_m is an abstraction which in this particular case essentially means that this step is repeated until all agreement issues are resolved.

We can achieve similar results using SSCGs. For example the matrix adj_f in M_{cz} can be represented by two scattered-context rules (ADJ, N_f) $\rightarrow$ (ADJ_f, N_f) and (N_f, ADJ) $\rightarrow$ (N_f, ADJ_f). Note that we need two rules, because the nonterminal order is important in SCG (this is one of the key differences between SMAT and SSCG). In this case, we need an additional rule in SSCG. However, this can also be an advantage, because it allows us to easily distinguish between left and right modifiers. For example, if we only have the first rule (ADJ, N_f) $\rightarrow$ (ADJ_f, N_f), it means that the adjective always has to occur on the left of the noun.

6.2 Rule-Restricted Transducers

In this section, we discuss the applications perspectives of RTs. The original results, observations, and examples presented here were first published in [10].

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

The cat caught the mouse.

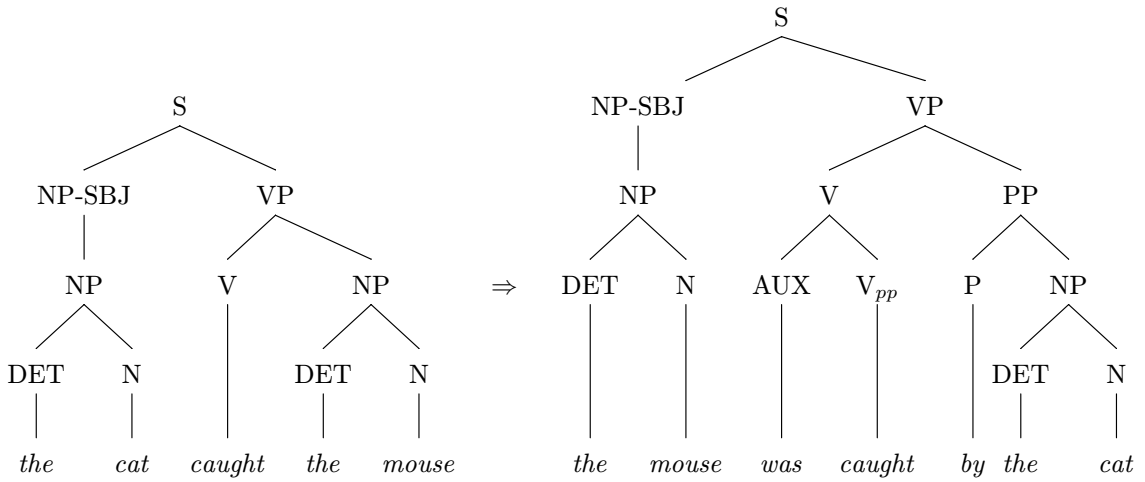


Figure 6.5: Example of the passive transformation in English

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 6.5 shows the corresponding syntax trees. Essentially, what we need to do is the following:

1. swap the subject and the object,
2. add the preposition *by* in correct position, and
3. change the verb into passive form, using the auxiliary verb *to be* in appropriate form.

The verb *to be* is irregular and has many different forms (paradigms) depending not only on tense, but also on person and number (see Table 6.1 on page 57). 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).

Example 6.4. Consider an RT $\Gamma = (M, G, \Psi)$, where $M = (Q, \Sigma, \delta, 0, F)$, $G = (N, T, P,$ S) such that

- $Q = \{0, 1, 2, 3, 4, 5, 6, 7, 8_a, 8_b, 8_c, 8_d, 8_e, 8_f, 9\}$,
- $\Sigma = \{\mathrm{N}_{1s}, \mathrm{N}_{2s}, \mathrm{N}_{3s}, \mathrm{N}_{1p}, \mathrm{N}_{2p}, \mathrm{N}_{3p}, \mathrm{V}_{pas}, \mathrm{V}_{ps}, \mathrm{V}_{pp}, \mathrm{DET}, \mathrm{P}, \mathrm{AUX}_{pas1s}, \mathrm{AUX}_{pas2s}, \mathrm{AUX}_{pas3s}, \mathrm{AUX}_{pas1p}, \mathrm{AUX}_{pas2p}, \mathrm{AUX}_{pas3p}, \mathrm{AUX}_{ps1s}, \mathrm{AUX}_{ps2s}, \mathrm{AUX}_{ps3s}, \mathrm{AUX}_{ps1p}, \mathrm{AUX}_{ps2p}, \mathrm{AUX}_{ps3p}\}$,
- $F = \{9\}$,
- $N = \{\mathrm{S}, \mathrm{NP\text{-}SBJ}, \mathrm{NP}, \mathrm{VP}, \mathrm{PP}, \mathrm{N}_?, \mathrm{V}_?, \mathrm{AUX}_{pas?}, \mathrm{AUX}_{ps?}\}$,
- $T = \Sigma$,

- $\delta = \left\{ \begin{array}{lll} r_1 : 0 \to 1, & r_2 : 1 \to 2, & r_3 : 2 \to 3, \\ r_4 : 3\text{DET} \to 4, & r_{5a} : 4\text{N}_{1s} \to 3, & r_{5b} : 4\text{N}_{2s} \to 3, \\ r_{5c} : 4\text{N}_{3s} \to 3, & r_{5d} : 4\text{N}_{1p} \to 3, & r_{5e} : 4\text{N}_{2p} \to 3, \\ r_{5f} : 4\text{N}_{3p} \to 3, & r_{6a} : 3\text{V}_{pas} \to 5, & r_{6b} : 3\text{V}_{ps} \to 5, \\ r_7 : 5 \to 6, & r_8 : 6\text{DET} \to 7, & r_{9a} : 7\text{N}_{1s} \to 8_a, \\ r_{9b} : 7\text{N}_{2s} \to 8_b, & \ldots & r_{9f} : 7\text{N}_{3p} \to 8_f, \\ r_{10a} : 8_a \to 9, & r_{10b} : 8_b \to 9, & r_{10f} : 8_f \to 9 \end{array} \right\}$,

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

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

For the sentence *the cat caught the mouse* from the above example, the computation can proceed as follows:

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

For clarity, in each computation step, the input symbol to be read (if any) and the nonterminal to be rewritten are underlined. Moreover, the text in angled brackets ($\langle\rangle$) 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 performed so far (for example, we cannot assign a word before we read the corresponding input token).

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 $\text{AUX}_{pas?}$. Later, when we read the object of the original sentence, we rewrite $\text{AUX}_{pas?}$ 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 and the form of the words reflects many grammatical categories, such as case, gender, or number (see [31], 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, namely *onna no hito* (*woman*) and *otoko no hito* (*man*).[2]

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 syntax trees in Figure 6.6.

[2]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. This is analogous for *otoko no hito*.

