ADAM MICKIEWICZ UNIVERSITY, POZNAŃ, POLAND

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Katarzyna Moroz

Algorithmic questions for pregroup grammars

PhD Thesis

Promotor:
Prof. Dr. Wojciech Buszkowski

POZNAŃ 2010

Acknowledgments

I wish to express my deep gratitude to my Promotor Prof. Wojciech Buszkowski for his time he devoted to me. He offered me so much advice, help and encouragement. He patiently supervised me. I have learned a lot from him.

I would like to thank my Master's thesis advisor, Dr Krzysztof Jassem, who encouraged me to continue my work and has always been willing to help.

I thank my colleagues at the Faculty of Mathematics and Computer Science of Adam Mickiewicz University, especially late Dariusz Okołowski, talking to whom was always very encouraging and inspiring.

Last but not least, this work would not have been achieved without the support and understanding of my family. I thank my husband who had to keep our house and take care of children when only at home and have always encouraged me to continue my work. Without support of my parents and sisters I would not be able to concentrate on my work.

Contents

Preface

Pregroups and pregroup grammars were introduced in 1999 by Lambek [Lam99] as a new tool for syntactical analysis of natural languages. The formalism of pregroup grammars belongs to the tradition of categorial grammars (or: type logical grammars). In general, they are part of a wide field of mathematical linguistics i.e. the theory of formal grammars and automata with applications in computer science, in particular in natural language processing.

The history of mathematical linguistics is not really long, in contrary to the research on formal languages of logics and mathematics. For a long time, mathematicians believed that the only languages that could be fully analyzed by means of mathematical methods are formal languages. Alfred Tarski [Tar33] regarded natural language as a domain which cannot be correctly formalized as a whole.

The fundamentals of mathematical linguistics are works of Noam Chomsky. In his book *Syntactic Structures* [Cho57] he described transformational-generative grammar. Chomsky also defined and studied a hierarchy of formal grammars.

Chomsky and his followers developed further different formalisms like generative grammars, head-driven phrase structure grammars or lexical functional grammars. However, approach of generative grammars is purely combinatorial, with no logic behind. Combining methods of Chomsky with standards of logical grammars gave rise to generalized phrase-structure grammars, definite clause grammars and others.

Richard Montague and Max J. Cresswell are logicians who initiated, in 1970s, an advanced logical analysis of natural languages, especially in semantics. They believed that all interesting phenomena in natural language semantics could be described by means of some existing formalisms of mathematical logic: Montague employed intentional type theory (in the form of lambda calculus with types and equality) whereas Cresswell higher-order possible worlds semantics.

Another type-theoretic approach to description of natural languages covers a wide variety of theories called categorial grammars. Actually, philosophical interest in categories may be traced back to the ancient times. Aristotle in his treatise *Categories*, attempted to enumerate the most general kinds into which entities in the world can be divided. He assumed that a *universal* is something identical in each of its *instances*. Therefore, instances of things can be characterized by some kind of *types*.

In modern times, a new approach to the distinction between *objects* and *meaning* or *concept* was proposed by Gottlob Frege [Fre92]. In a way, he simplified the Aristotelian theory. He formed the *principle of compositionality*, saying that the meaning of any complex expression is a function of the meanings of its parts and of the way they are *syntactically combined*. The *complete expressions* are names, i.e. denoting expressions. They consist of *simple names* of objects, *complex names* denoting objects and *sentences*, which are also complex names. Complex names are formed out of incomplete expressions, which are actually *functions*. This relation

between function, its arguments and the value of the function on the given argument is the basic idea of Frege's philosophy of language. He claimed that any sentence that expresses a singular proposition consists of an *expression*, that is a proper name or a name preceded by an article, together with a *predicate*, that is the verb *is*, with a name accompanied by the indefinite article or an adjective. An expression signifies an *object* and a predicate signifies a *concept*.

Ideas in considering the difference between *object* and *meaning* were further developed by Edmund Husserl and Bertrand Russell.

Husserl explicitly distinguished categories of *meanings* from categories of *objects*. Categories of *meanings* are called *formal*, whereas categories of *objects* are called *material*. Both, formal and material category systems form a hierarchy. However, formal and material categories are not mutually exclusive, since one and the same entity may be categorized either in terms of its material nature or its form. One can form *complete expressions* from *incomplete expressions* by means of *meaning connection rules*. In these rules, expressions belonging to the same *meaning category* may be freely interchanged. There follows the idea of *purely logical grammar*, which is a set of analytical laws common for all languages. It is important in this framework that some categories are basis to which other categories can be applied as operators, due to some laws, and the result can become basis to some new applications.

Russell invented the first *type theory* in response to finding some inconsistency in Frege's theories (Russell's paradox) by creating a *hierarchy of types*, which can be assigned to entities. In the type theory, only expressions belonging to the same logical types can be substituted for one another without changing *grammatical* or *well-formed expressions* into *ungrammatical* or *ill-formed expressions*. Only well-formed expressions may have meaning. The type theory was modified by the Russell and Whitehead in *Principia Mathematica* [WR13].

In 1930s Alonzo Church introduced lambda calculus, a formal system designed to investigate function definition, application and recursion. First an untyped version was proposed, later in 1940s a typed version of the calculus was introduced. The lambda calculus in its both versions turned out to be very useful in logic, linguistics, mainly in semantics, and programming languages, especially in functional programming. Typed lambda calculus served as the foundation for modern type systems in programming languages.

Russerl's type theory inspired Polish logicians. Stanisław Leśniewski [Leś29] readjusted the notion of a category, turning categories from classes of entities to classes of expressions. He developed a new hierarchy of types, which he called a *grammar of semantic categories*. He proposed two basic semantic categories: *proposition (sentence - s)* and *noun (name - n)*. Sentences are complex expressions that can be true or false from the logical point of view. Kazimierz Ajdukiewicz adapted Leśniewski's approach and formulated a general *theory of semantic categories*. It is described in the article *Syntactic Connection* (1935) [Ajd35]. The modern history of categorial grammars started with this essay. Ajdukiewicz introduced an algebraic notation to represent the hierarchy of semantic types in which each word is assigned a fraction representing some category. Fractions are built out of two primitive types

n and s that can be joined together forming complex expressions. He proposed a kind of parsing algorithm for any language in which functors precede arguments; it is the first parsing algorithm in the literature.

The term *categorial grammar* was first proposed by Yehoshua Bar-Hillel [BGS60] who enriched Ajdukiewicz's work with his own ideas. He proposed are to assign more than one category to the given word and to allow functor categories to be applied to the arguments occurring both on the right and on the left, thus avoiding Ajdukiewicz's constraint on the order of functors and arguments. Such grammars are called *classical categorial grammars*, and the calculus is nowadays called the *logic* ***AB***. In the paper [BGS60] Bar-Hillel together with Gaifman and Shamir presented some important results concerning categorial grammars, among others, so-called Gaifman Theorem, saying that classical categorial grammars are weakly equivalent to context-free grammars. The theorem is equivalent to the Greibach Normal Form Theorem for context-free grammars (independently proved in 1965).

The logic **AB** gave rise to different calculi, under a general name of *type logics*. Grammars of these systems are called *type logical grammars* or categorial grammars.

Joachim Lambek [Lam58] extended logic **AB** by introducing *Syntactic Calculus*, later called Lambek Calculus. Assuming there is a set of basic types, one obtains recursively new types by means of three operations: multiplication and two divisions (left and right). His system is deductive. Analysis of natural language sentences is performed similarly as in classical categorial grammars but there are more laws admitting some new operations on types. Thus, the calculus is more powerful than logic **AB**. Grammars based on Lambek Calculus are called *Lambek grammars*.

Concatenation, expressed in the Lambek Calculus by product, is a standard operation on languages, but residuals (left and right division) are not usually discussed in mathematical linguistics. Residuals correspond to the notion of functor in categorial grammars. A *functor* is an expression that has to be completed to form an expression of a given category. For example $pn\backslash s$ is a functor type of intransitive verb phrase, as it takes a proper noun as an argument, positioned on the left of the functor, and returns a sentence. In Lambek grammar, if a given word is assigned type A, then it can also be assigned complex types B determined by all laws $A \to B$ provable in the Lambek Calculus.

Lambek grammars were shadowed by Chomskyan theory of generative-transformational grammars for a few decades. A new interest in Lambek Calculus started in late 1970s and brought to some theoretical results and different extensions and refinements of the calculus. Earlier, Cohen [Coh67] had attempted a proof of the equivalence of Lambek grammars and context-free grammars, which was showed deficient [Bus85, Zie85]. For unidirectional Lambek grammars this equivalence was proved by Buszkowski [Bus85]. Finally, the equivalence theorem for Lambek grammars was obtained by Pentus [Pen93]. Computational and model-theoretic results on Lambek Calculus and Lambek grammars are due to Buszkowski [Bus78, Bus82, Bus86a] and Zielonka [Zie78, Zie81]. Buszkowski [Bus86b] and Kandulski [Kan88a, Kan88b] obtained results on strong equivalence for non-associative Lambek grammars and classical categorial grammars, which entail the weak equiv-

alence of non-associative Lambek grammars and context-free grammars.

Lambek [Lam58] gave a Gentzen-style axiomatization of his calculus and proved the decidability. There holds a cut-elimination theorem for this system, which gives a standard proof-search method. Algebraic models for the Lambek Calculus are based on residuated semigroups.

Girard [Gir87] introduced *linear logic*, which soon has become a subject of great interest. Linear logic can be classified in a more general family of *substructural logics*, including *relevant logics*, *BCI-* and *BCK-logics* and others. Linear logic can be seen as a subsystem of classical logic without structural rules of Weakening and Contraction in its Gentzen-style axiomatization. This property is characteristic of substructural logics. Lambek Calculus can be treated as the multiplicative fragment of noncommutative intuitionistic linear logic (there is no Exchange rule).

Lambek Calculus is naturally related to the lambda calculus, which was discussed by van Benthem, e.g. in [vB86, vB91]. Commutative Lambek Calculus is a kind of logic of types introduced by a fragment of lambda calculus. Consequently, parsing based on the Lambek Calculus and related systems can be associated with a procedure of meaning assignment. This led to a new, semantic, perspective of research on the Lambek calculi; see e.g. Moortgat [Moo88] and Morrill [Mor94].

Recent research in the field of Lambek Calculus is also combined with other ideas of various logical work, e.g. substructural logics, modal logics, information processing or labeled deduction of Gabbay. Steedman developed a similar approach basing on the Curry-Shaumyan tradition of combinatory grammars.

One of the recent extensions is the calculus of pregroups proposed by Lambek in [Lam99]. The calculus strengthens and, at the same time, simplifies the multiplicative fragment noncommutative linear logic. Instead of binary operations of residuation there are two unary operations ^{l} and ^{r} called *left* and *right adjointness*.

Especially *free pregroups* are useful in linguistic applications. Free pregroups can be represented by a formal system called *Compact Bilinear Logic*. This system can be defined in the form of a rewriting system. A derivation in this system is a rewriting procedure. An important property of the system (proved by Lambek [Lam99]) is that if a string of types reduces to a simple term or the empty string, then the reduction can be preformed by use of Contractions and Induced Steps only. Consequently, as proved by Buszkowski [Bus03], the calculus of pregroups is decidable in polynomial time. Moreover, pregroup grammars are weakly equivalent to context-free grammars ([Bus01]).

For natural language processing one defines a finite partially ordered set of *basic syntactical types*, including the type of a *sentence*, a *lexicon* in which each lexical entry is assigned a finite number of *types* (strings of iterated right and left adjoints of basic types) and a partial order on basic types. While parsing a sting of words with a pregroup grammar, a type for each word is chosen from the lexicon and pregroup laws are applied to the concatenation of those types. If a sentence type is derived, then the string of words is recognized as a sentence. Parsing algorithms for pregroup grammars were studied by Oehrle [Oeh04] and Preller [Pre07]. Preller discussed also a linear parsing algorithm for pregroup grammars with some restrictions.

Since their introduction, pregroups have been a subject of great interest. They are an efficient tool in natural language processing. Pregroup grammars have been successfully applied to various linguistic phenomena of many languages. A lot of work has been done by Lambek and his collaborators to describe English, see [Lam08]. However, large fragments of other languages have also been presented by means of pregroups, among others French [BL01], German [LP04], Italian [CL01], Polish [KM08] and Japanese [Car07].

In general, type logical grammars differ from other grammars by the fact that each word is assigned one or more types and there are no grammar rules. Types describe some characteristics of words. All linguistic data are encoded only in the lexicon. This property is called *lexicality*. Rules of type grammars are sequents provable in some logic; therefore, they do not depend on the particular language. To check whether a string of words is a sentence one finds appropriate types in the lexicon and applies rules of the calculus to that string of types. If the calculations is successful, then the string is considered a sentence. The advantage of such approach is that exactly the same algorithms can be used to parsing different natural languages. It suffices to define the lexicon of a given language in which appropriate types are assigned to each entry. Moreover, when a lexicon is expanded, the parsing algorithms need not be changed.

The calculus of pregroups is simpler in use and computationally less demanding than Lambek Calculus. Pregroup grammars can be parsed in polynomial time, whereas the Lambek Calculus is NP-complete [Pen08].

Structure of the Thesis

In this thesis we consider pregroup grammars and algorithmic questions for pregroup grammars and some extended pregroup grammars. We consider the equivalence of pregroup grammars and context-free grammars. Then we focus on parsing pregroup grammars. We give a new polynomial dynamic parsing algorithm for pregroup grammars and pregroup grammars with letter promotions. Later we discuss pregroup grammars with letter promotions with 1. Finally we present our tool for parsing pregroup grammars.

The structure of the thesis is as follows.

In the first two chapters we give basic definitions and theorems used further in the thesis. The next four chapters present main results of the thesis and the last chapter presents the application.

In **Chapter 1** we present basic definitions and ideas from the theory of automata and context-free grammars. The chapter consists of five parts. First we define deterministic and non-deterministic finite-state automata. In the second section we present pushdown automata. The next section treats on context-free grammars. We define a context-free grammar, we present the theorem on their equivalence with pushdown automata and a CYK algorithm for parsing context-free grammars. Further, categorial grammars are introduced. We define a classical categorial grammar, we give a few linguistic examples and we present some results on equivalence of categorial grammars and context-free grammars. In the final section we introduce type logics. We present Lambek Calculus and a cut-elimination theorem for it. We define Lambek grammars, we state the Pentus theorem on equivalence of Lambek grammars and context-free grammars (not allowing the empty strings). Finally, algebraic models are presented.

Chapter 2 is devoted to pregroups and pregroup grammars. First we briefly discuss bilinear algebras. In the second section we introduce pregroups in a formal way and we prove some basic pregroup laws. In the next part we discuss free pregroups, which were proposed by Lambek as a way of applying pregroups to natural languages. We describe a term rewriting system of Compact Bilinear Logic. We present also the Lambek Normalization Theorem. Then we introduce pregroup grammars. In the last section of this chapter we describe application of pregroup grammars to natural languages.

In **Chapter 3** the equivalence of pregroup grammars and context-free grammars is considered. At first, we present the Equivalence Theorem and briefly outline its proof (which are given in [Bus01]). Further, we discuss an idea of partial composition [Béc07] used to construct a **CFG** equivalent to the given pregroup grammar. However, the size of the constructed grammar is exponential. The main result of this chapter is a direct polynomial construction of a context-free grammar equivalent to the given pregroup grammar, based on the proof of the Equivalence Theorem. It was first presented during Poznań Linguistic Meeting in May 2006 (PLM), then

published in [BM08]. Next, we show that the size of the constructed **CFG** is polynomial. We continue with a construction of a pushdown automaton equivalent to the given pregroup grammar (both results were also presented on PLM). Some of these results were also presented on the First International Workshop on Non-Classical Formal Languages in Linguistics in Budapest in August 2007 and published in the Proceedings.

Chapter 4 presents a dynamic polynomial parsing algorithm for pregroup grammars with a proof of its correctness (**Section** 4.3). This chapter consists of four sections. We start with a brief description of problems that can be encountered while parsing pregroup grammars. In the second section, we present a recognition algorithm of Savateev [Sav09], which was defined for the Unidirectional Lambek Calculus ($\mathbf{L}^{\backslash}$). Further, we present our dynamic polynomial recognition algorithm for pregroup grammars. It is based on the algorithm of Savateev. We prove the correctness of the algorithm. We describe how to amend the algorithm to obtain a full parsing algorithm and we give some examples of how it works. This result was presented on the 14th Conference on Formal Grammar, Bordeaux, July 2009, and the article is to appear [Mor10]. Finally, some other recognition and parsing algorithms for pregroup grammars ([Oeh04] and [Pre07]) are discussed.

In **Chapter 5** we present a modification of our dynamic polynomial parsing algorithm for pregroup grammars working for pregroup grammars with letter promotions (**Section** 5.3) and the proof of its correctness. This chapter consists of four sections. The first two section describe results obtained by Buszkowski and Lin Zhe [BLZ09]. We start with an introduction of pregroup grammars with letter promotions. In the second section, we present the proof that the letter promotion problem for pregroups is solvable in polynomial time [BLZ09]. Further, we present a dynamic polynomial recognition algorithm for pregroup grammars with letter promotions. We show the proof of the correctness of the algorithm. Finally, we give some examples of pregroup grammars with letter promotions. The algorithm described in this chapter has not been published yet.

In **Chapter 6** we discuss **CBL** enriched with letter promotions and more general promotions: letter promotions with 1. In the first section we describe the calculus and give the first result: a normalization theorem for this system. In the second section we show that the letter promotion problem for **CBL** enriched with letter promotion with 1 is polynomial. We use similar reductions as proposed by Buszkowski and Lin Zhe [BLZ09] for the calculus with letter promotions. Finally, we present a dynamic polynomial recognition algorithm for pregroup grammars with letter promotions with 1. We show the proof of the correctness of the algorithm. None of the results of this chapter has been published so far.

Chapter 7 contains a presentation of a tool constructed to show application of our dynamic parsing algorithm for pregroup grammars. It is a Java application giving answer to the recognition problem for pregroup grammars and if it is positive, one of possible parsings of a given string of words. The tool was presented together with the algorithm on the International Multiconference on Computer Science and Information Technology, Mrągowo, October 2009 and published in [Mor09].

Chapter 1

Preliminaries

Contents

In this chapter we present basic definitions and ideas. It consists of five main parts. First, we introduce the theory of finite state automata. In the second section we present pushdown automata. Further there is a section on context-free grammars and the theorem on their equivalence with pushdown automata. Two final sections are devoted to categorial grammars and type logics which influenced the theory of pregroups. At the end we give some linguistic examples.

1.1 Finite state automata

Definitions and theorems in the first three sections are stated as in [HU03] or [JS08].

If Σ is an alphabet, then Σ^* denotes the set of all strings on the alphabet Σ, whereas Σ^+ is the set of all strings on the alphabet Σ without the empty string (ε). Any subset of Σ^* (Σ^+) is called a *language* on Σ (ε-free language on Σ, respectively).

Assume L_1, L_2 are ε-free languages on Σ. One defines:

$$L_1 \cdot L_2 = \{ab : a \in L_1 \text{ and } b \in L_2\},$$

$$L_1 \backslash L_2 = \{c \in \Sigma^+ : L_1 \cdot \{c\} \subseteq L_2\},$$

$$L_1 / L_2 = \{c \in \Sigma^+ : \{c\} \cdot L_2 \subseteq L_1\}.$$

For $L_1, L_2 \subseteq \Sigma^*$ these operations are defined in a similar way. Obviously, Σ^+ is replaced by Σ^*. The difference can be seen on the following example. If $L = \{a\}$, $a \neq \varepsilon$, then $L \backslash L = \emptyset$ if $L \subseteq \Sigma^+$ and $L \backslash L = \varepsilon$ if $L \subseteq \Sigma^*$.

Finite state automaton is a mathematical model that has been widely used in computer science for many years. It consists of a finite number of states and transitions between states. The automaton starts working in one of the *initial*

states. In some conditions, e.g. when the automaton reads the *input* it can switch to another state, that is it performs a *transition*. The input is a sequence of symbols of the finite *alphabet*. A transition can also be labeled by ε, which means that it can be performed without reading any input symbol. If the automaton is in one of its *final states* when it stops working, it is said to *accept* its input.

Definition 1.1 (Deterministic finite state automaton)**.** A *deterministic finite state automaton (**DFA**)* is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of *states*
- Σ is a finite *alphabet of input symbols*
- $q_0 \in Q$ is an *initial state*
- $F \subseteq Q$ is a set of *final states*
- δ is a *transition function* from $Q \times \Sigma$ to Q, that is δ assigns to a state $q \in Q$ and a symbol $a \in \Sigma$ a new state $q' = \delta(q, a)$, $q' \in Q$.

If one allows zero, one or more transitions from the given state for the same input string, then the automaton is *non-deterministic*. Formal definition is similar to the definition of a **DFA** with the only difference in transition function and initial states.

Definition 1.2 (Nondeterministic finite state automaton)**.** A *nondeterministic finite state automaton (**NFA**)* is a 5-tuple $A = (Q, \Sigma, \delta, Q_0, F)$, where

- Q is a finite set of *states*
- Σ is a finite *alphabet of input symbols*
- $Q_0 \subseteq Q$ is a set of *initial states*
- $F \subseteq Q$ is a set of *final states*
- δ is a transition function from $Q \times \Sigma$ to 2^Q, that is δ assigns to a state $q \in Q$ and a symbol $a \in \Sigma$ a set of new states P such that for $p \in P$ there exists a transition from q to p with the label a.

An automaton can be described as a directed graph. The graph is defined as follows: vertices are states of the automaton and arcs represent transitions. If there exists a transition from a state q to a state p for an input symbol a, then the graph contains an arc from the state q to p labeled with a. Usually, the states are represented by circles with the name of the state written inside. An initial state is denoted by an arrow having no source, pointing to this state (the arrow has no label). Final states are denoted by two concentric circles.

A sample **DFA** in a form of a graph is given in **Figure 1.1** and an **NFA** in **Figure 1.2**.

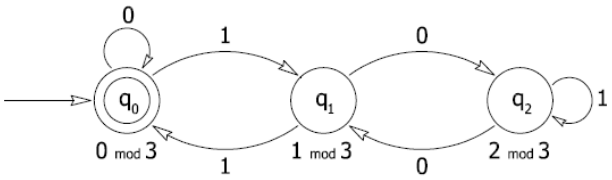


Figure 1.1: A sample DFA

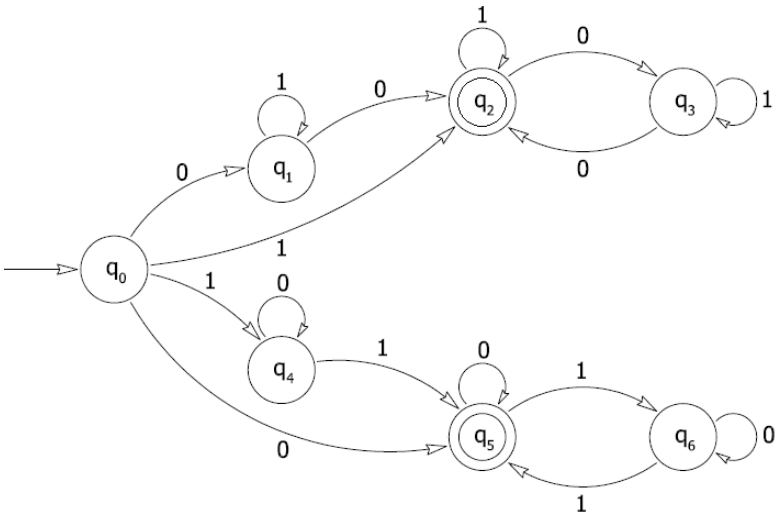


Figure 1.2: A sample NFA

One should notice, that in a **DFA** there can be only one transition with the same label starting at the given state, whereas in an **NFA** more such transitions are allowed.

We say that an input string w is *accepted* or *recognized* by a **FSA** $A = (Q, \Sigma, \delta, q_0, F)$ if A is in a final state when it stops working, i.e. there exists a route from the initial state (in case of a **DFA**) or one of the initial states (in case of a **NFA**) to one of the final states, labeled with all elements of string.

The *language* A is the set of all strings accepted by the given **FSA** and it is denoted by $L(A)$.

A language is called a *regular set* or simply *regular* if it is accepted by some **FSA**.

The language accepted by a **DFA** in **Figure 1.1** consists of all nonempty binary strings which are multiples of 3. The language accepted by an **NFA** in **Figure 1.2** consists of all nonempty binary strings in which either number of 0 or number of 1 is even.

The difference between **DFA**s and **NFA**s is that in the former for the given input

string w and state q there is exactly one route starting in q labeled by w, whereas in the latter, there can be more routes. Therefore, in an **NFA** all routes have to be checked to say that the given string is not accepted by the automaton. However, the class of languages accepted by **NFA**s is the same as the class of languages accepted by **DFA**s.

Theorem 1.1. *Let L be a set accepted by an **NFA**. Then there exists a **DFA** accepting the language L.*

The proof consists of showing that any **NFA** can be simulated by a **DFA**, that is for any **NFA** one can construct an *equivalent* **DFA** accepting the same language. A **DFA** simulates an **NFA** in such a way that states of the **DFA** correspond to the sets of states of the **NFA**.

1.2 Pushdown automata

Pushdown automata are **FSA**s supplied with a stack; they use the stack as follows:

- they can use the top of the stack to decide which transition to take,
- they can manipulate the top of the stack as a part of performing a transition.

The manipulation can be to push a string of symbols to the top of the stack, or to pop off the top of the stack. The automaton can also leave the stack as it is. The choice of manipulation (or no manipulation) is determined by the transition table.

Definition 1.3 (Pushdown automaton)**.** A *pushdown automaton (**PDA**)* is a system A of the form $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where

- Q is a finite set of *states*,
- Σ is a finite set called *alphabet of input symbols*,
- Γ is a finite set called *alphabet of stack symbols*,
- $q_0 \in Q$ is an *initial state*,
- $Z_0 \in \Gamma$ is an *initial stack symbol*,
- $F \subseteq Q$ is a set of *final states*,
- δ is a *transition function* from $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.

In order to formalize configuration of the pushdown automaton A at the given moment a description of the current situation is introduced. Any 3-tuple $(q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$ is called an *instantaneous description* (**ID**) of A. **ID** includes:

- the current state,

- the part of the input string that has not been read,
- the contents of the stack (the topmost symbol written first).

We write $(q, aw, Z\alpha) \vdash_A (p, w, \beta\alpha)$ if $\delta(q, a, Z)$ contains (p, β). We should notice that a can be any input symbol or ε. The reflexive and transitive closure of the relation $\vdash_A$ is denoted by $\vdash_A^*$. We write $I \vdash_A^* J$ for any **ID**s I, J, if there exists a sequence of **ID**s $(I_0, \ldots, I_n)$ such that $n \geqslant 0$, $I_0 = I, I_n = J$ and $I_{i-1} \vdash_A I_i$, for all $i = 1, \ldots, n$.

There are two ways of defining the language accepted by a **PDA** . The first one is acceptance *by empty stack*, which means that the language accepted by the **PDA** is the set of all input strings such that, after reading them, the automaton empties its stack. The other one is acceptance *by final state*, that is the language accepted by the **PDA** is the set of all input strings for which the automaton reaches an accepting state (any state of F). For a **PDA** accepting an input by empty stack there exists an equivalent **PDA** accepting an input by final state, and conversely.

1.3 Context-free grammars

A *context-free grammar* (**CFG**) is a grammar in which every *production rule* is of the form

$$A \mapsto w,$$

where A is a single *nonterminal symbol*, and w is a *string of terminals and/or nonterminals*. In some cases w can be the empty string, denoted by ε.

The formalism of context-free grammars was developed in the mid-1950's by Noam Chomsky. He called them *phrase-structure grammars*. Since then the formalism has been widely studied by linguists, mathematicians and computer scientists.

Context-free grammars play an important role in the description and design of programming languages and compilers. They are also used for analyzing the syntax of natural languages. However, not all important features of natural language syntax can be expressed by context-free grammars.

Below we present a formal definition of a **CFG**.

Definition 1.4 (Context-free grammar)**.** A *context-free grammar* is a 4-tuple $G = (V, T, P, S)$, where

- V is a finite set of *non-terminal symbols* or *variables*,
- T is a finite set of *terminal symbols*, one assumes that V and T are disjoint,
- P is a finite set of *production rules* which are of the form $A \mapsto \alpha$ where $A \in V$, $\alpha \in (V \cup T)^*$,
- S is a special non-terminal called the *initial symbol*.

A **CFG** is said to be *ε-free* if it does not contain nullary rules $A \mapsto \varepsilon$.

We want to define formally a language generated by a **CFG** $G = (V, T, P, S)$. First one defines two relations of *derivability* $\to_G$ and $\to_G^*$. If $A \mapsto \beta$ is a production from P, α and γ are arbitrary strings from $(V \cup T)^*$, then $\alpha A \gamma \to_G \alpha\beta\gamma$; we say that $\alpha\beta\gamma$ is *directly derivable* from $\alpha A\gamma$ in the grammar G. Two strings are connected by the relation $\to_G$ if, and only if, the second one can be obtained from the first one by an application of a production rule. If it is not confusing, the subscript G can be omitted: $\alpha A\gamma \to \alpha\beta\gamma$.

We say that a string β is *derivable* from a string α in G (we write $\alpha \to_G^* \beta$) if there exists a sequence $(\alpha_0, \ldots, \alpha_n)$ such that $n \geqslant 0$, $\alpha_0 = \alpha, \alpha_n = \beta$ and $\alpha_{i-1} \to_G \alpha_i$, for all $i = 1, \ldots, n$. Therefore, $\to_G^*$ is the reflexive and transitive closure of the relation $\to_G$.

Definition 1.5 (Context-free language)**.** The set of all strings generated by a grammar G:

$$L(G) = \{w : w \in T^* \text{ and } S \to_G^* w\}$$

is called the *language generated by the grammar G*. A language L is *context-free* if there exists a context-free grammar G which generates L.

Example 1.1 (Matching parentheses)**.** A canonical example of a **CFG** is a grammar generating strings of matching parentheses. The only nonterminal is the symbol S and there are two terminals "(" and ")". Production rules are as follows.

$S \mapsto SS$,
$S \mapsto (S)$,
$S \mapsto ()$.

A sample string generated by this grammar is (()(()()))(). It can be derived in the following way:

$S \to SS \to (S)S \to (SS)S \to (S(S))S \to (S(SS))S \to (()(SS))S \to$
$(()(()S))S \to (()(()()))S \to (()(()()))()$

Example 1.2 (Propositional Calculus)**.** Now we consider another example. Let $G = (\{S, R, Z\}, \{p, ', \neg, \Rightarrow, (,)\}, P, S)$. Assume P consists of the following productions:

$S \mapsto \neg S$,
$S \mapsto (S \Rightarrow S)$,
$S \mapsto Z$,
$Z \mapsto p$,
$Z \mapsto pR$,
$R \mapsto '$,
$R \mapsto ' R$.

The language generated by G is the set of propositional calculus formulae.
Below, we show a sample derivation in this grammar.

$S \to (S \Rightarrow S) \to (S \Rightarrow (S \Rightarrow S)) \to (Z \Rightarrow (S \Rightarrow S)) \to (p \Rightarrow (S \Rightarrow S)) \to (p \Rightarrow (Z \Rightarrow S)) \to (p \Rightarrow (pR \Rightarrow S)) \to (p \Rightarrow (p' \Rightarrow S)) \to (p \Rightarrow (p' \Rightarrow Z)) \to (p \Rightarrow (p' \Rightarrow p))$

We say that two grammars G_1 and G_2 are *weakly equivalent* if $L(G_1) = L(G_2)$.

There is an important theorem about the equivalence of context-free languages and pushdown automata.

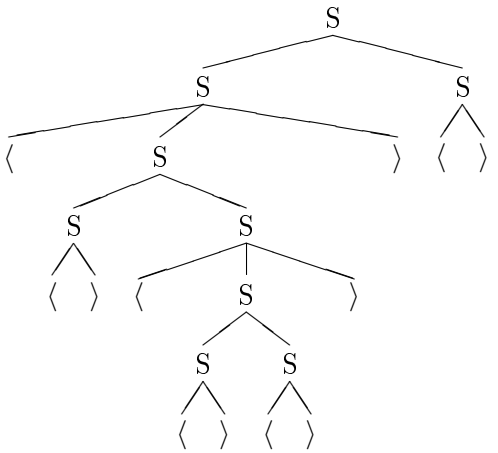
Theorem 1.2. *A language is context-free if and only if there exists a **PDA** that accepts it.*

In different applications one needs to represent derivations as *trees*. A *concrete syntax tree* or *derivation tree* is an ordered tree representing the syntactic structure of a string according to some formal grammar.

Definition 1.6 (Derivation tree)**.** A tree D is called a *derivation tree* for the given grammar $G = (V, T, P, S)$ if

- every node has a symbol from $V \cup T \cup \{\varepsilon\}$ as a label,
- the root of D is labeled with S,
- if any inner node of the tree is labeled with A, then A has to belong to V,
- if a node n of the tree D has a label A and nodes $n_1, n_2, ..., n_k$ are children of the node n, ordered from left to right and labeled with $X_1, X_2, ...X_k$ respectively, then $A \mapsto X_1X_2...X_k$ has to be a production in P,
- if a node n of the tree D has a label ε, then n is a leaf of the tree and the only child of its parent.

Example 1.3. We show a derivation tree of a string (()(()()))() generated by the grammar G from **Example 1.1**.



It is useful, in many proofs in the field of languages and computability, to represent grammars in some standard ways. One of widely used standards is *Chomsky Normal Form* (**CNF**).

Definition 1.7 (Chomsky Normal Form)**.** A **CFG** $G = (V, T, P, S)$ is said to be in **CNF** if all of its productions are in one of the following forms:

- $A \mapsto BC$
- $A \mapsto a$

where $A, B, C \in V$, $a \in T$.

Theorem 1.3. *Any context-free language not containing ε can be generated by a grammar in **CNF**.*

All rules of a grammar in **CNF** are *expansive*; that is each string of terminals and nonterminals in the derivation is always either the same length or longer than the previous one. The other important property is that a derivation tree based on a grammar in **CNF** is a binary tree, and the height of this tree is not grater than the length of the string. It is due to the fact that all rules deriving nonterminals transform one nonterminal to exactly two nonterminals.

These properties yield various efficient algorithms based on grammars in **CNF**, for example, the ***CYK*** (Cocke-Younger-Kasami) algorithm, which determines whether the given string can be generated by the given grammar (in **CNF**). It is a dynamic algorithm working in a cubic time for the given grammar, more precisely $O(k \cdot n^3)$, where k is the size of the grammar.

Let w be a string of length $n \geqslant 1$ and assume G is a grammar in **CNF**. For any $1 \leqslant i < j \leqslant n$ and any variable A one checks whether $A \to_G^* w_{ij}$, where w_{ij} is a substring of length j starting at i-th position. Obviously, $w = w_{1n}$. Let V_{ij} be the set of variables A for which $A \to_G^* w_{ij}$. A table of appropriate sets of values is formed. It is obvious that $w \in L(G)$ if, and only if, $S \in V_{1n}$.

Let us notice that we can assume $1 \leqslant i \leqslant n - j + 1$, as there does not exist any string of length greater than $n - i + 1$ starting at the i-th position in w. A formal statement of the algorithm is presented below.

Algorithm 1.1 CYK algorithm

for $i \leftarrow 1$ **to** n **do**
  $V_{i1} \leftarrow \{A : A \mapsto a$ is a production and a is the i-th symbol of $w\}$
end for
for $j \leftarrow 2$ **to** n **do**
  for $i \leftarrow 1$ **to** $n - j + 1$ **do**
    $V_{ij} \leftarrow \emptyset$
    for $k \leftarrow 1$ **to** $j - 1$ **do**
      $V_{ij} \leftarrow V_{ij} \cup \{A : A \mapsto BC$ is a production, $B \in V_{ik}$ and $C \in V_{i+k,j-k}\}$
    end for
  end for
end for

Example 1.4 (CYK). Let us consider a grammar G with the following production rules:

$S \mapsto AB|BC$
$A \mapsto BA|a$
$B \mapsto CC|b$
$C \mapsto AB|a,$

and a string $w = baaba$. Let us notice that $S \mapsto AB|BC$ stands for two production rules $S \mapsto AB$ and $S \mapsto BC$. A table of values V_{ij} for G and w is given below.

	1	2	3	4	5
1	B	A, C	A, C	B	A, C
2	S, A	B	S, C	S, A	-
3	$\emptyset$	B	B	-	-
4	$\emptyset$	S, A, C	-	-	-
5	S, A, C	-	-	-	-

$S \in V_{15}$, therefore the string $baaba \in L(G)a$.

Another widely used normal form of grammars is called *Greibach Normal Form* (**GNF**).

Definition 1.8 (Greibach Normal Form). A **CFG** $G = (V, T, P, S)$ is said to be in **GNF** if all of its productions are of the form:

$A \mapsto a\alpha$

where $A \in V$, $a \in T$ and α is a (possibly empty) string of variables.