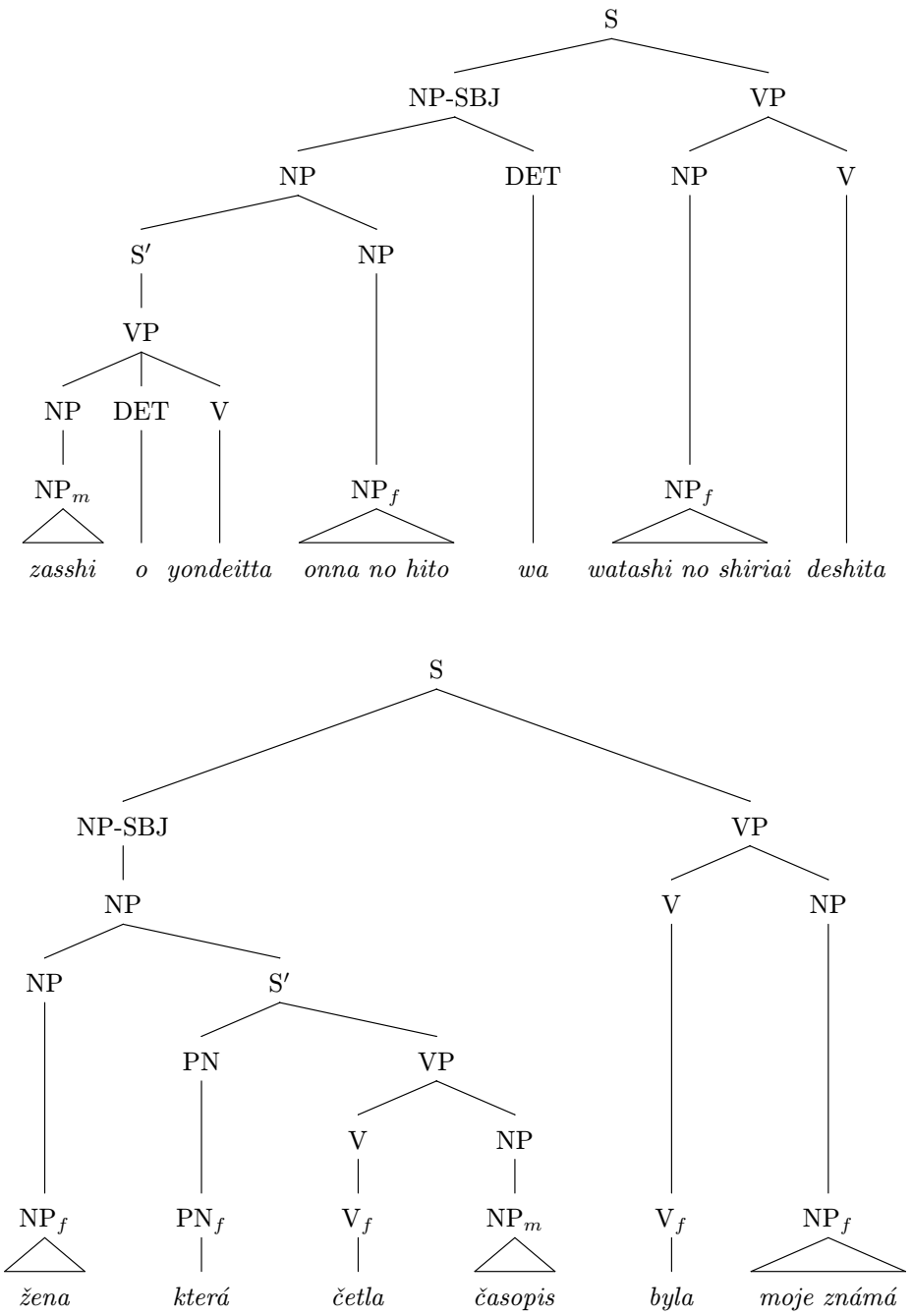


Figure 6.6: Syntax trees for Japanese (top) and Czech sentence

Example 6.5. Consider an RT $\Gamma = (M, G, \Psi)$, where $M = (Q, \Sigma, \delta, 0, F)$, $G = (N, T, P,$ S) such that

- $Q = \{0, m, m1, m2, f, f1, f2, n, n1, n2, 1m, 1f, 1n\}$,
- $\Sigma = \{\mathrm{NP}_m, \mathrm{NP}_f, \mathrm{NP}_n, \mathrm{NP}_?, \mathrm{V}, \mathrm{DET}, \#\}$,
- $F = \{1m, 1f, 1n\}$,
- $N = \{\mathrm{S}, \mathrm{S}', \text{NP-SBJ}, \mathrm{NP}_?, \mathrm{VP}, \mathrm{PN}_?, \mathrm{V}_?, \mathrm{X}\}$,
- $T = \{\mathrm{NP}_m, \mathrm{NP}_f, \mathrm{NP}_n, \mathrm{V}_m, \mathrm{V}_f, \mathrm{V}_n, \mathrm{PN}_m, \mathrm{PN}_f, \mathrm{PN}_n\}$,
- $\delta = \left\{ \begin{array}{lll} r_1 : 0\mathrm{V} \to 1, & r_2 : 1 \to 0, & r_3 : 0\mathrm{NP}_? \to 0, \\ r_4 : 0\mathrm{DET} \to 0, & r_{5m} : 0\mathrm{NP}_m \to m, & r_{5f} : 0\mathrm{NP}_f \to f, \\ r_{5n} : 0\mathrm{NP}_n \to n, & r_{m1} : m\mathrm{V} \to m1, & r_{m2} : m1 \to m2, \\ r_{m3} : m2 \to m, & r_{m4} : m\mathrm{DET} \to m, & r_{m5} : m\mathrm{NP}_? \to m, \\ r_{m5m} : m\mathrm{NP}_m \to m, & r_{m5f} : m\mathrm{NP}_f \to m, & r_{m5n} : m\mathrm{NP}_n \to m, \\ r_{m6} : m\# \to 1m, & r_{m7} : 1m \to 1m, & r_{f1} : f\mathrm{V} \to f1, \\ r_{f2} : f1 \to f2, & \ldots & r_{f7} : 1f \to 1f, \\ r_{n1} : n\mathrm{V} \to n1, & r_{n2} : n1 \to n2, & \ldots \\ r_{n7} : 1n \to 1n & & \end{array} \right\}$,
- $P = \left\{ \begin{array}{ll} p_1 : \mathrm{S} \to \text{NP-SBJ VP X}, & p_2 : \text{NP-SBJ} \to \mathrm{NP}_?, \\ p_{3m} : \mathrm{NP}_? \to \mathrm{NP}_m, & p_{3f} : \mathrm{NP}_? \to \mathrm{NP}_f, \\ p_{3n} : \mathrm{NP}_? \to \mathrm{NP}_n, & p_4 : \mathrm{NP}_? \to \mathrm{NP}_?, \\ p_5 : \mathrm{NP}_? \to \mathrm{NP}_? \ \mathrm{S}', & p_6 : \mathrm{VP} \to \mathrm{V}_? \ \mathrm{NP}_?, \\ p_{7m} : \mathrm{V}_? \to \mathrm{V}_m, & p_{7f} : \mathrm{V}_? \to \mathrm{V}_f, \\ p_{7n} : \mathrm{V}_? \to \mathrm{V}_n, & p_8 : \mathrm{S}' \to \mathrm{PN}_? \ \mathrm{VP}, \\ p_{9m} : \mathrm{PN}_? \to \mathrm{PN}_m, & p_{9f} : \mathrm{PN}_? \to \mathrm{PN}_f, \\ p_{9n} : \mathrm{PN}_? \to \mathrm{PN}_n, & p_{10} : \mathrm{X} \to \varepsilon \end{array} \right\}$,
- $\Psi = \{(r_1, p_1), (r_2, p_6), (r_3, p_4), (r_4, p_2), (r_{5m}, p_4), (r_{5f}, p_4), (r_{5n}, p_4), (r_{m1}, p_5), (r_{m2}, p_8), (r_{m3}, p_6), (r_{m4}, p_4), (r_{m5}, p_4), (r_{m5m}, p_{3m}), (r_{m5f}, p_{3f}), (r_{m5n}, p_{3n}), (r_{m6}, p_{10}), (r_{m7}, p_{3m}), (r_{m7}, p_{7m}), (r_{m7}, p_{9m}), (r_{f1}, p_5), (r_{f2}, p_8), (r_{f3}, p_6), (r_{f4}, p_4), (r_{f5}, p_4), (r_{f5m}, p_{3m}), (r_{f5f}, p_{3f}), (r_{f5n}, p_{3n}), (r_{f6}, p_{10}), (r_{f7}, p_{3f}), (r_{f7}, p_{7f}), (r_{f7}, p_{9f}), (r_{n1}, p_5), (r_{n2}, p_8), (r_{n3}, p_6), (r_{n4}, p_4), (r_{n5}, p_4), (r_{n5m}, p_{3m}), (r_{n5f}, p_{3f}), (r_{n5n}, p_{3n}), (r_{n6}, p_{10}), (r_{n7}, p_{3n}), (r_{n7}, p_{7n}), (r_{n7}, p_{9n})\}$.

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 can 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:

$(0\ \underline{\mathrm{V}\langle \textit{deshita} \rangle}\ \mathrm{NP}_?\langle \textit{watashi no shiriai} \rangle\ \mathrm{DET}\langle \textit{wa} \rangle\ \mathrm{NP}_f\langle \textit{onna no hito} \rangle$
$\mathrm{V}\langle \textit{yondeitta} \rangle\ \mathrm{DET}\langle o \rangle\ \mathrm{NP}_m\langle \textit{zasshi} \rangle\ \#,\ \underline{\mathrm{S}})$

$\Rightarrow$ $(1\ \mathrm{NP}_?\langle \textit{watashi no shiriai} \rangle\ \mathrm{DET}\langle \textit{wa} \rangle\ \mathrm{NP}_f\langle \textit{onna no hito} \rangle\ \mathrm{V}\langle \textit{yondeitta} \rangle$
$\mathrm{DET}\langle o \rangle\ \mathrm{NP}_m\langle \textit{zasshi} \rangle\ \#,\ \text{NP-SBJ}\ \underline{\mathrm{VP}}\ \mathrm{X})\ [(r_1, p_1)]$

$\Rightarrow$ $(0\ \underline{\mathrm{NP}_?\langle \textit{watashi no shiriai} \rangle}\ \mathrm{DET}\langle \textit{wa} \rangle\ \mathrm{NP}_f\langle \textit{onna no hito} \rangle\ \mathrm{V}\langle \textit{yondeitta} \rangle$
$\mathrm{DET}\langle o \rangle\ \mathrm{NP}_m\langle \textit{zasshi} \rangle\ \#,\ \text{NP-SBJ}\ \mathrm{V}_?\langle \textit{byl} \rangle\ \underline{\mathrm{NP}_?}\ \mathrm{X})\ [(r_2, p_6)]$

$\Rightarrow$ $(0\ \underline{\mathrm{DET}\langle \textit{wa} \rangle}\ \mathrm{NP}_f\langle \textit{onna no hito} \rangle\ \mathrm{V}\langle \textit{yondeitta} \rangle\ \mathrm{DET}\langle o \rangle\ \mathrm{NP}_m\langle \textit{zasshi} \rangle\ \#,$
$\underline{\text{NP-SBJ}}\ \mathrm{V}_?\langle \textit{byl} \rangle\ \mathrm{NP}_?\langle \textit{můj známý} \rangle\ \mathrm{X})\ [(r_3, p_4)]$

$\Rightarrow$ $(0\ \underline{\mathrm{NP}_f\langle \textit{onna no hito} \rangle}\ \mathrm{V}\langle \textit{yondeitta} \rangle\ \mathrm{DET}\langle o \rangle\ \mathrm{NP}_m\langle \textit{zasshi} \rangle\ \#,\ \underline{\mathrm{NP}_?}$
$\mathrm{V}_?\langle \textit{byl} \rangle\ \mathrm{NP}_?\langle \textit{můj známý} \rangle\ \mathrm{X})\ [(r_4, p_2)]$

$\Rightarrow$ $(f\ \underline{\mathrm{V}\langle \textit{yondeitta} \rangle}\ \mathrm{DET}\langle o \rangle\ \mathrm{NP}_m\langle \textit{zasshi} \rangle\ \#,\ \underline{\mathrm{NP}_?\langle \textit{žena} \rangle}\ \mathrm{V}_?\langle \textit{byl} \rangle$
$\mathrm{NP}_?\langle \textit{můj známý} \rangle\ \mathrm{X})\ [(r_{5f}, p_4)]$

$\Rightarrow$ $(f1\ \mathrm{DET}\langle o \rangle\ \mathrm{NP}_m\langle \textit{zasshi} \rangle\ \#,\ \mathrm{NP}_?\langle \textit{žena} \rangle\ \underline{\mathrm{S}'}\ \mathrm{V}_?\langle \textit{byl} \rangle\ \mathrm{NP}_?\langle \textit{můj známý} \rangle\ \mathrm{X})$
$[(r_{f1}, p_5)]$

$\Rightarrow$ $(f2\ \mathrm{DET}\langle o \rangle\ \mathrm{NP}_m\langle \textit{zasshi} \rangle\ \#,\ \mathrm{NP}_?\langle \textit{žena} \rangle\ \mathrm{PN}_?\langle \textit{který} \rangle\ \underline{\mathrm{VP}}\ \mathrm{V}_?\langle \textit{byl} \rangle$
$\mathrm{NP}_?\langle \textit{můj známý} \rangle\ \mathrm{X})\ [(r_{f2}, p_8)]$

$\Rightarrow$ $(f\ \underline{\mathrm{DET}\langle o \rangle}\ \mathrm{NP}_m\langle \textit{zasshi} \rangle\ \#,\ \mathrm{NP}_?\langle \textit{žena} \rangle\ \mathrm{PN}_?\langle \textit{který} \rangle\ \mathrm{V}_?\langle \textit{četl} \rangle\ \underline{\mathrm{NP}_?}\ \mathrm{V}_?\langle \textit{byl} \rangle$
$\mathrm{NP}_?\langle \textit{můj známý} \rangle\ \mathrm{X})\ [(r_{f3}, p_6)]$

$\Rightarrow$ $(f\ \underline{\mathrm{NP}_m\langle \textit{zasshi} \rangle}\ \#,\ \mathrm{NP}_?\langle \textit{žena} \rangle\ \mathrm{PN}_?\langle \textit{který} \rangle\ \mathrm{V}_?\langle \textit{četl} \rangle\ \underline{\mathrm{NP}_?}\ \mathrm{V}_?\langle \textit{byl} \rangle$
$\mathrm{NP}_?\langle \textit{můj známý} \rangle\ \mathrm{X})\ [(r_{f4}, p_4)]$