Theorem 1.4. *Any context-free language (not containing ε) can be generated by a **CFG** in **GNF**.*

Actually, this theorem can be strengthened: any context-free language (not containing ε) can be generated by a **CFG** in 2-**GNF** (all production rules are of the form: $A \mapsto aBC$, $A \mapsto aB$, $A \mapsto a$).

Example 1.5 ($|a| = |b|$ in **2-GNF**). Now we consider a grammar in **2-GNF**. Let $G = (\{S, A, B\}, \{a, b\}, P, S)$. Assume P consists of the following productions:

$S \mapsto aB|bA,$
$A \mapsto a|aS|bAA,$
$B \mapsto b|bS|aBB.$

The language generated by this grammar consists of all nonempty strings in which number of a is equal to the number of b.

Below we show a sample derivation in this grammar.

$S \to aB \to aaBB \to aabSB \to aabaBB \to aababB \to aababbS \to aababbbA \to aababbba$

1.4 Categorial grammars

Categorial grammars are grammars based on type logics. Their characteristic feature is use of types to describe grammatical aspects of words.

We present here briefly *Classical Categorial Grammars* (**CCG**s) based on logic **AB** of Ajdukiewicz and Bar-Hillel. For more details see e.g. [Bus07].

Let Pr be a set of *primitive types*, also called *atomic types*. *Types* or *formulae* are denoted by letters $A, B, C, ...$; they are formed out of primitive types by means of two conditionals $\rightarrow$ and $\leftarrow$, denoted by $\backslash$ and $/$. Therefore, if A and B are types, then $A\backslash B$ and A/B are types. Finite sequences or strings of types are denoted by capitals $X, Y, ...$ We write XY for a concatenation of the strings X and Y; in rules and sequents we write X, Y.

The logic **AB** can be defined as a rewriting system with the following rules.

$(AB1)$ $A, (A\backslash B) \Rightarrow B$,
$(AB2)$ $(B/A), A \Rightarrow B$,

where A, B are any formulae. A string of types X reduces to a type A in **AB**, if A is the result of finitely many (possibly zero) successive applications of rules $(AB1)$, $(AB2)$; we write $X \Rightarrow_{\mathbf{AB}} A$. In situations when it is not confusing we can omit the subscript **AB** and write $X \Rightarrow A$. Expressions of the form $X \Rightarrow A$ are called *sequents*. $\vdash_{\mathbf{AB}} X \Rightarrow A$ denotes that a sequent $X \Rightarrow A$ is provable in **AB**. It means that a sequence X reduces to A in **AB**. This terminology will be used for other systems.

Let us take as an example the following types of words:

some: $Det = np/n$
poet: n
dreams: $np\backslash s$,

then for a string *some poet dreams* there is the following reduction.

$(np/n), n, (np\backslash s) \Rightarrow np, (np\backslash s) \Rightarrow s$.

One defines different classes of categorial grammars based on different calculi.

Definition 1.9 (R-grammar)**.** An *R-grammar* is a quadruple $(\Sigma, I, B, \mathbf{R})$ such that

- Σ is nonempty, finite lexicon,
- I is a mapping assigning a finite set of types to each word from the lexicon,
- B is a distinguished type,
- R is a calculus of types.

Definition 1.10 (**CCG**)**.** An **AB**-grammar is called a *Classical Categorial Grammar* (**CCG**): $G = (\Sigma, I, B, \mathbf{AB})$.

Types of this grammar are formulae of the logic **AB**. If $(v, A) \in I$ then we write $G: \ v \mapsto A$. A grammar G *assigns* type A to a string $w = v_1 \dots v_n$, such that $v_i \in \Sigma$, $i = 1, ..., n$ (we write $w \mapsto_G A$) if there exist types A_i such that $G: \ v_i \mapsto A_i$ for $i = 1, ..., n$ and $A_1, ...A_n \Rightarrow_{\mathbf{AB}} A$.

Definition 1.11 (Language of a categorial grammar)**.** The set of all strings to which G assigns the principal type B:

$$L(G) = \{w \in \Sigma^+ : w \mapsto_G B\}$$

is called the *language of the grammar* G.

Below we give a few linguistic examples of sentences parsed by means of a **CCG**. In such **CCG**s the principal type is usually s.

Example 1.6.

John	*likes*	*fresh*	*milk.*
n	$(n\backslash s)/n$	n/n	n

with the following calculations:
$n,\ (n\backslash s)/n,\ n/n,\ n \ \Rightarrow \ n,\ (n\backslash s)/n,\ n \ \Rightarrow \ n,\ n\backslash s \ \Rightarrow \ s.$

He	*works.*
$s/(n\backslash s)$	$n\backslash s$

with the calculations:
$s/(n\backslash s),\ n\backslash s \ \Rightarrow \ s.$

He	*likes*	*Jane.*
$s/(n\backslash s)$	$(n\backslash s)/n$	n

with the calculations:
$s/(n\backslash s),\ (n\backslash s)/n,\ n \ \Rightarrow \ s/(n\backslash s),\ (n\backslash s) \ \Rightarrow \ s.$

Jane	*works*	*for*	*John.*
n	$n\backslash s$	$(s\backslash s)/n$	n

with the calculations:
$n,\ (n\backslash s),\ (s\backslash s)/n,\ n \ \Rightarrow \ n,\ n\backslash s,\ s\backslash s \ \Rightarrow \ n,\ n\backslash s \ \Rightarrow \ s.$

In the last sentence, one can see that parsing with **CCG**s, is not always very natural from linguistic point of view. One would prefer to consider first the verb phrase and then the subject, which cannot be done in a **CCG**. See the following example.

Example 1.7.

works	*for*	*John*
$n\backslash s$	$(s\backslash s)/n$	n

In a **CCG** one can obtain:
$n\backslash s,\ (s\backslash s)/n,\ n \ \Rightarrow \ n\backslash s,\ s\backslash s.$

But we would like to have simply:
$n\backslash s,\ (s\backslash s)/n,\ n \Rightarrow n\backslash s$.

Theorem 1.5 (Gaifman Theorem[BGS60]). ***CCG**s are weakly equivalent to ε-free **CFG**s. Moreover, every context-free language, not containing the empty string, is the language of some **CCG** whose mapping I uses only types of the form*

$$p,\ p/q,\ (p/q)/r$$

where p, q, r are atomic types.

It is easy to show that the language of every **CCG** is equivalent the language of some ε-free **CFG**.

Let G be a **CCG**, and let T_G denote the set of all types appearing in I_G. By $T(G)$ we denote the set of all subtypes of types from T_G. Clearly, $T(G)$ is finite and contains all types assigned by G to any strings. One defines a **CFG** $G_0 = (V_{G_0}, T_{G_0}, P_{G_o}, S_{G_0})$ such that

- $V_{G_0} = \Sigma_G$,
- $T_{G_0} = T(G)$,
- $S_{G_0} = s_G$,
- the set of production rules P_{G_o} consists of all rules of the following forms:

  $A \mapsto a$, where $A \in I_G(a)$
  $A \mapsto (A/B)B$
  $A \mapsto B(B\backslash A)$

  for $(A/B) \in T(G)$ and $(B\backslash A) \in T(G)$.

One easily proves:

$$\vdash_{\mathbf{AB}} A_1 \dots A_n \Rightarrow A \text{ iff } A \to^*_{G_0} A_1 \dots A_n,$$

for all $A_i, A \in T(G)$, and, therefore, $L(G) = L(G_0)$.

The converse is more sophisticated.

If a **CFG** is in **GNF** then all its production rules are of the form $A \to aA_1 \cdots A_n$. One can consider a **CCG** with $((\dots(A/A_n)/A_{n-1})\dots)/A_1 \in I(a)$ for each production of the **CFG**. To obtain an equivalent grammar in the required form one assumes $0 \leqslant n \leqslant 2$.

One can say that the following theorem is equivalent to the theorem on existence of a **GNF** for **CFG**s.

Theorem 1.6. *Every **CCG** is equivalent to a **CFG** in **CNF**.*

This theorem yields polynomial time decidability of categorial grammars. One can transform a **CCG** into a **CFG** and use algorithm like **CYK**. Therefore the recognition problem for **CCG**s can be solved in time proportional to n^3 where n is the number of words in the considered string.

Classical categorial grammars, unlike context-free grammars, are *lexical* that is all linguistic information is encoded in the lexicon, not in rules. Other kinds of type grammars are also lexical.

1.5 Type logics

A standard logic for type-logical grammars is *Lambek Calculus* **L**. The original calculus was called *Syntactic Calculus* and was introduced by Lambek in [Lam58]. Nowadays it is called *Lambek Calculus*. It is an extension of **AB**. It started with introducing *primitive or basic types*, for example s - of a sentence and n - of names and a recursive rule saying that if A and B are types, then expressions A/B (A over B), $B\backslash A$ (B under A) and $A \cdot B$ (product) are also types. For example the adjective *poor* is assigned type n/n as it modifies a name from the left producing a noun phrase. The predicate (intransitive verb) *works* is assigned type $n\backslash s$ because it transforms a name from the right to produce a sentence, see a sample sentence below.

(Poor John) works.
$n/n \quad n \quad n\backslash s \;\Rightarrow\; n\; n\backslash s \;\Rightarrow\; s.$

These calculations are based on the rules (AB1) and (AB2). However, it is a richer calculus. Below we give the algebraic axiomatization of the calculus **L**, assuming A, B, C are any types.

$$(1)\ A \Rightarrow A$$

$$(2)\ (A \cdot B) \cdot C \Rightarrow A \cdot (B \cdot C) \quad A \cdot (B \cdot C) \Rightarrow (A \cdot B) \cdot C$$

$$(3)\ \frac{A \cdot B \Rightarrow C}{B \Rightarrow A\backslash C} \quad \frac{A \cdot B \Rightarrow C}{A \Rightarrow C/B}$$

$$(4)\ \frac{A \Rightarrow B \;\; B \Rightarrow C}{A \Rightarrow C}$$

Rules (3) hold in both directions. The following laws are provable in **L**.

$$(5)\ A \cdot (A\backslash B) \Rightarrow B \quad (B/A) \cdot A \Rightarrow B$$

$$(6)\ (A\backslash B) \cdot (B\backslash C) \Rightarrow A\backslash C \quad (A/B) \cdot (B/C) \Rightarrow A/C$$

$$(7)\ A \Rightarrow (B/A)\backslash B \quad A \Rightarrow B/(A\backslash B)$$

$$(8)\ A\backslash(B/C) \Rightarrow (A\backslash B)/C \quad (A\backslash B)/C \Rightarrow A\backslash(B/C)$$

The laws (5) correspond to (AB1) and (AB2). Laws (6) are called Geach laws and (7) - Montague laws. The laws (8) mean that complex types $A\backslash(B/C)$ and $(A\backslash B)/C$ are equivalent. In practical applications it is easier not to distinguish between

them. Therefore, one writes simply $A\backslash B/C$. To avoid multiplication of parentheses, one may also write $A/B/C$ instead of $(A/B)/C$ and $C\backslash B\backslash A$ for $C\backslash(B\backslash A)$. However, parentheses must remain in the following compounds: $A/(B/C)$, $(C\backslash B)\backslash A$, $(A/B)\backslash C$ and $C/(B\backslash A)$.

L can also be axiomatized as a Gentzen-style system. It is useful for further considerations. Let X, Y, Z be sequences of types (where types are separated by commas). A Gentzen-style system for the Lambek Calculus admits the following axioms and rules

$$(Id)\ A \Rightarrow A,$$

$$(\backslash L)\ \frac{X, B, Y \Rightarrow C;\ Z \Rightarrow A}{X, Z, A\backslash B, Y \Rightarrow C} \qquad (\backslash R)\ \frac{A, X \Rightarrow B}{X \Rightarrow A\backslash B}$$

$$(/L)\ \frac{X, B, Y \Rightarrow C;\ Z \Rightarrow A}{X, A/B, Z, Y \Rightarrow C} \qquad (/R)\ \frac{X, A \Rightarrow B}{X \Rightarrow B/A}$$

$$(\cdot L)\ \frac{X, A, B, Y \Rightarrow C}{X, A \cdot B, Y \Rightarrow C} \qquad (\cdot R)\ \frac{X \Rightarrow A; Y \Rightarrow B}{X, Y \Rightarrow A \cdot B}.$$

$$(CUT)\ \frac{Z \Rightarrow A;\ X, A, Y \Rightarrow B}{X, Z, Y \Rightarrow B}.$$

In rules $(\backslash R)$ and $(/R)$ X cannot be empty. If we admit $X = \varepsilon$ in these rules we obtain Lambek Calculus with (possibly) empty antecedents ($\mathbf{L}^*$).

Theorem 1.7 (Cut-elimination Theorem [Lam58])**.** *Any sequent provable in* ***L*** *can be proved in* ***L*** *without (CUT).*

This theorem implies the *subformula property.*

Theorem 1.8. *In a cut-free proof of* $A_1, \ldots, A_n \Rightarrow A_{n+1}$ *every formula of every sequent is a sub-formula of some formula* A_i $(1 \leqslant i \leqslant n+1)$*.*

A consequence of these theorems is the decidability of the calculus **L**. If we want to prove a sequent, there are only finitely many rule instances to try, since the cut-rule is not needed. Premises of each instance are of a complexity less than the conclusion.

For both calculi **L** and $\mathbf{L}^*$ the provability problem is NP-complete [Pen08].

Another variant of the Lambek Calculus is **L1** which is **L** with a new constant 1 and a new rule and an axiom.

$$(1L)\ \frac{X, Y \Rightarrow A}{X, 1, Y \Rightarrow A} \qquad (1R)\ \Rightarrow 1.$$

$\mathbf{L}^*$ is a conservative fragment of **L1**.

All these systems admit cut elimination and are decidable.

Another formalism is $\mathbf{L}^{\backslash}$ (or $\mathbf{L}^{/}$) - the Lambek Calculus with one residual $\backslash$ ($/$ respectively) and without product called also *Unidirectional Lambek Calculus* which will be discussed later on.

Definition 1.12 (**L**-grammar)**.** An ***L**-grammar* is a grammar $G = (\Sigma, I, s, \mathbf{L})$.

Definition 1.13 ($\mathbf{L}^*$-grammar)**.** An ***L***-grammar* is a grammar $G = (\Sigma, I, s, \mathbf{L}^*)$.

L-grammars and $\mathbf{L}^*$-grammars are also called *Lambek grammars.*

Definition 1.14 (Language of a Lambek grammar)**.** The set of all strings to which a grammar G assigns the principal type s:

$$L(G) = \{w \in \Sigma^+ : w \mapsto_G s\}$$

is called the *language of the grammar* G.

Theorem 1.9 ([Pen93])**.** ***L**-grammars are equivalent to ε-free **CFG**s.*

Below we give linguistic examples of a few sentences and their type assignment in an **L**-grammar.

Example 1.8.
The calculations in the first example can be also performed in a **CCG**, but parsing using **L**-grammar is linguistically more natural, compare **Example 1.7**.

Jane	*works*	*for*	*John.*
n	$n\backslash s$	$(s\backslash s)/n$	n

with the calculations:
$n,\ n\backslash s,\ (s\backslash s)/n,\ n \Rightarrow n,\ (n\backslash s),\ (s\backslash s) \Rightarrow n,\ n\backslash s \Rightarrow s$ by rules (5) and (6).

The following example cannot be parsed using **CCG**s.

He	*likes*	*her.*
$s/(n\backslash s)$	$n\backslash s/n$	$(s/n)\backslash s$

with the calculations:
$s/(n\backslash s),\ n\backslash(s/n),\ (s/n)\backslash s \Rightarrow s/(n\backslash s),\ (n\backslash s) \Rightarrow s$ by rules (5) and (6).

For more examples of sentences that can and cannot be parsed using **CCG**s see **Example C.1** and **Example C.2**.

MODELS

A *partially ordered set* is a set with a binary relation of partial order $\leqslant$ which is reflexive, transitive and antisymmetric.

A *semigroup* is an algebraic structure consisting of a nonempty set S together with an associative binary operation $\cdot$ on S, so for all $a, b, c \in S$: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

A *monoid* is a semigroup with a unit element satisfying $a \cdot 1 = a = 1 \cdot a$ for all elements a.

A *partially ordered semigroup* is both a semigroup and a partially ordered set such that for all elements a, b, c the *monotonicity condition* is satisfied:

(MON) if $a \leqslant b$ then $ca \leqslant cb$ and $ac \leqslant bc$.

A *partially ordered monoid* is a partially ordered semigroup with 1. We will use this notion in the following chapters.

Abstract algebraic models of **L** are *residuated semigroups*. A residuated semigroup is a structure $(S, \leqslant, \cdot, \backslash, /)$ such that $(S, \leqslant, \cdot)$ is a partially ordered semigroup and $\backslash, /$ are binary operations on S called *residuals* and satisfying the following equivalences:

$$\text{(RES)}\ ab \leqslant c \text{ iff } b \leqslant a\backslash c \text{ iff } a \leqslant c/b.$$

Abstract models for $\mathbf{L}^*$ and **L1** are *residuated monoids*. A residuated monoid is a residuated semigroup with 1. A residuated monoid such that $(M, \leqslant)$ is a lattice is called a *residuated lattice*.

In Lambek's approach the operations $\backslash$, $/$ and $\cdot$ are interpreted as operations in the *algebra of languages*. Algebras of languages are instances of residuated semigroups. Proving the fact that all sequents provable in **L** are true in the algebra of ε-free languages is easy. Pentus [Pen95] solved much more difficult problem - the completeness theorem for **L**, that is the fact that all sequents true in this algebra are provable in **L**. The completeness theorem for the $(\backslash, /)$-fragment of **L** was proved earlier by Buszkowski [Bus86a].

Chapter 2

Pregroups and Pregroup Grammars

Contents

Pregroups and pregroup grammars have been an area of research since 1999. Pregroup grammars were introduced by Lambek [Lam99] as an algebraic tool for the syntactic analysis of sentences. They belong to the tradition of categorial grammars. Many grammatical aspects of quite a few natural languages have been described in terms of pregroup grammars. Some examples are different types of verbs, nouns, adjectives, adverbs, noun phrases, negative sentences, wh-questions, yes-or-no questions and relative clauses. All of them are discussed for English in [Lam08]. Some other languages to which pregroup grammars have been applied are French [BL01], German [LP04], Italian [CL01], Polish [KM08] and Japanese [Car07].

In the first section we briefly describe bilinear algebras. Then we introduce pregroups in a formal way. In the next part we discuss free pregroups, which were proposed as a way of applying pregroups to natural languages. Then we introduce pregroup grammars and finally, we describe how pregroup grammars are applied in linguistics.

2.1 Bilinear algebras

Linear logic is a *substructural logic* proposed by Girard [Gir87] as a refinement of *classical* and *intuitionistic logic*. It derives from an analysis of classical sequent calculus in the absence of the structural rules of *weakening* and *contraction*. Linear logic is related to many fields, such as programming languages, quantum physics and linguistics. A version of linear logic appropriate for linguistics was developed by Abrusci [Abr91] and Lambek [Lam93]. It is called *non-commutative linear logic* ***NcLL*** or *classical bilinear logic* and it extends the Lambek Calculus.

Algebraic models for **NcLL** are *bilinear algebras*. The term was introduced by Lambek. A bilinear algebra is a structure $\mathcal{M} = (M, \leqslant, \cdot, \backslash, /, 1, 0)$ such that $(M, \leqslant, \cdot, \backslash, /, 1)$ is a residuated monoid, and $0 \in M$ satisfies the following equations:

$$a = (0/a)\backslash 0,$$

$$a = 0/(a\backslash 0),$$

for any $a \in M$. One can notice that $a \leqslant (0/a)\backslash 0$, and $a \leqslant 0/(a\backslash 0)$ hold in all residuated semigroups. 0 is called a *dualizing element*.

One defines $a^r = a\backslash 0$ and $a^l = 0/a$. Then one can show that $(b^l a^l)^r = (b^r a^r)^l$, for all elements $a, b \in M$. This element is denoted by $a \oplus b$, where $\oplus$ corresponds to the operation "par" of Girard and it can be viewed as a De Morgan dual of $\otimes$ (in our notation, used in the former Chapter, it is simply $\cdot$).

In 1999 Lambek [Lam99] introduced *Compact Bilinear Logic*, which strengthens and, at the same time, simplifies the multiplicative fragment of **NcLL**. One identifies $\oplus$ with $\otimes$, and 1 with 0. Algebraic models of *Compact Bilinear Logic* are pregroups.

2.2 Pregroups

The idea of pregroups, as other categorial grammars, is that sentences are built from words by use of the lexical rules only. We assume that words have syntactic properties which can be described by a finite set of pregroup types [Lam01]. A string of words is considered a *sentence* if, and only if, for each word we can find such a type in the lexicon that the concatenation of those types can be transformed into a sentence by some calculations performed on it. Performing the calculations means here applying some pregroup laws.

The advantage of pregroups in particular, is that the calculus of pregroups is computationally simpler, even though it is stronger, than Lambek Calculus, which is definitely more powerful than **AB**. In **Section 2.5** we give some examples of English sentences, their type assignment and checking their correctness. In parsing some sentences calculus **AB** suffices, and for some other sentences, rules not admissible in the calculus **AB** are required.

Definition 2.1 (Pregroup)**.** A *pregroup* is a structure $\mathcal{M} = (M, \leqslant, \cdot, l, r, 1)$ such that:

- $(M, \leqslant, \cdot, 1)$ is a partially ordered monoid
- l, r are unary operations on M, satisfying the adjoint laws

  (Al) $a^l a \leqslant 1 \leqslant a a^l$,
  (Ar) $a a^r \leqslant 1 \leqslant a^r a$,

  for all $a \in M$.

The element a^l (resp. a^r) is called the *left* (resp. *right*) *adjoint* of a.

Proposition 2.1 (Uniqueness of adjoints). *Let $\mathcal{M}$ be a partially ordered monoid. Then, for each element $a \in M$ there exists at most one element $b \in M$ satisfying $ba \leqslant 1 \leqslant ab$ and at most one element $c \in M$ satisfying $ac \leqslant 1 \leqslant ca$.*

Proof. Let us assume $ba \leqslant 1 \leqslant ab$ and $b'a \leqslant 1 \leqslant ab'$. Using these inequalities, we obtain:

$$b = b \cdot 1 \leqslant b(ab') = (ba)b' \leqslant 1 \cdot b' = b'.$$

In an analogous way we obtain $b' \leqslant b$, whence $b = b'$. Similarly, assuming $ac \leqslant 1 \leqslant ca$ and $ac' \leqslant 1 \leqslant c'a$, $c = c'$ is proved. □

Consequently, in a pregroup a^l is the only element b such that $ba \leqslant 1 \leqslant ab$ and a^r is the only element c such that $ac \leqslant 1 \leqslant ca$.

It is worth noticing that the following laws are easily derivable from (Al), (Ar).

(i) $1^l = 1 = 1^r$,

(ii) $(a^l)^r = a = (a^r)^l$,

(iii) $(ab)^l = b^l a^l, (ab)^r = b^r a^r$,

(iv) $a \leqslant b$ iff $b^l \leqslant a^l$ iff $b^r \leqslant a^r$.

We will prove these laws for the part of left adjoint; the part with right adjoints can be done analogously.

We start with (i). From (Al) we have $1^l \cdot 1 \leqslant 1$ that is $1^l \leqslant 1$ and similarly $1 \leqslant 1^l$, whence $1 = 1^l$.

Now we prove (ii). $(a^l)^r \leqslant (aa^l)(a^l)^r = a(a^l(a^l)^r) \leqslant a$. On the other hand $a \leqslant ((a^l)^r a^l)a = (a^l)^r(a^l a) \leqslant (a^l)^r$. Hence $a = (a^l)^r$.

To prove (iii) we can use the fact that adjoints are unique. It suffices to prove that $(b^l a^l)(ab) \leqslant 1 \leqslant (ab)(b^l a^l)$ which is obvious.

Finally, we will show that (iv) holds. Let us assume $a \leqslant b$. Then $b^l \leqslant b^l aa^l \leqslant b^l ba^l \leqslant a^l$. Now assume $b^l \leqslant a^l$. Then $a \leqslant bb^l a \leqslant ba^l a \leqslant b$.

In any pregroup one can define $a\backslash b = a^r b$ and $a/b = ab^l$. These operations satisfy

$$\text{(RES) } ab \leqslant c \text{ iff } b \leqslant a\backslash c \text{ iff } a \leqslant c/b.$$

To prove it assume $ab \leqslant c$. Then $b \leqslant a^r ab \leqslant a^r c$. Conversely, assume $b \leqslant a^r c$. Then $ab \leqslant aa^r c \leqslant c$. Thus $ab \leqslant c$ iff $b \leqslant a^r c = a\backslash c$. Similarly, one proves that $ab \leqslant c$ iff $a \leqslant cb^l$. Therefore, one obtains a *residuated monoid*. In this way pregroups can be expanded to residuated monoids.

As a consequence, all sequents provable in Lambek Calculus with empty antecedents ($\mathbf{L}^*$) are valid in all pregroups. However, the converse does not hold. For example $(ab)/c = a(b/c)$ is true in all pregroups but not in all residuated monoids, as observed by Lambek [Lam99]. Buszkowski [Bus07] gave another example of a sequent not provable in $\mathbf{L}^*$, but translation is true in pregroups. The sequent is $(a/((b/b)/c))/c \Rightarrow a$, where a, b, c are atomic. One easily sees that its translation $ac^{ll}b^{ll}b^l c^l = a(c^{ll}(b^{ll}b^l)c^l) \leqslant a(c^{ll}c^l) \leqslant a$ is valid in pregroups. The translation is obtained as follows:

$(a/((b/b)/c))/c = (a/((b/b)/c))c^l = a(((b/b)/c)^l c^l = a(bb^l c^l)^l c^l = ac^{ll}b^{ll}b^l c^l$.

If in a residuated monoid one defines $a^l = 1/a$ and $a^r = a\backslash 1$, then the conditions $a^l a \leqslant 1$ and $aa^r \leqslant 1$ are satisfied, but the inequalities $1 \leqslant aa^l$ and $1 \leqslant a^r a$ need not hold. As observed by Buszkowski [Bus07], a residuated monoid is a pregroup iff it satisfies:

$$a(b/c) = (ab)/c \text{ and } (a\backslash b)c = a\backslash(bc).$$

This is the only possible pregroup structure on this partially ordered monoid, since adjoints are uniquely determined by the partially ordered monoid structure, see **Proposition** 2.1. However, in an arbitrary partially ordered monoid, the operations of adjoints and residuals need not be defined for all elements.

Pregroups are a generalization of *partially ordered groups*. Partially ordered groups are structures of the form $(M, \leqslant, \cdot, ()^{-1}, 1)$, such that $(M, \leqslant, \cdot, 1)$ is a partially ordered monoid and $()^{-1}$ is a unary operation satisfying $aa^{-1} = 1 = a^{-1}a$, for all $a \in M$. In a partially ordered group one can define $a^r = a^l = a^{-1}$, then (Ar) and (Al) hold, and such group is an example of a pregroup. Conversely, if in a pregroup $a^l = a^r$ for all $a \in M$, then one can set $a^{-1} = a^l = a^r$ and the pregroup is a partially ordered group.

A pregroup is said to be *proper*, if it is not a partially ordered group. So a pregroup is proper if, for some $a \in M$, $a^l \neq a^r$. We should notice that if the operation $\cdot$ is commutative, then $a^l = a^{-1} = a^r$ since $a^l a = 1 = aa^l$ and $aa^r = 1 = a^r a$. Hence, a commutative pregroup is simply a partially ordered Abelian group. Consequently, proper pregroups cannot be commutative. Further we are interested only in proper pregroups, as partially ordered groups are not useful in applications in linguistics.

Assume now $(P, \leqslant)$ is a poset and $\mathbf{M}(P)$ is the set of all order-preserving functions $f : P \mapsto P$. A function $f : P \mapsto P$ is said to be *order-preserving* if it satisfies the condition:

if $x \leqslant y$, then $f(x) \leqslant f(y)$, for all $x, y \in P$.

One defines a relation $\leqslant$ on the set $\mathbf{M}(P)$ as follows:

$f \leqslant g$ iff, for all $x \in P$, $f(x) \leqslant g(x)$.

Let $I(x)$ be the *identity function*: $I(x) = x$ and let $\circ$ denote the operation of composition of functions: $(f \circ g)(x) = f(g(x))$. A structure $(\mathbf{M}(P), \leqslant, \circ, I)$ is a partially ordered monoid.

Definition 2.2 (Pregroup of functions)**.** A pregroup is called a *pregroup of functions*, if its reduct is a substructure of the partially ordered monoid $\mathbf{M}(P)$, for some partially ordered set $(P, \leqslant)$.

Buszkowski [Bus01] proved the following facts.

Proposition 2.2. *Each pregroup is isomorphic with a pregroup of functions.*

Proof. Let $(M, \leqslant, l, r, 1)$ be a pregroup. Let us consider the partially ordered set $(M, \leqslant)$. For $a \in M$, f_a is defined: $f_a(x) = ax$. One can easily observe that it is an order-preserving function from M to M. Moreover, $f_{ab} = f_a \circ f_b$, $f_1 = I$, and $a \leqslant b$ iff $f_a \leqslant f_b$. Then, the mapping $h(a) = f_a$ is an isomorphic embedding of $(M, \leqslant, \cdot, 1)$ into $\mathbf{M}(M)$. Adjoints are defined as follows: $f_a^l = f_{a^l}$ and $f_a^r = f_{a^r}$. Then, (Al) and (Ar) hold, and h is the required isomorphism. □

Proposition 2.3. *Let F be a pregroup of functions on a partially ordered set $(P, \leqslant)$. For each $f \in F$ and for all $x \in P$ there holds:*

$$f^l(x) = min\{y \in P : x \leqslant f(y)\},$$
$$f^r(x) = max\{y \in P : f(y) \leqslant x\}.$$

Proof. By (Al) $f^l(f(x)) \leqslant x \leqslant f(f^l(x))$ for all $x \in P$. Therefore, $f^l(x)$ is in the set of all y such that $x \leqslant f(y)$. Assume z belongs to the same set. Then $x \leqslant f(z)$. Hence, $f^l(x) \leqslant f^l(f(z)) \leqslant z$, which proves the first equality. The proof of the second equality is dual. □

Proposition 2.4. *Let F be a substructure of the monoid $\mathbf{M}(P)$. If for each $f \in F$ functions f^r and f^l, defined as above, exist and belong to F, then F is a pregroup of functions.*

Proof. We prove (Al). $f^l(f(x)) = min\{y : f(x) \leqslant f(y)\} \leqslant x$ and $x \leqslant f(f^l(x))$. (Ar) is proved analogously. □

Corollary 2.1. *If F is a pregroup of functions on a poset $(P, \leqslant)$, then all functions in F are* unbounded*, which means that $\forall x \exists y(x \leqslant f(y))$ and $\forall x \exists y(f(y) \leqslant x)$.*

The natural example of a proper pregroup given by Lambek [Lam99] is a pregroup consisting of all unbounded, order preserving functions from the set of integers into itself. For $f(n) = 2n$, one obtains $f^l(n) = [\frac{(n+1)}{2}]$ and $f^r = [\frac{n}{2}]$ (where $[x]$ denotes the greatest integer $m \leqslant x$). So, $f^l \neq f^r$. Hence the pregroup is proper. This pregroup is called the *Lambek pregroup*.

One defines *iterated adjoints* $a^{(n)} = a^{rr\ldots r}$ and $a^{(-n)} = a^{ll\ldots l}$, where n is a non-negative integer, ^{r} and ^{l} are iterated n times and a is any element of M. By definition $a^{(0)} = a$.

The following laws are provable in pregroups.

$(a^{(n)})^l = a^{(n-1)}, (a^{(n)})^r = a^{(n+1)}$, for any integer n.
$a^{(n)}a^{(n+1)} \leqslant 1 \leqslant a^{(n+1)}a^{(n)}$, for any integer n.
$(a^{(n)})^{(m)} = a^{(n+m)}$, for any integers n, m.
$(a_1^{(n_1)}\ldots a_m^{(n_m)})^l = a_m^{(n_m-1)}\ldots a_1^{(n_1-1)}$, for any integers $n_1, \ldots, n_m$.
$(a_1^{(n_1)}\ldots a_m^{(n_m)})^r = a_m^{(n_m+1)}\ldots a_1^{(n_1+1)}$, for any integers $n_1, \ldots, n_m$.
$a \leqslant b$ iff $a^{(n)} \leqslant b^{(n)}$, for any even integer n.
$a \leqslant b$ iff $b^{(n)} \leqslant a^{(n)}$, for any odd integer n.

2.3 Free pregroups and Compact Bilinear Logic

Lambek's approach to syntactic analysis is based on the notion of a *free pregroup*, generated by a finite, partially ordered set of *basic types* $(P, \leqslant)$. Most definitions in this section follow Lambek [Lam99]. Elements of the set P are also called *atoms* or *basic types* and denoted by letters p, q, r, s. Elements of P are treated as constant symbols.

For any $p \in P$ we construct a set of formal expressions of the form $p^{(n)}$ for any integer n; one defines $p^{(0)} = p$. Expressions of the form $p^{(n)}$, $p \in P, n \in \mathcal{Z}$ are called *simple terms* ($\mathcal{Z}$ denotes the set of integers).

A *term* is a finite sequence (string) of simple terms. Terms are also called *types*. They are denoted by capitals $X, Y, Z, U, ...$ Types are assigned to different words in the lexicon. They refer to the role the given word has in sentences. Usually one word can be assigned many types.

One defines a binary relation $\Rightarrow$ on the set of terms as the least reflexive and transitive relation, satisfying the following clauses.

(CON) $X, p^{(n)}, p^{(n+1)}, Y \Rightarrow X, Y$,
(EXP) $X, Y \Rightarrow X, p^{(n+1)}, p^{(n)}, Y$,
(IND) $X, p^{(n)}, Y \Rightarrow X, q^{(n)}, Y$, if $p \leqslant q$ and n is even or $q \leqslant p$ and n is odd.

(CON), (EXP), (IND) are called Contraction, Expansion and Induced Step, respectively. They can be treated as rules of a term rewriting system. $X \Rightarrow Y$ is true iff X can be rewritten into Y by a finite number of applications of these rules. This rewriting system is Lambek's original form of the logic of pregroups which is also called *Compact Bilinear Logic* (**CBL**).

It is useful to define X^r and X^l for any type X:

$$\varepsilon^l = \varepsilon = \varepsilon^r;$$
$$X^r = (p_1^{(n_1)} ... p_k^{(n_k)})^r = p_k^{(n_k+1)} ... p_1^{(n_1+1)},$$
$$X^l = (p_1^{(n_1)} ... p_k^{(n_k)})^l = p_k^{(n_k-1)} ... p_1^{(n_1-1)},$$

where $n_1, ..., n_m$ are arbitrary integers.

A structure consisting of he set of all terms with the relation $\Rightarrow$, operations $\cdot$ (concatenation), $^l, ^r$, and the unit element ε is a *preordered* pregroup. *Preorder* is the reflexive and transitive relation (the difference with partial order is that preorder need not be antisymmetric). *Preordered pregroups* are defined like pregroups except that $\leqslant$ can be a preorder and in monoid equations $=$ is replaced by $\sim$, where: a $\sim$ b iff $a \leqslant b$ and $b \leqslant a$. The relation $\sim$ is an equivalence relation, compatible with $\cdot, ^l, ^r$. If $\mathcal{M}$ is a preordered pregroup, then $\mathcal{M}/\sim$ is a pregroup.

For types, $X \sim Y$ iff $X \Rightarrow Y$ and $Y \Rightarrow X$. The relation $\sim$ is nontrivial even for a trivial partially ordered set $(P, =)$. For instance $p, p^{(1)}, p \sim p$ and $p, p^{(-1)}, p \sim p$. (In pregroups, $aa^r a = a$ and $aa^l a = a$.) It is a congruence on the preordered pregroup, defined above. The quotient-structure with the ordering defined by:

$$[X] \leqslant [Y] \text{ iff } X \Rightarrow Y$$

is a pregroup, called the *free pregroup generated by* $(P, \leqslant)$, and denoted $F(P, \leqslant)$, or $F(P)$. One easily proves that it is free in the sense that every function f from P to a pregroup $\mathcal{M}$ which preserves the order has a unique extension to an (order-preserving) homomorphism from $F(P)$ to $\mathcal{M}$. This yields the completeness theorem.

Theorem 2.1. *$X \Rightarrow Y$ iff, for all pregroups $\mathcal{M}$, $f([X]) \leqslant f([Y])$, for any order-preserving function f from P to $\mathcal{M}$.*

After Lambek [Lam99] one also defines a *Generalized Contraction* that combines Contraction with Induced Step

(GCON) $X, p^{(n)}, q^{(n+1)}, Y \Rightarrow X, Y$,

and *Generalized Expansion* combining Expansion with Induced Step

(GEXP) $X, Y \Rightarrow X, p^{(n+1)}, q^{(n)} Y$,

where in both cases either $p \leqslant q$ and n is even, or $q \leqslant p$ and n is odd. Obviously, (GCON) amounts to an Induced Step followed by a Contraction and (GEXP) to an Expansion followed by an Induced Step. Clearly, (CON) and (EXP) are special cases of (GCON) and (GEXP), respectively.

Lambek [Lam99] proves an important property called *the Switching Lemma* or *Normalization Theorem.*

Theorem 2.2 (Normalization Theorem)**.** *If $X \Rightarrow Y$ in **CBL**, then there exist Z and U, such that $X \Rightarrow Z$ applying only (GCON), $Z \Rightarrow U$ by (IND) only and $U \Rightarrow Y$ using (GEXP) only.*

Consequently, there holds.

Corollary 2.2. *If $X \Rightarrow t$, where t is a simple term or ε, then X can be reduced to t by (CON) and (IND) only.*

Such reductions are easily computable and can be simulated by a context-free grammar. This yields the polynomial time decidability of **CBL** [Bus01, Bus07]. We discuss the problem in **Chapter 3**.

Observe that for any types X, Y there holds:

$X \Rightarrow Y$ iff $XY^r \Rightarrow \varepsilon$ iff $Y^l X \Rightarrow \varepsilon$

Therefore, to check whether $X_1 ... X_n \Rightarrow Y$ holds in **CBL**, one can verify the condition $X_1 ... X_n Y^r \Rightarrow \varepsilon$.

2.4 Pregroup grammars

Definition 2.3 (Pregroup grammar)**.** A *pregroup grammar **PG*** is a quintuple $G = (\Sigma, P, \leqslant, s, I)$ such that:

- Σ is a nonempty, finite alphabet,

- $(P, \leqslant)$ is a finite poset,
- $s \in P$,
- I is a finite relation between symbols from Σ and nonempty types (on P); for $a \in \Sigma$, $I(a)$ denotes the set of all types X such that $(a, X) \in I$.

Let $x \in \Sigma^+$, $x = a_1...a_n$ $(a_i \in \Sigma)$. One says that the grammar G *assigns* type Y to x, if there exist types $X_i \in I(a_i)$, $i = 1, ..., n$, such that $X_1, ..., X_n \Rightarrow Y$ in **CBL**; we write $x \rightarrow_G Y$. One defines the language of a pregroup grammar G as follows:

Definition 2.4 (The language of a pregroup grammar)**.** Let $G = (\Sigma, P, \leqslant, s, I)$ be a pregroup grammar. The *language of the pregroup grammar* G is the set:
$L(G) = \{x \in \Sigma^+ : x \rightarrow_G s\}$.

Due to the Normalization Theorem, by **Corollary 2.2**, while parsing pregroup grammars, one may restrict the rules to (CON)s and (IND)s.

2.5 Linguistic examples

Further in the thesis we give examples of application of pregroup grammars to English. However, actually we could give examples in any language already described in terms of pregroups, as parsing pregroup grammars does not depend on the particular language, see **Appendix B**.

Applying pregroup grammars to natural languages consists of constructing a pregroup grammar for a fragment of the given language. First, one defines a set of basic types, a partial order on this set and a lexicon in which to each word one or more types are assigned. Then, to check the correctness of a string of words in the given language, an algorithm of parsing pregroup grammars is run on that grammar.

Atoms are characteristic for the natural language described. Some appear in most languages, like π_1 - first person subject or i - infinitive, and others are specific for a particular language, like declination forms of nouns in Polish. For English the most important basic types are as follows (after Lambek [Lam08]):

π - subject when person is not important,
n - noun,
i - infinitive of intransitive verb,
j - infinitive of any complete verb phrase,
o - direct object,
q - yes-or-no question,
$\bar{q}$ - question,
p_1 - present participle,
p_2 - past participle,
s - declarative sentence.

When analyzing a language, some new types, subtypes and supertypes are added. Subtypes are denoted by subscripts, for example π_i (i-th person singular subject,

$i = 1, 2, 3$) and s_i, q_i (declarative or interrogative sentence in present tense for $i = 1$ or past tense for $i = 2$). Supertypes are created using some symbol drawn over the type symbol, e.g. $\bar{\ },\tilde{\ },\hat{\ }$, like $\bar{q}$.

In our examples we need the following partial order.

$$\pi_i \leqslant \pi, \quad i \leqslant j, \quad s_i \leqslant s,\ q_i \leqslant q,\ q \leqslant \bar{q}.$$

A full list of English pregroup basic types and partial order is given in **Appendix A**.

It is useful to represent reductions by means of *links*. A link indicates a pair of simple terms that reduce to the empty string. Further we say that the string *reduces to 1*, meaning the reduction to the empty string. If a whole string reduces to 1, then each simple term appearing in the string is connected by a link with another simple term. Each term can be a member of only one link and links cannot cross. We can see the links in the following example.

Example 2.1.

I	*will*	*find*	*him.*
$(\pi_1)\ (\pi^r$	s_1	$j^l)\ (i$	$o^l)\ (o)$

We should notice that the only type that does not take part in any link is s_1, so the given string reduces to this type and it is therefore a grammatically (syntactically) correct sentence (since $s_1 \Rightarrow s$).

Example 2.2.

Whom		*will*	*she*	*go*	*with?*	
$(\bar{q}$	o^{ll}	$q^l)\ (q_1$	j^l	$\pi^l)\ (\pi_3)$	$(i)\ (j^r$	$i\ o^l)$

The only type that does not take part in any link is $\bar{q}$, so it is a grammatically (syntactically) correct question.

For more linguistic examples in English see **Section 4.1**, **Section 4.3.4**, or **Appendix C**, and in other languages see **Appendix B**.

Chapter 3

Pregroup Grammars and Context-free Grammars

Contents

In this chapter we consider the equivalence of pregroup grammars and context-free grammars. At first, we present the Equivalence Theorem and briefly outline its proof (which are given in [Bus01]). In one direction the proof is based on the fact that context-free languages are closed under homomorphism and inverse homomorphism.

Further, we discuss an idea of partial composition [Béc07] used to construct a **CFG** equivalent to the given pregroup grammar. However, the size of the constructed grammar is exponential. We give a direct polynomial construction based on the proof of the Equivalence Theorem. The size of constructed **CFG** is also polynomial. We continue with the construction of a pushdown automaton equivalent to the given pregroup grammar.

3.1 The Equivalence Theorem

Theorem 3.1 (The Equivalence Theorem [Bus01])**.** *Pregroup grammars are weakly equivalent to $\varepsilon-$free **CFG**s.*

We recall the proof. It consists of two main parts. In the first one it is showed that every $\varepsilon-$free **CFG** is equivalent to some **PG**. The second one consists of proving that every **PG** is equivalent to an ε-free **CFG**.

We start with a brief description of the first part. One uses **Theorem 1.5** proved in [BGS60] that every $\varepsilon-$free **CFG** G is equivalent to an **AB**-grammar G', whose types are in one of the forms: p, p/q, $(p/q)/r$, where p, q, r are atoms.

One defines a translation of any type A to a pregroup type $T(A)$ as follows:

$T(p) = p$;
$T(A\backslash B) = T(A)^r\ T(B)$;
$T(A/B) = T(A)\ T(B)^l$.

For example, if $A = (p/q)/r$ then $T(A) = pq^l r^l$.

Let $G_0 = (\Sigma, P, =, s, I)$ be a pregroup grammar such that

- Σ is the (input) alphabet of G',
- P is a set of all atoms appearing in G',
- s is the basic type of G',
- $(a, T(A)) \in I$ iff there exists a type A, assigned to a by G'.

One also proves: $A_1, \ldots, A_n \Rightarrow s$ in the reduction system of **CCG**'s iff $T(A_1), \ldots, T(A_n) \Rightarrow s$ in the free pregroup. Therefore, every ε-free **CFG** is equivalent to some **PG** which is based on a trivial poset $(P, =)$ with neither right adjoints, nor iterated left adjoints $p^{ll\ldots l}$ (due to the restriction on types of the **CCG** G').

Now we recall main lines of the second part of the proof, that is showing that every pregroup grammar is equivalent to an $\varepsilon-$free **CFG**. One sets a pregroup grammar $G = (\Sigma, P, \leqslant, s, I)$. For $a_i \in \Sigma$, $i = 1, \ldots, n$ there holds

$a_1 \ldots a_n \in L(G)$ if, and only if, $X_1, \ldots, X_n \Rightarrow s$,

for some types $X_i \in I(a_i)$, $i = 1, \ldots, n$. By Lambek Normalization Theorem, it suffices to consider only applications of the rules (CON) and (IND) in reductions to the sentence type s.

Let $T(G)$ be the set of all types involved in I and let $S(G)$ be the set of all simple terms appearing in types from $T(G)$. Let G_1 be a **CFG** such that $L(G_1) = X \in (S(G))^+ : X \Rightarrow s\}$, where X is a string of types. One sets

- $S(G)$ - the terminal alphabet of G_1,
- N - the nonterminal alphabet of G_1 containing an additional symbol E and all types $q^{(n)}$ such that $p^{(n)} \leqslant q^{(n)}$, for some $p^{(n)} \in S(G)$,
- s - the start symbol of G_1,
- production rules of G_1 are all possible rules in any of the forms:

  (i) $q^{(n)} \mapsto p^{(n)}$,
  (ii) $\mathbb{X} \mapsto E\mathbb{X}$,
  (iii) $\mathbb{X} \mapsto \mathbb{X}E$,
  (iv) $E \mapsto p^{(n)}p^{(n+1)}$,

  where all symbols are in N ($\mathbb{X}$ is an arbitrary element from N) and $p^{(n)} \leqslant q^{(n)}$ in the first rule.

Warning: this **CFG** violates the condition of disjointness of the terminal and nonterminal alphabets. This condition can easily be fulfilled by replacing terms in N by their copies.

Using Normalization Theorem it is proved that

$$X \Rightarrow t \text{ in the free pregroup iff } t \Rightarrow^* X \text{ in a } \mathbf{CFG}\ G_1,$$

where X is an arbitrary string on $N \setminus \{E\}$ and $t \in N$ - a simple term.

Let a be an element of the alphabet Σ such that $I(a) = \{X_1, \ldots, X_k\}$, $k > 0$. For each such element a there are chosen k new symbols $a(1), \ldots, a(k)$. All of these symbols corresponding to the symbols from Σ form a new alphabet Σ'. One defines a homomorphism h from $(\Sigma')^*$ to $(S(G))^*$, and a homomorphism g from $(\Sigma')^*$ to Σ^* as follows

$$h(a(i)) = X_i,$$
$$g(a(i)) = a.$$

Hence, $L(G) = g(h^{-1}(L(G_1)))$, so it is a context-free language, see [Bus01].

3.2 Partial composition

Béchet [Béc07] notices that the proof of the Equivalence Theorem described in **Section 3.1** does not yield any direct construction of a **CFG** equivalent to the given pregroup grammar. The proof discussed uses the fact that the class of context-free languages is closed under homomorphisms and inverse homomorphisms. However, actually we proposed a direct polynomial construction of a **CFG** equivalent to the given **PG** using these closure properties (see [BM08] and **Section 3.3**). Béchet gave another construction based on so called *partial composition*. Below we present briefly his construction.

Partial composition is a series of Generalized Contractions performed at the same time on a certain number of pairs of simple terms. It is defined as follows:

$$(PC)\ X p_1^{(n_1)} \ldots p_k^{(n_k)}, q_k^{(n_{k+1})} \ldots q_1^{(n_1+1)} Y \Rightarrow XY,$$

provided that $p_i^{(n_i)} \leqslant q_i^{(n_i)}$, for all $i = 1, \ldots, k$. We should notice that in this notation a comma separates types, where for pregroup grammars it is interpreted as the concatenation of strings. Hence, after each application of (PC) on the given string of types, the number of types appearing in that string decreases. It is due to the fact that (PC) acts on two types, canceling a few simple terms and concatenating the remaining simple terms of both types into one type. We should notice that the length of the resulting type can exceed maximal length of two original types. Therefore, there can be put a constraint on the rule (PC), such that the size of the resulting type is not greater than sizes of two original types. Such restricted version of (PC) is called *functional composition*.

Let $l(X)$ be the *length* of the type X, that is the total number of atoms occurring in X. The following lemma is crucial. The formulation of the lemma, as we give it, was proposed in [BM08], and it is slightly different from the original one.

Lemma 3.1 (Béchet's lemma). *If $X = X_1, \ldots, X_n \Rightarrow p$, $n \geqslant 2$, in the free pregroup, then there are $1 \leqslant i < n$ and a type Y such that*

- $X_i, X_{i+1} \Rightarrow Y$ *without (EXP)*
- $X_1, \ldots, X_{i-1}, Y, X_{i+2}, \ldots, X_n \Rightarrow p$
- $l(Y) \leqslant max\{l(X_j) : \ j = 1, \ldots, n\}$.

Béchet [Béc07] first proves a similar lemma for $X \Rightarrow 1$. The proof is based on some properties of *outer-planar graphs* and induction on the number of types in the derivation. An outer-planar graph is a graph in which all vertices are placed on a straight line and whose edges are drawn in one semi plane determined by this line. Vertices of the graph are types X_i, $i = 1, \ldots, n$ and an edge between two vertices is drawn if there exists a link corresponding to a generalized contraction between them. Obviously, such a graph is outer-planar. Using properties of outer-planar graph one chooses a type from which changes in the derivation begin, in order to obtain the searched result.

Lemma 3.1 can be derived from the described lemma, since $X \Rightarrow p$ iff $X, p^r \Rightarrow \varepsilon$.

Let $G = (\Sigma, P, \leqslant, s, I)$ be a pregroup grammar, ml the maximum lengths of the types used by G and m the maximum of absolute values of exponents of atoms used by G. Assume V is a set of types of length not greater than ml with exponents from the set $\{-m, \ldots, m\}$. One defines a context-free grammar $\tilde{G} = (\Sigma, V, s, R)$ equivalent to the given pregroup grammar $G = (\Sigma, P, \leqslant, s, I)$ with relation R defined as follows:

- $y \mapsto x, x' : \ y, x, x' \in V, \ xx' \Rightarrow y$,
- $x \mapsto a : \ a \in \Sigma, \ x \in I(a)$.

Using Béchet's lemma, one proves $L(\tilde{G}) = L(G)$. The size of that grammar is exponential in the size of the **PG** G.

3.3 Polynomial construction of a context-free grammar equivalent to a pregroup grammar

One of possible solutions to the recognition problem for pregroup grammars is constructing a context-free grammar (**CFG**), in Chomsky Normal Form, equivalent to the given pregroup grammar (**PG**). We propose such construction. It is polynomial in the size of the pregroup grammar [BM08]. The basic idea of the construction is based on the proof of the Equivalence Theorem described in **Section 3.1**.

Let $G = (\Sigma, P, \leqslant, s, I)$ be a pregroup grammar. Let $T(G)$, $S(G)$ and N be as in **Section 3.1**.

First we construct a **CFG** G_1 such that

$$L(G_1) = \{X \in (S(G))^+ : X \Rightarrow s\}$$

as in **Section 3.1**.

Now we construct a **CFG** G_2. For any $a \in \Sigma$, we create a new nonterminal symbol $X(a)$. The idea is that grammar G_2 could insert symbols $X(a)$, in front of terms from $S(G)$, within the strings generated by G_1. The grammar G_2 is defined as follows

- $S(G) \cup \{X(a) : a \in \Sigma\}$ - the terminal alphabet of G_2,
- N - nonterminal alphabet of G_2 (defined as in **Section 3.1**),
- s - the start symbol,
- the production rules are:

  (i) all rules of G_1,
  (ii) all possible rules of the form $\mathbb{Y} \mapsto X(a)\mathbb{Y}$, for $\mathbb{Y} \in S(G)$.

We define $Q(G)$ as the set of all types Y such that

$$XY \in T(G), \text{ for some } X.$$

Now we define a new **CFG** G_3 such that

- Σ - the terminal alphabet of G_3,
- NT - the nonterminal alphabet of G_3 consisting of symbols of the form $([X], \mathbb{Y}, [Z])$, where $X, Z \in Q(G)$ and $\mathbb{Y}$ is a terminal or nonterminal symbol of G_2,
- $([\varepsilon], s, [\varepsilon])$ - the start symbol of G_3,
- the production rules of G_3 are all possible rules in any of the four forms

  (I) $([X_1], \mathbb{X}, [X_3]) \mapsto ([X_1], \mathbb{Y}, [X_2])([X_2], \mathbb{Z}, [X_3])$,
  for any rule $\mathbb{X} \mapsto \mathbb{Y}\mathbb{Z}$ of G_2 and all $X_1, X_2, X_3 \in Q(G)$.
  (II) $([X_1], \mathbb{X}, [X_2]) \mapsto ([X_1], \mathbb{Y}, [X_2])$,
  for all rules $\mathbb{X} \mapsto \mathbb{Y}$ of G_2 and $X_1, X_2 \in Q(G)$.
  (III) $([tX], t, [X]) \mapsto \varepsilon$, for all $X \in Q(G)$ and $t \in S(G)$.
  (IV) $([\varepsilon], X(a), [X]) \mapsto a$, for all $a \in \Sigma, X \in I(a)$.

We [BM08] prove the following fact.

Proposition 3.1. $L(G) = L(G_3)$.

Proof. We should notice that in a derivation of a string $x \in L(G_3)$ rules (III), (IV) can be applied at the very end. Neither ε nor any symbol from Σ appears on the left-hand side of any rule in G_3, so the result of application of rules of the form (III) and (IV) cannot be used as an input to any new rule.

We denote $X_i = t^i_1 \dots t^i_{k(i)}$, $X^i_j = t^i_j t^i_{j+1} \dots t^i_{k(i)}$, for $i = 1, \dots, n$, $j = 1, \dots, k(i)$. We also denote:

$$Y_i = ([\varepsilon], X(a_i), [X_i])([X^i_1], t^i_1, [X^i_2]) \dots ([t^i_{k(i)}], t^i_{k(i)}, [\varepsilon]).$$

One easily notices that $X^i_1 = X_i$ and $X^i_{k(i)} = t^i_{k(i)}$.

Assume $a_1, \dots, a_n \in \Sigma$ and $a_1 \dots a_n \in L(G)$, which means there exist $X_1 \in I(a_1), \dots, X_n \in I(a_n)$ such that $X_1, \dots, X_n \Rightarrow s$. Then $X_1 \dots X_n$ is derivable from s in G_1. Hence, the string $X(a_1)X_1X(a_2)X_2 \dots X(a_n)X_n$ is derivable from s in G_2. This string can be written as follows:

$$X(a_1)\ t^1_1 \dots t^1_{k(1)}\ X(a_2)\ t^2_1 \dots t^2_{k(2)} \dots X(a_n)\ t^n_1 \dots t^n_{k(n)}.$$

Then the string $Y_1 \dots Y_n$ is derivable from $([\varepsilon], s, [\varepsilon])$ in G_3. This string can be derived by means of the rules (I) and (II) exclusively, as only these rules are related to the rules of the grammar G_2 and a derivation in G_2 is transformed into a derivation in G_3. Now rules (III) and (IV) can be applied to the string $Y_1 \dots Y_n$. Thus, one obtains $a_1 \dots a_n$, which means $a_1 \dots a_n \in L(G_3)$.

Conversely, let us assume $a_1, \dots, a_n \in \Sigma$ and $a_1 \dots a_n \in L(G_3)$. Then, there exists a derivation of $a_1, \dots a_n$ in G_3 from $([\varepsilon], s, [\varepsilon])$. The derivation must contain steps $([\varepsilon], X(a_i), [X_i]) \to a_i$ such that $X_i \in I(a_i)$ for each $i = 1, \dots, n$. To obtain a string containing elements $([\varepsilon], X(a_i), [X_i])$ in a derivation in G_3 one can use only rules (I) and (II). All other elements of the string thus obtained have to be in a form allowing application of the rule (IV). To satisfy this condition, in a string derived from $([\varepsilon], s, [\varepsilon])$ each element $([\varepsilon], X(a_i), [X_i])$ have to be followed by the string $([X^i_1], t^i_1, [X^i_2]) \dots ([t^i_{k(i)}], t^i_{k(i)}, [\varepsilon])$, where $X_i = t^i_1 \dots t^i_{k(i)}$. Hence, this string is of the form $Y_1 \dots Y_n$. But application of rules (I) and (II) reflects a derivation in the grammar G_2, thus the string $X(a_1)X_1X(a_2)X_2 \dots X(a_n)X_n$ is derivable from s in G_2. By omitting elements $X(a_i)$ we obtain that $X_1 \dots X_n$ is derivable from s in G_1. It means that $a_1 \dots a_n \in L(G)$, which finishes the proof. □

G_3 contains nullary rules. A standard procedure can be applied to eliminate them. We use the following lemma:

Lemma 3.2. *$([X], \mathbb{Y}, [Z]) \to^* \varepsilon$ in G_3 if, and only if, there exist $t_1, \dots, t_n \in S(G)$, $n > 0$, such that $X = t_1 \dots t_n Z$ and $\mathbb{Y} \to^* t_1 \dots t_n$ in G_1.*

Proof. First the "if" part is proved. Assume there exist $t_1, \dots, t_n \in S(G)$, $n > 0$, such that $X = t_1 \dots t_n Z$ and $\mathbb{Y} \to^* t_1 \dots t_n$ in G_1. Then

$$([X], \mathbb{Y}, [Z]) \to^*_{G_3} ([X], t_1, [t_2 \cdots t_n Z])([t_2 \cdots t_n Z], t_2, [t_3 \cdots t_n Z]) \cdots ([t_n Z], t_n, [Z])$$

After applying rule (III) n times one obtains $([X], \mathbb{Y}, [Z]) \to^* \varepsilon$.

The "only if" part is proved by induction on the length of the derivation of ε from $([X], \mathbb{Y}, [Z])$ in G_3.

If ε is derivable in one step, then the only rule that can be applied is of the form (III) with $\mathbb{Y} = t$. The rule takes the form $([tZ], t, [Z]) \mapsto \varepsilon$, hence there exist $X = tZ$ and $t \mapsto t$ in G_1, which means $\mathbb{Y} \to^* t$ in G_1. Therefore the thesis is satisfied.

Now assume that the derivation has k steps, $k > 1$. Two cases have to be considered, depending on which rule is applied in the first step.

1. The derivation starts with rule (I), that is a rule of the form

   $([X_1], \mathbb{Y}, [X_3]) \mapsto ([X_1], \mathbb{X}, [X_2])([X_2], \mathbb{Z}, [X_3])$, where $\mathbb{Y} \mapsto \mathbb{X}\mathbb{Z}$ is a rule of G_2.

   Due to the fact that the considered derivation leads to ε, there holds $([X_1], \mathbb{X}, [X_2]) \to^* \varepsilon$, $([X_2], \mathbb{Z}, [X_3]) \to^* \varepsilon$, both in less than k steps. By the induction hypothesis, there exist $t_1, \ldots, t_m \in S(G)$ and $u_1, \ldots, u_n \in S(G)$, $m, n > 0$, such that

   - $X_1 = t_1 \ldots t_m X_2$,
   - $X_2 = u_1 \ldots u_n X_3$,
   - $\mathbb{X} \to^*_{G_1} t_1 \cdots t_m$,
   - $\mathbb{Z} \to^*_{G_1} u_1 \cdots u_n$.

   $\mathbb{Y} \mapsto \mathbb{X}\mathbb{Z}$ is a rule of G_1, hence $\mathbb{X}, \mathbb{Z} \in N$. It suffices to notice that $X_1 = t_1 \cdots t_m u_1 \cdots u_n X_3$ and $\mathbb{Y} \to^* t_1 \cdots t_m u_1 \cdots u_n$ in G_1 to obtain the thesis.

2. The derivation starts with rule (II), that is a rule of the form

   $([X_1], \mathbb{Y}, [X_2]) \mapsto ([X_1], \mathbb{Z}, [X_2])$, where $\mathbb{Y} \mapsto \mathbb{Z}$ is a rule of G_2.

   Due to the fact that the considered derivation leads to ε, there holds $([X_1], \mathbb{Z}, [X_2]) \to^* \varepsilon$, in less than k steps. By the induction hypothesis, there exist $t_1, \ldots, t_m \in S(G)$, $m > 0$, such that

   - $X_1 = t_1 \ldots t_m X_2$,
   - $\mathbb{Z} \to^*_{G_1} t_1 \ldots t_m$.

   $\mathbb{Y} \mapsto \mathbb{Z}$ is a rule of G_1, so $\mathbb{Z} \in N$. It suffices to notice that $\mathbb{Y} \to^* t_1 \ldots t_m$ in G_1 to obtain the thesis.

□

We transform the grammar G_3 into a grammar G_4, which has no nullary rules and generates the same language as G_3. Rules (III) are dropped and new rules of the following forms are added.

- $([X_1], \mathbb{X}, [X_3]) \mapsto ([X_2], \mathbb{Z}, [X_3])$, for any rule (I) such that $([X_1], \mathbb{Y}, [X_2]) \to^* \varepsilon$ in G_3,
- $([X_1], \mathbb{X}, [X_3]) \mapsto ([X_1], \mathbb{Y}, [X_2])$, for any rule (I) such that $([X_2], \mathbb{Z}, [X_3]) \to^* \varepsilon$ in G_3.

A standard argument of formal language theory yields:

Proposition 3.2. $L(G_3) = L(G_4)$.

Hence by **Proposition 3.1** we obtain the following theorem.

Theorem 3.2. $L(G) = L(G_4)$.

We need still one more step, as G_4 contains unary rules (that is rules of the type: $([X_1], \mathbb{X}, [X_2]) \mapsto ([X_{1'}], \mathbb{Y}, [X_{2'}])$). In a usual way, we can eliminate them, and thus obtain a grammar in Chomsky Normal Form. The resulting grammar G_5 is equivalent to the grammar G_4. Unary rules of G_4 are of the type (II) and those obtained while transforming G_3 into G_4. The procedure is as follows: we take any unary rule

$$([X_1], \mathbb{X}, [X_2]) \mapsto ([Y_1], \mathbb{Y}, [Y_2])$$

we drop such a rule and we add for each rule

$([Y_1], \mathbb{Y}, [Y_2]) \mapsto u$ where u is of one of the forms:

- a,
- $([Z_1], \mathbb{Z}, [Z_2])$,
- $([Z_1], \mathbb{Z}, [Z_2])([Z_2], \mathbb{Z}', [Z_3])$,

a new rule

$([X_1], \mathbb{X}, [X_2]) \mapsto u$ (unless this was a rule previously removed).

We repeat the action until there are no more unary rules. So production rules of G_5 are of the form:

- $([X_1], \mathbb{X}, [X_2]) \mapsto ([X_3], \mathbb{Y}, [X_4])([X_4], \mathbb{Z}, [X_5])$,
where $([X_1], \mathbb{X}, [X_2]) \to^* ([X_3], \mathbb{Y}, [X_4])([X_4], \mathbb{Z}, [X_5])$ in G_4,
- $([X_1], \mathbb{X}, [X_2]) \mapsto a$,
where $([X_1], \mathbb{X}, [X_2]) \to^* ([\varepsilon], X(a), [Y])$ in G_4 and $a \in \Sigma,\ Y \in I(a)$.

The size of grammars G_3, G_4 and G_5 is polynomial in the size of G. Moreover, all of them can be constructed in polynomial time. For more details on the complexity see **Section 3.3.1**. Therefore, any standard recognition algorithm for **CFG**'s (e.g. **CYK**) yields a polynomial time recognition algorithm for the given pregroup grammar G. However, a practical implementation of this procedure is rather involved. In **Chapter 4** we present a direct recognition algorithm for pregroup grammars.

3.3.1 Computational complexity of the construction

In this section we show that the construction described above can be performed in polynomial time. It depends on the size of the initial pregroup grammar. We have to estimate sizes of all grammars appearing in the construction and show that each step of the construction requires polynomial time only.

We count the size of a pregroup grammar as the sum of lengths of types assigned by I to the elements of the alphabet Σ (that is all types in $T(G)$) plus the sum of

absolute values of all exponents appearing in types from $T(G)$ plus the cardinality of the set of primitive types P. Let us assume $|P|$ is the cardinality of the set P and TLE the sum of lengths of types assigned to words in the lexicon plus the sum of absolute values of all exponents appearing in those types. Let k be the cardinality of the alphabet Σ and n_i a number of types assigned to the symbol a_i. Then x_{ij} is the j-th type assigned to the element a_i. Hence,

$$TLE = \sum_{i=1}^{k} \sum_{j=1}^{n_i} |x_{ij}|,$$

where $|x_{ij}|$ is counted as the sum of the number of simple terms in the type x_{ij} and the sum of absolute values of the exponents appearing in x_{ij}. Therefore, $|G| = TLE + |P|$. We should notice that if $I(a) \neq \emptyset$ for any $a \in \Sigma$, then the cardinality of Σ is not greater than $|G|$.

The size of a **CFG** is counted as the sum of numbers of its nonterminal symbols, terminal symbols and number of its rules. Let S be the cardinality of the set $S(G)$, that is the number of terminal symbol of G_1. If m is the maximum absolute value of exponents of elements of the set $S(G)$, then $S \leqslant (2m+1) \cdot |P| = O(|G|^2)$. Let $|N|$ be the number of nonterminal symbols of the grammar G_1: $|N| \leqslant (2m+1) \cdot |P| + 1 = O(|G|^2)$. We count the number of rules of G_1: $|N| \cdot (3 + |P|) = O(|G|^3)$. Details are presented in **Table 3.1**.

(i) $q^{(n)} \mapsto p^{(n)}$;	$\|N\| \cdot \|P\|$
(ii) $\mathbb{X} \mapsto E\mathbb{X}$;	$\|N\|$
(iii) $\mathbb{X} \mapsto \mathbb{X}E$;	$\|N\|$
(iv) $E \mapsto p^{(n)}p^{(n+1)}$,	$\|N\|$
total number	$\|N\| \cdot (3 + \|P\|)$

Table 3.1: Number of rules of the grammar G_1

Therefore, the size of the grammar G_1 is $O(|G|^3)$.

Now, we estimate size of the grammar G_2. Number of its terminal symbols is $|T_{G_2}| = S + |\Sigma| \leqslant 2|G| = O(|G|^2)$, and there are $|N|$ nonterminals as in G_1. The number of rules is $|N| \cdot (3 + |P| + |\Sigma|) = O(|G|^3)$. For detailed count of number of rules see **Table 3.2**.

(i) rules of the grammar G_1	$\|N\| \cdot (3 + \|P\|)$
(ii)rules of the form $\mathbb{Y} \mapsto X(a)\mathbb{Y}$	$\|N\| \cdot \|\Sigma\|$
total number	$\|N\| \cdot (3 + \|P\| + \|\Sigma\|)$

Table 3.2: Number of rules of the grammar G_2

The size of G_2 is hence $O(|G|^3)$.

Further, we consider the grammar G_3. The number of its terminal symbols is $|\Sigma|$. The cardinality of $Q(G)$ is $Q = O(|G|)$. So, the number of nonterminal symbols

(I) $([X_1], \mathbb{X}, [X_3]) \mapsto ([X_1], \mathbb{Y}, [X_2])([X_2], \mathbb{Z}, [X_3])$	$Q^3 \cdot (3 \cdot S + \lvert\Sigma\rvert \cdot S) = O(\lvert G\rvert^6)$
(II) $([X_1], \mathbb{X}, [X_2]) \mapsto ([X_1], \mathbb{Y}, [X_2])$	$Q^2 \cdot \lvert N\rvert \cdot \lvert P\rvert = O(\lvert G\rvert^5)$
(III) $([tX], t, [X]) \mapsto \varepsilon$	$Q \cdot S = O(\lvert G\rvert^3)$
(IV) $([\varepsilon], X(a), [X]) \mapsto a$	$Q \cdot \lvert\Sigma\rvert = O(\lvert G\rvert^3)$
total number	$O(\lvert G\rvert^6)$

Table 3.3: Number of rules of the grammar G_3

is $|N_{G_3}| = Q \cdot (|T_{G_2}| + |N|) \cdot Q = O(|G|^4)$. Again, we count the number of rules in a table, see **Table 3.3**.

The size of the grammar G_3 is mostly affected by the number of rules and it is also $O(|G|^6)$. The number of terminal and nonterminal symbols of grammars G_4 and G_5 remain the same as for G_3. In **Table 3.4** we count a number of rules of G_4.

(I) $([X_1], \mathbb{X}, [X_3]) \mapsto ([X_1], \mathbb{Y}, [X_2])([X_2], \mathbb{Z}, [X_3])$	$Q^3 \cdot (3 \cdot S + \lvert\Sigma\rvert \cdot S) = O(\lvert G\rvert^6)$
(Ia) $([X_1], \mathbb{X}, [X_3]) \mapsto ([X_2], \mathbb{Z}, [X_3])$	$Q^3 \cdot (3 \cdot S + \lvert\Sigma\rvert \cdot S) = O(\lvert G\rvert^6)$
(Ib) $([X_1], \mathbb{X}, [X_3]) \mapsto ([X_1], \mathbb{Y}, [X_2])$	$Q^3 \cdot (3 \cdot S + \lvert\Sigma\rvert \cdot S) = O(\lvert G\rvert^6)$
(II) $([X_1], \mathbb{X}, [X_2]) \mapsto ([X_1], \mathbb{Y}, [X_2])$	$Q^2 \cdot \lvert N\rvert \cdot \lvert P\rvert = O(\lvert G\rvert^5)$
(IV) $([\varepsilon], X(a), [X]) \mapsto a$	$Q \cdot \lvert\Sigma\rvert = O(\lvert G\rvert^3)$
total number	$O(\lvert G\rvert^6)$

Table 3.4: Number of rules of the grammar G_4

Therefore, the size of G_4 is $O(|G|^6)$. The size of G_5 is the same since the number of symbols does not change and number of new rules is not greater than the number of deleted rules.

It is easily seen that grammars G_1, G_2 and G_3 can be constructed from the grammar G in polynomial time. The construction of G_4 from G_3 requires application of **Lemma 3.2**. The condition of the lemma can be checked in time $O(|G|^3)$, with applying a cubic algorithm for provability in **CBL**. The elimination of unary rules can also be done in polynomial time and the grammar G_5 can be constructed in polynomial time.

Thus we have presented a polynomial construction of a **CFG** equivalent to the given **PG**.

3.4 Transformation of a pregroup grammar into a push-down automaton in polynomial time

We propose a construction of a **PDA** equivalent to a **PG** $G = (\Sigma, P, \leqslant, s, I)$ that can be performed in polynomial time in the size of G. It is done in a similar way as the construction presented in **Section 3.3**.

We assume a **PDA** accepts a string $x \in \Sigma^*$, if it reads the input and terminates in the unique final state with the empty stack. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a

PDA such that

- Q is a set of states consisting of the symbols of the form $[X]$ for any $X \in Q(G)$ and the only final state f ($Q(G)$ defined as in **Section 3.3**),
- Σ is the input alphabet,
- $N_1 = N \setminus E$ is the stack alphabet,
- $q_0 = [\varepsilon]$,
- $F = \{f\}$,
- δ is a transition function described as follows

  (i) in state $[\varepsilon]$ read a and go to state $[X]$, for $X \in I(a)$,

  (ii) in any state except f replace $p^{(n)}$ on the stack by $q^{(n)}$ such that $p^{(n)} \leqslant q^{(n)}$,

  (iii) in state $[p^{(n)}X]$ push $p^{(n)}$ on the stack and go to state $[X]$,

  (iv) in state $[p^{(n+1)}X]$ pull $p^{(n)}$ of the stack and go to state $[X]$,

  (v) in state $[\varepsilon]$ pull s of the stack and go to state f.

Chapter 4

Parsing Pregroup Grammars

Contents

This chapter consists of five main sections. We start with a brief description of problems that can be encountered while parsing pregroup grammars. In the second section, we present a recognition algorithm of Savateev [Sav09], which was defined for the Unidirectional Lambek Calculus ($\mathbf{L}^{\backslash}$).

Further, we present our dynamic polynomial recognition algorithm for pregroup grammars. It is based on the algorithm of Savateev. We show the proof of the correctness of the algorithm. We describe how to amend the algorithm to obtain a full parsing algorithm and we give some examples of how it works. We also discuss some other recognition and parsing algorithms for pregroup grammars ([Oeh04] and [Pre07]).

4.1 Problems with parsing pregroup grammars

Parsing in linguistics and computer science is a process of syntactic analysis of some text, that is a process of determining the grammatical structure of a given text consisting of some tokens (words) with respect to some grammar. To perform this action grammar has to be formalized in some way. The most often approach to parsing natural languages is to present the grammar in a form of a **CFG** or an equivalent grammar and use the **CYK** algorithm or its modification.

The problem of parsing pregroup grammars is complex for different reasons. The main reason is that pregroup grammars allow ambiguities of different types.

We start with lexical ambiguities. The first kind of ambiguity arises from the fact that the same word can be assigned different types, depending on the role it takes in the sentence. It is the most important factor in the problem of parsing pregroup grammars. For example **do** is of the type $\pi_k^r s_1 i^l$ in a declarative sentence and of the type $q_1 i^l \pi_k^l$ in questions where $k = 1, 2$ and the types denote:

π_i - i-th person (in English π_2 denotes also all persons plural) subject,
s_1 - sentence in present tense,
i - infinitive,
q_1 - yes-or-no question in present tense.