$\Rightarrow$ $(f\ \underline{\#},\ \mathrm{NP}_?\langle \textit{žena} \rangle\ \mathrm{PN}_?\langle \textit{který} \rangle\ \mathrm{V}_?\langle \textit{četl} \rangle\ \mathrm{NP}_m\langle \textit{časopis} \rangle\ \mathrm{V}_?\langle \textit{byl} \rangle$
$\mathrm{NP}_?\langle \textit{můj známý} \rangle\ \underline{\mathrm{X}})\ [(r_{f5m}, p_{3m})]$

$\Rightarrow$ $(1f,\ \underline{\mathrm{NP}_?\langle \textit{žena} \rangle}\ \mathrm{PN}_?\langle \textit{který} \rangle\ \mathrm{V}_?\langle \textit{četl} \rangle\ \mathrm{NP}_m\langle \textit{časopis} \rangle\ \mathrm{V}_?\langle \textit{byl} \rangle$
$\mathrm{NP}_?\langle \textit{můj známý} \rangle)\ [(r_{f6}, p_{10})]$

$\Rightarrow$ $(1f,\ \mathrm{NP}_f\langle \textit{žena} \rangle\ \underline{\mathrm{PN}_?\langle \textit{který} \rangle}\ \mathrm{V}_?\langle \textit{četl} \rangle\ \mathrm{NP}_m\langle \textit{časopis} \rangle\ \mathrm{V}_?\langle \textit{byl} \rangle$
$\mathrm{NP}_?\langle \textit{můj známý} \rangle)\ [(r_{f7}, p_{3f})]$

$\Rightarrow$ $(1f,\ \mathrm{NP}_f\langle \textit{žena} \rangle\ \mathrm{PN}_f\langle \textit{která} \rangle\ \underline{\mathrm{V}_?\langle \textit{četl} \rangle}\ \mathrm{NP}_m\langle \textit{časopis} \rangle\ \mathrm{V}_?\langle \textit{byl} \rangle$
$\mathrm{NP}_?\langle \textit{můj známý} \rangle)\ [(r_{f7}, p_{9f})]$

$\Rightarrow$ $(1f,\ \mathrm{NP}_f\langle \textit{žena} \rangle\ \mathrm{PN}_f\langle \textit{která} \rangle\ \mathrm{V}_f\langle \textit{četla} \rangle\ \mathrm{NP}_m\langle \textit{časopis} \rangle\ \underline{\mathrm{V}_?\langle \textit{byl} \rangle}$
$\mathrm{NP}_?\langle \textit{můj známý} \rangle)\ [(r_{f7}, p_{7f})]$

$\Rightarrow$ $(1f,\ \mathrm{NP}_f\langle \textit{žena} \rangle\ \mathrm{PN}_f\langle \textit{která} \rangle\ \mathrm{V}_f\langle \textit{četla} \rangle\ \mathrm{NP}_m\langle \textit{časopis} \rangle\ \mathrm{V}_f\langle \textit{byla} \rangle$
$\underline{\mathrm{NP}_?\langle \textit{můj známý} \rangle})\ [(r_{f7}, p_{7f})]$

$\Rightarrow$ $(1f,\ \mathrm{NP}_f\langle \textit{žena} \rangle\ \mathrm{PN}_f\langle \textit{která} \rangle\ \mathrm{V}_f\langle \textit{četla} \rangle\ \mathrm{NP}_m\langle \textit{časopis} \rangle\ \mathrm{V}_f\langle \textit{byla} \rangle$
$\mathrm{NP}_f\langle \textit{moje známá} \rangle)\ [(r_{f7}, p_{3f})]$

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.

As already noted in the previous section, Czech is considered a free-word-order language. That is, 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.

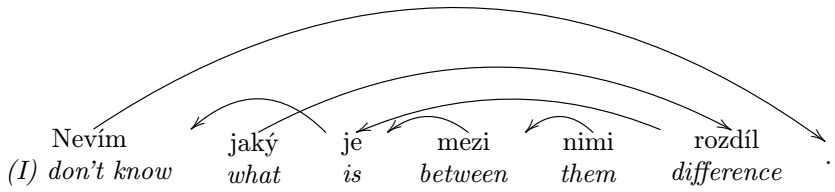


Figure 6.7: Non-projective dependency tree (Czech)

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

I ate a cake yesterday which was delicious.

from Section 3.1.1 and its dependency tree shown in Figure 3.2. 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 6.7) is also non-projective.

For further information about projectivity, and the issue of non-projectivity in the Czech language in particular, see [32].

The following Example 6.6 illustrates how our formalism can account for some non-projectivity.

Example 6.6. Consider an RT $\Gamma = (M, G, \Psi)$, where $M = (Q, \Sigma, \delta, 0, F)$, $G = (N, T, P, \mathrm{S})$ such that

- $Q = \{0, 1, 2, 3, 4, 5, 6m, 6f, 6n\}$,
- $\Sigma = \{\mathrm{N}_m, \mathrm{N}_f, \mathrm{N}_n, \mathrm{V}, \mathrm{PN}, \mathrm{PN}_i, \mathrm{DET}, \mathrm{P}\}$,
- $F = \{0\}$,
- $N = \{\mathrm{S}, \mathrm{NP\text{-}SBJ}, \mathrm{NP}, \mathrm{VP}, \mathrm{PP}, \mathrm{V}_?\}$,
- $T = \{\mathrm{N}_m, \mathrm{N}_f, \mathrm{N}_n, \mathrm{V}, \mathrm{PN}_m, \mathrm{PN}_f, \mathrm{PN}_n, \mathrm{P}\}$,
- $\delta = \left\{ \begin{array}{lll} r_{1m} : 0\mathrm{N}_m \to 6m, & r_{1f} : 0\mathrm{N}_f \to 6f, & r_{1n} : 0\mathrm{N}_n \to 6n, \\ r_2 : 0\mathrm{V} \to 0, & r_3 : 0\mathrm{PN} \to 1, & r_4 : 0\mathrm{PN}_i \to 3, \\ r_5 : 0\mathrm{DET} \to 0, & r_6 : 0\mathrm{P} \to 0, & r_7 : 1 \to 2, \\ r_8 : 2 \to 0, & r_9 : 3 \to 4, & r_{10} : 4 \to 5, \\ r_{11} : 5 \to 0, & r_{12m} : 6m \to 0, & r_{12f} : 6f \to 0, \\ r_{12n} : 6n \to 0, & r_{13} : 0\mathrm{PN} \to 0 & \end{array} \right\}$,