Example 4.1. To illustrate the problem better, let us see some sentences. We take from the lexicon the following words:

come	$i,\ p_2,\ \pi_1^r s_1, \pi_2^r s_1, ...$
does	$\pi_3^r s_1 i^l, q_3 i^l \pi_3^l, ...$
has	$\pi_3^r s_1 p_2^l,\ q_1 p_2^l \pi_3^l, ...$
not	$jj^l,\ ii^l,\ p_1 p_1^l,\ p_2 p_2^l, ...$
she	π_3
would	$\pi^r s_2 j^l,\ q_2 j^l \pi^l,$

where the new types are:

p_1 - present participle,
p_2 - past participle,
s_2 - sentence in past tense.

All pregroup types defined for English in [Lam08] are listed in **Appendix A.1** whereas partial order is given in **Appendix A.2** (also after [Lam08]). Here, we define the partial order as follows:

$i \leqslant i'$,
$i' \leqslant j'$,
$j' \leqslant j$,
$\pi_3 \leqslant \pi$.

We should notice that by the transitivity, from the first three inequalities, we obtain also $i \leqslant j$.

One can see below that both *come* and *not* are assigned different types in different sentences.

She *does* *not come.*
$(\pi_3)(\pi_3^r\ s_1\ i^l)(i\, i^l)\ (i)$

She *has* *not* *come.*
$(\pi_3)(\pi_3^r\ s_1\ p_2^l)(p_2\ p_2^l)(p_2)$

She would not come.
$(\pi_3)(\pi^r\ s_2\ j^l)(j\ j^l)(i)$

Clearly, the number of all possible strings of types assigned to a given string of words can increase exponentially with length of the string of words.

The other type of lexical ambiguity is when there are more reductions to the sentence type for the same string of words. Different reductions to the sentence type can come from either different type assignments or the same type assignment. It may be connected with different semantical reading of the same structure. Obviously, pregroup grammars, in their basic form, do not deal with semantic problems but some of them appear anyway, as can be noticed in **Example 4.2**.

Example 4.2.
I promised to see her today.
$(\pi_1)\ (\pi^r\ s_2\ j^l\ i\ \bar{j}^l)\ (\bar{j}\ i^l)\ (i\ o^l)\ (o)\ \ (i^r\ i)$

I promised to see her today.
$(\pi_1)\ (\pi^r\ s_2\ j^l\ i\ \bar{j}^l)\ (\bar{j}\ i^l)\ (i\ o^l)\ (o)\ \ (i^r\ i)$

In this case the two reductions show difference in semantics, as *today* modifies the verb *see* in the first sentence and the verb *promise* in the second one.

However, not only in strings reducing to the sentence type, but also in many others, there are more possible reductions. It is another problem to solve while parsing pregroup grammars. If in a string there appear three types: $a^l\mathbb{U}a\mathbb{V}a^r$, such that $\mathbb{U} \to 1$ and $\mathbb{V} \to 1$, then there are two possible contractions $a^l a \leqslant 1$ and $aa^r \leqslant 1$. In some situations both of them can lead to the searched result, producing two different reductions, in other cases only one or none of them is useful. The problem of choosing the appropriate link becomes even more complex when we consider pregroups with partial order.

Clearly, similar problems appear when parsing **CFGs** or **CCG**s , but we concentrate here on pregroup grammars and these problems are crucial.

Actually, a given string of types can have up to $C_n = \frac{1}{n+1}\begin{pmatrix} 2n \\ n \end{pmatrix}$ different reductions to the sentence type (where n is the number of simple terms in the string). Therefore an efficient parsing algorithm cannot return all possible parsings.

4.2 Algorithm of Savateev

The algorithm of Savateev [Sav09] is a recognition algorithm for $\mathbf{L}^{\backslash}$-grammars. In order to describe the algorithm we need to present the calculus $\mathbf{L}^{\backslash}$ (Unidirectional Lambek Calculus). It is the \-fragment of the Lambek Calculus. There is a set of atomic types P: $p, q, r, \ldots$. Atomic types form complex types. Types of $\mathbf{L}^{\backslash}$ consist

of atomic types connected by means of the operation $\backslash$ which is the only connective of this calculus. The type formation rule is the same as the rule for $\backslash$ in Lambek Calculus: if A and B are types, then the expression $B\backslash A$ is also a type.

An $\mathbf{L}^{\backslash}$-grammar and its language are defined similarly to other Lambek grammars. Therefore, an $\mathbf{L}^{\backslash}$-grammar is a quadruple $(\Sigma, I, s, \mathbf{L}^{\backslash})$. Each symbol from the alphabet Σ is assigned a finite number of types.

Expressions of the form p^n such that p is an atomic type and n is a positive integer are called *atoms*. The set of atoms is denoted by Atn and $FS = (Atn)^+$. Savateev defines a mapping γ from types to elements of a free semigroup (FS) in the following way:

$$\gamma(p) = p^1,$$
$$\gamma(A\backslash B) = (\gamma(A))^{\top}\gamma(B),$$

where $(p^i)^{\top} = p^{i+1}$ and $(p^{n_1}...p^{n_k})^{\top} = p^{n_k+1}...p^{n_1+1}$. Such a mapping is precisely the translation of types in the formalism of free pregroups except that p^{n+1} in this notation stands for $p^{(n)}$ in pregroups.

A string of atoms has a good pairing, if there exists a system of non-crossing links (as in pregroup reductions) such that

(1) each atom is a left or right end of exactly one link,

(2) if p^n is the left end of a link, then its right end is p^{n+1},

(3) if p^n is the left end of a link and n is even, then the interval between p^n and the right end p^{n+1} contains an atom p^m with $m < n$.

Lemma 4.1 ([Sav09]). *$A_1, ..., A_n \to B$ is provable in $\boldsymbol{L}^{\backslash}$ if and only if the string $\gamma(A_1)...\gamma(A_n)(\gamma(B))^{\top}$ has a good pairing.*

We should notice that conditions **(1)** and **(2)** yield the reducibility to 1 in **CBL** (after replacing p^{n+1} by $p^{(n)}$) of the given string. **(3)** is an additional constraint. The constraint is necessary, since $\mathbf{L}^{\backslash}$ is weaker than **CBL**.

4.2.1 The recognition algorithm

Assume $G = (\Sigma, I, B, \mathbf{L}^{\backslash})$ is a $\mathbf{L}^{\backslash}$-grammar. The input is a string x of the form $a_1...a_n \in \Sigma^+$. The algorithm checks whether $x \in L(G)$. All types assigned to each element a_i are searched and replaced by their translation, that is if A is assigned to a_i one takes $\gamma(A)$. We denote:

- $\mathcal{Z}$ - the set of integers,
- $Atn = \{p^n : p \in P, n \in \mathcal{Z}\}$ - the set of atoms
- $A, B, C, ..$ - types,
- $k^a = |I(a)|$,
- A^a_j - the j-th possible assignment of type to a, $1 \leqslant j \leqslant k^a$ (hence $I(a) = \{A^a_1, ...A^a_{k^a}\}$),

- $\mathbb{A}$ - element of FS,
- $\mathbb{A}^a_j = \gamma(A^a_j)$,
- $\mathbb{B} = \gamma(B)^\top$,
- $Q^a = \langle *\mathbb{A}^a_1 * \mathbb{A}^a_2 * ... * \mathbb{A}^a_{k^a} *\rangle$,
- $W^x = Q^{a_1}...Q^{a_n}\langle *\mathbb{B}$, $W^x \in (Atn \cup \{*, \langle, \rangle\})^*$
- W^x_i - the i-th symbol of the string W^x, $1 \leqslant i \leqslant |W^x|$
- $W^x_{[i,j]} = W^x_i W^x_{i+1}...W^x_j$ - the substring of W^x, $1 \leqslant i < j \leqslant |W^x|$.

The goal is to check whether there exist numbers $m_i \leqslant k^{a_i}, 1 \leqslant i \leqslant n$ such that $\mathbf{L}^\backslash \vdash A^{a_1}_{m_1} \dots A^{a_1}_{m_1} \Rightarrow B$. However, given **Lemma 4.1**, one can check instead whether the string $\gamma(A^{a_1}_{m_1}) \dots \gamma(A^{a_n}_{m_n})\gamma(B)^\top = \mathbb{A}^{a_1}_{m_1} \dots \mathbb{A}^{a_n}_{m_n}\mathbb{B}$ has a good pairing.

One defines a function $M(i,j)$ $(1 \leqslant i < j \leqslant |W^x|)$ as follows: $M(i,j) = 1$ if one of the following cases holds:

- **1.** $W^x_{[i,j]}$ lies in FS and has a good pairing.
- **2.** $W^x_{[i,j]}$ is of the form $\langle ...\rangle ...\langle \mathbf{V} * \mathbb{C}$, where:
  - $\mathbb{C} \in FS$,
  - $\mathbf{V}$ contains no angle brackets,
  - in $W^x_{[i,j]}$ there are g $(g \geqslant 0)$ pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*\mathbb{D}_h*$ in between them, such that $\mathbb{D}_h \in FS$ and the string $\mathbb{D}_1...\mathbb{D}_g\mathbb{C}$ has a good pairing.
- **3.** $W^x_{[i,j]}$ is of the form $\mathbb{D} * \mathbf{U}\rangle ...\langle \mathbf{V} * \mathbb{C}$, where:
  - $\mathbb{C}, \mathbb{D} \in FS$,
  - $\mathbf{U}, \mathbf{V}$ contain no angle brackets,
  - in $W^x_{[i,j]}$ there are g $(g \geqslant 0)$ pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*\mathbb{E}_h*$ in between them, such that $\mathbb{E}_h \in FS$ and the string $\mathbb{D}\mathbb{E}_1...\mathbb{E}_g\mathbb{C}$ has a good pairing.

In all other cases $M(i,j) = 0$.

Obviously, the whole string W^x is of the second form. Therefore, $M(1, |W^x|) = 1$ if and only if there exists a string $\mathbb{D}_1 \dots \mathbb{D}_n\mathbb{B}$ that has a good pairing, which is equivalent to $x \in L(G)$.

One computes $M(i,j)$ dynamically. The initial case is for strings of length two. One computes $M(i, i+1) = 1$ iff for all $W^x_i = p^{2m+1}$, $W^x_{i+1} = p^{2m+2}$ for some $p \in P$ and $m \in \mathcal{Z}$.

If one already knows $M(g,h)$, for all $1 \leqslant g < h \leqslant |W^x|$ such that $h - g < j - i$, $M(i,j)$ can be computed. There are several cases:

- **A1.** $W^x_i, W^x_j \in Atn$. If there exists k such that $i < k < j - 1$ and $W^x_k \in Atn$, $W^x_{(k+1)} \in Atn$ and $M(i,k) = 1$ and $M(k+1,j) = 1$, then we put $M(i,j) = 1$.
- **A2.** $W^x_i = p^m$, $W^x_j = p^{m+1}$, for some $p \in P$ and $m \in \mathcal{Z}$. If $M(i+1, j-1) = 1$, m is odd or m is even and there exists k such that $i < k < j$ and the superscript of W^x_k is less than m, then $M(i,j) = 1$.

- **A3.**$W^x_{[i,j]}$ is of the form $p^1 * \mathbf{U}\rangle\langle\mathbf{V} * p^2$, $p \in P$, where $\mathbf{U}, \mathbf{V}$ contain no angle brackets. Then we put $M(i,j) = 1$.
- **A4.** $W^x_{[i,j]}$ is of the form $\langle ...\rangle ...\langle ...p^{(m)}$, $p \in P, m \in \mathcal{Z}$. If there exists k such that $i < k < j$ and $W^x_k = *, W^x_{[i+1,k]}$ contains no angle brackets and $M(k+1,j) = 1$, then $M(i,j) = 1$.
- **A5.** $W^x_{[i,j]}$ is of the form $p^1 * ...\rangle ...\langle ...p^2$. If $M(k,j-1) = 1$, where k is the position of the first left angle bracket in the string $W^x_{[i,j]}$, then we put $M(i,j) = 1$.

In all other cases $M(i,j) = 0$.

Lemma 4.2. *The algorithm described above computes correctly values of the function* $M(i,j)$.

The proof given by Savateev shows: if $M(i,j) = 1$, then the algorithm computes $M(i,j) = 1$ correctly for all substrings. It is done by induction on $j - i$.

The proof depends strongly on the fact that all types appearing in the string W^x are translation of types of an $\mathbf{L}^{\backslash}$-grammar. Types thus obtained can only have positive exponents and they start with an atom with superscript 1.

Our algorithm for pregroup grammars has to be essentially modified, so that it admits arbitrary simple terms, both with positive and negative integers, and allowing any simple term as the first and the last symbol of the type; our pregroup types need not be restricted to translation of types of $\mathbf{L}^{\backslash}$. Moreover, we did not need constraints on a string to have a good pairing, but instead we require that the string reduces to 1 using Generalized Contractions, which means that some of simple terms can be reduced with more than one simple terms.

4.3 A dynamic parsing algorithm for pregroup grammars

Our goal is to check whether a given string $x = a_1, \ldots, a_n$, $a_i \in \Sigma$, $i = 1, \ldots, n$ is a member of the language generated by a pregroup grammar G, that is whether $x \in L(G)$. If this is the case, we want to obtain the appropriate derivation. We recall that the condition $X_1 ... X_n \Rightarrow s$ in **CBL** is equivalent to the condition $X_1 ... X_n s^{(1)} \Rightarrow \varepsilon$. In what follows, we write 1 for ε.

We define the algorithm in a style proposed by Savateev (see the previous section) for Unidirectional Lambek Calculus. It is a dynamic algorithm working on a special form of a string, containing all possible type assignments for words of the sentence to parse.

4.3.1 Definition of the function M

We fix a pregroup grammar $G = (\Sigma, P, \leqslant, s, I)$. We take a string of words $x \in \Sigma^+$ such that $x = a_1 ... a_n$. We use special symbols $*, \langle, \rangle$. Let us denote:

- $\mathcal{Z}$ - the set of integers,
- $T = \{p^{(n)} : p \in P,\ n \in \mathcal{Z}\}$ - the set of simple terms,
- $X, Y, Z, ..$ - elements of T^*,
- $k^a = |I(a)|$,
- X_j^a - the j-th possible assignment of type to a, $1 \leqslant j \leqslant k^a$ (hence $I(a) = \{X_1^a, ...X_{k^a}^a\}$),
- $Q^a = \langle *X_1^a * X_2^a * ... * X_{k^a}^a *\rangle$,
- $W^x = Q^{a_1}...Q^{a_n}\langle *s^{(1)}$, $W^x \in (T \cup \{*, \langle, \rangle\})^*$
- W_i^x - the i-th symbol of the string W^x, $1 \leqslant i \leqslant |W^x|$,
- $W_{[i,j]}^x = W_i^x W_{i+1}^x ... W_j^x$ - the substring of W^x, $1 \leqslant i < j \leqslant |W^x|$.

Let $M(i,j),\ 1 \leqslant i < j \leqslant |W^x|$ be a function, such that $M(i,j) = 1$ iff one of the following conditions holds.

- **M1.** $W_{[i,j]}^x \in T^+$ and it reduces to 1.
- **M2a.** $W_{[i,j]}^x$ is of the form $\langle ...\rangle ...\langle \mathbf{V} * Z$, where:
  - $Z \in T^+$,
  - $\mathbf{V}$ contains no angle brackets,
  - in $W_{[i,j]}^x$ there are precisely g ($g \geqslant 0$) pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them, such that $X_h \in T^+$ and the string $X_1...X_g Z$ reduces to 1.
- **M2b.** $W_{[i,j]}^x$ is of the form $Y * \mathbf{U}\rangle ...\langle ...\rangle$, where:
  - $Y \in T^+$,
  - $\mathbf{U}$ contains no angle brackets,
  - in $W_{[i,j]}^x$ there are precisely g ($g \geqslant 0$) pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them, such that $X_h \in T^+$ and the string $Y X_1...X_g$ reduces to 1.
- **M3.** $W_{[i,j]}^x$ is of the form $Y * \mathbf{U}\rangle ...\langle \mathbf{V} * Z$, where:
  - $Y, Z \in T^+$,
  - $\mathbf{U}, \mathbf{V}$ contain no angle brackets,
  - in $W_{[i,j]}^x$ there are precisely g ($g \geqslant 0$) pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them, such that $X_h \in T^+$ and the string $Y X_1...X_g Z$ reduces to 1.
- **M4.** $W_{[i,j]}^x$ is of the form $\langle ...\rangle ...\langle ...\rangle$, where:
  - in $W_{[i,j]}^x$ there are precisely g ($g \geqslant 1$) pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them, such that $X_h \in T^+$ and the string $X_1...X_g$ reduces to 1.

In all other cases $M(i,j) = 0$.

Obviously, the whole string W^x is of the form **M2a**. Therefore, $M(1, |W^x|) = 1$ iff there exists a string $X_1 \dots X_n s^r$ which reduces to 1. Each X_i is the searched type for a_i, and $x \in L(G)$.

4.3.2 The recognition algorithm

We compute $M(i,j)$ dynamically. There are two initial cases.

- **I1** $M(i,i+1) = 1$ only if $W_i^x = p^{(m)}$, $W_{i+1}^x = q^{(m+1)}$ and the following condition holds

  (GC) $p, q \in P$, $m \in \mathcal{Z}$ and either $p \leqslant q$ in P and m is even, or $q \leqslant p$ in P and m is odd.

- **I2** If $W_{[i,j]}^x$ is of the form $p^{(m)} * \mathbf{U}\rangle\langle \mathbf{V} * q^{(m+1)}$, the condition (GC) holds and strings $\mathbf{U}$, $\mathbf{V}$ contain no angle brackets. Then, we put $M(i,j) = 1$.

When we already know $M(g,h)$, for all $1 \leqslant g < h \leqslant |W^x|$ such that $h-g < j-i$, we can compute $M(i,j)$. There are several cases:

- **A1a.** $W_i^x, W_j^x \in T$. If there exists k such that $i < k < j-1$ and $W_k^x \in T$, $W_{(k+1)}^x \in T$ and $M(i,k) = 1$ and $M(k+1,j) = 1$, then we put $M(i,j) = 1$.

- **A1b.** $W_i^x, W_j^x \in T$. If there exists k such that $i < k < j-1$ and $W_k^x = \rangle$, $W_{(k+1)}^x = \langle$ and $M(i,k) = 1$ and $M(k+1,j) = 1$, then we put $M(i,j) = 1$.

- **A2.** $W_i^x = p^{(m)}$, $W_j^x = q^{(m+1)}$ and the condition (GC) holds.
  If $M(i+1,j-1) = 1$, then $M(i,j) = 1$.

- **A3a.** $W_{[i,j]}^x$ is of the form $\langle ... \rangle ... \langle ... p^{(m)}$, $p \in P, m \in \mathcal{Z}$. If there exists k such that $i < k < j$ and $W_k^x = *, W_{[i+1,k]}^x$ contains no angle brackets and $M(k+1,j) = 1$, then $M(i,j) = 1$.

- **A3b.** $W_{[i,j]}^x$ is of the form $p^{(m)} ... \rangle ... \langle ... \rangle$, $p \in P, m \in \mathcal{Z}$. If there exists k such that $i < k < j$ and $W_k^x = *, W_{[k,j-1]}^x$ contains no angle brackets and $M(i,k-1) = 1$, then we put $M(i,j) = 1$.

- **A4a.** $W_{[i,j]}^x$ is of the form $p^{(m)} * ... \rangle ... \langle ... q^{(m+1)}$ and the condition (GC) holds. If $M(k,j-1) = 1$, where k is the position of the first left angle bracket in the string $W_{[i,j]}^x$, then we put $M(i,j) = 1$.

- **A4b.** $W_{[i,j]}^x$ is of the form $p^{(m)} ... \rangle ... \langle ... * q^{(m+1)}$ and the condition (GC) holds. If $M(i+1,k) = 1$, where k is the position of the last right angle bracket in the string $W_{[i,j]}^x$, then $M(i,j) = 1$.

- **A4c.** $W_{[i,j]}^x$ is of the form $p^{(m)} * ... \rangle ... \langle ... * q^{(m+1)}$ where the string "..." in between the angle brackets is not empty and the condition (GC) holds. If $M(k,k') = 1$, where k is the position of the first left angle bracket in the string $W_{[i,j]}^x$ and k' is the position of the last right angle bracket in the string $W_{[i,j]}^x$, then $M(i,j) = 1$.

- **A5.** $W^x_{[i,j]}$ is of the form $\langle ...\rangle ...\langle ...\rangle$. If $M(k,k')=1$, where W^x_k is a simple term in between the first pair of angle brackets, $W^x_{k'}$ is a simple term in between the last pair of angle brackets in the string $W^x_{[i,j]}$ and $W^x_{k-1}=*$ and $W^x_{k'+1}=*$, then $M(i,j)=1$.

In all other cases $M(i,j)=0$.

Algorithm 4.1 Recognition algorithm for a pregroup grammar G

Input: string of words x and the lexicon.

1. **Compute** W^x.
2. **For each** $1 \leqslant i < |W^x|$ **compute** $M(i,i+1)$(**I1**).
3. **For each pair** (i,j) such that $1 \leqslant i < j \leqslant |W^x|$ compute $M(i,j)$ using **I2**.
4. **Compute** dynamically $M(i,j)$ for $1 \leqslant i < j \leqslant |W^x|$.
5. **If** $M(1,|W^x|)=1$ **then** $x \in L(G)$,
   else $x \notin L(G)$.

We claim:

Theorem 4.1. *The algorithm computes $M(i,j)$ correctly.*

Proof. The proof given by Savateev consists only of showing that the algorithm correctly computes $M(i,j)$ for all substrings. In case of pregroup grammars it is not sufficient. In our proof we show that if the algorithm computes $M(i,j)=1$, then $M(i,j)=1$ according to the definition of M and then that the algorithm finds correctly all substrings for which the function $M(i,j)=1$.

At first we show that, if the algorithm computes $M(i,j)=1$, then $M(i,j)=1$ according to the definition of M. We prove it by induction on the length of the string.

For strings of length two, the algorithm computes $M(i,j)=1$ only in case when $W^x_i = p^{(m)}$ and $W^x_{i+1} = q^{(m+1)}$ and the condition (GC) holds. Then $W^x_{[i,j]}$ is of the form **(M1)**, since $W^x_{[i,j]} \in T^+$ and the string $W^x_{[i,j]}$ reduces to 1. So, $M(i,j)=1$ according to the definition of M.

The other initial case when the algorithm computes $M(i,j)=1$ is when $W^x_{[i,j]}$ is of the form $p^{(m)} * \mathbf{U} \rangle \langle \mathbf{V} * q^{(m+1)}$, the condition (GC) holds and the strings $\mathbf{U}$ and $\mathbf{V}$ contain no angle brackets. Then $W^x_{[i,j]}$ is of the form **(M3)**, since we can assume $Y = p^{(m)}$ and $Z = q^{(m+1)}$ and $g=0$. So, YZ reduces to 1. Hence, $M(i,j)=1$ according to the definition of M.

Now let us consider the recursive cases when the algorithm computes $M(i,j)=1$ (all cases of the description of the algorithm).

A1a. $W^x_i, W^x_j \in T$ and there exists k such that $i < k < j-1$ and $W^x_k \in T$, $W^x_{(k+1)} \in T$, $M(i,k)=1$ and $M(k+1,j)=1$. We illustrate the case below.

$$W^x_{[i,j]} = \underbrace{p_1^{(m_1)} ... p_2^{(m_2)}}_{M(i,k)=1} \underbrace{p_3^{(m_3)} ... p_4^{(m_4)}}_{M(k+1,j)=1}$$
$$\underbrace{i \qquad k}_{M(i,k)=1} \underbrace{k+1 \qquad j}_{M(k+1,j)=1}$$

Then the substrings $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ are shorter than $W^x_{[i,j]}$, therefore, by the induction hypothesis, both $M(i,k)$ and $M(k+1,j)$ are equal to 1 according to the definition of M. $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ can be of the form **(M1)** or **(M3)**. There are the following cases.

- If both substrings $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ are of the form **(M1)**, then $W^x_{[i,j]}$ also consists of simple terms and reduces to 1, as both $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ reduce to 1. $W^x_{[i,j]}$ is therefore of the form **(M1)**. Hence, $M(i,j) = 1$ in accordance with the definition of M.
- $W^x_{[i,k]}$ is of the form **(M1)**. Therefore, it consists of simple terms and reduces to 1 and $W^x_{[k+1,j]}$ is of the form **(M3)**. Then $W^x_{[i,j]}$ is also of the form **(M3)**. Hence, $M(i,j) = 1$ according to the definition of M.
- $W^x_{[i,k]}$ is of the form **(M3)** and $W^x_{[k+1,j]}$ is of the form **(M1)**. So, it consists of simple terms and reduces to 1. Then $W^x_{[i,j]}$ is also of the form **(M3)**. Hence, $M(i,j) = 1$ according to the definition of M.
- $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ are of the form **(M3)**. Hence, the whole string $W^x_{[i,j]}$ is also of the form **(M3)**. Then $M(i,j) = 1$ in accordance with the definition of M.

A1b. $W^x_i, W^x_j \in T$ and there exists k such that $i < k < j-1$ and W^x_k is a right angle bracket, $W^x_{(k+1)}$ is a left angle bracket, $M(i,k) = 1$ and $M(k+1,j) = 1$.

$$W^x_{[i,j]} = \underbrace{p_1^{(m_1)} \ldots \rangle}_{M(i,k)=1} \underbrace{\langle \ldots \ p_2^{(m_2)}}_{M(k+1,j)=1}$$
$$\underbrace{i \qquad k}_{M(i,k)=1} \underbrace{k+1 \quad j}_{M(k+1,j)=1}$$

Then the substrings $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ are shorter than $W^x_{[i,j]}$, therefore, by the induction hypothesis, both $M(i,k)$ and $M(k+1,j)$ are equal to 1 according to the definition of M. $W^x_{[i,k]}$ is of the form **(M2b)**, so there is a string $YX_1...X_{g_1}$ that reduces to 1 (such that the first simple term of Y is W^x_i, $X_h \in T^+$ is a substring in between the h-th pair of matching angle brackets in the string $W^x_{[i,k]}$). The substring $W^x_{[k+1,j]}$ is of the form **(M2a)**, so there is a string $X'_1...X'_{g_2}Z'$ that reduces to 1 (such that the last simple term of Z' is W^x_j, $X'_h \in T^+$ is a substring in between the h-th pair of matching angle brackets in the string $W^x_{[k+1,j]}$). Hence, the whole string $W^x_{[i,j]}$ is of the form **(M3)** (with the string $YX_1...X_{g_1}X'_1...X'_{g_2}Z'$ reducing to 1).

A2. $W^x_i = p^{(m)}$, $W^x_j = q^{(m+1)}$, the condition (GC) holds and $M(i+1,j-1) = 1$. $M(i+1,j-1) = 1$ is computed according to the definition of M by the induction hypothesis, as $W^x_{[i+1,j-1]}$ is shorter than $W^x_{[i,j]}$.

$$W^x_{[i,j]} = \underset{i}{p^{(m)}} \underbrace{\underset{i+1}{\_} \ldots \underset{j-1}{\_}}_{M(i+1,j-1)=1} \underset{j}{q^{(m+1)}}$$, (GC) holds

Then the substring $W^x_{[i+1,j-1]}$ can be:

- of the form **(M1)**, then the whole string $W^x_{[i,j]}$ consists of simple terms and reduces to 1, so it is also of the form **(M1)**. Hence, $M(i,j) = 1$ according to the definition of M.
- of the form **(M3)**, that is $Y * \mathbf{U}\rangle ... \langle \mathbf{V} * Z$, where $\mathbf{U}$ and $\mathbf{V}$ contain no angle brackets and the string $YX_1...X_gZ$ reduces to 1. We can assume $Y' = p^{(m)}Y$ and $Z' = Zq^{(m+1)}$. Then $W^x_{[i,j]} = Y' * U\rangle ... \langle V * Z'$ and the string $Y'X_1...X_gZ'$ reduces to 1. Hence, $M(i,j) = 1$ according to the definition of M.

We should notice that the string $W^x_{[i+1,j-1]}$ cannot be of the form **(M2a)**, **(M2b)** or **(M4)**, since a simple term can be followed (preceded) only by another simple term or an asterisk.

A3a. $W^x_{[i,j]}$ is of the form $\langle ... \rangle ... \langle ... p^{(m)}$, $p \in P, m \in \mathcal{Z}$ and there exists k such that $i < k < j$ and $W^x_k = *$, the string $W^x_{[i+1,k]}$ contains no angle brackets and $M(k+1,j) = 1$.

$$W^x_{[i,j]} = \underset{i}{\langle} \underbrace{\underset{i+1}{*} \ldots \underset{k}{*}}_{no\ \langle,\rangle} \underset{\_}{\ldots\ \rangle} \ldots \langle \ldots \underbrace{\underset{k+1}{p^{(m)}} \ldots \underset{j}{}}_{M(k+1,j)=1}$$

$W^x_{[k+1,j]}$ is shorter than $W^x_{[i,j]}$ so $M(k+1,j) = 1$ is computed according to the definition of M by the induction hypothesis. The string $W^x_{[k+1,j]}$ ends with a simple term, so it can be of the form **(M1)**, **(M2a)** or **(M3)**. But it cannot begin with a left angle bracket and it contains angle brackets, so it must be of the form **(M3)**. Hence, there is a string $YX_1...X_gZ$ reducing to 1. So, $W^x_{[i,j]}$ is of the form **(M2a)** with the string $YX_1...X_gZ$ reducing to 1. Therefore, $M(i,j) = 1$ according to the definition of M.

A3b. $W^x_{[i,j]}$ is of the form $p^{(m)}...\rangle ... \langle ...\rangle$, where $p \in P, m \in \mathcal{Z}$ and there exists k such that $i < k < j$, $W^x_k = *, W^x_{[k,j-1]}$ contains no angle brackets and $M(i,k-1) = 1$.

$$W^x_{[i,j]} = \underbrace{\underset{i}{p^{(m)}} \ldots \rangle \ldots \langle \ \ldots \underset{k-1}{\_}}_{M(i,k-1)=1} \underbrace{\underset{k}{*} \ldots \underset{j-1}{*}}_{no\ \langle,\rangle} \underset{j}{\rangle}$$

The string $W^x_{[i,k-1]}$ is shorter than $W^x_{[i,j]}$, so, by the induction hypothesis, $M(i,k-1) = 1$ is computed according to the definition of M. The string $W^x_{[i,k-1]}$ begins with a simple term, so it can be of the form **(M1)**, **(M2b)** or **(M3)**. But it cannot end with a right angle bracket and it contains angle

brackets, so it must be of the form **(M3)**. Hence, there is a string $YX_1...X_gZ$ reducing to 1. So, $W^x_{[i,j]}$ is of the form **(M2b)** with the string $YX_1...X_gZ$ reducing to 1. Therefore, $M(i,j) = 1$ according to the definition of M.

A4a. $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle...\langle...q^{(m+1)}$, the condition (GC) holds and $M(k, j-1) = 1$, where k is the position of the first left angle bracket in the string $W^x_{[i,j]}$.

$$\begin{array}{l} W^x_{[i,j]} = p^{(m)} * ...*\rangle\langle...\rangle\langle...r^{(n)}q^{(m+1)}, \text{ (GC) holds} \\ \qquad\quad i \quad \underbrace{\qquad\qquad}_{no\ \langle,\rangle} \ \underbrace{k \qquad\quad j-1}_{M(k,j-1)=1} \ j \end{array}$$

$W^x_{[k,j-1]}$ is shorter than $W^x_{[i,j]}$. Hence, by the induction hypothesis, $M(k, j-1) = 1$ according to the definition of M. The string $W^x_{[k,j-1]}$ must be therefore of the form **(M2a)**, since it is the only form starting with an angle bracket and ending with an element different from a bracket. Hence, $W^x_{[k,j-1]} = \langle...\rangle...\langle \mathbf{V} * Z$, where Z is the string of simple terms, $\mathbf{V}$ contains no angle brackets and there are precisely g pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1...X_gZ$ reduces to 1. Let $Z' = Zq^{(m+1)}$, $Y = p^{(m)}$. Then $YX_1...X_gZ'$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M3)**. Then $M(i,j) = 1$ in accordance with the definition of M.

A4b. $W^x_{[i,j]}$ is of the form $p^{(m)}...\rangle...\langle... * q^{(m+1)}$, the condition (GC) holds, and $M(i+1, k) = 1$, where k is the position of the last right angle bracket in the string $W^x_{[i,j]}$.

$$\begin{array}{l} W^x_{[i,j]} = p^{(m)} \ r^{(n)} \ ...\rangle\langle...\rangle\langle*... * q^{(m+1)}, \text{ (GC) holds} \\ \qquad\quad i \ \underbrace{i+1 \qquad\quad k}_{M(i+1,k)=1} \ \underbrace{\qquad\quad j}_{no\ \langle,\rangle} \end{array}$$

$W^x_{[i+1,k]}$ is shorter than $W^x_{[i,j]}$. Hence, by the induction hypothesis, $M(i+1, k) = 1$ according to the definition of M. The string $W^x_{[i+1,k]}$ must be therefore of the form **(M2b)**, since it is the only form ending with an angle bracket but starting with an element different from a bracket. Hence, $W^x_{[i+1,k]} = Y * \mathbf{U}\rangle...\langle...\rangle$, where Y is the string of simple terms, $\mathbf{U}$ contains no angle brackets and there are precisely g pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $YX_1...X_g$ reduces to 1. Let $Y' = p^{(m)}Y$, $Z = q^{(m+1)}$. Then $Y'X_1...X_gZ$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M3)**. Then $M(i,j) = 1$ in accordance with the definition of M.

A4c. $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle...\langle... * q^{(m+1)}$, where the string "..." in between the angle brackets is not empty and the condition (GC) holds. $M(k, k') = 1$, where k is the position of the first left angle bracket in the string $W^x_{[i,j]}$ and k' the position of the last right angle bracket in the string $W^x_{[i,j]}$.

$$W^x_{[i,j]} = \underbrace{p^{(m)} * \ ...}_{i,\ no\ \langle,\rangle}\rangle\langle\ \underbrace{.\ .\ .}_{k,\ k',\ M(k,k')=1}\ \rangle\langle ... * \underbrace{q^{(m+1)}}_{j,\ no\ \langle,\rangle}, \text{ (GC) holds}$$

$W^x_{[k,k']}$ is shorter than $W^x_{[i,j]}$. Hence, by the induction hypothesis, $M(k,k')=1$ according to the definition of M. The string $W^x_{[k,k']}$ must be therefore of the form **(M4)**. Hence $W^x_{[k,k']} = \langle ... \rangle ... \langle ... \rangle$, where there are precisely g pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1 ... X_g$ reduces to 1. Let $Y = p^{(m)}$, $Z = q^{(m+1)}$, $\mathbf{U} = W^x_{[i+2,k-1]}$ (if $k > i+3$ else $\mathbf{U} = 1$) and $\mathbf{V} = W^x_{[k'+1,j-2]}$ (if $k' < j-3$ else $\mathbf{V} = 1$). Then $YX_1 ... X_g Z$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M3)**. Then $M(i,j) = 1$ according to the definition of M.

A5. $W^x_{[i,j]}$ is of the form $\langle ... \rangle ... \langle ... \rangle$, and $M(k,k') = 1$, where W^x_k is a simple term in between the first pair of angle brackets and $W^x_{k'}$ is a simple term in between the last pair of angle brackets in the string $W^x_{[i,j]}$ and $W^x_{k-1} = *$ and $W^x_{k'+1} = *$.

$$W^x_{[i,j]} = \underset{i}{\langle} \underbrace{* \ . \ . \ . \ *}_{i+1,\ k-1,\ no\ \langle,\rangle} \underbrace{p_1^{(m_1)} ... \rangle ... \langle ... p_2^{(m_2)}}_{k,\ k',\ M(k,k')=1} \underbrace{* \ . \ . \ *}_{k'+1,\ j-1,\ no\ \langle,\rangle} \underset{j}{\rangle}$$

$W^x_{[k,k']}$ is shorter than $W^x_{[i,j]}$. Hence $M(k,k') = 1$ according to the definition of M. $W^x_{[k,k']}$ must be then of the form **(M3)**, so the whole string $W^x_{[i,j]}$ is of the form **(M4)** and therefore, $M(i,j) = 1$ in accordance with the definition of M.

We will prove that the algorithm finds correctly all substrings for which the function $M(i,j) = 1$ by induction on the length of the substring. Let us notice that there are no such substrings that contain asterisk but no angle brackets.

The only strings of length two, for which $M(i,j) = 1$, are of the form $p^{(m)}q^{(m+1)}$, where the condition (GC) holds (that is of the form **(M1)**). The algorithm finds them correctly.

Now let us consider the substrings of the length $l > 2$ such that for all $l' < l$ the algorithm finds the substrings of the length l' correctly (all forms of the definition of M).

1. $W^x_{[i,j]}$ is of the form **(M1)**, that is $W^x_{[i,j]} \in T^+$ and it reduces to 1. There are two possible cases:

- $W^x_i W^x_j$ does not reduce to 1 in the whole reduction of the string $W^x_{[i,j]}$ to 1. Then $W^x_{[i,j]}$ can be divided into two substrings. Each of them reduces to 1, is shorter than $W^x_{[i,j]}$, so they are found by the algorithm correctly (by the induction hypothesis). Hence, by (**A1a**), $M(i,j) = 1$.

- $W_i^x W_j^x$ reduces to 1 in the whole reduction of $W_{[i,j]}^x$ to 1. Then $W_{[i+1,j-1]}^x$ is a shorter substring reducing to 1. By the induction hypothesis, the string is found by the algorithm correctly, so $M(i,j) = 1$ (case (**A2**)).

2a. $W_{[i,j]}^x$ is of the form **(M2a)**. If $M(i,j) = 1$, then there exists a substring of the form $*X_1*$ in between the first pair of angle brackets such that X_1 takes part in some reduction to 1 (if $g > 0$). Let i' be the position of the first simple term in X_1. The substring $W_{[i',j]}^x$ is of the form **(M3)**, it is shorter than $W_{[i,j]}^x$ and $M(i',j) = 1$. By the induction hypothesis, the string is found by the algorithm correctly, so $M(i,j) = 1$ (case (**A3a**)). If $g = 0$, then Y reduces to 1 and it is of the form **(M1)** and shorter than $W_{[i,j]}^x$. So, by the induction hypothesis, it is found by the algorithm correctly. Then $M(i,j) = 1$ by (**A3a**).

2b. $W_{[i,j]}^x$ is of the form **(M2b)**. If $M(i,j) = 1$, then there exists the substring of the form $*X_g*$ (if $g > 0$) in between the last pair of angle brackets such that X_g takes part in some reduction to 1. Let j' be the position of the last simple term in X_g. The substring $W_{[i,j']}^x$ is of the form **(M3)**, it is shorter than $W_{[i,j]}^x$ and $M(i,j') = 1$. By the induction hypothesis, the string is found by the algorithm correctly, so $M(i,j) = 1$ (case (**A3b**)). If $g = 0$ then Y reduces to 1 and it is of the form **(M1)** and shorter than $W_{[i,j]}^x$. So, by the induction hypothesis, it is found by the algorithm correctly. Then $M(i,j) = 1$ by (**A3b**).

3. $W_{[i,j]}^x$ is of the form **(M3)**. $W_i^x W_j^x$ takes part in the reduction to 1 in $W_{[i,j]}^x$. Let us assume $W_i^x W_{i'}^x$ reduces to 1 where $i' \neq j$. Then $W_{i'+1}^x$ can be:

- simple term. Then $W_{[i,i']}^x$ and $W_{[i'+1,j]}^x$ are of the form **(M1)** or **(M3)**, they are shorter and both $M(i,i') = 1$ and $M(i'+1,j) = 1$. Hence, by the induction hypothesis, these strings are found by the algorithm correctly, so $M(i,j) = 1$ (case (**A1a**)).
- asterisk. Then $W_{[i,i']}^x$ is of the form **(M1)** or **(M3)**, it is shorter and $M(i,i') = 1$. Hence, by the induction hypothesis, the string is found by the algorithm correctly. Let k be the position of the first right angle bracket following i'. $W_{[i,k]}^x$ is shorter than $W_{[i,j]}^x$ and it is of the form **(M2b)**. So, by the induction hypothesis, $M(i,k) = 1$ (case (**A3b**)). Similarly, the substring $W_{[k+1,j]}^x$ is of the form **(M2a)** is shorter than $W_{[i,j]}^x$. So, by the induction hypothesis, $M(k+1,j) = 1$ (case (**A3a**)). Hence, by the induction hypothesis, $M(i,j) = 1$ (case (**A1b**)).

Let us assume $i' = j$ so $W_i^x W_j^x$ reduces to 1. There are the following cases:

- W_{i+1}^x, W_{j-1}^x are simple terms. Then the substring $W_{[i+1,j-1]}^x$ is of the form **(M3)**, it is shorter than $W_{[i,j]}^x$, hence $M(i+1,j-1) = 1$. By the induction hypothesis, that string is found by the algorithm correctly and thus $M(i,j) = 1$ (case (**A2**)).

- $W^x_{i+1} = *, W^x_{j-1} \in T$. So, $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle ...\langle ...q^{(m+1)}$. Let j' be the position of the first left angle bracket in $W^x_{[i,j]}$. We know there exists $j' < k < j-1$ such that $W^x_k W^x_{j-1}$ reduces to 1. Hence, $W^x_{[j',j-1]}$ is of the form **(M2a)**, it is shorter than $W^x_{[i,j]}$, so $M(j', j-1) = 1$. By the induction hypothesis, that string is found by the algorithm correctly and thus $M(i,j) = 1$ (case (**A4a**)).
- $W^x_{i+1} \in T$, $W^x_{j-1} = *$. So, $W^x_{[i,j]}$ is of the form $p^{(m)} ...\rangle ...\langle ... * q^{(m+1)}$. Let i' be the position of the last right angle in $W^x_{[i,j]}$. We know there exists $i+1 < k < i'$ such that $W^x_{i+1} W^x_k$ reduces to 1. Hence, $W^x_{[i+1,i']}$ is of the form **(M2b)**, it is shorter than $W^x_{[i,j]}$, so $M(i+1, i') = 1$. By the induction hypothesis, that string is found by the algorithm correctly and thus $M(i,j) = 1$ (case (**A4b**)).
- $W^x_{i+1} = *, W^x_{j-1} = *$. Then the string $W^x_{[i,j]}$ can be of the form $p^{(m)} * \mathbf{U}\rangle\langle \mathbf{V} * q^{(m+1)}$ and then $M(i,j) = 1$,by the initial case of the description of the algorithm. If $W^x_{[i,j]}$ contains more brackets, then there is a string $W^x_{[k,k']}$, where k is the index of the first left angle bracket in the string $W^x_{[i,j]}$ and k' is the index of the last right angle bracket in the string $W^x_{[i,j]}$. $W^x_{[k,k']}$ is of the form **(M4)**, it is shorter than $W^x_{[i,j]}$, therefore, by the induction hypothesis $M(k,k') = 1$. So,, $M(i,j) = 1$ by (**A4c**).

4. $W^x_{[i,j]}$ is of the form **(M4)**. Then let k be the index of the first simple term that participates in the reduction (that is the first simple term in the substring X_1, so it is obviously in between the first pair of matching angle brackets) and k' be the index of the last simple term that participates in the reduction (that is the last simple term in the substring X_g, so it is obviously in between the last pair of matching angle brackets). The substring $W^x_{[k,k']}$ is of the form **(M3)**, it is shorter than $W^x_{[i,j]}$, so, by induction hypothesis, $M(k,k') = 1$. Therefore, $M(i,j) = 1$ by (**A5**).

□

The algorithm is polynomial. One can observe that all conditions for $W^x_{[i,j]}$ can be checked in $O(j-i)$ steps. The number of the substrings is equal to $O(|W^x|^2)$. Therefore, one can compute the function $M(1, |W^x|)$ in time $O(k \cdot |W^x|^3)$, where k is the size of the grammar.

4.3.3 Obtaining the reduction

The algorithm can be easily modified to become a parsing algorithm. Each obtained reduction is described by the set of links that take part in it. If we want to obtain only one reduction, the complexity of the algorithm does not increase. The set of links $L(i,j)$ represents a reduction of some term to 1. Links are denoted by pairs of integers (k,l) such that $i \leqslant k < l \leqslant j$. We find the set of links by backtracking the indices of the function $M(i,j) = 1$, obviously starting with $M(1, |W^x|)$. We

also define an auxiliary function $Prev(i,j)$ to help us follow the backtracking (as the value of the function $M(i,j)$ does not tell how it was obtained). The value of the function $Prev(i,j)$ is a sequence of three pairs $((l_1,l_2),(m1_1,m1_2),(m2_1,m2_2))$, where l_1,l_2 are indices of the link, $m1_1,m1_2,m2_1,m2_2$ are indices of function M on which the computation $M(i,j)=1$ is based. If any of the values is not used, it is set to 0. Every time when the algorithm computes the value of the function $M(i,j)=1$ we set the value of the function $Prev(i,j)$ in the following way:

- for any initial case, $Prev(i,j)=((i,j),(0,0),(0,0))$,
- if it is a computation by one of the cases: (**A2**), (**A4a**), (**A4b**), (**A4c**), then $Prev(i,j)=((i,j),(k,l),(0,0))$, where (k,l) is the pair of indices for which the value the function M was 1 in the current computation (that is e.g. in (**A2**) a pair $(k,l)=(i+1,j-1)$),
- if it is a computation by one of the cases: (**A3a**), (**A3b**), (**A5**), then $Prev(i,j)=((0,0),(k,l),(0,0))$ where (k,l) is the pair of indices for which the value the function M was 1 in the current computation,
- if it is a computation by one of the cases: (**A1a**), (**A1b**), then $Prev(i,j)=((0,0),(i,k),(k+1,j))$.

Obviously, one can choose whether the algorithm should remember the first computed reduction or the last computed reduction. In the first case if $Prev(i,j)\neq((0,0),(0,0),(0,0))$, then it cannot modified. In the latter, $Prev(i,j)$ is updated every time when the algorithms computes $M(i,j)=1$. When the computation of the functions M and $Prev$ is finished, we easily compute the set $L(1,|W^x|)$. The definition of the function $L(i,j)$ is as follows:

- if $Prev(i,j)=((i,j),(0,0),(0,0))$, where $0<i<j$, then $L(i,j)=\{(i,j)\}$,
- if $Prev(i,j)=((i,j),(k,l),(0,0))$, where $0<i\leqslant k<l\leqslant j$, then $L(i,j)=L(k,l)\cup\{(i,j)\}$,
- if $Prev(i,j)=((0,0),(k,l),(0,0))$, where $0<i\leqslant k<l\leqslant j$, then $L(i,j)=L(k,l)$,
- if $Prev(i,j)=((0,0),(i,k),(k+1,j))$, where $0<i<k<j$, then $L(i,j)=L(i,k)\cup L(k+1,j)$.

4.3.4 Examples

We run our algorithm on some linguistic examples from [Lam08], [Oeh04].

Example 4.3. Let us take a sample dictionary:

I: π_1
will: $\pi^r s_1 j^l,\ q_1 j^l \pi^l$
meet: io^l
him: o

with

$$\pi_1 \leqslant \pi, \quad i \leqslant j, \quad s_1 \leqslant s.$$

Let us consider the sentence:

I will meet him.

We construct a string W_x (in the illustration below we put only these numbers of these elements of W^x that can be starting points or endpoints for some $M(i,j) = 1$):
$\langle *\pi_1^{(0)} * \rangle \langle *\pi^{(1)} s_1^{(0)} j^{(-1)} * q_1^{(0)} j^{(-1)} \pi^{(-1)} * \rangle \ \langle *i^{(0)} \ o^{(-1)} * \rangle \ \langle \ * \ o^{(0)} * \rangle \ \langle *s^{(1)}$
1 3 5 6 8 9 10 12 13 14 16 17 19 20 22 23 25 27 28 30

We can compute the function $M(i,j)$. There are no substrings of the length 2 such that $M(i,j) = 1$. We start with the following $M(i,j)$:

1. $M(3,8) = 1$, $Prev(3,8) = ((3,8),(0,0),(0,0))$
2. $M(10,19) = 1$, $Prev(10,19) = ((10,19),(0,0),(0,0))$
3. $M(20,25) = 1$, $Prev(20,25) = ((20,25),(0,0),(0,0))$.

Then we compute:

4. $M(10,25) = 1$ by the lines 2,3 and the case (**A1a**),
$Prev(10,25) = ((0,0),(10,19),(20,25))$
5. $M(10,27) = 1$ by the line 4 and the case (**A3b**),
$Prev(10,27) = ((0,0),(10,25),(0,0))$
6. $M(9,30) = 1$ by the line 5 and the case (**A4b**),
$Prev(9,30) = ((9,30),(10,27),(0,0))$
7. $M(3,30) = 1$ by the lines 1,6 and the case (**A1a**),
$Prev(3,30) = ((0,0),(3,8),(9,30))$
8. $M(1,30) = 1$ by the line 7 and the case (**A3a**),
$Prev(1,30) = ((0,0),(3,30),(0,0))$.

Therefore, there exists a type assignment for a given string of words which reduces to the sentence type s.

The output of the parsing algorithm is:
{(3,8), (9,30), (10,19) (20,25)}.

Example 4.4. The dictionary:

did: $q_2 i^l \pi^l, \ \pi^r s_2 i^l$
he: π_3
give: $io^l, i\overline{n}^l o'^l$
books: n
to: $i^r i o^l, \overline{j} i^l$
her: o

with

$\pi_j \leqslant \pi,\ s_i \leqslant s,\ q_k \leqslant q \leqslant s,\ \overline{j} \leqslant \pi_3,\ o' \leqslant o,\ n \leqslant \overline{n} \leqslant o.$

We consider the sentence:

Did he give books to her?

We construct a string W_x:

$\langle *\pi^r s_2 i^l * q_2 i^l \pi^l * \rangle\ \langle *\pi_3 * \rangle\ \langle *\ io^l * i\overline{n}^l o'^l * \rangle\ \langle *n* \rangle\ \langle *i^r i o^l * \overline{j} i^l * \rangle\ \langle *o* \rangle\ \langle *s^r$

1 7 8 9 14 16 19 20 29 34 36 44 49

We can compute the function $M(i,j)$. There are no substrings of the length 2 such that $M(i,j) = 1$. The algorithm computes $M(i,j) = 1$:

1. $M(9,14) = 1$ by initial case, $Prev(9,14) = ((9,14),(0,0),(0,0))$
2. $M(9,16) = 1$ by line 1 and (**A3b**),
$Prev(9,16) = ((0,0),(9,14),(0,0))$
3. $M(8,19) = 1$ by line 2 and (**A4b**) - useless,
$Prev(8,19) = ((8,19),(9,16),(0,0))$
4. $M(8,22) = 1$ by line 2 and (**A4b**) - useless,
$Prev(8,22) = ((8,22),(9,16),(0,0))$
5. $M(20,29) = 1$ by initial case, $Prev(20,29) = ((20,29),(0,0),(0,0))$
6. $M(8,29) = 1$ by lines 3,5 and (**A1a**) - useless,
$Prev(8,29) = ((0,0),(8,19),(20,29))$
7. $M(20,31) = 1$ by line 5 and (**A3b**),
$Prev(20,31) = ((0,0),(20,29),(0,0))$
8. $M(8,31) = 1$ by line 6 and (**A3b**) - useless,
$Prev(8,31) = ((0,0),(8,29),(0,0))$
9. $M(19,34) = 1$ by line 7 and (**A4b**),
$Prev(19,34) = ((19,34),(20,31),(0,0))$
10. $M(17,34) = 1$ by line 9 and (**A3a**),
$Prev(17,34) = ((0,0),(19,34),(0,0))$
11. $M(9,34) = 1$ by lines 2,10 and (**A1b**),
$Prev(9,34) = ((0,0),(9,16),(17,34))$
12. $M(8,35) = 1$ by line 11 and (**A2**),
$Prev(8,35) = ((8,35),(9,34),(0,0))$
13. $M(36,44) = 1$ by initial case, $Prev(36,44) = ((36,44),(0,0),(0,0))$
14. $M(8,44) = 1$ by lines 12,13 and (**A1a**),
$Prev(8,44) = ((0,0),(8,35),(36,44))$
15. $M(36,46) = 1$ by line 13 and (**A3b**) - useless,
$Prev(36,46) = ((0,0),(36,44),(0,0))$
16. $M(8,46) = 1$ by line 14 and (**A3b**),
$Prev(8,46) = ((0,0),(8,44),(0,0))$
17. $M(7,49) = 1$ by line 16 and (**A4b**),
$Prev(7,49) = ((7,49),(8,46),(0,0))$
18. $M(1,49) = 1$ by line 17 and (**A3a**),
$Prev(1,19) = ((0,0),(7,49),(0,0))$.

Therefore there exists a type assignment for a given string which reduces to s. The output of the parsing algorithm is:
{(7,49), (8,35), (9,14), (19,34), (20,29), (36,44)}

Example 4.5. An example from [Oeh04].

The dictionary:

Kim: np
mailed: $np^r\ s\ pp^l\ np^l, np^r\ s\ np^l$
the: $np\ n^l$
letter: $n\ pp^l,\ n$
to: $pp\ np^l$
Sandy: np

We consider the sentence:

Kim mailed the letter to Sandy.

We construct a string W_x:

$\langle * \ np\ * \rangle \langle * np^r\ s\ pp^l\ np^l * np^r\ s\ np^l * \rangle \langle * np\ n^l * \rangle \langle * n\ pp^l * n * \rangle \langle * pp\ np^l * \rangle \langle * \ np\ * \rangle \langle * s^r$

1 3 8 9 10 11 17 20 21 29 32 34 35 40 45

1. $M(3,8) = 1$ by initial case -(r1)
2. $M(1,8) = 1$ by line 1 and (**A3a**) - useless
3. $M(3,13) = 1$ by initial case -(r2)
4. $M(1,13) = 1$ by line 3 and (**A3a**) - useless
5. $M(15,20) = 1$ by initial case - (r2)
6. $M(11,20) = 1$ by initial case - (r1)
7. $M(21,26) = 1$ by initial case - (r2)
8. $M(15,26) = 1$ by lines 5,7 and (**A1a**) - (r2')
9. $M(11,26) = 1$ by line 6,7 and (**A1a**) - useless
10. $M(21,29) = 1$ by initial case - (r1)
11. $M(15,29) = 1$ by lines 5,10 and (**A1a**)
12. $M(11,29) = 1$ by lines 6,10 and (**A1a**)- (r1)
13. $M(21,31) = 1$ by line 10 and (**A3b**) - useless
14. $M(15,31) = 1$ by line 11 and (**A3b**) - useless
15. $M(11,31) = 1$ by line 12 and (**A3b**) - (r1)
16. $M(27,34) = 1$ by initial case - (r2)
17. $M(21,34) = 1$ by lines 7,16 and (**A1a**) - (r2")
18. $M(15,34) = 1$ by lines 5,17 and (**A1a**) or by line 8,16 and (**A1a**) - both leading to (r2)
19. $M(11,34) = 1$ by lines 6,17 and (**A1a**) - useless
20. $M(10,34) = 1$ by line 15 and (**A4b**) - (r1)
21. $M(35,40) = 1$ by initial case - (r1) and (r2)
22. $M(27,40) = 1$ by lines 16,21 and (**A1a**) - (r2)

23. $M(21,40) = 1$ by lines 7,22 and (**A1a**) - (r2) or by lines 17,20 and (**A1a**)
24. $M(15,40) = 1$ by lines 5,23 and (**A1a**) - (r2) or by lines 8,22 - (r2') and (**A1a**) or by lines 18,21 and (**A1a**) - (r2")
25. $M(11,40) = 1$ by lines 6,23 and (**A1a**) - useless
26. $M(10,40) = 1$ by lines 20,21 and (**A1a**) - (r1)
27. $M(35,42) = 1$ by line 21 and (**A3b**) - useless
28. $M(27,42) = 1$ by line 22 and (**A3b**) - useless
29. $M(21,42) = 1$ by line 23 and (**A3b**) - useless
30. $M(15,42) = 1$ by line 24 and (**A3b**) - (r2)
31. $M(11,42) = 1$ by line 25 and (**A3b**) - useless
32. $M(10,42) = 1$ by line 26 and (**A3b**) - (r1)
33. $M(14,45) = 1$ by line 30 and (**A4b**) - (r2)
34. $M(9,45) = 1$ by line 32 and (**A4b**) - (r1)
35. $M(3,45) = 1$ by lines 1,34 and (**A1a**) - (r1) or lines 3,33 and (**A1a**) - (r2)
36. $M(1,45) = 1$ by line 35 and (**A3a**) - (r1) and (r2).

There are 2 possible reductions (the algorithm at each run can find only one of them):
the first computed reduction: {(3,8), (9,45), (10,34), (11,20), (21,29), (35,40)} and
the last computed reduction: {(3,13), (14,45), (15,20), (21,26), (27,34), (35,40)}.

4.3.5 Conclusion

We have presented a new dynamic parsing algorithm for pregroup grammars. The algorithm works in polynomial time. We have given some examples of the run of the algorithm. The algorithm can be easily modified for other similar grammars. For example, pregroup grammars with letter promotions, see **Section 5.3**.

The algorithm is based on the algorithm of Savateev described in **Section 4.2**. However, several details are essentially different. For instance, Savateev's algorithm and the proof of its correctness rely upon the fact that the right-most atom in $\gamma(A)$ has the form p^1, and other peculiarities of the translation. The algorithm of Savateev can be applied only to strings of atoms of the form described above (only with positive exponents) whereas our algorithm is more general. It handles arbitrary strings of terms, admits both positive and negative integers n (i.e. both right and left adjoints), and needs no constraints on structure of types used.

4.4 Other parsing algorithms for pregroup grammars

Parsing pregroup grammars has been a matter of interest in recent years. Oehrle [Oeh04] proposed a parsing algorithm, based on lexical graphs. We describe the algorithm of Oehrle below. Some other ideas on parsing pregroup grammars were given by Béchet in [Béc07]. His concept of partial composition and functional composition is presented in **Section 3.2**. Béchet proposes an algorithm, which is a

modification of **CYK** algorithm with some restrictions, allowing to use functional composition. The drawback of his approach is that functional composition yields a **CFG** whose size is exponential in the size of the given **PG**; hence his algorithm is really exponential (see **Section 3.2**). Degeilh and Preller [DP05] gave a cubic algorithm for a pregroup grammar recognition problem. Preller [Pre07] proposed also a linear parsing algorithm for some restricted pregroup grammars. We describe it at the end of this section.

Oehrle's algorithm is essentially different. The original algorithm is a recognition algorithm. Oehrle's algorithm is based on so-called *analytic lexical graphs*. The main idea is that at first a separate directed linear graph is built for each word of the string of words $x = a_1 \dots a_n$, then they are joined together and finally new arcs corresponding to possible contractions are added. The procedure in more details follows.

Each type A assigned to a word a_i by G is represented as a linear graph of simple terms. The source node and target node are 1. One draws arcs from the source to the first simple terms of these types and from the last simple terms of these types to the target. Obviously simple terms of each type are also connected by arcs. Then one identifies the target of the graph of a_i with the source of the graph of a_{i+1}.

The algorithm modifies the graph obtained for a given string. It adds new arcs to the resulting graphs. The added arcs denote reduction steps. They connect either simple terms separated by 1 (for any path $t_1 \to 1 \to t_2$, one adds $t_1 \to t_2$), or separated by two simple terms that can be contracted (for any path $t_1 \to p^{(n)} \to p^{(n+1)} \to t_2$, one adds $t_1 \to t_2$). We should notice that the algorithm does not regard poset rules, therefore the only contractions applied are of form $p^{(n)}p^{(n+1)} \Rightarrow 1$. The procedure of adding new arcs is executed recursively from left to right. Finally, a set of *winners* is returned. A *winner* is a simple term t for which there exists a path $1 \to t \to 1$ from the source of a_1 to the target of a_n in the resulting graph.

The run of Oehrle's algorithm on the last example is presented on pp. 67-71 of [Oeh04], so it takes much more space than our presentation (see **Example 4.5**).

Oehrle [Oeh04] shows how to change the algorithm to become a parsing one. The algorithm remains cubic, but uses even more memory. All reductions performed have to be remembered. Each node is represented then not only by its name but by a pair consisting of a node name and a list of reduction steps performed so far.

Oehrle [Oeh04] shows that the time is $O(n^3)$, but the size of G is treated as a constant. It can be shown that the time is cubic also in our sense, that is depending on the string $|W^x|$. However, execution of the algorithm seems more complicated than ours. At each step all predecessors and successors of every vertex have to be remembered and some of them updated. Moreover, our algorithm can easily be modified to admit an arbitrary type Y in the place of s to search for phrases of different types. In our algorithm it suffices to replace s^r by Y^r. This generalization is not directly applicable to Oehrle's algorithm. To conclude our approach is less memory-consuming and slightly more general.

Another interesting algorithm for pregroup grammars was proposed by Preller and Degeilh [DP05]. They present a cubic algorithm to solve a recognition problem

for pregroup grammars. As in our algorithm the constant n^3 depends on a bound for the number of types per word and a bound for their length. The algorithm processes a string of words $a_1, \ldots, a_n$ from left to right in different stages. Each stage is described by three features:

- i - an index of a processed word,
- a type $t \in I(a_i)$
- a position p of the simple term in t.

All types assigned to a given word a_i are examined in different stages. Therefore, simple terms of different types assigned to the same words are never compared. Simple terms occurring before the position p in different type assignments (for words $a_1, ..., a_{i-1}$) are examined. The simple term considered is either contracted with the denoted *nearest left parenthesis* that is the last simple term that can be contracted, or remembered as a new *nearest left parenthesis* awaiting for another type to contract. If a contraction is performed a simple term preceding the former *nearest left parenthesis* becomes now the new *nearest left parenthesis*. The algorithm does not remember the contractions already performed. Obviously one can remember the history of contractions, but then the complexity increases.

The approach of [Pre07] to linear parsing of pregroup grammars is called *Lazy Parsing*. It is a modification of the algorithm described above. A given string of words is also processed from left to right in stages. As before searching for a reduction is combined with choosing appropriate type assignment. At each stage there are two possibilities: a type for a considered word is chosen or the assigned type is processed by reading its simple terms from left to right. A stage is defined by a triple consisting of:

- an index i of the word examined,
- a string of types assigned to the first i words
- a position p in the i-th type of the simple term processed.

Lazy Parsing means that if, at a given stage, a contraction is possible, then it is performed. Consequently, the algorithm in general is correct, but not complete. Indeed, if it finds a reduction to the sentence type, then a given string is a correct grammatical sentence, but if the algorithm does not find any reduction to the sentence type, then nothing can be said about grammatical correctness of the string. The algorithm becomes complete if some restrictions are imposed on the lexicon. Preller [Pre07] notices that one can restrict dictionary of a pregroup grammar so that when parsing such a grammar no *critical triples* appear. A *critical triple* is a triple of simple terms that can be contracted in two ways, like in $a^l \ldots a \ldots a^r = (a^l a) \ldots a^r \Rightarrow a^r$ or $a^l \ldots a \ldots a^r = a^l \ldots (aa^r) \Rightarrow a^l$ (all simple terms between a^l and a reduce to 1 and so do all simple terms between a and a^r). A reduction is then computed in time proportional to the length of the appropriate type assignment.

Chapter 5

Pregroup Grammars with Letter Promotions

Contents

In this chapter we consider a complete system of **CBL** enriched with letter promotions. First we describe the calculus, following [BLZ09]. Then we present a proof of a fact that the letter promotion problem for pregroups can be solved in polynomial time, which was presented in [BLZ09]. In the next section we adjust our main algorithm to parsing pregroup grammars with letter promotions. Finally, we give some examples of such grammars.

5.1 CBL enriched with letter promotions

CBL can be enriched with more general assumptions than partial order. Mater and Fix [MF05] consider such pregroup grammars. They show that **CBL** enriched with finitely many assumptions of the form $X \Rightarrow Y$ can be undecidable. An interesting problem they propose is called a *letter promotion problem for pregroups*. To define it, one needs a system with *letter promotions*, that is assumptions of the form $p^{(m)} \Rightarrow q^{(n)}$.

Let us consider a complete system of **CBL** with letter promotions obtained by modifying (IND) to Promotion Rules (PRO):

> (PRO) $X, p^{(m+k)}, Y \Rightarrow X, q^{(n+k)}, Y$, if either k is even and $p^{(m)} \Rightarrow q^{(n)}$ is an assumption, or k is odd and $q^{(n)} \Rightarrow p^{(m)}$ is an assumption.

The letter promotion problem for pregroups (**LPPP**) can be stated as follows:

> For the given finite set R, of letter promotions, and terms t, u, verify whether $t \Rightarrow u$ in **CBL** enriched with all promotions from R as assumptions.

Mater and Fix proved that the problem is NP-hard, assuming the binary (or decimal) representation of $p^{(n)}$. However, Buszkowski and Lin Zhe [BLZ09], show that the problem can be solved in polynomial time, if the size of n in $p^{(n)}$ is counted as $|n|+1$. This way of counting is more natural, since we can treat $p^{(n)}$ as an abbreviation for $p^{l\cdots l}$ or $p^{r\cdots r}$. Moreover, in linguistic applications, $|n|$ does not exceed 3. Actually, in most cases one uses single or double adjoints only. Further we assume such a representation.

Now we continue with presentation of pregroups with letter promotions and some results given in [BLZ09]. Assume that R is a set of letter promotions, then $R \vdash_{\mathbf{CBL}} X \Rightarrow Y$ denotes that X can be transformed into Y by a finite number of applications of (CON), (EXP) and (PRO), restricted to the assumption from R. In this system we do not consider poset rules, they can be presented in a form of assumptions from the set R.

We write $t \Rightarrow_R u$, if $t \Rightarrow u$ is an instance of (PRO), restricted to the assumptions from R (X, Y are empty). We write $t \Rightarrow_R^* u$, if there exist terms $t_0, \ldots, t_k$ such that $k \geqslant 0$, $t_0 = t$, $t_k = u$, and $t_{i-1} \Rightarrow_R t_i$, for all $i = 1, \ldots, k$. Therefore, $\Rightarrow_R^*$ is the reflexive and transitive closure of the relation $\Rightarrow_R$. One defines Generalized Contraction and Generalized Expansion for **CBL** with letter promotions. They are generalization on of (CON) and (EXP) derivable in **CBL** with assumptions from R.

(GCON-R) $X, p^{(m)}, q^{(n+1)}, Y \Rightarrow X, Y$ if $p^{(m)} \Rightarrow_R^* q^{(n)}$,
(GEXP-R) $X, Y \Rightarrow X, p^{(m+1)}, q^{(n)}, Y$ if $p^{(m)} \Rightarrow_R^* q^{(n)}$.

Clearly, (CON) is a special instance of (GCON-R) and (EXP) is a special instance of (GEXP-R). One can treat any iteration of (PRO) as a single step.

(PRO-R) $X, t, Y \Rightarrow X, u, Y$ if $t \Rightarrow_R^* u$.

Mater and Fix [MF05] proved only a weaker version of Lambek's Normalization Theorem for **CBL** with letter promotions (only for sequents $X \Rightarrow \varepsilon$). Buszkowski and Lin Zhe [BLZ09] gave the proof of the Normalization Theorem for this system in its full version.

Theorem 5.1 ([BLZ09]). *If $R \vdash_{\mathbf{CBL}} X \Rightarrow Y$, then there exist Z, U such that $X \Rightarrow Z$ by a finite number of instances of (GCON-R), $Z \Rightarrow U$ by a finite number of instances of (PRO-R), and $U \Rightarrow Y$ by a finite number of instances of (GEXP-R).*

The proof is based on induction on the length of a derivation of $X \Rightarrow Y$ from R in **CBL**. A derivation denotes here that there exists a sequence $X_0, \ldots, X_k$ such that $X = X_0$, $Y = X_k$ and for any $i = 1, \ldots, k$ $X_{k-1} \Rightarrow X_k$ is an instance of (GCON-R) or (GEXP-R) or (PRO-R). A derivation is *normal* if all applications of (GCON-R)

are at the beginning of the derivation, then there are all applications of (PRO-R) followed by all applications of (GEXP-R). It is proved that each derivation can be transformed into a normal derivation not longer then the initial one. Our proof of the normalization theorem for **CBL** with letter promotions with 1 presented in **Section 6.1** extends this proof.

As a consequence we obtain the following lemma.

Lemma 5.1. *$R \vdash_{\mathbf{CBL}} t \Rightarrow u$ if, and only if, $t \Rightarrow_R^* u$.*

Therefore, **LPPP** can be stated shortly:

given R, t, u verify whether $t \Rightarrow_R^* u$.

To show that the problem is solvable in polynomial time Buszkowski and Lin Zhe [BLZ09] offer the procedure described below.

5.2 Polynomial time decidability of CBL enriched with letter promotions

We present results of Buszkowski and Lin Zhe published in [BLZ09]. We give the reductions in details, as our results in **Chapter 6** use these constructions.

First **LPPP** is reduced into a graph-theoretic problem of the existence of a route of the given weight in a weighted directed graph with integer weights. The graph is constructed for the set of assumptions R. Integers are represented in unary notations, e.g 4 is a string of four digits. Further, this problem is reduced to the emptiness problem for context-free languages. The graph is transformed into an **NFA** . Both reductions can be performed in polynomial time and the final problem is solvable in polynomial time. Consequently, the provability problem for **CBL** enriched with letter promotions is solvable in polynomial time.

First the letter promotion problem for pregroups with letter promotions is reduced to a graph theoretic problem. One starts with defining a finite, directed weighted graph $G(R)$. Let $P(R)$ denote the set of all atoms appearing in promotions from R. The vertices of $G(R)$ are elements p_0, p_1, for all $p \in P(R)$. Assume n is any integer, then one sets $\pi(n) = 0$ if n is even and $\pi(n) = 1$ if n is odd. One defines $\pi^*(n) = 1 - \pi(n)$. If $p^{(m)} \Rightarrow q^{(n)}$ is a promotion from R, then the graph $G(R)$ contains an arc from $p_{\pi(m)}$ to $q_{\pi(n)}$ with weight $n - m$ and an arc from $q_{\pi^*(m)}$ to $p_{\pi^*(n)}$ with weight $m - n$. Thus, there are two arcs in $G(R)$ for each promotion form R.

An arc from v to w of weight k is represented by a triple (v, k, w). A *route* from a vertex v to a vertex w in $G(R)$ is defined, as usually, as a sequence of arcs $(v_o, k_1, v_1), \ldots, (v_{r-1}, k_r, v_r)$, where $v_o = v$, $v_r = w$ and the target of each arc, except the last arc, is the source of the following arc. The *length* of such route is equal to r and its *weight* is $k_1 + \ldots + k_r$. The *trivial* route from v to v is of length 0 and weight 0.

The following results are proved.

Lemma 5.2 ([BLZ09]). *If* $p^{(m)} \Rightarrow_R q^{(n)}$, *then* $(p_{\pi(m)}, n-m, q_{\pi(n)})$ *is an arc in* $G(R)$.

Proof. Assume $p^{(m)} \Rightarrow_R q^{(n)}$. There are two cases.

(I) Let $m = m' + k$ and $n = n' + k$ and k be even. Assume that $p^{(m')} \Rightarrow q^{(n')}$ belongs to R. Then $(p_{\pi(m')}, n'-m', q_{\pi(n')})$ is an arc in $G(R)$. Since k is even, $\pi(m) = \pi(m')$ and $\pi(n) = \pi(n')$ and consequently $n - m = n' - m'$, which yields the thesis.

(II) Let $m = m' + k$ and $n = n' + k$ and k be odd. Assume also $q^{(n')} \Rightarrow p^{(m')}$ belongs to R. Then $(p_{\pi^*(m')}, n'-m', q_{\pi^*(n')})$ is an arc in $G(R)$. Since k is odd, $\pi(m) = \pi^*(m')$ and $\pi(n) = \pi^*(n')$ and consequently $n - m = n' - m'$, which yields the thesis.

□

Lemma 5.3 ([BLZ09]). *Let* $(v, r, q_{\pi(m)})$ *be an arc in* $G(R)$. *Then, there exists* $p \in P(R)$ *such that* $v = p_{\pi(n-r)}$ *and* $p^{(n-r)} \Rightarrow_R q^{(n)}$.

Proof. There are two cases.

(I) Assume $(v, r, q_{\pi(n)})$ equals the arc $(p_{\pi(m')}, n'-m', q_{\pi(n')})$ and $p^{(m')} \Rightarrow q^{(n')}$ belongs to R. Then $r = n' - m'$ and $\pi(n) = \pi(n')$. There is $n = n' + k$ for an even integer k and $n - r = m' + k$. This yields $\pi(n-r) = \pi(m')$ and $p^{(n-r)} \Rightarrow_R q^{(n)}$.

(II) Assume $(v, r, q_{\pi(n)})$ equals the arc $(p_{\pi^*(m')}, n'-m', q_{\pi^*(n')})$ and $q^{(n')} \Rightarrow p^{(m')}$ belongs to R. Then $r = n' - m'$ and $\pi(n) = \pi^*(n')$. There is $n = n' + k$ for an odd integer k and $n - r = m' + k$. This yields $\pi(n-r) = \pi^*(m')$ and $p^{(n-r)} \Rightarrow_R q^{(n)}$.

□

Theorem 5.2 ([BLZ09]). *Let* $p, q \in P(R)$. *Then* $p^{(m)} \Rightarrow_R^* q^{(n)}$ *if, and only if, there exists a route from* $p_{\pi(m)}$ *to* $q_{\pi(n)}$ *of weight* $n - m$ *in* $G(R)$.

Proof. The "only if" part easily follows from **Lemma 5.2**. The "if" part can be proved by induction on the length of a route from $p_{\pi(m)}$ to $q_{\pi(n)}$ in $G(R)$. **Lemma 5.3** is used.

At first a trivial route is considered. Then $p = q$ and $n - m = 0$. Hence, $m = n$ and it yields $p^{(m)} \Rightarrow p^{(m)}$.