- $P = \left\{ \begin{array}{lllll} p_1 & : & \text{S} \rightarrow \text{NP-SBJ VP}, & p_2 & : & \text{NP-SBJ} \rightarrow \text{NP}, \\ p_3 & : & \text{NP-SBJ} \rightarrow \varepsilon, & p_{4m} & : & \text{NP} \rightarrow \text{N}_m, \\ p_{4f} & : & \text{NP} \rightarrow \text{N}_f, & p_{4n} & : & \text{NP} \rightarrow \text{N}_n, \\ p_5 & : & \text{NP} \rightarrow \text{S}, & p_{6m} & : & \text{NP} \rightarrow \text{PN}_m, \\ p_{6f} & : & \text{NP} \rightarrow \text{PN}_f, & p_{6n} & : & \text{NP} \rightarrow \text{PN}_n, \\ p_7 & : & \text{VP} \rightarrow \text{V}_? \text{ PP NP}, & p_8 & : & \text{V}_? \rightarrow \text{V}, \\ p_9 & : & \text{PP} \rightarrow \text{P NP}, & p_{10} & : & \text{PP} \rightarrow \varepsilon, \end{array} \right\}$,
- $\Psi = \{(r_{1m}, p_{4m}), (r_{1f}, p_{4f}), (r_{1n}, p_{4n}), (r_2, p_8), (r_3, p_1), (r_4, p_{10}), (r_5, p_7), (r_6, p_9), (r_7, p_3), (r_8, p_7), (r_9, p_5), (r_{10}, p_1), (r_{11}, p_2), (r_{12m}, p_{6m}), (r_{12f}, p_{6f}), (r_{12n}, p_{6n}), (r_{13}, p_{6m})\}$.

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 $\underline{\text{PN}\langle I\rangle}$ V⟨*don't know*⟩ $\text{PN}_i\langle what\rangle$ DET⟨*the*⟩ $\text{N}_m\langle difference\rangle$ P⟨*between*⟩ PN⟨*them*⟩ V⟨*is*⟩, $\underline{\text{S}}$)

⇒ (1 V⟨*don't know*⟩ $\text{PN}_i\langle what\rangle$ DET⟨*the*⟩ $\text{N}_m\langle difference\rangle$ P⟨*between*⟩ PN⟨*them*⟩ V⟨*is*⟩, $\underline{\text{NP-SBJ}}$ VP) $[(r_3, p_1)]$

⇒ (2 V⟨*don't know*⟩ $\text{PN}_i\langle what\rangle$ DET⟨*the*⟩ $\text{N}_m\langle difference\rangle$ P⟨*between*⟩ PN⟨*them*⟩ V⟨*is*⟩, $\underline{\text{VP}}$) $[(r_7, p_3)]$

⇒ (0 $\underline{\text{V}\langle don't\ know\rangle}$ $\text{PN}_i\langle what\rangle$ DET⟨*the*⟩ $\text{N}_m\langle difference\rangle$ P⟨*between*⟩ PN⟨*them*⟩ V⟨*is*⟩, $\underline{\text{V}_?}$ PP NP) $[(r_8, p_7)]$

⇒ (0 $\underline{\text{PN}_i\langle what\rangle}$ DET⟨*the*⟩ $\text{N}_m\langle difference\rangle$ P⟨*between*⟩ PN⟨*them*⟩ V⟨*is*⟩, V⟨*nevím*⟩ $\underline{\text{PP}}$ NP) $[(r_2, p_8)]$

⇒ (3 DET⟨*the*⟩ $\text{N}_m\langle difference\rangle$ P⟨*between*⟩ PN⟨*them*⟩ V⟨*is*⟩, V⟨*nevím*⟩ $\underline{\text{NP}}$) $[(r_4, p_{10})]$

⇒ (4 DET⟨*the*⟩ $\text{N}_m\langle difference\rangle$ P⟨*between*⟩ PN⟨*them*⟩ V⟨*is*⟩, V⟨*nevím*⟩ $\underline{\text{S}}$) $[(r_9, p_5)]$

⇒ (5 DET⟨*the*⟩ $\text{N}_m\langle difference\rangle$ P⟨*between*⟩ PN⟨*them*⟩ V⟨*is*⟩, V⟨*nevím*⟩ $\underline{\text{NP-SBJ}}$ VP) $[(r_{10}, p_1)]$

⇒ (0 $\underline{\text{DET}\langle the\rangle}$ $\text{N}_m\langle difference\rangle$ P⟨*between*⟩ PN⟨*them*⟩ V⟨*is*⟩, V⟨*nevím*⟩ NP⟨*jaký*⟩ $\underline{\text{VP}}$) $[(r_{11}, p_2)]$

⇒ (0 $\underline{\text{N}_m\langle difference\rangle}$ P⟨*between*⟩ PN⟨*them*⟩ V⟨*is*⟩, V⟨*nevím*⟩ NP⟨*jaký*⟩ $\text{V}_?$ PP $\underline{\text{NP}}$) $[(r_5, p_7)]$

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

⇒ (0 $\underline{\text{P}\langle between\rangle}$ PN⟨*them*⟩ V⟨*is*⟩, V⟨*nevím*⟩ $\text{PN}_m\langle jaký\rangle$ $\text{V}_?$ $\underline{\text{PP}}$ $\text{N}_m\langle rozdíl\rangle$) $[(r_{12m}, p_{6m})]$

⇒ (0 $\underline{\text{PN}\langle them\rangle}$ V⟨*is*⟩, V⟨*nevím*⟩ $\text{PN}_m\langle jaký\rangle$ $\text{V}_?$ P⟨*mezi*⟩ $\underline{\text{NP}}$ $\text{N}_m\langle rozdíl\rangle$) $[(r_6, p_9)]$

⇒ (0 $\underline{\text{V}\langle is\rangle}$, V⟨*nevím*⟩ $\text{PN}_m\langle jaký\rangle$ $\underline{\text{V}_?}$ P⟨*mezi*⟩ $\text{PN}_m\langle nimi\rangle$ $\text{N}_m\langle rozdíl\rangle$) $[(r_{13}, p_{6a})]$

⇒ (0, V⟨*nevím*⟩ $\text{PN}_m\langle jaký\rangle$ V⟨*je*⟩ P⟨*mezi*⟩ $\text{PN}_m\langle nimi\rangle$ $\text{N}_m\langle rozdíl\rangle$) $[(r_2, p_8)]$

The corresponding derivation tree of G is shown in Figure 6.8.

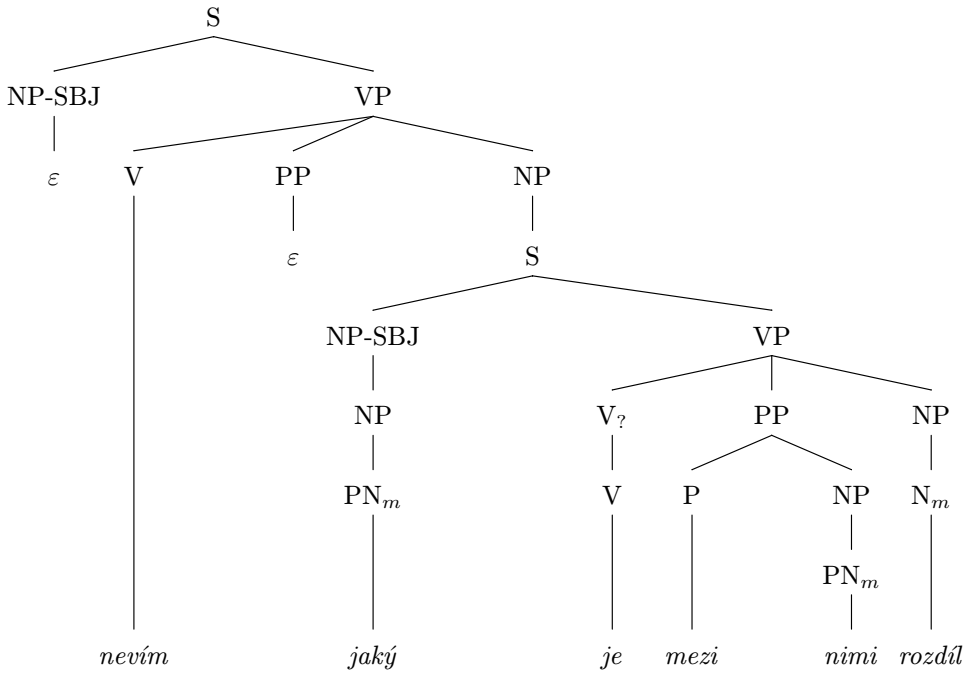


Figure 6.8: Derivation tree of G

6.3 Summary

In the first two sections of this chapter, we have tried to point out and illustrate the key advantages of the proposed formal models using select case studies from the Czech, English, and Japanese language. Here, we summarize them, and compare the respective strengths and weaknesses of the new synchronous grammars and RTs. The observations presented are based on our previously published papers [10], [39], and [41].

One of the main advantages of both types of models is their power. As shown in Chapters 4 and 5, both synchronous grammars (with linked rules) and RTs (without leftmost restriction) are able to describe even some non-context-free languages. Although arguably relatively rare in practice, there are some features of natural languages that are difficult or impossible to properly capture with CFGs only (such as cross-dependencies). Furthermore, even in cases when a purely context-free description is possible, it may require a high number of rules. Our new models can provide a more economical description thanks to their increased generative power and, in case of RTs, also accepting power.

Another advantage of our new synchronous grammars is their high flexibility, especially if we synchronize models that have higher generative power themselves, such as regulated grammars. In particular, let us consider the case of SMAT. As shown above (Theorem 4.7), if we synchronize MATs in the proposed fashion, we do not obtain any further increase in power of the whole system compared to RSCFG or MAT. However, more powerful individual components allow for easier—and again, more economical—description of each individual language.

Unlike synchronous grammars, which are symmetric and therefore can be used for bidirectional translation, RTs can only describe translation in one direction. Furthermore, because their components are relatively simple (an FA and a CFG), RTs are also less flexible than, for example, SMATs and SSCGs. Consequently, the description of linguistic

structures and features can be more complex (essentially, requiring more rules).

On the other hand, the simplicity of components can also be seen as an important advantage of RT, especially from a practical viewpoint. Both FAs and CFGs are well-known and well-studied not only from a theoretical point of view, but also with regards to practical implementations. For example, there are well-known methods of efficient parsing for CFGs.

Another advantage of RT lies in its the straightforward and intuitive basic principle (read input with an FA, generate output with a CFG), which directly corresponds to the translation task in practice. In contrast, in synchronous grammars, both components generate sentences.

Finally, note that both types of introduced formal models can be extended for use in statistical natural processing as well. We can, for example, assign weights (or probabilities) to rules similarly to probabilistic CFGs or weighted synchronous grammars.

Chapter 7

Conclusion

In this doctoral thesis, we have presented new grammar systems that can formally describe translations (or, more specifically, transformations of syntactic structures). We have discussed some of the theoretical properties of the new models, in particular their generative and accepting power.

More specifically, we have introduced the idea of synchronization based on linked rules as a modification of the well-known synchronous grammars. We have extended this principle beyond CFGs, to models with regulated rewriting, defining sychronous MATs and synchronous SCGs.

Further, we have introduced the rule-restricted automaton-grammar transducer, based on the natural idea of reading some input with an FA and producing an appropriate output with a CFG, and provided precise formal definitons. We have also considered two of its variants, namely leftmost restricted RTs and RTs with appearance checking.

We have established the following main results:

1. Rule-synchronized CFGs are more powerful than CFGs, as they characterize the same class of languages as MATs (see Section 4.1).

2. Synchronous MATs have the same power as MATs (see Section 4.3).

3. Synchronous SCGs are able to generate all recursively enumerable languages (see Section 4.2).

4. RTs can generate any language that can be generated by some MAT, and they can accept any language that can be accepted by some k-PBCA (see Section 5.1).

5. Leftmost restricted RTs can only accept and generate context-free languages (see Section 5.2). Note that this is still an increase in accepting power compared to FAs.

6. RTs with appearance checking can both accept and generate all recursively enumerable languages (see Section 5.3).

Figure 7.1 summarizes the results in a graphical representation.

We have also discussed application perspectives of the new models in translation of natural languages, using select case studies from Czech, English, and Japanese to illustrate (see Chapter 6). Besides natural language processing, the models can be useful in other translation and transformation tasks, such as programming language compilation.

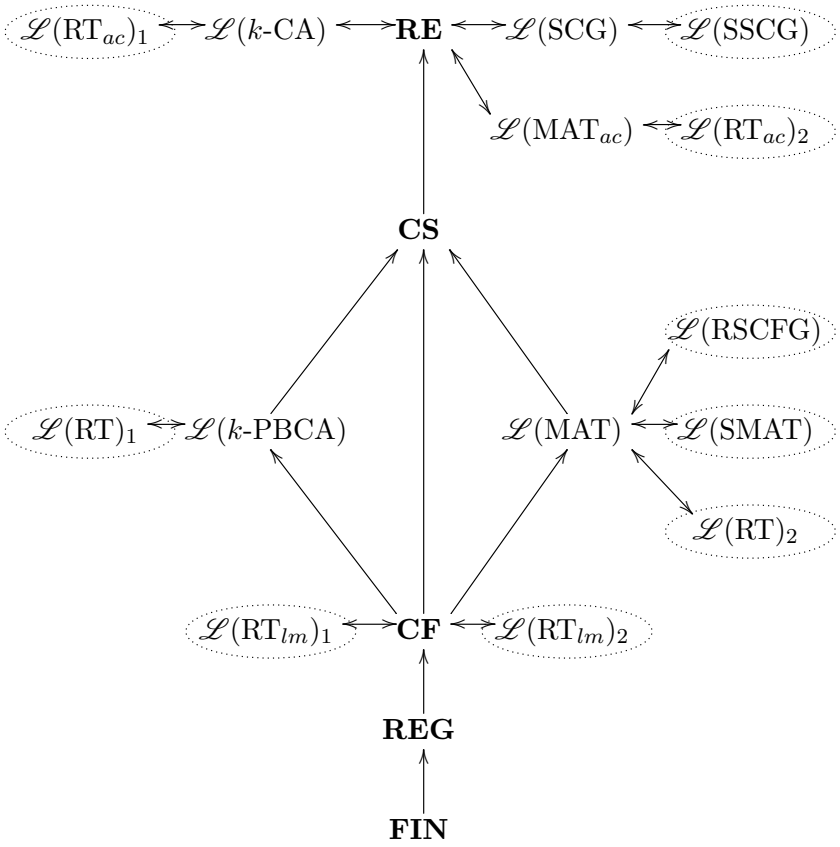


Figure 7.1: Hierarchy of select language classes (for $k \geq 2$) incorporating new results (highlighted by dotted frames). $\mathscr{L}_1 \rightarrow \mathscr{L}_2$ denotes $\mathscr{L}_1 \subset \mathscr{L}_2$ and $\mathscr{L}_1 \leftrightarrow \mathscr{L}_2$ denotes $\mathscr{L}_1 = \mathscr{L}_2$

7.1 Further Research Prospects

Further research prospects include the study of other theoretical properties of the proposed models, such as descriptional complexity. Although we have already shown which language classes our new models define, how efficiently they can do so remains an open problem. That is, we can investigate the effects of different limits placed on, for example, the number of nonterminal symbols in grammars, states in automata, or rules in both. In MATs, we can also limit the length of matrices, and similarly in SCGs, the length of scattered context rules (as sequences of context-free rules).