Now assume that $(p_{\pi(m)}, r_1, v_1), (v_1, r_2, v_2), \ldots, (v_k, r_{k+1}, q_{\pi(n)})$ is a route of length $k+1$ and weight $n - m$ in $G(R)$. By **Lemma 5.3**, there exists $s \in P$ such that $v_k = s_{\pi(n-r_{k+1})}$ and $s^{(n-r_{k+1})} \Rightarrow_R q^{(n)}$. The weight of the initial subroute of length k is $n - m - r_{k+1} = n - r_{k+1} - m$. By the induction hypothesis $p^{(m)} \Rightarrow_R^* s^{(n-r_{k+1})}$, which yields $p^{(m)} \Rightarrow_R^* q^{(n)}$. □

To verify whether $R \vdash p^{(m)} \Rightarrow q^{(n)}$ one considers two cases. If $p, q \in P(R)$, then by **Lemma 5.1** and **Theorem 5.2**, the answer is YES iff there exists a route from $p_{\pi(m)}$ to $q_{\pi(n)}$ of weight $n - m$ in $G(R)$. Otherwise, $R \vdash p^{(m)} \Rightarrow q^{(n)}$ iff $p = q$ and $m = n$.

LPPP is thus reduced to the following problem: given a finite weighted, directed graph G with integer weights, two vertices v and w, and an integer k, verify whether

there exists a route from v to w of weight k in G. One reduces that problem to the emptiness problem for context-free languages. A trivial route in G exists only if $v = w$ and $k = 0$, therefore one can restrict the problem to nontrivial routes only.

First an **NFA** $M(G)$ is constructed.

- $\Sigma_{M(G)} = \{+, -\}$,
- $Q_{M(G)} = V_G \cup \mathit{Auxiliary}$ - the set of states, where V_G is the set of vertices of the graph G and $\mathit{Auxiliary}$ is the set of some auxiliary states,
- q_0 - the initial state, if $k = 0$ then $q_o = v$, otherwise $q_0 = i_1$, $i_1 \in \mathit{Auxiliary}$, defined below
- $q_f = w$ - the final state,
- transitions are described as follows:
  If (v', n, w') is an arc in G, $n > 0$, then v' is linked with w' by n transitions $v' \to s_1 \to \ldots \to s_n = w'$, all labeled by $+$. $s_1, \ldots, s_n$ are new states. Similarly, v' is linked with w' for $n < 0$, with the only difference that transitions are labeled with $-$. For $n = 0$ v' is linked with w' by two transitions $v' \to s \to w'$, one with label $+$, the other one labeled by $-$, where s is a new state. If $k \neq 0$, then k new states are added $i_1, \ldots i_k$ with transitions $i_1 \to i_2 \to \ldots i_k$ and $i_k \to v$, all labeled by $-$ if $k > 0$ and by $+$, if $k < 0$. $\mathit{Auxiliary}$ consists of all new states.

There holds the following equivalence: there exists a nontrivial route from v to w of weight k in G iff there exists a nontrivial route from the start state to the final state that visits as many pluses as minuses.

The right-hand side of the above equivalence is equivalent to the condition $L(M(G)) \cap L \neq \emptyset$, where L is the context-free language consisting of all nonempty strings on $\{+, -\}$ which contain as many pluses as minuses.

A **CFG** for L consists of the following production rules:

$S \mapsto SS$,
$S \mapsto +S-$,
$S \mapsto -S+$,
$S \mapsto +-$,
$S \mapsto -+$.

This grammar can be transformed into an equivalent grammar G_{+-} in **CNF** in constant time. Finally, G_{+-} is modified to a **CFG** $\bar{G}$ for $L(M(G)) \cap L$ in a routine way.

- $V = \{(q, A, q') : \ q, q' \in Q_{M(G)}, \ A \in V_{G_{+-}}\}$ - the set of variables,
- (q_o, S, q_f), - initial symbol, such that q_0 and q_f are the initial and final state of $M(G)$, respectively, and S is the initial symbol of G_{+-},

- the production rules are as follows:

  $(q_1, A, q_3) \mapsto (q_1, B, q_2)(q_2, C, q_3)$, for any rule $A \mapsto BC$ of G_{+-},
  $(q_1, A, q_2) \mapsto a$, if there exists a rule $A \mapsto a$ of G_{+-}, and $M(G)$ admits the transition from q_1 to q_2 labeled by $a \in \{+, -\}$.

The construction of $G(R)$ can be performed in linear time. The size of a graph G is defined as the sum of the following numbers: the number of vertices, the number of arcs, and the sum of absolute values of weights of arcs. $M(G)$ can be constructed in $O(n^2)$, where n is the size of G. A **CFG** for $L(M(G)) \cap L$ can be constructed in $O(n^3)$, where n is the size of $M(G)$ counted as the number of transitions. Assume the size of a **CFG** is the sum of the number of variables and the number of rules. The emptiness problem for a context-free language can be solved in $O(n^2)$, where n is the size of the **CFG** for the language. Consequently, the following theorem holds.

Theorem 5.3. *The letter promotion problem for pregroups with letter promotions can be solved in polynomial time.*

Algorithm 5.1 Algorithm for verifying whether $t \Rightarrow_R^* u$ in non-trivial case

Input: simple terms t, u, set of assumptions R

1. Construct a graph $G(R)$.
2. Construct $M(G(R))$ for given t, u.
3. Construct G_{+-}.
4. Construct $\bar{G}$.
5. **If** $L(\bar{G}) \neq \emptyset$, **then** $t \Rightarrow_R^* u$
   else $t \not\Rightarrow_R^* u$.

A *pregroup grammar with letter promotions* is a tuple $G = (\Sigma, P, R, s, I)$ where R is the set of assumptions (letter promotions), such that $P(R) \subseteq P$. Σ, P, s, I are defined in the same way as for pregroup grammars. The set of assumptions R defines a relation on simple terms and substitutes the partial order. One can notice that all Generalized Contractions $t, u \Rightarrow 1$ derivable from R in **CBL** for arbitrary simple terms $t, u \in T(G)$, where $T(G)$ is the set of all terms appearing in I, can be determined in polynomial time. To determine the contractions one uses **Algorithm 5.1**.

5.3 A dynamic parsing algorithm for pregroup grammars with letter promotions

The algorithm is a modification of our algorithm for pregroup grammars. First we have to determine the set *Pairs* of all possible Generalized Contractions $t, u \Rightarrow 1$

in the given pregroup grammar with letter promotions. Then the algorithm works in an analogous way as the algorithm described in **Section 4.3**, with the only difference that it checks for contractions in the set $Pairs$, not using the condition (GC).

Our goal is to check whether the given string of symbols $x = a_1 \dots a_n$ is a member of the language generated by a pregroup grammar with letter promotions G, that is whether $x \in L(G)$. If this is the case, we want to obtain the appropriate derivation.

We fix $G = (\Sigma, P, R, \leqslant, s, I)$ and $x \in \Sigma^+,\ x = a_1 \dots a_n$. Again, we use special symbols $*, \langle, \rangle$. We adopt the same notation as in **Section 4.3**.

The definition of the function M is similar as in **Section 4.3**. The only difference is that by a reduction to 1 here we mean a reduction in the **CBL** with letter promotions.

Let $M(i,j), 1 \leqslant i < j \leqslant |W^x|$ be a function such that $M(i,j) = 1$ iff one of the following conditions holds:

- **M1.** $W^x_{[i,j]} \in T^+$ and it reduces to 1.
- **M2a.** $W^x_{[i,j]}$ is of the form $\langle ... \rangle ... \langle \mathbf{V} * Z$, where:
  - $Z \in T^+$
  - $\mathbf{V}$ contains no angle brackets
  - in $W^x_{[i,j]}$ there are precisely g $(g \geqslant 0)$ pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $X_1 ... X_g Z$ reduces to 1
- **M2b.** $W^x_{[i,j]}$ is of the form $Y * \mathbf{U} \rangle ... \langle ... \rangle$, where:
  - $Y \in T^+$
  - $\mathbf{U}$ contains no angle brackets
  - in $W^x_{[i,j]}$ there are precisely g $(g \geqslant 0)$ pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $Y X_1 ... X_g$ reduces to 1
- **M3.** $W^x_{[i,j]}$ is of the form $Y * \mathbf{U} \rangle ... \langle \mathbf{V} * Z$, where:
  - $Y, Z \in T^+$
  - $\mathbf{U}, \mathbf{V}$ contain no angle brackets
  - in $W^x_{[i,j]}$ there are precisely g $(g \geqslant 0)$ pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $Y X_1 ... X_g Z$ reduces to 1
- **M4.** $W^x_{[i,j]}$ is of the form $\langle ... \rangle ... \langle ... \rangle$, where:
  - in $W^x_{[i,j]}$ there are precisely g $(g \geqslant 1)$ pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $X_1 ... X_g$ reduces to 1

In all other cases $M(i,j) = 0$.

Obviously, as before, the whole string W^x is of the form **M2a**. Therefore, $M(1,|W^x|)=1$ entails existence of a string $X_1 \dots X_n s^r$ reducing to 1. Each X_i is the searched type for a_i. Thus, a solution to the recognition problem is found, i.e. $x \in L(G)$. On the other hand, if $M(1,|W^x|)=0$, then there is no string reducing to 1, in which exactly one element comes from each pair of angle brackets and which reduces to 1. It means $x \notin L(G)$.

We start the algorithm by determining the set $Pairs$ of all pairs $(p^{(m)}, q^{(n+1)})$ such that $p,q \in P$ and $p^{(m)} \Rightarrow_R^* q^{(n)}$, which can be done in polynomial time, see [BLZ09].

Then we compute $M(i,j)$ dynamically. First, we look for two adjacent simple terms W_i^x and W_{i+j}^x belonging to the set Pairs. If $(W_i^x, W_{i+j}^x) \in Pairs$ then we put $M(i,i+1)=1$.

The other initial case is when $W_{[i,j]}^x$ is of the form $p^{(m)} * \mathbf{U}\rangle\langle \mathbf{V} * q^{(n+1)}$, such that $(p^{(m)}, q^{(n+1)}) \in Pairs$ and strings $\mathbf{U}, \mathbf{V}$ contain no angle brackets. Then we put $M(i,j)=1$.

When we already know $M(g,h)$, for all $1 \leqslant g < h \leqslant |W^x|$ such that $h-g < j-i$, we can compute $M(i,j)$. There are several cases:

- **A1a.** $W_i^x, W_j^x \in T$. If there exists k such that $i < k < j-1$, $W_k^x \in T$, $W_{(k+1)}^x \in T$ and both $M(i,k)$ and $M(k+1,j)$ are equal to 1, then we put $M(i,j)=1$.

- **A1b.** $W_i^x, W_j^x \in T$. If there exists k such that $i < k < j-1$, $W_k^x = \rangle$, $W_{(k+1)}^x = \langle$ and both $M(i,k)$ and $M(k+1,j)$ are equal to 1, then we put $M(i,j)=1$.

- **A2.** $W_i^x = p^{(m)}$, $W_j^x = q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$.
  If $M(i+1,j-1)=1$, then $M(i,j)=1$.

- **A3a.** $W_{[i,j]}^x$ is of the form $\langle \dots \rangle \dots \langle \dots p^{(m)}$, $p \in P, m \in \mathcal{Z}$. If there exists k such that $i < k < j$, $W_k^x = *, W_{[i+1,k]}^x$ contains no angle brackets and $M(k+1,j) = 1$, then $M(i,j)=1$.

- **A3b.** $W_{[i,j]}^x$ is of the form $p^{(m)} \dots \rangle \dots \langle \dots \rangle$, $p \in P, m \in \mathcal{Z}$. If there exists k such that $i < k < j$, $W_k^x = *, W_{[k,j-1]}^x$ contains no angle brackets and $M(i,k-1) = 1$, then we put $M(i,j)=1$.

- **A4a.** $W_{[i,j]}^x$ is of the form $p^{(m)} * \dots \rangle \dots \langle \dots q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$. If $M(k,j-1)=1$, where k is the position of the first left angle bracket in the string $W_{[i,j]}^x$, then we put $M(i,j)=1$.

- **A4b.** $W_{[i,j]}^x$ is of the form $p^{(m)} \dots \rangle \dots \langle \dots * q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$. If $M(i+1,k)=1$, where k is the position of the last right angle bracket in the string $W_{[i,j]}^x$, then $M(i,j)=1$.

- **A4c.** $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle ...\langle ... * q^{(n+1)}$, where the string "..." in between the angle brackets is not empty and $(p^{(m)}, q^{(n+1)}) \in Pairs$. If $M(k,k') = 1$, where k is the position of the first left angle bracket in the string $W^x_{[i,j]}$ and k' is the position of the last right angle bracket in the string $W^x_{[i,j]}$, then $M(i,j) = 1$.

- **A5.** $W^x_{[i,j]}$ is of the form $\langle ...\rangle ...\langle ...\rangle$. If $M(k,k') = 1$, where W^x_k is a simple term in between the first pair of angle brackets, $W^x_{k'}$ is a simple term in between last pair of angle brackets in the string $W^x_{[i,j]}$ and $W^x_{k-1} = *$ and $W^x_{k'+1} = *$, then $M(i,j) = 1$.

In all other cases $M(i,j) = 0$.

Theorem 5.4. *The algorithm computes $M(i,j)$ correctly.*

Proof. The proof below is very similar to the proof given in **Section 4.3**. We state it here as it will be useful in **Section 6.3**.

We will show at first that, if the algorithm computes $M(i,j) = 1$, then $M(i,j) = 1$ according to the definition of M. We will prove it by induction on the length of the string.

For strings of length two, the algorithm computes $M(i,j) = 1$ only in case when $W^x_i = p^{(m)}$ and $W^x_{i+1} = q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$. $W^x_{[i,j]}$ is then of the form **(M1)**, since $W^x_{[i,j]} \in T^+$ and the string $W^x_{[i,j]}$ reduces to 1. Hence, $M(i,j) = 1$ according to the definition of M.

The other initial case when the algorithm computes $M(i,j) = 1$ is when $W^x_{[i,j]}$ is of the form $p^{(m)} * \mathbf{U}\rangle\langle \mathbf{V} * q^{(n+1)}$, $(p^{(m)}, q^{(n+1)}) \in Pairs$ and the strings $\mathbf{U}$ and $\mathbf{V}$ contain no angle brackets. $W^x_{[i,j]}$ is then of the form **(M3)**, since we can assume $X = p^{(m)}$, $Y = q^{(n+1)}$ and $g = 0$. So, XY reduces to 1. Hence, $M(i,j) = 1$ according to the definition of M.

Now let us consider the recursive cases when the algorithm computes $M(i,j) = 1$ (all cases of the description of the algorithm).

A1a. $W^x_{[i,j]}$ is of the form **A1a**. Then the substrings $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ are shorter than $W^x_{[i,j]}$, therefore, by the induction hypothesis, both $M(i,k)$ and $M(k+1,j)$ are equal to 1 according to the definition of M. $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ can be of the form **(M1)** or **(M3)**. There are the following cases.

- If both substrings $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ are of the form **(M1)**, then $W^x_{[i,j]}$ also consists of simple terms and reduces to 1, as both $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ reduce to 1. $W^x_{[i,j]}$ is therefore of the form **(M1)**. Hence, $M(i,j) = 1$ in accordance with the definition of M.

- $W^x_{[i,k]}$ is of the form **(M1)**. Therefore, it consists of simple terms and reduces to 1 and $W^x_{[k+1,j]}$ is of the form **(M3)**. Then $W^x_{[i,j]}$ is also of the form **(M3)**. Hence, $M(i,j) = 1$ according to the definition of M.

- $W^x_{[i,k]}$ is of the form **(M3)** and $W^x_{[k+1,j]}$ is of the form **(M1)**. So, it consists of simple terms and reduces to 1. Then $W^x_{[i,j]}$ is also of the form **(M3)**. Hence, $M(i,j) = 1$ according to the definition of M.
- $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ are of the form **(M3)**. Hence, the whole string $W^x_{[i,j]}$ is also of the form **(M3)**. Then $M(i,j) = 1$ in accordance with the definition of M.

A1b. $W^x_{[i,j]}$ is of the form **A1b**. Then the substrings $W^x_{[i,k]}$ and $W^x_{[k+1,j]}$ are shorter than $W^x_{[i,j]}$. Therefore, by the induction hypothesis, both $M(i,k)$ and $M(k+1,j)$ are equal to 1 according to the definition of M. $W^x_{[i,k]}$ is of the form **(M2b)** so there is a string $YX_1...X_{g_1}$ that reduces to 1 (such that the first simple term of Y is W^x_i, $X_h \in T^+$ is a substring in between the h-th pair of matching angle brackets in the string $W^x_{[i,k]}$). The substring $W^x_{[k+1,j]}$ is of the form **(M2a)** so there is a string $X'_1...X'_{g_2}Y'$ that reduces to 1 (such that the last simple term of Y' is W^x_j, $X'_h \in T^+$ is a substring in between the h-th pair of matching angle brackets in the string $W^x_{[k+1,j]}$). So the whole string $W^x_{[i,j]}$ is of the form **(M3)** (with the string $YX_1...X_{g_1}X'_1...X'_{g_2}Y'$ reducing to 1).

A2. $W^x_{[i,j]}$ is of the form **A2**. $M(i+1,j-1) = 1$ is computed according to the definition of M by the induction hypothesis, as $W^x_{[i+1,j-1]}$ is shorter than $W^x_{[i,j]}$. Then the substring $W^x_{[i+1,j-1]}$ can be:

- of the form **(M1)**, then the whole string $W^x_{[i,j]}$ consists of simple terms and reduces to 1, so it is also of the form **(M1)**. Hence, $M(i,j) = 1$ according to the definition of M.
- of the form **(M3)**, that is $Y * \mathbf{U}\rangle...\langle\mathbf{V} * Z$, where $\mathbf{U}$ and $\mathbf{V}$ contain no angle brackets and the string $YX_1...X_gZ$ reduces to 1. We can assume $Y' = p^{(m)}Y$ and $Z' = Zq^{(n+1)}$. Then $W^x_{[i,j]} = Y' * \mathbf{U}\rangle...\langle\mathbf{V} * Z'$ and the string $Y'X_1...X_gZ'$ reduces to 1. Hence $M(i,j) = 1$ according to the definition of M.

We should notice that the string $W^x_{[i+1,j-1]}$ cannot be of the form **(M2a)**, **(M2b)** or **(M4)** since a simple term can be followed (preceded) only by another simple term or an asterisk.

A3a. $W^x_{[i,j]}$ is of the form **A3a**. $W^x_{[k+1,j]}$ is shorter than $W^x_{[i,j]}$, so $M(k+1,j) = 1$ is computed according to the definition of M by the induction hypothesis. The string $W^x_{[k+1,j]}$ ends with a simple term, so it can be of the form **(M1)**, **(M2a)** or **(M3)**. But it cannot begin with a left angle bracket and it contains angle brackets, so it must be of the form **(M3)**. So, there is a string $YX_1...X_gZ$ reducing to 1. Hence, $W^x_{[i,j]}$ is of the form **(M2a)** with the string $YX_1...X_gZ$ reducing to 1. Therefore, $M(i,j) = 1$ according to the definition of M.

A3b. $W^x_{[i,j]}$ is of the form **A3b**. The string $W^x_{[i,k-1]}$ is shorter than $W^x_{[i,j]}$, so M(i,k-1)=1 is computed according to the definition of M, by the induction hypothesis. The string $W^x_{[i,k-1]}$ begins with a simple term, so it can be of the form

(M1), **(M2b)** or **(M3)**. But it cannot end with a right angle bracket and it contains angle brackets, so it must be of the form **(M3)**. So there is a string $YX_1...X_gZ$ reducing to 1. Hence $W^x_{[i,j]}$ is of the form **(M2b)** with the string $YX_1...X_gZ$ reducing to 1. Therefore $M(i,j) = 1$ according to the definition of M.

A4a. . $W^x_{[i,j]}$ is of the form **A4a**. $W^x_{[k,j-1]}$ is shorter than $W^x_{[i,j]}$. Hence, by the induction hypothesis, $M(k, j-1) = 1$ according to the definition of M. The string $W^x_{[k,j-1]}$ must be therefore of the form **(M2a)**, since it is the only form starting with an angle bracket and ending with an element different from a bracket. Hence, $W^x_{[k,j-1]} = \langle ...\rangle ...\langle \mathbf{V} * Z$, where Z is the string of simple terms, $\mathbf{V}$ contains no angle brackets and there are precisely g pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1...X_gZ$ reduces to 1. Let $Z' = Zq^{(n+1)}$, $Y = p^{(m)}$. Then $YX_1...X_gZ'$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M3)**. Then $M(i,j) = 1$ in accordance with the definition of M.

A4b. $W^x_{[i,j]}$ is of the form **A4b**. $W^x_{[i+1,k]}$ is shorter than $W^x_{[i,j]}$. Hence, by the induction hypothesis, $M(i+1, k) = 1$ according to the definition of M. The string $W^x_{[i+1,k]}$ must be therefore of the form **(M2b)**, since it is the only form ending with an angle bracket but starting with an element different from a bracket. Hence, $W^x_{[i+1,k]} = Y * \mathbf{U}\rangle ...\langle ...\rangle$, where Y is the string of simple terms, $\mathbf{U}$ contains no angle brackets and there are precisely g pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $YX_1...X_g$ reduces to 1. Let $Y' = p^{(m)}Y$, $Z = q^{(n+1)}$. Then $Y'X_1...X_gZ$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M3)**. Then $M(i,j) = 1$ in accordance with the definition of M.

A4c. $W^x_{[i,j]}$ is of the form **A4c**. $W^x_{[k,k']}$ is shorter than $W^x_{[i,j]}$. Hence, by the induction hypothesis, $M(k, k') = 1$ according to the definition of M. The string $W^x_{[k,k']}$ must be therefore of the form **(M4)**. Hence, $W^x_{[k,k']} = \langle ...\rangle ...\langle ...\rangle$, where there are precisely g pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1...X_g$ reduces to 1. Let $Y = p^{(m)}$, $Z = q^{(n+1)}$, $\mathbf{U} = W^x_{[i+2,k-1]}$ (if $k > i+3$, else $\mathbf{U} = 1$) and $\mathbf{V} = W^x_{[k'+1,j-2]}$ (if $k' < j-3$, else $\mathbf{V} = 1$). Then $YX_1...X_gZ$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M3)**. Then $M(i,j) = 1$ according to the definition of M.

A5. $W^x_{[i,j]}$ is of the form **A5**. $W^x_{[k,k']}$ is shorter than $W^x_{[i,j]}$ hence $M(k, k') = 1$ according to the definition of M. $W^x_{[k,k']}$ must be then of the form **(M3)**, so the whole string $W^x_{[i,j]}$ is of the form **(M4)** and therefore $M(i,j) = 1$ in accordance with the definition of M.

We will prove that the algorithm finds correctly all substrings for which the function $M(i,j) = 1$ by induction on the length of the substring. Let us notice that there are no such substrings that contain asterisk but no angle brackets.

The only strings of length two, for which $M(i,j)=1$, are of the form $p^{(m)}q^{(n+1)}$, where $(p^{(m)},q^{(n+1)}) \in Pairs$ (that is of the form **(M1)**), and the algorithm finds them correctly.

Let us consider now the substrings of the length $l>2$ such that for all $l'<l$ the algorithm finds the substrings of the length l' correctly (all forms of the definition of M).

1. $W^x_{[i,j]}$ is of the form **(M1)** that is $W^x_{[i,j]} \in T^+$ and it reduces to 1. There are two possible cases:

- $W^x_i W^x_j$ does not reduce to 1 in the whole reduction of the string $W^x_{[i,j]}$ to 1. Then $W^x_{[i,j]}$ can be divided into 2 substrings. Each of them reduces to 1, is shorter than $W^x_{[i,j]}$, so they are found by the algorithm correctly (by the induction hypothesis). Hence by (**A1a**), $M(i,j)=1$.
- $W^x_i W^x_j$ reduces to 1 in the whole reduction of $W^x_{[i,j]}$ to 1. Then $W^x_{[i+1,j-1]}$ is shorter substring reducing to 1. By the induction hypothesis the string is found by the algorithm correctly, so $M(i,j)=1$ (case (**A2**)).

2a. $W^x_{[i,j]}$ is of the form **(M2a)**. If $M(i,j)=1$ then there exists a substring of the form $*X_1*$ in between the first pair of angle brackets such that X_1 takes part in some reduction to 1 (if $g>0$). Let i' be the position of the first simple term in X_1. The substring $W^x_{[i',j]}$ is of the form **(M3)**, is shorter than $W^x_{[i,j]}$ and $M(i',j)=1$. By the induction hypothesis, the string is found by the algorithm correctly, so $M(i,j)=1$ (case (**A3a**)). If $g=0$, then Z reduces to 1 and it is of the form **(M1)** and shorter than $W^x_{[i,j]}$. So, by the induction hypothesis, it is found by the algorithm correctly. Then $M(i,j)=1$ by (**A3a**).

2b. $W^x_{[i,j]}$ is of the form **(M2b)**, that is $W^x_{[i,j]} = C*\mathbf{U}\rangle ... \langle ... * D_g * ...\rangle$.

If $M(i,j)=1$, then there exists the substring of the form $*X_g*$ (if $g>0$) in between the last pair of angle brackets, such that X_g takes part in some reduction to 1. Let j' be the position of the last simple term in X_g. The substring $W^x_{[i,j']}$ is of the form **(M3)**, it is shorter than $W^x_{[i,j]}$ and $M(i,j')=1$. By the induction hypothesis, the string is found by the algorithm correctly, so $M(i,j)=1$ (case (**A3b**)). If $g=0$, then Y reduces to 1 and it is of the form **(M1)**, it is shorter than $W^x_{[i,j]}$, therefore, by the induction hypothesis, it is found by the algorithm correctly. Then $M(i,j)=1$ by (**A3b**).

3. $W^x_{[i,j]}$ is of the form **(M3)**. $W^x_i W^x_j$ takes part in the reduction to 1 in $W^x_{[i,j]}$. Let us assume $W^x_i W^x_{i'}$ reduce to 1 where $i' \neq j$. Then $W^x_{i'+1}$ can be:

- simple term. Then $W^x_{[i,i']}$ and $W^x_{[i'+1,j]}$ are of the form **(M1)** or **(M3)**, they are shorter and both $M(i,i')=1$ and $M(i'+1,j)=1$. Hence, by the induction hypothesis, these strings are found by the algorithm correctly, so $M(i,j)=1$ (case (**A1a**)).

- asterisk. Then $W^x_{[i,i']}$ is of the form **(M1)** or **(M3)**, it is shorter and $M(i,i') = 1$. Hence, by the induction hypothesis, the string is found by the algorithm correctly. Let k be the position of the first right angle bracket following i'. $W^x_{[i,k]}$ is shorter than $W^x_{[i,j]}$ and it is of the form **(M2b)**. So, by the induction hypothesis $M(i,k) = 1$ (case (**A3b**)). Similarly, the substring $W^x_{[k+1,j]}$ is of the form **(M2a)** , it is shorter than $W^x_{[i,j]}$. So, by the induction hypothesis $M(k+1,j) = 1$ (case (**A3a**)). Hence, $M(i,j) = 1$, by the induction hypothesis (case (**A1b**)).

Let us assume $i' = j$ so $W^x_i W^x_j$ reduces to 1. There are the following cases.

- W^x_{i+1}, W^x_{j-1} are simple terms. Then the substring $W^x_{[i+1,j-1]}$ is of the form **(M3)**. It is shorter than $W^x_{[i,j]}$, therefore $M(i+1,j-1) = 1$. By the induction hypothesis, that string is found by the algorithm correctly so $M(i,j) = 1$ (case (**A2**)).
- $W^x_{i+1} = *, W^x_{j-1} \in T$. So $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle ...\langle ...q^{(n+1)}$. Let j' be the position of the first left angle bracket in $W^x_{[i,j]}$. We know there exists $j' < k < j-1$ such that $W^x_k W^x_{j-1}$ reduces to 1. Hence, $W^x_{[j',j-1]}$ is of the form **(M2a)**. It is shorter than $W^x_{[i,j]}$, therefore $M(j',j-1) = 1$. By the induction hypothesis, that string is found by the algorithm correctly, so $M(i,j) = 1$ (case (**A4a**)).
- $W^x_{i+1} \in T$, $W^x_{j-1} = *$. So $W^x_{[i,j]}$ is of the form $p^{(m)} ...\rangle ...\langle ... * q^{(n+1)}$. Let i' be the position of the last right angle in $W^x_{[i,j]}$. We know there exists $i+1 < k < i'$ such that $W^x_{i+1} W^x_k$ reduces to 1. Hence, $W^x_{[i+1,i']}$ is of the form **(M2b)**. It is shorter than $W^x_{[i,j]}$, therefore $M(i+1,i') = 1$. By the induction hypothesis, that string is found by the algorithm correctly, so $M(i,j) = 1$ (case (**A4b**)).
- $W^x_{i+1} = *, W^x_{j-1} = *$. Then the string $W^x_{[i,j]}$ can be of the form $p^{(m)} * \mathbf{U}\rangle\langle \mathbf{V} * q^{(n+1)}$ and then $M(i,j) = 1$ by the initial case of the description of the algorithm. If $W^x_{[i,j]}$ contains more brackets, then there is a string $W^x_{[k,k']}$, where k is the index of the first left angle bracket in the string $W^x_{[i,j]}$ and k' is the index of the last right angle bracket in the string $W^x_{[i,j]}$. $W^x_{[k,k']}$ is of the form **(M4)**. It is shorter than $W^x_{[i,j]}$, therefore, by the induction hypothesis, $M(k,k') = 1$. So, $M(i,j) = 1$ by (**A4c**).

4. $W^x_{[i,j]}$ is of the form **(M4)**. Let k be the index of the first simple term that participates in the reduction (that is the first simple term in the substring X_1, so it is obviously in between the first pair of matching angle brackets) and k' be the index of the last simple term that participates in the reduction (that is the last simple term in the substring X_g, so it is obviously in between the last pair of matching angle brackets). The substring $W^x_{[k,k']}$ is of the form **(M3)**, it is shorter than $W^x_{[i,j]}$, so by induction hypothesis $M(k,k') = 1$. Therefore $M(i,j) = 1$ by (**A5**).

□

Obviously, the reduction can be found exactly in the same way as in **Section 4.3**.

The algorithm is polynomial and it works in time proportional to n^3 and the size of the grammar, where n is the length of the string W_x, assuming the set $Pairs$ is determined. The procedure for computing the set $Pairs$ is polynomial, see [BLZ09].

5.4 Examples

As an example of use of letter promotions in natural language processing we can think of a self-reducing type of negation ν, such that $\nu\nu \Rightarrow 1$. Equivalently, one can add a letter promotion of the form $\nu \Rightarrow \nu^l$. In such a way a double negation law on the syntactic level can be expressed.

Example 5.1. We give an example of a formal language. Let us consider the language L over the alphabet $\Sigma = \{A, B\}$ consisting of the strings in which number of A's is equal to the number of B's. Let us assume the set $P = \{a, b\}$, $I(A) = \{a\}$, $I(B) = \{b\}, s = 1$ and the following set R of letter promotions:

$$a \Rightarrow b^l,$$
$$b \Rightarrow a^l.$$

To verify whether the given string is a member of the language L one can search for a reduction to 1 of the string of types in the pregroup grammar with letter promotions $G = (\Sigma, P, R, \leqslant, I)$.

Let us take a string $x = ABAABABBBA$. Obviously, there is only one possible type assignment for x. The string of assigned types has a few reductions to 1.

$$\begin{array}{l} A\ B\ A\ A\ B\ A\ B\ B\ B\ A \\ \underbrace{a\ \ b}\ a\ \ \underbrace{a\ b}\ \ \underbrace{a\ b}\ \ b\ \ \underbrace{b\ \ a}, \\ \underbrace{a\ \ \underbrace{b\ a}\ \ \underbrace{a\ b}\ \ \underbrace{a\ b}\ \ b}\ \ \underbrace{b\ a}, \\ \underbrace{a\ \ \underbrace{b\ a}\ \ \underbrace{a\ \underbrace{b\ \underbrace{a\ b}}}\ \ b}\ \ \underbrace{b\ \ a}. \end{array}$$

Example 5.2. Now we consider a second example of a formal language. We define a grammar whose language is A^nB^n. The alphabet is $\Sigma = \{A, B\}$, the set $P = \{a, b, s, t\}$, $I(A) = \{sat, sa\}$, $I(B) = \{b\}$ and there is the following set R of letter promotions:

$$a \Rightarrow b^l,$$
$$t \Rightarrow s^l.$$

To verify whether the given string is a member of the language L one can search for a reduction to s of the string of types in the pregroup grammar with letter promotions $G = (\Sigma, P, R, \leqslant, I)$.

Let us consider the string $AABB$. There are possible four type assignments, but only one of them can be reduced to the designated type s.

$$
\begin{array}{l}
A\ A\ B\ B \\
sat\ sat\ b\ b \\
s\ \underbrace{a\ t\ s\ a\ b}\ b \\
sa\ \ sat\ b\ b \\
sa\ \ s\ a\ b\ \ b
\end{array}
$$

Now consider a string not belonging to the language of G, e.g. $ABBA$ or $ABAB$. For each string there are possible four type assignments, none of which reducing to the designated type s.

$$
\begin{array}{l}
A\ B\ B\ A \\
sat\ b\ b\ sat \\
sat\ b\ b\ sa \\
s\ a\ b\ \ b\ sat \\
s\ a\ b\ \ b\ sa
\end{array}
$$

$$
\begin{array}{l}
A\ B\ A\ B \\
sat\ b\ sat\ b \\
sat\ b\ s\ a\ b \\
s\ a\ b\ \ sat\ b \\
s\ a\ b\ \ s\ a\ b
\end{array}
$$

Chapter 6

Pregroup Grammars with Letter Promotions with 1

Contents

In this chapter we consider **CBL** enriched with letter promotions and more general promotions: letter promotions with 1. First we describe the calculus and we prove a normalization theorem. In the second section we show that the letter promotion problem for **CBL** enriched with letter promotion with 1 is polynomial. Finally, we adjust our main algorithm to parsing pregroup grammars with letter promotions with 1.

6.1 CBL enriched with letter promotions with 1

Letter promotions with 1 are promotions allowing 1, that is letter promotions of the form: $p^{(m)} \Rightarrow 1$ or $1 \Rightarrow q^{(n)}$. We add 1 to the set of simple terms.

To simplify further notation and calculations we should notice that an assumption $p^{(m)} \Rightarrow 1$ is equivalent to the assumption $p \Rightarrow 1$ if m is even or to the assumption $1 \Rightarrow p$ if m is odd. Similarly, an assumption $1 \Rightarrow p^{(m)}$ is equivalent to the assumption $1 \Rightarrow p$ if m is even or to the assumption $p \Rightarrow 1$ if m is odd. It follows from pregroup laws (in pregroups $a^{(n)} \leqslant 1$ if, and only if, $a \leqslant 1$, for any even integer n and $a^{(n)} \leqslant 1$ if, and only if, $1 \leqslant a$, for any odd integer n and similarly for $1 \leqslant a^{(n)}$).

Therefore, in what follows, we consider only letter promotions with 1 of the form $p \Rightarrow 1$ and $1 \Rightarrow q$. We add 1 to the set of simple terms.

A complete system of **CBL** with letter promotions with 1 is obtained by adding two new rules to **CBL** with letter promotions (described in **Section 5.2**).

(PRO-C) $X, p^{(m)}, Y \Rightarrow X, Y$, if either m is even and $p \Rightarrow 1$ is an assumption, or m is odd and $1 \Rightarrow p$ is an assumption,

and

(PRO-E) $X, Y \Rightarrow X, q^{(n)}, Y$, if either n is even and $1 \Rightarrow q$ is an assumption, or n is odd and $q \Rightarrow 1$ is an assumption.

Consequently, (PRO-C) is a contracting promotion step, (PRO-E) is an expanding promotion step and (PRO) is a neutral promotion step.

The letter promotion problem for pregroups with letter promotions with 1 is as follows: given a finite set $R1$ of letter promotions, possibly containing letter promotions with 1, and terms t, u, verify whether $t \Rightarrow u$ in **CBL** enriched with all promotions from $R1$.

We write $t \Rightarrow_{R1} u$ if $t \Rightarrow u$ is an instance of (PRO), (PRO-C) or (PRO-E) restricted to the set of assumptions $R1$ (X, Y are empty). We write $t \Rightarrow^*_{R1} u$ if there exists terms $t_0, ..., t_k$ such that $k \geqslant 0,\ t_0 = t,\ t_k = u,\ t_{i-1} \Rightarrow_{R1} t_i$, for all $i = 1, ..., k$.

We introduce derivable rules of Generalized Contraction and Generalized Expansion for **CBL** with letter promotions with 1. They are generalization of (CON) and (EXP), respectively, derivable in **CBL** with assumptions from $R1$.

(GCON-R1) $X, p^{(m)}, q^{(n+1)}, Y \Rightarrow X, Y$, if $p^{(m)} \Rightarrow^*_{R1} q^{(n+1)}$

(GEXP-R1) $X, Y \Rightarrow X, p^{(n+1)}, q^{(m)}, Y$, if $p^{(n)} \Rightarrow^*_{R1} q^{(m)}$

Clearly, (CON) is a special instance of (GCON-R) and (EXP) is a special instance of (GEXP-R). We also treat any iteration of (PRO), (PRO-C) and (PRO-E) steps as a single step:

(PRO-R1) $X, t, Y \Rightarrow X, u, Y$ if $t \Rightarrow^*_{R1} u$ and $t \neq 1$ and $u \neq 1$.

Moreover, we define (PRO-C-R1) as a single step combining iterations of (PRO) with an iteration of (PRO-C)

(PRO-C-R1) $X, t, Y \Rightarrow X, Y$ if $t \Rightarrow^*_{R1} 1$ and $t \neq 1$,

and a (PRO-E-R1) as a single step combining an iteration of (PRO-E) with iterations of (PRO):

(PRO-E-R1) $X, Y \Rightarrow X, u, Y$ if $1 \Rightarrow^*_{R1} u$ and $u \neq 1$.

Theorem 6.1 (Normalization Theorem for **CBL** with Letter Promotions with 1)**.** *If $R1 \vdash_{\mathbf{CBL}} X \Rightarrow Y$, then there exist Z, U such that $X \Rightarrow Z$ by a finite number of instances of (GCON-R1) and (PRO-C-R1), $Z \Rightarrow U$ by a finite number of instances of (PRO-R1) and $U \Rightarrow Y$ by a finite number of instances of (GEXP-R1) and (PRO-E-R1).*

Proof. Let us start with some notions. A sequence $X_0, \ldots, X_k$ such that $X = X_0,\ Y = X_k$ and, for any $i = 1, \ldots, k,\ X_{i-1} \Rightarrow X_i$ is an instance of (GCON-R1), (GEXP-R1), (PRO-E-R1), (PRO-C-R1) or (PRO-R1) is called a *derivation of*

$X \Rightarrow Y$ *from the set of assumptions R1.* Clearly, $R1 \vdash X \Rightarrow Y$ iff there exists a derivation of $X \Rightarrow Y$ of this form. k is the length of the derivation. If a derivation has a form required by the Theorem, then it is called a *normal derivation.*

We prove that every derivation $X_0, \ldots, X_k$ of $X \Rightarrow Y$ can be transformed into a normal derivation of length not greater than k. We proceed by induction on k.

We should notice that for $k = 0$ and $k = 1$ the initial derivation is normal. For $k = 0$, it suffices to take $X = Z = U = Y$. For $k = 1$, if $X \Rightarrow Y$ is an instance of (GCON-R1) or (PRO-C-R1), one takes $Z = U = Y$, if $X \Rightarrow Y$ is an instance of (GEXP-R1) or (PRO-E-R1), one takes $X = Z = U$, and if $X \Rightarrow Y$ is an instance of (PRO-R1), one takes $X = Z$ and $U = Y$.

Assume now $k > 1$. The derivation $X_1, \ldots, X_k$ is shorter, whence it can be transformed into a normal derivation $Y_1, \ldots, Y_l$ such that $X_1 = Y_1$, $X_k = Y_l$ and $l \leqslant k$. If $l < k$, then $X_0, Y_1, \ldots, Y_l$ is a derivation of $X \Rightarrow Y$ of length less than k, whence it can be transformed into a normal derivation, by the induction hypothesis. So assume $l = k$.

Case 1. $X_0 \Rightarrow X_1$ is an instance of (GCON-R1). Then $X_0, Y_1, \ldots., Y_l$ is a normal derivation of $X \Rightarrow Y$ from $R1$.

Case 2. $X_0 \Rightarrow X_1$ is an instance of (PRO-C-R1). Then $X_0, Y_1, \ldots., Y_l$ is a normal derivation of $X \Rightarrow Y$ from $R1$.

Case 3. $X_0 \Rightarrow X_1$ is an instance of (GEXP-R1), assume $X_0 = UV$; $X_1 = Up^{(n+1)}q^{(m)}V$, and $p^{(n)} \Rightarrow^*_{R1} q^{(m)}$. We consider two subcases.

Case 3.1. Neither any (GCON-R1)-step nor any (PRO-C-R1)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrences of $p^{(n+1)}, q^{(m)}$. If also no (PRO-R1)-step of $Y_1, \ldots, Y_l$ acts on these designated terms, then we drop $p^{(n+1)}q^{(m)}$ from all types appearing in (GCON-R1)-steps, (PRO-C-R1)-steps and (PRO-R1)-steps of $Y_1, \ldots, Y_l$, then we introduce $p^{(n+1)}q^{(m)}$ by a single instance of (GEXP-R1), and continue the (GEXP-R1)-steps and (PRO-E-R1)-steps of $Y_1, \ldots, Y_l$; this yields a normal derivation of $X \Rightarrow Y$ of length k. Otherwise, let $Y_{i-1} \Rightarrow Y_i$ be the first (PRO-R1)-step of $Y_1, \ldots, Y_l$ which acts on $p^{(n+1)}$ or $q^{(m)}$.

(I) If $Y_{i-1} \Rightarrow Y_i$ acts on $p^{(n+1)}$, then there exists a term $r^{(m')}$ and types T, W such that $Y_{i-1} = Tp^{(n+1)}W$, $Y_i = Tr^{(m')}W$ and $p^{(n+1)} \Rightarrow^*_{R1} r^{(m')}$. Consequently, $r^{(m'-1)} \Rightarrow^*_{R1} p^{(n)}$, whence $r^{(m'-1)} \Rightarrow^*_{R1} q^{(m)}$. Then we can replace the derivation $X_0, Y_1, \ldots, Y_l$ by a shorter derivation: first apply (GEXP-R1) of the form $U, V \Rightarrow U, r^{(m')}, q^{(m)}, V$, then derive $Y_1, \ldots, Y_{i-1}$ in which $p^{(n+1)}$ is replaced by $r^{(m')}$, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. By the induction hypothesis, this derivation can be transformed into a normal derivation of length less than k.

(II) If $Y_{i-1} \Rightarrow Y_i$ acts on $q^{(m)}$, then there exist a term $r^{(m')}$ and types T, W such that $Y_{i-1} = Tq^{(m)}W$, $Y_i = Tr^{(m')}W$ and $q^{(m)} \Rightarrow^*_{R1} r^{(m')}$. Consequently, $p^{(n)} \Rightarrow^*_{R1} r^{(m')}$, and we can replace the derivation $X_0, Y_1, \ldots, Y_l$

by a shorter derivation: first apply (GEXP-R1) of the form $U, V \Rightarrow U, p^{(n+1)}, r^{(m')}, V$, then derive $Y_1, \ldots, Y_{i-1}$ in which $q^{(m)}$ is replaced by $r^{(m')}$, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. Again we apply the induction hypothesis.

Case 3.2. Some (GCON-R1)-step of $Y_1, \ldots, Y_l$ acts on (some of) the designated occurrences of $p^{(n+1)}, q^{(m)}$. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. There are three possibilities.

(I) This step acts on both $p^{(n+1)}$ and $q^{(m)}$. Then, the derivation $X_0, Y_1, \ldots, Y_l$ can be replaced by a shorter derivation: drop the first application of (GEXP-R1), then derive $Y_1, \ldots, Y_{i-1}$ in which $p^{(n+1)}q^{(m)}$ is omitted, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. We apply the induction hypothesis.

(II) This step acts on $p^{(n+1)}$ only. Then, $Y_{i-1} = Tr^{(m')}p^{(n+1)}q^{(m)}W$, $Y_i = Tq^{(m)}W$ and $r^{(m')} \Rightarrow^*_{R1} p^{(n)}$. The derivation $X_0, Y_1, \ldots, Y_l$ can be replaced by a shorter, normal derivation: drop the first application of (GEXP-R1), then derive $Y_1, \ldots, Y_{i-1}$ in which $p^{(n+1)}q^{(m)}$ is omitted, derive Y_i by a (PRO-R1)-step (notice $r^{(m')} \Rightarrow^*_{R1} q^{(m)}$), and continue $Y_{i+1}, \ldots, Y_l$.

(III) This step acts on $q^{(m)}$ only. Then, $Y_{i-1} = Tp^{(n+1)}q^{(m)}r^{(m'+1)}W$, $Y_i = Tp^{(n+1)}W$ and $q^{(m)} \Rightarrow^*_{R1} r^{(m')}$. The derivation $X_0, Y_1, \ldots, Y_l$ can be replaced by a shorter derivation: drop the first application of (GEXP-R1), then derive $Y_1, \ldots, Y_{i-1}$ in which $p^{(n+1)}q^{(m)}$ is omitted, derive Y_i by a (PRO-R1)-step (notice $r^{(m'+1)} \Rightarrow^*_{R1} p^{(n+1)}$), and continue $Y_{i+1}, \ldots, Y_l$. We apply the induction hypothesis.

Case 3.3. Some (PRO-C-R1)-step of $Y_1, \ldots, Y_l$ acts on (some of) the designated occurrences of $p^{(n+1)}, q^{(m)}$. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. There are two possibilities.

(I) This step acts on $p^{(n+1)}$. Then $Y_{i-1} = Tp^{(n+1)}W$, $Y_i = TW$ and $p^{(n+1)} \Rightarrow^*_{R1} 1$. Thus, $1 \Rightarrow^*_{R1} p^{(n)}$, and hence $1 \Rightarrow^*_{R1} q^{(m)}$. We can replace the derivation $X_0, Y1, \ldots, Y_l$ by a shorter derivation: start with an application of (PRO-E-R1) of the form $UV \Rightarrow Uq^{(m)}V$ derive $Y_1, \ldots, Y_{i-1}$ in which $p^{(n+1)}$ is replaced by 1, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. Again we apply the induction hypothesis.

(II) This step acts on $q^{(m)}$. Then, $Y_{i-1} = Tq^{(m)}W$, $Y_i = TW$ and $q^{(m)} \Rightarrow^*_{R1} 1$. Thus, $p^{(n)} \Rightarrow^*_{R1} 1$, and we can replace the derivation $X_0, Y_1, \ldots, Y_l$ by a shorter derivation: start with an application of (PRO-E-R1) of the form $U, V \Rightarrow U, p^{(n+1)}, V$, derive $Y_1, \ldots, Y_{i-1}$ in which $q^{(m)}$ is replaced by 1, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. Again we apply the induction hypothesis.

Case 4. $X_0 \Rightarrow X_1$ is an instance of (PRO-E-R1), assume $X_0 = UV$, $X_1 = Uq^{(m)}V$, and $1 \Rightarrow^*_{R1} q^{(m)}$. There are three subcases.

Case 4.1. Neither any (GCON-R1)-step nor any (PRO-C-R1) of $Y_1, \ldots, Y_l$ acts on the designated occurrence of $q^{(m)}$. If also no (PRO-R1)-step of $Y_1, \ldots, Y_l$ acts on this designated term, then we drop the first application of the (PRO-E-R1)-step, omit $q^{(m)}$ in all types appearing in (GCON-R1)-steps, (PRO-C-R1)-steps and (PRO-R1)-steps of $Y_1, \ldots, Y_l$, then introduce $q^{(m)}$ by a single instance of (PRO-E-R1), and continue with the (GEXP-R1)-steps of $Y_1, \ldots, Y_l$; this yields a normal derivation of $X \Rightarrow Y$ of length k.

Otherwise, let $Y_{i-1} \Rightarrow Y_i$ be the first (PRO-R1)-step of $Y_1, \ldots, Y_l$ which acts on $q^{(m)}$. Then, there exist a term $r^{(m')}$ and types T, W such that $Y_{i-1} = Tq^{(m)}W$, $Y_i = Tr^{(m')}W$ and $q^{(m)} \Rightarrow^*_{R1} r^{(m')}$. Thus $1 \Rightarrow^*_{R1} r^{(m')}$, and we can replace the derivation $X_0, Y_1, \ldots, Y_l$ by a shorter derivation: first apply (PRO-E-R1) of the form $UV \Rightarrow Ur^{(m')}V$, then derive $Y_1, \ldots, Y_{i-1}$ in which $q^{(m)}$ is replaced by $r^{(m')}$, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. Again we apply the induction hypothesis.

Case 4.2. Some (GCON-R1)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrence of $q^{(m)}$. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. Then, $Y_{i-1} = Tq^{(m)}r^{(m'+1)}W$, $Y_i = TW$ and $q^{(m)} \Rightarrow^*_{R1} r^{(m')}$. The derivation $X_0, Y_1, \ldots, Y_l$ can be replaced by a shorter derivation: drop the first application of (PRO-E-R1), then derive $Y_1, \ldots, Y_{i-1}$ in which $q^{(m)}$ is omitted, derive Y_i by a (PRO-C-R1)-step (notice $r^{(m'+1)} \Rightarrow^*_{R1} 1$), and continue $Y_{i+1}, \ldots, Y_l$. We apply the induction hypothesis.

Case 4.3. Some (PRO-C-R1)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrence of $q^{(m)}$. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. Then, $Y_{i-1} = Tq^{(m)}W$, $Y_i = TW$ and $q^{(m)} \Rightarrow^*_{R1} 1$. Then we can replace the derivation $X_0, Y_1, \ldots, Y_l$ by a shorter, normal derivation: drop the first application of the (PRO-E-R1)-step, derive $Y_1, \ldots, Y_{i-1}$ in which $q^{(m)}$ is omitted, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$.

Case 5. $X_0 \Rightarrow X_1$ is an instance of (PRO-R1), say $X_0 = UtV$, $X_1 = UuV$, $t \Rightarrow^*_{R1} u$, $u \neq 1$ and $t \neq 1$.

Case 5.1. Neither any (GCON-R1)-step nor any (PRO-C-R1)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrence of u. Then $X_0, Y_1, \ldots, Y_l$ can be transformed into a normal derivation of the length k: drop the first application of (PRO-R1), apply all (GCON-R1)-steps of $Y_1, \ldots, Y_l$ in which the designated occurrence of u is replaced by t and apply all (PRO-C-R1)-steps, then apply a (PRO-R1)-step which changes t into u, and continue the remaining steps of $Y_1, \ldots, Y_l$.

Case 5.2. Some (GCON-R1)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrence of u. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. There are two possibilities.

(I) $Y_{i-1} = Tuq^{(n+1)}W$, $Y_i = TW$ and $u \Rightarrow^*_{R1} q^{(n)}$. Since $t \Rightarrow^*_{R1} q^{(n)}$, then $X, Y_1, \ldots, Y_l$ can be transformed into a shorter derivation: drop the

first application of (PRO-R1), derive $Y_1, \ldots, Y_{i-1}$ in which the designated occurrence of u is replaced by t, derive Y_i by a (GCON-R1)-step of the form $T, t, q^{(n+1)}, W \Rightarrow T, W$, and continue $Y_{i+1}, \ldots, Y_l$. We apply the induction hypothesis.
(II) $u = q^{(n+1)}, Y_{i-1} = Tp^{(m)}uW$, $Y_i = TW$ and $p^{(m)} \Rightarrow^*_{R1} q^{(n)}$. Let $t = r(n')$. We have $q^{(n)} \Rightarrow^*_{R1} r^{(n'-1)}$, whence $p^{(m)} \Rightarrow^*_{R1} r^{(n'-1)}$. The derivation $X_0, Y_1, \ldots, Y_l$ can be transformed into a shorter derivation: drop the first application of (PRO-R1), derive $Y_1, \ldots, Y_{i-1}$ in which the designated occurrence of u is replaced by t, derive Y_i by a (GCON-R1)-step of the form $T, p^{(m)}, r^{(n')}, W \Rightarrow T, W$, and continue $Y_{i+1}, \ldots, Y_l$. We apply the induction hypothesis.

Case 5.3. Some (PRO-C)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrence of u. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. Then there exists types T, W, such that $Y_{i-1} = TuW$, $Y_i = TW$ and $u \Rightarrow^*_{R1} 1$. Thus $t \Rightarrow^*_{R1} 1$. The derivation $X_0, Y_1, \ldots, Y_l$ can be transformed into a normal derivation of length k: drop the first application of (PRO-R1), apply a (PRO-C)-step of the form: $TuW \Rightarrow^*_{R1} TW$ derive $Y_1, \ldots, Y_{i-1}$ in which the designated occurrence of u is omitted, drop Y_i and continue $Y_{i+1}, \ldots, Y_l$.

□

Consequently, there holds:

Corollary 6.1. *If $R1 \vdash_{\mathbf{CBL}} X \Rightarrow t$, where t is a simple term, then X can be reduced to t by (GCON-R1), (PRO-C-R1) and (PRO-R1) only.*

As as consequence of **Theorem 6.1**, the following lemma can be obtained (it is similar as for **LPPP**):

Lemma 6.1. *$R1 \vdash t \Rightarrow u$ if, and only if, $t \Rightarrow^*_{R1} u$.*

Proof. The "if" part is obvious. The "only if" parts depends on **Theorem 6.1**. Assume $R1 \vdash_{\mathbf{CBL}} t \Rightarrow u$. Then there exists a normal derivation of $t \Rightarrow u$ from the set of assumptions $R1$. The derivation cannot start with (GCON-R1), since (GCON-R1) cannot be applied to a single term. Therefore, in the normal derivation of $t \Rightarrow u$, (GCON-R1) cannot appear. Similarly, the derivation cannot end with (GEXP-R1), as it produces two terms. Hence, in the normal derivation of $t \Rightarrow u$, (GEXP-R1) cannot appear.

Consequently, the only steps that can be applied are (PRO-R1), (PRO-C-R1) or (PRO-E-R1), with empty X, Y. □

Therefore, our problem can be stated as follows.

Verify, whether, given t, u and the set of letter promotions with 1, $t \Rightarrow^*_{R1} u$.

6.2 Polynomial time decidability of CBL enriched with letter promotions with 1

We give an algorithm solving the letter promotion problem for pregroups with letter promotions with 1 in polynomial time. Let R be a set of all letter promotions from $R1$, in which 1 does not occur. First we define the same finite, directed weighted graph $G(R)$ as the graph for **LPPP**, see **Section 5.2**. Then we add conditions for promotions with 1.

From **Theorem 5.2** for pregroups with letter promotions there follows directly:

Lemma 6.2. *Let $p, q \in P$. If there exists a route from $p_{\pi(m)}$ to $q_{\pi(n)}$ of weight $n - m$ in $G(R)$, then $p^{(m)} \Rightarrow^*_{R1} q^{(n)}$.*

The following two lemmas define conditions, on how to treat letter promotions with 1.

Lemma 6.3. *Let $p \in P$. Then $p^{(m)} \Rightarrow^*_{R1} 1$ iff one of the conditions holds:*

(i) *there exists a vertex q_0 in the graph $G(R)$ such that $q \Rightarrow_{R1} 1$ and there is a route in $G(R)$ from $p_{\pi(m)}$ to q_0 of even weight if m is even or of odd weight if m is odd*

(ii) *there exists a vertex q_1 in the graph $G(R)$ such that $1 \Rightarrow_{R1} q$ and there is a route from $p_{\pi(m)}$ to q_1 of odd weight if m is even and of even weight if m is odd*

Proof. We prove the "if" part.

Assume (i). There is a route in the graph $G(R)$ from $p_{\pi(m)}$ to q_0 of weight k and there holds $q \Rightarrow_{R1} 1$. Then, by **Lemma 6.2** $p^{(m)} \Rightarrow^*_{R1} q^{(k+m)}$ and by assumption $k + m$ is even (either k and m are even, or both are odd). Since $q_o \Rightarrow_{R1} 1$, there holds $q^{(k+m)} \Rightarrow^*_{R1} 1$. Consequently, we have $p^{(m)} \Rightarrow^*_{R1} 1$.

Now assume (ii). There is a route in the graph $G(R)$ from $p_{\pi(m)}$ to q_1 of weight k and there holds $1 \Rightarrow_{R1} q$. Then, by **Lemma 6.2** $p^{(m)} \Rightarrow^*_{R1} q^{(k+m)}$ and by assumption $k + m$ is odd (either k is even and m is odd, or conversely k is odd and m is even). Since $1 \Rightarrow_{R1} q_1$, there holds $q^{(k+m)} \Rightarrow^*_{R1} 1$. Consequently, we have $p^{(m)} \Rightarrow^*_{R1} 1$.

Conversely, now assume there holds $p^{(m)} \Rightarrow^*_{R1} 1$. If 1 appears more than once in a derivation of $p^{(m)} \Rightarrow^*_{R1} 1$, then it suffices to consider only the derivation to the first occurrence of 1. If the length of the derivation is 1, then by assumption $p^{(m)} \Rightarrow_{R1} 1$ and there holds $p \Rightarrow_{R1} 1$ if m is even or $1 \Rightarrow_{R1} p$ if m is odd. Then we take $p = q$; and there exist trivial routes in $G(R)$ from p_0 to q_0 and from q_1 to p_1 of even weight. Thus both (i) and (ii) are satisfied.

If the length of the derivation is greater than one, then there exists q and an integer $n = k + m$, such that $p^{(m)} \Rightarrow^*_{R1} q^{(k+m)}$ and $q^{(k+m)} \Rightarrow^*_R 1$. By **Theorem 5.2** there exists a route in $G(R)$ from $p_{\pi(m)}$ to $q_{\pi(m+k)}$ of weight k. First assume that $k + m$ is even. Then $q_{\pi(k+m)} = q_0$ and either k and m are even, or k and m are

odd. Moreover, since $k+m$ is even, $q \Rightarrow_{R1} 1$. Thus (i) holds. Assume that $k+m$ is odd. Then $q_{\pi(k+m)} = q_1$ and either k is odd and m is even, or k is even and m is odd. Moreover, since $k+m$ is odd, $1 \Rightarrow_{R1} q$. Thus (ii) holds. □

Lemma 6.4. *Let $p \in P$. Then $1 \Rightarrow^*_{R1} q^{(n)}$ iff one of the conditions holds:*

(i) *there exists a vertex p_0 in the graph $G(R)$ such that $1 \Rightarrow_{R1} p$ and there is a route in $G(R)$ from p_0 to $q_{\pi(n)}$ of even weight if n is even or of odd weight if n is odd*

(ii) *there exists a vertex p_1 in the graph $G(R)$ such that $p \Rightarrow_{R1} 1$ and there is a route from p_1 to $q_{\pi(n)}$ of odd weight if n is even or of even weight if n is odd*

Proof. The proof is analogous to the previous proof. □

Theorem 6.2. *Let t, u be simple terms. Then, $t \Rightarrow^*_{R1} u$ if, and only if, one of the following conditions is satisfied:*

(i) $t = u$,

(ii) $t = 1$, $u = q^{(n)}$ *and one of the conditions of* ***Lemma 6.4*** *is satisfied,*

(iii) $t = p^{(m)}$, $u = 1$ *and one of the conditions of* ***Lemma 6.3*** *is satisfied,*

(iv) $t \neq 1$, $u \neq 1$, $t \Rightarrow^*_{R1} 1$, *and one of the conditions of* ***Lemma 6.3*** *holds and* $1 \Rightarrow^*_{R1} u$ *and one of the conditions of* ***Lemma 6.4*** *holds,*

(v) $t = p^{(m)}$, $u = q^{(n)}$, *and there exists a route in $G(R)$ from $p_{\pi(m)}$ to $q_{\pi(n)}$ of weight $n - m$.*

Proof. The proof follows from earlier theorems and lemmas. □

To find answer to our problem for pregroups with letter promotions with 1, that is to verify whether $R1 \vdash t \Rightarrow u$ we consider some cases. The answer is YES iff one of the conditions of **Theorem 6.2** is satisfied.

To describe procedures for verifying the conditions of the theorem we will need the following constructions.

We construct an **NFA** $A(G(R))$ in a similar way as $M(G)$ for $G = G(R)$ in **Section 5.2** except that we do not introduce new states $i_1, \ldots i_k$.

- $\Sigma_{A(G(R))} = \{+, -\}$,
- $Q_{A(G(R))} = V_{G(R)} \cup Auxiliary$ - the set of states, where $V_{G(R)}$ is the set of vertices of the graph $G(R)$ and $Auxiliary$ is the set of some auxiliary states,
- transitions are described as follows:
  If (v', n, w') is an arc in $G(R)$, $n > 0$, then v' is linked with w' by n transitions $v' \to s_1 \to \ldots \to s_n = w'$, all labeled by $+$. $s_1, \ldots, s_n$ are new states. Similarly, v' is linked with w' for $n < 0$, with the only difference that transitions are labeled with -. For $n = 0$ v' is linked with w' by two transitions $v' \to s \to w'$, one with label $+$, the other one labeled by -, where s is a new state.

In our problem we have to check whether the given weight is even or not. We construct a **DFA** *Even* checking if an input is of even or odd length. There are only two states: e (even), which is the initial state and a state o (odd). Transitions are simply defined: in the given state, for any input symbol, go to the opposite state.

Finally we construct a product automaton $P(G(R))$. It is defined as follows:

- $\Sigma = \{+, -\}$
- $Q = \{[q, e], [q, o] : q \in Q_{A(G(R))}\}$ - the set of states,
- the transition function δ is defined as follows - $\delta([q, e], +) = \{[q', o] : q' \in \delta_{A(G(R))}(q, +)\}$
  - $\delta([q, e], -) = \{[q', o] : q' \in \delta_{A(G(R))}(q, -)\}$
  - $\delta([q, o], +) = \{[q', e] : q' \in \delta_{A(G(R))}(q, +)\}$
  - $\delta([q, o], -) = \{[q', e] : q' \in \delta_{A(G(R))}(q, -)\}$

To find the answer to the letter promotion problem for pregroups with letter promotions with 1 we have to check whether any of the conditions of **Theorem 6.2** is satisfied.

The condition **(i)** is trivial. If $t = u$, then $t \Rightarrow^*_{R1} u$.

Now, let us consider the condition **(ii)**. So, we assume that $t = 1$ and $u = q^{(n)}$. We have to verify conditions of **Lemma 6.4**. First we determine from the set $R1$ the set P_1 of all basic types p such that $1 \Rightarrow p$. Then we construct the automaton $P(G(R))$. For each $p \in P_1$ we run the automaton $P(G(R))$ assuming that the initial state of $P(G(R))$ is $[p_0, e]$ and the final state is either $[q_{\pi(n)}, e]$, if n is even or $[q_{\pi(n)}, o]$ if n is odd. If $L(P(G(R))) \neq \emptyset$ in any run of $P(G(R))$, then $1 \Rightarrow^*_{R1} q^{(n)}$. Otherwise we verify the second condition of the lemma.

We determine the set P_2 of all basic types p such that $p \Rightarrow 1$ in $R1$. For each $p \in P_2$ we run the automaton $P(G(R))$ assuming that the initial state of $(P(G(R))$ is $[p_1, e]$ and the final state is either $[q_{\pi(n)}, o]$, if n is even or $[q_{\pi(n)}, e]$, if n is odd. If $L(P(G(R))) \neq \emptyset$ in any run of $P(G(R))$, then $1 \Rightarrow^*_{R1} q^{(n)}$. Otherwise, we have to check further conditions of **Theorem 6.2**.

The condition **(iii)** is checked in a similar way, using **Lemma 6.3**. The condition **(iv)** consists of verifying whether **(iii)** holds for $p^{(m)}$ and 1 and whether **(ii)** holds for 1 and $q^{(n)}$. If we obtain $p^{(m)} \Rightarrow^*_{R1} 1$ and $1 \Rightarrow^*_{R1} q^{(n)}$, then $p^{(m)} \Rightarrow^*_{R1} q^{(n)}$. Finally, the condition **(v)** is checked using **Algorithm 5.1**.

If we obtain the answer $t \Rightarrow^*_{R1} u$, then the answer to our problem is YES, otherwise, the answer is NO, which means that it is not true that $t \Rightarrow^*_{R1} u$.

The size of a graph $G(R)$ is defined as the sum of the following numbers: the number of vertices, the number of arcs, and the sum of absolute values of weights of arcs. The construction o $G(R)$ can be performed in linear time. The time of the construction of $A(G(R))$ is $O(n^2)$, where n is the size of $G(R)$. The time of the construction of the automaton *Even* is a small constant. The time of the construction of the final automaton is thus also $O(n^2)$. The solution to the emptiness problem for regular language is solvable in time. Consequently, the following theorem holds.

Theorem 6.3. *The letter promotion problem for pregroups with letter promotions with 1 can be solved in polynomial time.*

6.3 A dynamic parsing algorithm for pregroup grammars with letter promotions with 1

A *pregroup grammar with letter promotions with 1* is a tuple $G = (\Sigma, P, R1, s, I)$, where $R1$ is the set of assumptions (letter promotions), such that $P(R1) \subseteq P$. Σ, P, s, I are defined in the same way as for pregroup grammars. We define now a polynomial, dynamic, parsing algorithm for pregroup grammars with letter promotions with 1 on the basis of the algorithm for pregroup grammars described in **Section 4.3**.

Our goal is to check whether the given string $x = a_1, \ldots, a_n$, $a_i \in \Sigma$, $i = 1, \ldots, n$ is a member of the language generated by a pregroup grammar with letter promotions with 1 G, that is whether $x \in L(G)$. If this is the case, we want to obtain the appropriate derivation.

We fix a pregroup grammar with letter promotions with 1 $G = (\Sigma, P, R1, s, I)$. We take a string of words $x \in \Sigma^+$ such that $x = a_1 ... a_n$. We use special symbols $*, \langle, \rangle$. We assume the same notation as in **Section 4.3**. The definition of the function M is the same as in **Section 4.3**. Obviously, by a reduction to 1 here we mean a reduction in **CBL** with letter promotions with 1.

Let $M(i,j), 1 \leqslant i \leqslant j \leqslant |W^x|$ be a function such that $M(i,j) = 1$ iff one of the following conditions holds:

- **M1.** $W^x_{[i,j]} \in T^+$ and it reduces to 1.
- **M2a.** $W^x_{[i,j]}$ is of the form $\langle ... \rangle ... \langle \mathbf{V} * Z$, where:
  - $Z \in T^+$
  - $\mathbf{V}$ contains no angle brackets
  - in $W^x_{[i,j]}$ there are precisely g ($g \geqslant 0$) pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $X_1 ... X_g Z$ reduces to 1
- **M2b.** $W^x_{[i,j]}$ is of the form $Y * \mathbf{U} \rangle ... \langle ... \rangle$, where:
  - $Y \in T^+$
  - $\mathbf{U}$ contains no angle brackets
  - in $W^x_{[i,j]}$ there are precisely g ($g \geqslant 0$) pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $Y X_1 ... X_g$ reduces to 1
- **M3.** $W^x_{[i,j]}$ is of the form $Y * \mathbf{U} \rangle ... \langle \mathbf{V} * Z$, where:
  - $Y, Z \in T^+$
  - $\mathbf{U}, \mathbf{V}$ contain no angle brackets
  - in $W^x_{[i,j]}$ there are precisely g ($g \geqslant 0$) pairs of matching angle brackets; for

the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $YX_1...X_gZ$ reduces to 1

- **M4.** $W^x_{[i,j]}$ is of the form $\langle ...\rangle ...\langle ...\rangle$, where:
  - in $W^x_{[i,j]}$ there are precisely g ($g \geqslant 1$) pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $X_1...X_g$ reduces to 1

In all other cases $M(i,j) = 0$.

Clearly, as before, the whole string W^x is of the form **M2a**. Therefore, $M(1,|W^x|) = 1$ entails the existence of a string $X_1 \dots X_n s^r$ reducing to 1. Each X_i is the searched type for a_i. Thus, a solution to the recognition problem is found, i.e. $x \in L(G)$. On the other hand, if $M(1,|W^x|) = 0$, then there is no string reducing to 1, in which exactly one element comes from each pair of angle brackets and which reduces to 1. It means $x \notin L(G)$.

We start the algorithm by determining the set $Pairs$ of all pairs $(p^{(m)}, q^{(n+1)})$ such that $p, q \in P$ and $p^{(m)} \Rightarrow^*_{R1} q^{(n)}$, and a set $Reducible$ of terms t such that $t \Rightarrow^*_{R1} 1$, which can be done in polynomial time, see **Section 5.3**.

We compute $M(i,j)$ dynamically. There are three initial cases. The first one computes $M(i,i) = 1$ in case if $W^x_i = t$ and $t \in Reducible$. Secondly, one looks for two adjacent simple terms W^x_i and W^x_{i+j} belonging to the set Pairs. If $(W^x_i, W^x_{i+j}) \in Pairs$ then we put $M(i,i+1) = 1$.

The last initial case is when $W^x_{[i,j]}$ is of the form $p^{(m)} * \mathbf{U}\rangle\langle \mathbf{V} * q^{(n+1)}$, $(p^m, q^{(n+1)}) \in Pairs$ and strings $\mathbf{U}$, $\mathbf{V}$ contain no angle brackets. Then, we put $M(i,j) = 1$.

When we already know $M(g,h)$, for all $1 \leqslant g < h \leqslant |W^x|$ such that $h-g < j-i$, we can compute $M(i,j)$. There are several cases:

- **A1a.** $W^x_i, W^x_j \in T$. If there exists k such that $i \leqslant k < j$, $W^x_k \in T$, $W^x_{(k+1)} \in T$ and both $M(i,k)$ and $M(k+1,j)$ are equal to 1, then we put $M(i,j) = 1$.

- **A1a'.** $W^x_i, W^x_j \in T$. If there exists k such that $i < k \leqslant j$, $W^x_k \in T$, $W^x_{(k+1)} \in T$ and both $M(i,k-1)$ and $M(k,j)$ are equal to 1, then we put $M(i,j) = 1$.

- **A1b.** $W^x_i, W^x_j \in T$. If there exists k such that $i < k < j-1$, $W^x_k = \rangle$, $W^x_{(k+1)} = \langle$ and both $M(i,k)$ and $M(k+1,j)$ are equal to 1, then we put $M(i,j) = 1$.

- **A2.** $W^x_i = p^{(m)}$, $W^x_j = q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$.
  If $M(i+1,j-1) = 1$, then $M(i,j) = 1$.

- **A3a.** $W^x_{[i,j]}$ is of the form $\langle ...\rangle ...\langle ...p^{(m)}$, $p \in P, m \in \mathcal{Z}$. If there exists k such that $i < k < j$, $W^x_k = *, W^x_{[i+1,k]}$ contains no angle brackets and $M(k+1,j) = 1$, then $M(i,j) = 1$.

- **A3b.** $W^x_{[i,j]}$ is of the form $p^{(m)}...\rangle...\langle...\rangle$, $p \in P, m \in \mathcal{Z}$. If there exists k such that $i < k < j$, $W^x_k = *, W^x_{[k,j-1]}$ contains no angle brackets and $M(i, k-1) = 1$, then we put $M(i, j) = 1$.

- **A4a.** $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle...\langle...q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$. If $M(k, j-1) = 1$, where k is the position of the first left angle bracket in the string $W^x_{[i,j]}$, then we put $M(i, j) = 1$.

- **A4b.** $W^x_{[i,j]}$ is of the form $p^{(m)}...\rangle...\langle... * q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$. If $M(i+1, k) = 1$, where k is the position of the last right angle bracket in the string $W^x_{[i,j]}$, then $M(i, j) = 1$.

- **A4c.** $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle\langle...*q^{(n+1)}$, where the string "..." in between the angle brackets is not empty and $(p^{(m)}, q^{(n+1)}) \in Pairs$. If $M(k, k') = 1$, where k is the position of the first left angle bracket in the string $W^x_{[i,j]}$ and k' is the position of the last right angle bracket in the string $W^x_{[i,j]}$, then $M(i, j) = 1$.

- **A4d.** $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle...\langle...$, and $p^{(m)} \in Reducible$. If $M(k, j) = 1$, where k is the position of the first left angle bracket in the string $W^x_{[i,j]}$, then we put $M(i, j) = 1$..

- **A4e.** $W^x_{[i,j]}$ is of the form $...\rangle...\langle... * q^{(n)}$, and $q^{(n)} \in Reducible$. If $M(i, k) = 1$, where k is the position of the last right angle bracket in the string $W^x_{[i,j]}$, then $M(i, j) = 1$.

- **A5.** $W^x_{[i,j]}$ is of the form $\langle...\rangle...\langle...\rangle$. If $M(k, k') = 1$, where W^x_k is a simple term in between the first pair of angle brackets, $W^x_{k'}$ is a simple term in between last pair of angle brackets in the string $W^x_{[i,j]}$ and $W^x_{k-1} = *$ and $W^x_{k'+1} = *$, then $M(i, j) = 1$.

- **A6a.** $W^x_{[i,j]}$ is of the form $p^{(m)}q^{(n)}...$, and $p^{(m)} \in Reducible$. If $M(i+1, j) = 1$, then we put $M(i, j) = 1$..

- **A6b.** $W^x_{[i,j]}$ is of the form $...p^{(m)}q^{(n)}$, and $q^{(n)} \in Reducible$. If $M(i, j-1) = 1$, then we put $M(i, j) = 1$.

In all other cases $M(i, j) = 0$.

We claim:

Theorem 6.4. *The algorithm computes $M(i, j)$ correctly.*

Proof. We will show at first that, if the algorithm computes $M(i, j) = 1$, then $M(i, j) = 1$ according to the definition of M. We will prove it by induction on the length of the string.

For strings of length one, $M(i, j) = 1$ only in case when $W^x_i = p^{(m)}$ and $p^{(m)} \in Reducible$. $W^x_{[i,i]}$ is then of the form **(M1)**, since $W^x_{[i,i]} \in T^+$ and the string $W^x_{[i,i]}$ reduces to 1. Hence, $M(i, i) = 1$ according to the definition of M.

Consider now the strings of length two. The algorithm computes $M(i,i+1)=1$ only in case when $W_i^x = p^{(m)}$ and $W_{i+1}^x = q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$. $W_{[i,j]}^x$ is then of the form **(M1)**, since $W_{[i,i+1]}^x \in T^+$ and the string $W_{[i,i+1]}^x$ reduces to 1. Hence, $M(i,i+1)=1$ according to the definition of M.