As we have done with RTs by introducing an appearance checking and a leftmost restriction, we can consider other variants of our models and investigate their properties. For example, we could restrict SSCGs by using propagating SCGs (which are known to be strictly weaker than SCGs with erasing rules). We can also introduce and study systems consisting of other well-known grammars and automata.

Extension to more than two components is possible as well. In such case, we could further investigate the relations to known grammar systems (see [18], [61], or [65]) and automata systems (see [11], [19], or [57]).

Finally, note that although our synchronous grammars and RTs represent different approaches and, consequently, are defined differently, there is a significant similarity in their basic principles. In essence, they are all systems in which the cooperation of components is

achieved by synchronization of their rules. It might be useful to introduce a more general formalism allowing for various components, and thus encompassing all such rule-synchronized systems.

From a more practical viewpoint, an important area to investigate is syntax analysis. For practical applications, we need to be able to parse sentences efficiently. There are well-known parsing methods for CFGs, such as (generalized) LR parsing or chart parsing, but for models with regulated rewriting, the situation is more complicated. While there have been some research in this area, particularly for SCGs (see [46] or [73]), efficient parsing with matrix grammars and scattered context grammars still represents an open problem.

In the examples presented in this work, we have made two important assumptions. First, we already have the input sentence analysed on a low level—that is, we know where every word starts and ends (which may be a non-trivial problem in itself in some languages, such as Japanese) and have some basic grammatical information about it. Furthermore, we assume that we know the translation of the individual words.

For practical applications in natural language translation, we would need a more complex system, with at least two other components: a part-of-speech tagger (lexical analyzer), and a dictionary to translate the actual meanings of the words (although we can do this directly within a grammar by using dictionary rules, as shown in Example 6.3, a separate dictionary generally allows for more efficient encoding). Then, the component based on the discussed formal models could be used to transform the syntactic structure of a sentence and ensure that the words in the translated sentence are in the correct 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, and Tools*. Pearson/Addison Wesley, 2007.

[3] A. V. Aho and J. D. Ullman. Syntax directed translations and the pushdown assembler. *J. Comput. Syst. Sci.*, 3(1):37–56, February 1969.

[4] J. Allen. *Natural language understanding (2nd edition)*. Benjamin/Cummings series in computer science. Benjamin/Cummings Pub. Co., 1995.

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

[7] R. Borsley and K. Börjars, editors. *Non-Transformational Syntax: Formal and Explicit Models of Grammar*. Wiley, 2011.

[8] P. J. Cameron. *Introduction to Algebra*. Oxford science publications. Oxford University Press, 1998.

[9] A. Carnie. *Syntax: A Generative Introduction*. Introducing Linguistics. Wiley, 2012.

[10] M. Čermák, P. Horáček, and A. Meduna. Rule-restricted automaton-grammar tranduscers: Power and linguistic applications. *Mathematics for Applications*, 1(1):13–35, 2012.

[11] 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.

[12] D. Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 263–270, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

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

[14] D. Chiang. Hierarchical phrase-based translation. *Comput. Linguist.*, 33(2):201–228, June 2007.

[15] D. Chiang. *Grammars for Language and Genes: Theoretical and Empirical Investigations.* Theory and applications of natural language processing. Springer, 2011.

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

[17] N. Chomsky. *Syntactic Structures.* Mouton, The Hague, 1957.

[18] E. Csuhaj-Varju, J. Kelemen, Gh. Paun, 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.

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

[20] 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.

[21] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, 1989.

[22] R. Debusmann, D. Duchier, and G.J. M. Kruijff. Extensible dependency grammar: A new methodology. In *Proceedings of the COLING 2004 Workshop on Recent Advances in Dependency Grammar*, Geneva/SUI, 2004.

[23] K. J. Devlin. *The Joy of Sets: Fundamentals of Contemporary Set Theory.* Springer Undergraduate Texts in Mathematics and Technology. Springer-Verlag, 1993.

[24] Y. Falk. *Lexical-functional Grammar: An Introduction to Parallel Constraint-based Syntax.* CSLI lecture notes. CSLI, 2001.

[25] H. Fernau. Regulated grammars under leftmost derivation. *Grammars*, 3(1):37–62, 2000.

[26] G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag. *Generalized Phrase Structure Grammar.* Blackwell, Oxford, 1985.

[27] S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages.* Elsevier Science Inc., New York, NY, USA, 1975.

[28] S. A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theor. Comput. Sci.*, 7:311–324, 1978.

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

[30] 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.

[31] J. Hajič. *Disambiguation of Rich Inflection: Computational Morphology of Czech.* Wisconsin Center for Pushkin Studies. Karolinum, 2004.

[32] E. Hajičová, P. Sgall, and D. Zeman. Issues of projectivity in the prague dependency treebank. In *Prague Bulletin of Mathematical Linguistics*, 2004.

[33] A. Hajnal and P. Hamburger. *Set Theory.* London Mathematical Society Student Texts. Cambridge University Press, 1999.

[34] M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978.

[35] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 2000.

[36] P. Horáček. Formal models in processing of japanese language. In *Proceedings of the 16th Conference and Competition STUDENT EEICT 2010 Volume 5*, pages 161–165. Brno University of Technology, 2010.

[37] P. Horáček. Parse driven translation. In *Proceedings of the 17th Conference and Competition STUDENT EEICT 2011 Volume 3*, pages 480–484. Brno University of Technology, 2011.

[38] P. Horáček. On generative power of synchronous grammars with linked rules. In *Proceedings of the 18th Conference STUDENT EEICT 2012 Volume 3*, pages 376–380. Brno University of Technology, 2012.

[39] P. Horáček. Application perspectives of synchronous matrix grammars. In *Proceedings of the 19th Conference STUDENT EEICT 2013 Volume 3*, pages 202–206. Brno University of Technology, 2013.

[40] P. Horáček and A. Meduna. Regulated rewriting in natural language translation. In *7th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 35–42, Brno, CZ, 2011. Brno University of Technology.

[41] P. Horáček and A. Meduna. Synchronous versions of regulated grammars: Generative power and linguistic applications. *Theoretical and Applied Informatics*, 24(3):175–190, 2012.

[42] K. Hrbacek and T. Jech. *Introduction to Set Theory, Third Edition, Revised and Expanded.* Chapman & Hall/CRC Pure and Applied Mathematics. Taylor & Francis, 1999.

[43] J. Hromkovič. *Theoretical Computer Science: Introduction to Automata, Computability, Complexity, Algorithmics, Randomization, Communication, and Cryptography.* An EATCS series. Springer, 2004.

[44] R. A. Hudson. *Word Grammar.* Blackwell, Oxford, 1984.

[45] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge University Press, 2004.

[46] O. Jirák and D. Kolář. Comparison of classical and lazy approach in scg compiler. In *NUMERICAL ANALYSIS AND APPLIED MATHEMATICS ICNAAM 2011: International Conference on Numerical Analysis and Applied Mathematics*, volume 1389, pages 873–876. American Institute of Physics, 2011.

[47] 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.

[48] M. Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632, 1998.

[49] A. K. Joshi and Y. Schabes. *Tree-adjoining Grammars and Lexicalized Grammars.* LINC LAB: LINC LAB. University of Pennsylvania, School of Engineering and Applied Science, Department of Computer and Information Science, 1991.

[50] 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.

[51] P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

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

[53] S. G. Krantz. *Handbook of Logic and Proof Techniques for Computer Science.* Birkhauser, 2002.

[54] P. M. Lewis and R. E. Stearns. Syntax-directed transduction. *J. ACM*, 15(3):465–488, July 1968.

[55] P. Linz. *An Introduction to Formal Languages and Automata.* Jones and Bartlett Publishers, 2006.

[56] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, Cambridge, MA, USA, 1999.

[57] C. Martín and V. Mitrana. Parallel communicating automata systems. *Journal of Applied Mathematics and Computing*, pages 237–257, 2008.

[58] 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.

[59] A. Meduna. A trivial method of characterizing the family of recursively enumerable languages by scattered context grammars. *EATCS Bulletin*, 1995(56):1–3, 1995.

[60] A. Meduna. *Automata and Languages: Theory and Applications*. Springer, London, 2000.

[61] A. Meduna and R. Lukáš. Multigenerative grammar systems. *Schedae Informaticae*, 2006(15):175–188, 2006.

[62] A. Meduna and M. Švec. *Grammars with Context Conditions and Their Applications*. John Wiley & Sons, 2005.

[63] A. Meduna and J. Techet. *Scattered Context Grammars and their Applications*. WIT Press, UK, GB, 2010.

[64] A. Meduna and P. Zemek. One-sided random context grammars. *Acta Informatica*, 48(3):149–163, 2011.

[65] R. Meersman and G. Rozenberg. Cooperating grammar systems. In J. Winkowski, editor, *Mathematical Foundations of Computer Science 1978*, volume 64 of *Lecture Notes in Computer Science*, pages 364–373. Springer Berlin Heidelberg, 1978.

[66] I. A. Mel'čuk. *Dependency Syntax: Theory and Practice*. Daw Book Collectors. State University Press of New York, 1988.

[67] 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.

[68] R. Mitkov, editor. *The Oxford Handbook of Computational Linguistics*. Oxford University Press, 2003.

[69] M. Mohri. Finite-state transducers in language and speech processing. *Comput. Linguist.*, 23(2):269–311, June 1997.

[70] R. N. Moll, M. A. Arbib, and A. J. Kfoury. *An introduction to formal language theory*. Texts and monographs in computer science. Springer-Verlag, 1988.

[71] S. S. Muchnick. *Advanced compiler design and implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[72] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. University of Chicago Press, 1994.

[73] F. Popowich. Chart parsing of scattered context grammars. *Applied Mathematics Letters*, 7(1):35–40, 1994.

[74] E. Rich. *Automata, Computability and Complexity: Theory and Applications*. Pearson Prentice Hall, 2008.

[75] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

[76] P. Šaloun. Parallel LR parsing. In *Proceedings of the Fifth International Scientific Conference Electronic Computers and Informatics 2002*. The University of Technology Košice, 2002.

[77] P. Sgall, E. Hajicová, J. Panevová, and J. Mey. *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Springer, 1986.

[78] S. M. Shieber and Y. Schabes. Synchronous tree-adjoining grammars. In *Proceedings of the 13th Conference on Computational Linguistics - Volume 3*, COLING '90, pages 253–258, Stroudsburg, PA, USA, 1990. Association for Computational Linguistics.

[79] L. Tesnière. *Éléments de syntaxe structurale*. Éditions Klinksieck, 1959.

[80] A. Venugopal, A. Zollmann, and V. Stephan. An efficient two-pass approach to synchronous-CFG driven statistical MT. In Candace L. Sidner, Tanja Schultz, Matthew Stone, and ChengXiang Zhai, editors, *HLT-NAACL*, pages 500–507. The Association for Computational Linguistics, 2007.

[81] H. Wang. A variant to Turing's theory of computing machines. *J. ACM*, 4(1):63–92, January 1957.

[82] A. Weber. On the valuedness of finite transducers. *Acta Informatica*, 27:749–780, 1990. 10.1007/BF00264285.

[83] Y. Wilks. *Machine Translation: Its Scope and Limits*. Springer, 2008.

[84] K. Yamada and K. Knight. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 523–530, Stroudsburg, PA, USA, 2001. Association for Computational Linguistics.

[85] E. Zámečníková and P. Horáček. Formal models in natural language processing. In *Proceedings of the 18th Conference STUDENT EEICT 2012*, vol. 3, pages 425–429. Brno University of Technology, 2012.

[86] 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.

Appendix A

Index to Language Classes

The following table lists all language classes discussed throughout this text. For each class, the list includes the abbreviated notation, the full description, and the number of the page where it is first introduced.

Notation	*Language class*	*Page*
CF	Context-free languages	14
CS	Context-sensitive languages	14
FIN	Finite languages	18
RE	Recursively enumerable languages	14
REG	Regular languages	14
$\mathscr{L}(k\text{-CA})$	Accepted by k-counter automata ($k \in \mathbb{N}$)	18
$\mathscr{L}(k\text{-PBCA})$	Accepted by partially blind k-counter automata	18
$\mathscr{L}(\text{FA})$	Accepted by finite automata	15
$\mathscr{L}(\text{MAT})$	Generated by matrix grammars	17
$\mathscr{L}(\text{MAT}_{ac})$	Generated by MATs with appearance checking	17
$\mathscr{L}(\text{PDA})$	Accepted by pushdown automata	15
$\mathscr{L}(\text{RSCFG})$	Generated by rule-synchronized context-free grammars	35
$\mathscr{L}(\text{RT})_1$	Accepted by rule-restricted transducers	44
$\mathscr{L}(\text{RT})_2$	Generated by rule-restricted transducers	44
$\mathscr{L}(\text{RT}_{ac})_1$	Accepted by RTs with appearance checking	51
$\mathscr{L}(\text{RT}_{ac})_2$	Generated by RTs with appearance checking	51
$\mathscr{L}(\text{RT}_{lm})_1$	Accepted by RTs with leftmost restriction	48
$\mathscr{L}(\text{RT}_{lm})_2$	Generated by RTs with leftmost restriction	48
$\mathscr{L}(\text{SCG})$	Generated by scattered context grammars	17
$\mathscr{L}(\text{SMAT})$	Generated by synchronous matrix grammars	40
$\mathscr{L}(\text{SSCG})$	Generated by synchronous scattered context grammars	39