The other initial case when the algorithm computes $M(i,j)=1$ is when $W_{[i,j]}^x$ is of the form $p^{(m)} * \mathbf{U}\rangle\langle \mathbf{V} * q^{(n+1)}$, $(p^{(m)}, q^{(n+1)}) \in Pairs$ and the strings $\mathbf{U}$ and $\mathbf{V}$ contain no angle brackets. $W_{[i,j]}^x$ is then of the form **(M3)**, since we can assume $X = p^{(m)}$, $Y = q^{(n+1)}$ and $g = 0$. So, XY reduces to 1. Hence, $M(i,j)=1$ according to the definition of M.

Now let us consider the recursive cases when the algorithm computes $M(i,j)=1$ (all cases of the description of the algorithm). We consider only the cases that differ from the proof presented in **Section 5.3**.

A4d. . $W_{[i,j]}^x$ is of the form $W_{[i,j]}^x = \underset{i}{p^{(m)}} * \underbrace{...*\rangle\langle...}_{no\ \langle,\rangle} \underbrace{\underset{k}{\_}...\underset{j}{\_}}_{M(k,j)=1}$,

where $p^{(m)} \in Reducible$ and k is the position of the first left angle bracket in the string $W_{[i,j]}^x$. $W_{[k,j]}^x$ is shorter than $W_{[i,j]}^x$. Hence, by the induction hypothesis, $M(k,j)=1$ according to the definition of M. The string $W_{[k,j]}^x$ must be then of the form:

- **(M2a)**. Then $W_{[k,j]}^x = \langle...\rangle...\langle \mathbf{V} * Z$, where Z is the string of simple terms, $\mathbf{V}$ contains no angle brackets and there are precisely g $(g \geqslant 0)$ pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1...X_gZ$ reduces to 1. Let $Y = p^{(m)}$. Then $YX_1...X_gZ$ also reduces to 1, and the string $W_{[i,j]}^x$ is therefore of the form **(M3)**. Then $M(i,j)=1$ in accordance with the definition of M.
- **(M4)**. Then $W_{[k,j]}^x = \langle...\rangle...\langle...\rangle$ and there are precisely g $(g > 0)$ pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1...X_g$ reduces to 1. Let $Y = p^{(m)}$. Then $YX_1...X_g$ also reduces to 1, and the string $W_{[i,j]}^x$ is therefore of the form **(M2b)**. Then $M(i,j)=1$ in accordance with the definition of M.

A4e. $W_{[i,j]}^x$ is of the form $W_{[i,j]}^x = \underbrace{\underset{i}{\_}...\underset{k}{\_}}_{M(i,k)=1} \underbrace{\rangle\langle *... * q^{(n)}}_{no\ \langle,\rangle}\ \underset{j}{}$,

where $q^{(n)} \in Reducible$ and k is the position of the last right angle bracket in the string $W_{[i,j]}^x$. $W_{[i,k]}^x$ is shorter than $W_{[i,j]}^x$. Hence, by the induction hypothesis, $M(i,k)=1$ according to the definition of M. The string $W_{[i,k]}^x$ can be therefore of the form:

- **(M2b)**. Then $W_{[i,k]}^x = Y * \mathbf{U}\rangle...\langle...\rangle$, where Y is the string of simple terms, $\mathbf{U}$ contains no angle brackets and there are precisely g $(g \geqslant 0)$ pairs of

matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $YX_1...X_g$ reduces to 1. Let $Z = q^{(n)}$. Then $YX_1...X_gZ$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M3)**. Then $M(i,j) = 1$ in accordance with the definition of M.

- **(M4)**. Then $W^x_{[i,k]} = \langle ...\rangle ...\langle ...\rangle$ and there are precisely g ($g > 0$) pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1...X_g$ reduces to 1. Let $Z = q^{(n)}$. Then $X_1...X_gZ$ also reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M2a)**. Then $M(i,j) = 1$ in accordance with the definition of M.

A6a. . $W^x_{[i,j]}$ is of the form $W^x_{[i,j]} = \underset{i}{p^{(m)}}\ \underbrace{\underset{i+1}{q^{(n)}} \ldots \underset{j}{\_}}_{M(i+1,j)=1}$,

where $p^{(m)} \in$ *Reducible*. $W^x_{[i+1,j]}$ is shorter than $W^x_{[i,j]}$. Hence, by the induction hypothesis, $M(i+1,j) = 1$ according to the definition of M. The string $W^x_{[i+1,j]}$ can be then of the form:

- **(M1)**. Then $W^x_{[i+1,j]} \in T^+$ and $W^x_{[i+1,j]}$ reduces to 1. Then $W^x_{[i,j]}$ is also of the form **(M4)** and it reduces to 1. Thus, $M(i,j) = 1$ in accordance with the definition of M..
- **(M2b)**. Then $W^x_{[i+1,j]} = Y*\mathbf{U}\rangle ...\langle ...\rangle$, where Y is the string of simple terms, $\mathbf{U}$ contains no angle brackets and there are precisely g ($g \geqslant 0$) pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $YX_1...X_g$ reduces to 1. Let $Y' = p^{(m)}Y$. Then $Y'X_1...X_g$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore also of the form **(M2b)**. Then $M(i,j) = 1$ in accordance with the definition of M.
- **(M3)**. Then $W^x_{[i+1,j]} = Y*\mathbf{U}\rangle ...\langle ...\rangle \mathbf{V}*Z$, where Y, Z are the strings of simple terms, $\mathbf{U}, \mathbf{V}$ contain no angle brackets and there are precisely g ($g \geqslant 0$) pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $YX_1...X_gZ$ reduces to 1. Let $Y' = p^{(m)}Y$. Then $Y'X_1...X_gZ$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore also of the form **(M3)**. Then $M(i,j) = 1$ in accordance with the definition of M.

A6b. $W^x_{[i,j]}$ is of the form $W^x_{[i,j]} = \underbrace{\underset{i}{\_} \ldots \underset{j-1}{p^{(m)}}}_{M(i,j-1)=1} \underset{j}{q^{(n)}}$,

where $q^{(n)} \in$ *Reducible*. $W^x_{[i,j-1]}$ is shorter than $W^x_{[i,j]}$. Hence, by the induction hypothesis, $M(i,j-1) = 1$ according to the definition of M. The string $W^x_{[i,j-1]}$ can be therefore of the form:

- **(M1)**. Then $W^x_{[i,j-1]} \in T^+$ and $W^x_{[i,j-1]}$ reduces to 1. Then $W^x_{[i,j]}$ is also of the form **(M4)** and it reduces to 1. Thus, $M(i,j) = 1$ in accordance with the definition of M..
- **(M2a)**. Then $W^x_{[i,j-1]} = \langle ... \rangle ... \langle \mathbf{V} * Z$, where Z is the string of simple terms, $\mathbf{V}$ contains no angle brackets and there are precisely g ($g \geqslant 0$) pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1...X_gZ$ reduces to 1. Let $Z' = Zq^{(n)}$. Then $X_1...X_gZ'$ also reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M2a)**. Then $M(i,j) = 1$ in accordance with the definition of M.
- **(M3)**. Then $W^x_{[i,j-1]} = Y * \mathbf{U} \rangle ... \langle ... \rangle \mathbf{V} * Z$, where Y, Z are the strings of simple terms, $\mathbf{U}, \mathbf{V}$ contain no angle brackets and there are precisely g ($g \geqslant 0$) pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $YX_1...X_gZ$ reduces to 1. Let $Z' = Zq^{(n)}$. Then $YX_1...X_gZ'$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore also of the form **(M3)**. Then $M(i,j) = 1$ in accordance with the definition of M.

We will prove that the algorithm finds correctly all substrings for which the function $M(i,j) = 1$ by induction on the length of the substring. Let us notice that there are no such substrings that contain asterisk but no angle brackets.

The only strings of length one for which $M(i,j) = 1$ is of the form $p^{(m)}$, where $p^{(m)} \in T$ and $p^{(m)} \in \mathit{Reducible}$ and the algorithm finds them correctly (the string is of the form **M1**). The only strings of length two, for which $M(i,j) = 1$, are of the form $p^{(m)}q^{(n+1)}$, where $(p^{(m)}, q^{(n+1)}) \in \mathit{Pairs}$ (that is of the form **(M1)**), and the algorithm finds them correctly.

Let us consider now the substrings of the length $l > 2$ such that for all $l' < l$ the algorithm finds the substrings of the length l' correctly (all forms of the definition of M). Again we consider only the case that differs from the proof in **Section 5.3**.

3. $W^x_{[i,j]}$ is of the form **(M3)**. $W^x_i W^x_j$ takes part in the reduction to 1 in $W^x_{[i,j]}$.

First, let us assume $W^x_i W^x_{i'}$ reduce to 1 where $i' \neq j$. Then $W^x_{i'+1}$ can be:

- simple term. Then $W^x_{[i,i']}$ and $W^x_{[i'+1,j]}$ are of the form **(M1)** or **(M3)**, they are shorter and both $M(i,i') = 1$ and $M(i'+1,j) = 1$. Hence, by the induction hypothesis, these strings are found by the algorithm correctly, so $M(i,j) = 1$ (case (**A1a**)).
- asterisk. Then $W^x_{[i,i']}$ is of the form **(M1)** or **(M3)**, it is shorter and $M(i,i') = 1$. Hence, by the induction hypothesis, the string is found by the algorithm correctly. Let k be the position of the first right angle bracket following i'. $W^x_{[i,k]}$ is shorter than $W^x_{[i,j]}$ and it is of the form **(M2b)**. So, by the induction hypothesis $M(i,k) = 1$ (case (**A3b**)). Similarly, the substring $W^x_{[k+1,j]}$ is of the form **(M2a)**, it is shorter than

$W^x_{[i,j]}$. So, by the induction hypothesis $M(k+1,j) = 1$ (case (**A3a**)). Hence, $M(i,j) = 1$, by the induction hypothesis (case (**A1b**)).

Otherwise, let us assume $W^x_i \in \mathit{Reducible}$. Then W^x_{i+1} can be:

- simple term. Then $W^x_{[i+1,j]}$ is of the form **(M3)**, it is shorter, so by the induction hypothesis $M(i+1,j) = 1$. Moreover, $W^x_{[i,i]}$ is of the form **M1**, it is shorter and $M(i,i) = 1$. Hence, by the induction hypothesis, these strings are found by the algorithm correctly. Then $M(i,j) = 1$ (case (**A1a**)).
- asterisk. Let k be the position of the first right angle bracket following i. $W^x_{[i,k]}$ is shorter than $W^x_{[i,j]}$ and it is of the form **(M2b)**. So, by the induction hypothesis $M(i,k) = 1$. Similarly, the substring $W^x_{[k+1,j]}$ is of the form **(M2a)**, it is shorter than $W^x_{[i,j]}$. So, by the induction hypothesis $M(k+1,j) = 1$. Hence, $M(i,j) = 1$, by the induction hypothesis (case (**A1b**)).

Let us assume $i' = j$ so $W^x_i W^x_j$ reduces to 1. There are the following cases.

- W^x_{i+1}, W^x_{j-1} are simple terms. Then the substring $W^x_{[i+1,j-1]}$ is of the form **(M3)**. It is shorter than $W^x_{[i,j]}$, therefore $M(i+1,j-1) = 1$, as by the induction hypothesis, that string is found by the algorithm correctly. Then $M(i,j) = 1$ (case (**A2**)).
- $W^x_{i+1} = *, W^x_{j-1} \in T$. So $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle ...\langle ...q^{(n+1)}$. Let i' be the position of the first left angle bracket in $W^x_{[i,j]}$. There exists $i' < k \leqslant j-1$ such that $W^x_k W^x_{j-1}$ reduces to, or if $k = j-1$, then $W^x_{j-1} \in \mathit{Reducible}$. Hence, $W^x_{[i',j-1]}$ is of the form **(M2a)**. It is shorter than $W^x_{[i,j]}$, therefore $M(i',j-1) = 1$, as by the induction hypothesis, that string is found by the algorithm correctly. Then $M(i,j) = 1$ (case (**A4a**)).
- $W^x_{i+1} \in T$, $W^x_{j-1} = *$. It is proved dually to the case $W^x_{i+1} = *, W^x_{j-1} \in T$.
- $W^x_{i+1} = *, W^x_{j-1} = *$ - proved as in **Section 5.2**.

□

Obviously the reduction can be found exactly in the same way as in **Section 4.3**.

The algorithm is polynomial and works in time proportional to n^3, where n is the length of the string W_x, assuming the set *Pairs* and *Reducible* are determined. The procedures for computing the set *Pairs* and the set *Reducible* are polynomial, see **Section 6.1**.

Chapter 7

Java Application of the Dynamic Parsing Algorithm for Pregroup Grammars

Contents

In this chapter we present a tool constructed to show application of the dynamic parsing algorithm for pregroup grammars described in **Chapter 4**. It is a Java application giving answer to the recognition problem for pregroup grammars, and if it is positive, one of possible parsings of a given string of words.

We have constructed two sample lexicons for English but obviously the algorithm can be run on dictionaries for any language.

7.1 Use of the application

The tool is a JSwing application. It permits to give one or more sentences to parse and offers a few choices.

First of all one can choose a grammar to apply. By grammar we mean here a pregroup grammar defined in a text file as a list of atoms, partial order dependencies and a lexicon. We propose two grammars for English. The main one is based on [Lam08] and consists of about 200 words. The dictionary is extended to work on our own examples too. Another grammar is a very tiny one which was built to illustrate examples of Oehrle. User can also choose his or her own grammar. It is crucial that the grammar file is of a given scheme. The file consists of three main parts separated by %. Each entry in every part must be put in a new line.

- A list of atoms.
- Partial order: for each inequality of the form $a \leqslant b$ we put $a\ b$.

- Lexicon, each entry starts with a lexical unit followed by all types assigned to it; types are separated by commas and terms within a given type are separated by whitespace; our lexicon is listed in **Appendix A.3**.

There is a button in the application showing the grammar file. **Figure 7.1** shows a screen shot of such a file for a tiny grammar for Oehrle's examples. One can notice there is no definition of partial order as Oehrle does not consider poset rules.

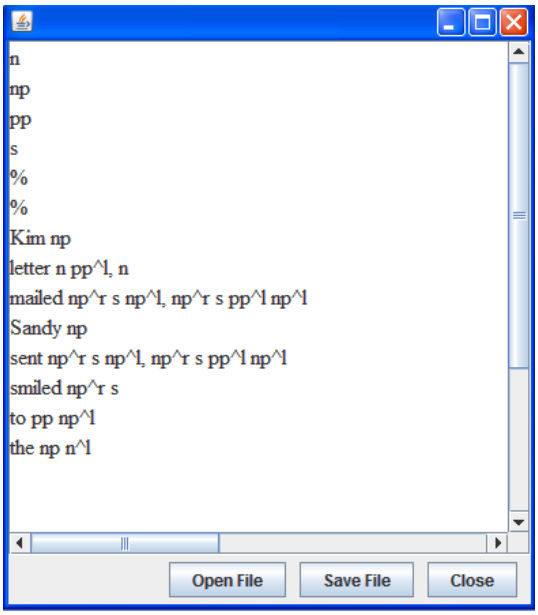


Figure 7.1: Grammar File.

A sentence or a text to parse can be entered directly in an appropriate window or loaded from user's file or any of a few files defined as examples.

Finally one can choose whether the tool is to give as an output the first or the last computed reduction (provided one exists), as described below.

An output is given in two text areas. In the first one a string W^x for each sentence is shown whereas the second one contains simple terms taking part in the computed reduction and an appropriate set of links.

Figure 7.2 shows an interface of the tool.

7.2 Classes

We use the following classes.

- Algorithm - computes values of the function M and $Prev$ and if possible determines a reduction.

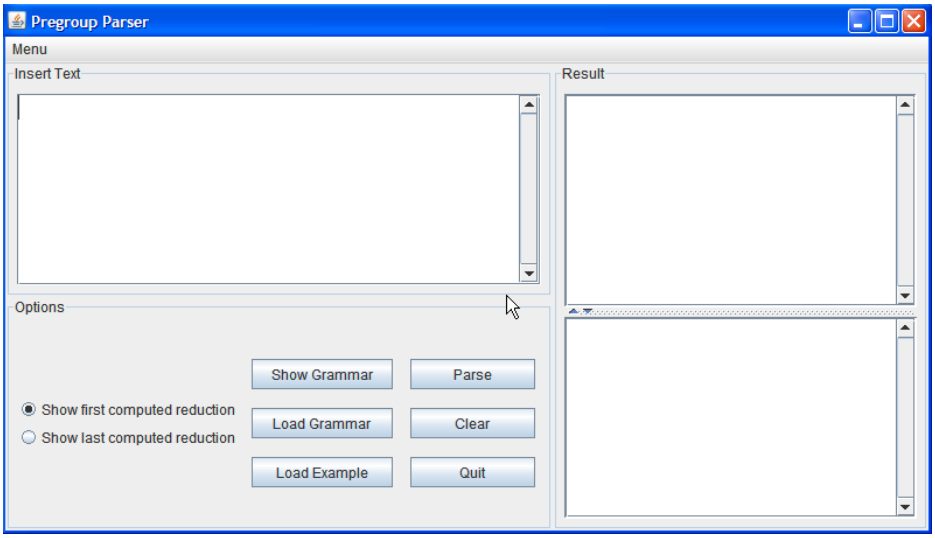


Figure 7.2: Java Pregroup Parser.

- Parser - main class of the application, for a chosen grammar it creates appropriate *Pregroup* and launches *Algorithm* on a given string (or strings) of words and the *Pregroup*.
- Pregroup - depends on the pregroup grammar defined in a given text file, remembers a list of atoms, determines the partial order on atoms and forms a lexicon out of list of words and types assigned to them.
- Term - constituent of *WString*. It is one of simple terms of a given pregroup with exponent and its position in the String W^x as attributes or an auxiliary symbol ("*", "$\langle$" or "$\rangle$") with its position in the String W^x as an attribute.
- WString - constructs representation of a string W^x for a given string x of words using data from *Pregroup*; the string W^x is represented by a list of *Terms*.

7.3 Parsing a sentence

When parsing is chosen the input is divided into strings that may be sentences. We assume sentences are separated by punctuation marks: ".", "*!*" and "*?*". Before parsing begins a pregroup grammar from a chosen file is constructed. Initialization of the *Pregroup* consists of creating a list of atoms, determining partial order from a list given in the grammar file and loading a lexicon.

For each sentence at first a *WString* is constructed. It consists of appropriate *Terms*. We recall that a string W^x consists of all types assigned to each word separated by asterisks.

Algorithm 7.1 Fragment of a procedure of computing function M and $Prev$ (both initial cases and cases (A1a), (A1b) are given)

Require: $W^x, startInd, endInd, first$
  $start \leftarrow$ term at $startInd$
  $end \leftarrow$ term at $endInd$
  if $endInd - startInd < 2$ **then**
    if $start$ and end reduce to 1 **then**
      $M[startInd][endInd] \leftarrow 1$
      if (**not** $first$) **or** ($first$ **and** $Prev[startInd][endInd] = null$) **then**
        $Prev[startInd][endInd] \leftarrow ((startInd, endInd), (0,0), (0,0))$
      end if
    end if//initial case 1
  end if
  call $computeMP$ on $w, startInd + 1, endInd, first$
  if $start$ and end are atoms **then**
    for $i = startInd + 1$ to $endInd$ **do**
      if term at W_i^x is atom **and** term at W_{i+1}^x is atom **and** $M[startInd][i] = 1$ **and** $M[i+1][endInd] = 1$ **then**
        $M[startInd][endInd] \leftarrow 1$
        if (**not** $first$) **or** ($first$ **and** $Prev[startInd][endInd]$ is null) **then**
          $Prev[startInd][endInd] \leftarrow ((0,0), (startInd, i), ((i+1), endInd))$ //case (A1a)
        else if term at W_i^x is $\rangle$ **and** term at W_{i+1}^x is $\langle$ **and** $M[startInd][i] = 1$ **and** $M[i+1][endInd] = 1$ **then**
          $M[startInd][endInd] = 1$
          if (**not** $first$) **or** ($first$ **and** $Prev[startInd][endInd]$ is null) **then**
            $Prev[startInd][endInd] \leftarrow ((0,0), (startInd, i), (i+1, endInd)$
          end if
        end if//case (A1b)
      end if
    end for
  else if $start$ and end reduce to 1 **then**
    if term at $W_{startInd+1}^x$ is " $*$ " **and** term at $W_{endInd-1}^x$ is " $*$ " **then**
      if last index of "$\rangle$" = 1+ first index of "$\rangle$" (both in $W_{[startInd,endInd]}^x$) **then**
        $M[startInd][endInd] = 1$
        if (**not** $first$) **or** ($first$ **and** $Prev[startInd][endInd]$ is null) **then**
          $Prev[startInd][endInd] = ((startInd, endInd), (0,0), (0,0))$
        end if//initial case 2
        //other cases are computed in a similar way
      end if
    end if
  end if

First answer to the recognition problem for each string is given. If the answer is positive, that is $M(1, |W^x|) = 1$, which means that there exists a reduction to the sentence type for a given string of words, then an appropriate set of links is determined and showed. The resulting reduction is given in a form of a set of pairs of indices of the String W^x of simple terms that are contracted and a list of simple terms taking part in the reduction. Each contraction in the latter is denoted by a pair of round brackets.

We show a sample output of the application in **Figure 7.3**.

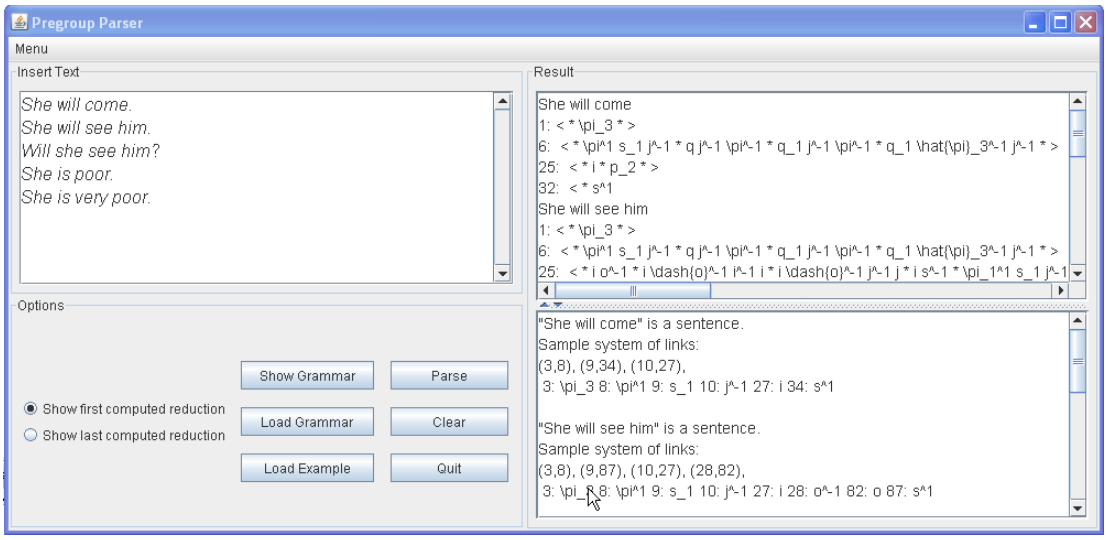


Figure 7.3: Sample Output of Java Pregroup Parser.

Below we present more examples of output of the Java Pregroup Parser. We start with a few questions.

Example 7.1.
Did he give books to her?

1: $\langle * \pi^1 s_2 i^{-1} * q i^{-1} \pi^{-1} * \rangle$
12: $\langle * \pi_3 * \rangle$
17: $\langle * i o^{-1} * i \hat{o}^{-1} i^{-1} i * i \hat{o}^{-1} j^{-1} j * i \overline{n}^{-1} o p^{-1} * i \hat{o}^{-1} i^{-1} i * \rangle$
42: $\langle * n_2 * \rangle$
47: $\langle * i^1 i o^{-1} * i^1 i \hat{o}^{-1} i^{-1} i * i^1 i \hat{o}^{-1} j^{-1} j * \overline{j} j^{-1} * j i^{-1} * \rangle$
72: $\langle * o * \rangle$
77: $\langle * s^1$

"Did he give books to her" is a sentence.

Sample system of links:
$(q\ (i^{-1}\ (\pi^{-1}\ \pi_3)\ (i\ (o^{-1}\ n_2)\ i^1)\ i)\ (o^{-1}\ o)\ s^1)$
(7,79), (8,50), (9,14), (19,49), (20,44), (51,74)

Example 7.2.
What did he give to her?

1: $\langle *os^1\overline{q} * \overline{q}\hat{o}^{-2}q^{-1} * \overline{q}s^{-1}\pi_3 * \overline{q}\hat{\pi}_3^{-2}q^{-1} * tj^{-1}*\rangle$
23: $\langle *\pi^1 s_2 i^{-1} * qi^{-1}\pi^{-1}*\rangle$
34: $\langle *\pi_3*\rangle$
39: $\langle *io^{-1} * i\hat{o}^{-1}i^{-1}i * i\hat{o}^{-1}j^{-1}j * i\overline{n}^{-1}op^{-1} * i\hat{o}^{-1}i^{-1}i*\rangle$
64: $\langle *i^1io^{-1} * i^1i\hat{o}^{-1}i^{-1}i * i^1i\hat{o}^{-1}j^{-1}j * \overline{j}j^{-1} * ji^{-1}*\rangle$
89: $\langle *o*\rangle$
94: $\langle *s^1$

"What did he give to her" is a sentence.

Sample system of links:
$(\overline{q}\ (\hat{o}^{-2}\ (q^{-1}q)\ (i^{-1}\ (\pi^{-1}\ \pi_3)\ i)\ \hat{o}^{-1})\ (i^{-1}\ (i\ i^1)\ i)\ (o^{-1}\ o)\ s^1)$
(7,96), (8,59), (9,29), (30,58), (31,36), (60,67), (61,66), (68,91)

Example 7.3.
What was given to her?

1: $\langle *os^1\overline{q} * \overline{q}\hat{o}^{-2}q^{-1} * \overline{q}s^{-1}\pi_3 * \overline{q}\hat{\pi}_3^{-2}q^{-1} * tj^{-1}*\rangle$
23: $\langle *\pi_1^1 s_2 j^{-1} i\hat{o}^{-2}p_2^{-1} * \pi_3^1 s_2 o^{-2}p_2^{-1} * \pi_3^1 s_2\hat{o}^{-2}p_2^{-1} * q_2p_1^{-1}\pi_1^{-1} * q_2p_1^{-1}\pi_3^{-1} *$
$q_2o^{-2}p_2^{-1}\pi_1^{-1} * q_2o^{-2}p_2^{-1}\pi_3^{-1} * q_2\hat{\pi}_3^{-1}\hat{o}^{-2}p_2^{-1} * q_2\hat{\pi}_3^{-1}p_1^{-1}*\rangle$
70: $\langle *p_2o^{-1} * p_2\hat{o}^{-1}i^{-1}i * p_2\hat{o}^{-1}j^{-1}j * p_2\overline{n}^{-1}op^{-1} * p_2\hat{o}^{-1}i^{-1}i*\rangle$
95: $\langle *i^1io^{-1} * i^1i\hat{o}^{-1}i^{-1}i * i^1i\hat{o}^{-1}j^{-1}j * \overline{j}j^{-1} * ji^{-1}*\rangle$
120: $\langle *o*\rangle$
125: $\langle *s^1$

"What was given to her" is a sentence.

Sample system of links:
$(\overline{q}\ (s^{-1}\ (\pi_3\ \pi_3^1)\ s_2)\ (\hat{o}^{-2}\ (p_2^{-1}\ p_2)\ \hat{o}^{-1})\ (i^{-1}\ (i\ i^1)\ i)\ (o^{-1}\ o)\ s^1)$ (11,127), (12,38), (13,37), (39,90), (40,89), (91,98), (92,97), (99,122)

Finally we give a result for a more complicated sentence.

Example 7.4.
I do not know who put these beautiful flowers on the table.

1: $\langle *\pi_1 * \rangle$
6: $\langle *i * \pi_1^1 s_1 i^{-1} * q^1 i^{-1} \pi_1^{-1} * \pi_2^1 s_1 i^{-1} * q^2 i^{-1} \pi_2^{-1} * \rangle$
27: $\langle *ii^{-1} * jj^{-1} * p_1 p_1^{-1} * p_2 p_2^{-1} * \overline{aa}^{-1} * \rangle$
45: $\langle *io^{-1} * i\hat{o}^{-1} i^{-1} i * i\hat{o}^{-1} j^{-1} j * it^{-1} * \pi_1^1 s_1 t^{-1} * q^1 t^{-1} \pi_1^{-1} * \pi_1^1 s_1 o^{-1} * q^1 o^{-1} \pi_1^{-1} * \pi_1^1 s_1 * \rangle$
83: $\langle *\bar{q}\hat{\pi}^{-2} q^{-1} * n_0^1 n_0 s^{-1} \pi_3 * n_0^1 n_0 s^{-1} \pi_3 * n_1^1 n_1 s^{-1} \pi_3 * n_2^1 n_2 s^{-1} \pi_2 * t\hat{\pi}^{-2} q^{-1} * tj^{-1} * \rangle$
117: $\langle *\pi^1 s_2 j^{-1} i o^{-1} * q_2 \hat{\pi}^{-1} j^{-1} i o^{-1} * \rangle$
132: $\langle *\overline{n_2} n_2^{-1} * \rangle$
138: $\langle *a * aa^1 n_2 n_2^{-1} * \rangle$
148: $\langle *n_2 * \rangle$
153: $\langle *i^1 i o^{-1} * i^1 i\hat{o}^{-1} i^{-1} i * i^1 i\hat{o}^{-1} j^{-1} j * \rangle$
172: $\langle *\overline{n_1} n_1^{-1} * \rangle$
178: $\langle *n_1 * \rangle$
183: $\langle *s^1$

"I do not know who put these beautiful flowers on the table." is a sentence.

Sample system of links:
$(\pi_1\ \pi_1^1)\ (s_1\ (i^{-1}\ i)\ (i^{-1}\ i)\ (t^l\ t)\ (\hat{\pi}^{-2}\ (q^{-1}\ q_2)\ \hat{\pi}^{-1})\ (j^{-1}\ (i\ (o^{-1}\ \overline{n}_2)$
$(n_2^{-1}\ (a\ a^1)\ n_2)\ (n_2^{-1}\ n_2)\ i^1)\ i)\ (o^{-1}\ \overline{n}_1)\ (n_1^{-1}\ n_1)\ s_1)$
(3,10), (11,185), (12,29), (30,60), (61,109), (110,126), (111,125), (127,156), (128,155), (129,134), (135,144), (142,143), (145,150), (157,174), (175,180)

Conclusion

In this thesis we discuss algorithmic questions for pregroup grammars. We start with giving a polynomial construction of a **CFG** equivalent to a given **PG**. Similarly, a construction of a **PDA** equivalent to a given **PG** is given.

However, these constructions are not really interesting for practical use. Thus, we give a new polynomial dynamic parsing algorithm for pregroup grammars. The algorithm improves slightly other cubic algorithms known so far. We prove correctness of the algorithm.

Further we consider some interesting extensions of pregroup grammars. We change our algorithm so that it works for pregroup grammars with letter promotions. It is still a polynomial algorithm and we provide the proof of its correctness (which is actually similar to the first proof). We discuss also pregroup grammars with letter promotions with 1. We prove a Normalization Theorem for **CBL** enriched with letter promotions with 1. We give a procedure to show that the letter promotion problem for pregroups with letter promotions with 1 is solvable in polynomial time. We describe how to employ our algorithm to pregroup grammars with letter promotions with 1.

Finally, we present a practical application of our parsing algorithm for pregroup grammars. It is a Java tool giving an answer to the recognition problem and in case of a positive answer returning one of possible reductions.

Appendix A

A Pregroup Grammar for a Fragment of English

The grammar is based on [Lam08]. The types and partial order are the same as listed in [Lam08]. The lexicon consists of words and types extracted from the linguistic examples in [Lam08] and we have added some new words.

A.1 List of basic types for English

Type	Description
a	predicative adjective
i	infinitive or intransitive verb
i'	one of intermediate types between i and j
j	infinitive of complete verb phrase
j'	one of intermediate types between i and j
n	name
n_0	mass noun
n_1	count noun
n_2	plural noun
o	direct object
o'	indirect object
p_1	present participle
p_2	past participle
q	yes-or-no question
q_1	yes-or-no question in the present tense
q_2	yes-or-no question in the past tense
s	sentence type
s_1	present sentence
s_2	past sentence
t	indirect question
$\hat{o}$	pseudo-object
$\hat{\pi}_3$	pseudo-subject
$\overline{a}$	predicative adjectival phrase
$\overline{j}$	complete infinitive with *to*
$\overline{n}$	complete noun phrase
$\overline{n}_0$	complete noun phrase with mass noun

$\overline{n}_1$	complete noun phrase with count noun
$\overline{n}_2$	complete noun phrase with plural noun
$\overline{q}$	question
$\overline{s}$	indirect statement
π	subject
π_1	first person singular subject
π_2	second person singular or plural subject
π_3	third person singular subject

A.2 Partial order

$a \leqslant \overline{a}$
$i \leqslant i'$
$i' \leqslant j'$
$j' \leqslant j$
$n \leqslant o$
$n \leqslant \pi_3$
$n_0 \leqslant \overline{n_0}$
$n_2 \leqslant \overline{n_2}$
$o \leqslant o'$
$q \leqslant s$
$q_1 \leqslant q$
$q_1 \leqslant s$
$q_2 \leqslant q$
$q_2 \leqslant s$
$s_1 \leqslant s$
$s_2 \leqslant s$
$\hat{o} \leqslant o$
$\hat{\pi}_3 \leqslant \pi_3$
$\overline{j} \leqslant \pi_3$
$\overline{n} \leqslant o$
$\overline{n}_0 \leqslant o$
$\overline{n}_0 \leqslant \pi_3$
$\overline{n}_0 \leqslant \overline{n}$
$\overline{n}_1 \leqslant o$
$\overline{n}_1 \leqslant \pi_3$
$\overline{n}_1 \leqslant \overline{n}$
$\overline{n}_2 \leqslant o$
$\overline{n}_2 \leqslant \pi_2$
$\overline{n}_2 \leqslant \overline{n}$
$\overline{q} \leqslant s$
$\pi_1 \leqslant \pi$
$\pi_2 \leqslant \pi$
$\pi_3 \leqslant \pi$

A.3 The lexicon

a $\overline{n_1}n_1^l$
am $\pi_1^r s_1 p_1^l, \pi_1^r s_1 o^{ll} p_2^l, q_1 p_1^l \pi_1^l, q_1 o^{ll} p_2^l \pi_1^l$
arrive $i, \pi_2^r s_1 j^l i$
arrived $p_2 j'^l i$
arriving $p_1 i'^l i$
ask $io^l, \pi_1^r s_1 t^l, \pi_2^r s_1 t^l, \pi_1^r s_1 j^l i t^l o^l, \pi_2^r s_1 j^l i t^l o^l$
asked $\pi^r s_2 t^l, \pi^r s_2 j^l i t^l o^l$
author n_1
avoid io^l

be $io^l, j' p_1^l, i' \hat{o}^{ll} p_2^l, i' \overline{a}^l$
beaten $p_2 o^l o p_2^r n_2 n_2^l$
bed n_1
being $p_1 o^{ll} p_2^l, p_1 i'^l i' \overline{a}^l$
been $p_2 p_1^l, p_2 o^{ll} p_2^l, p_2 j'^l j' p_1^l$
book n_1
books n_2
boy n_1

came $\pi_3^r s_2 j^l i, q_2 \pi_3^l j^l i$
cat n_1
cats n_2
colourless $aa^r n_2 n_2^l$
come i, p_2
coming $p_1, p_1 i'^l i$
completely $jj^l, p_2 p_2^l, ii^r$
consult i, io^l
cover n_1

dead a
did $\pi^r s_2 i^l$
die i
dislike io^l
do $i, \pi_2^r s_1 i^l$
does $\pi_3^r s_1 i^l, q_1 i^l \pi_3^l$
dog n_1
dogs n_2

eager $a, a\overline{j}^l$
easy $a, a\hat{o}^{ll}\overline{j}^l$
eating $n_1^r n_1 o^l$
entertain $\pi_2 r s_1 o^l$

expect $i, \pi_1^r s_1 o^l$

fish $n_0 n_1$
flee i
flower n_1
flowers n_2
frighten i, io^l
frightened a
furiously $i^r i$

gave $\pi^r s_2 j^l i o^l o p^l, \pi^r s_2 o^l o p^l, q_2 \hat{\pi}^l j^l i o^l$
get $io^l, i\hat{o}^{ll}$
give $io^l, i\overline{n}^l o p^l, i\hat{o}^l i^l i$
given $p_2 o^l, p_2 \overline{n}^l o p^l, p_2 \hat{o}^l i^l i$
go $i, \pi_1^r s_1 j^l i, \pi_2^r s_1 j^l i, \pi_1^r s_1, \pi_2^r s_1$
going $p_1 j^l i, p_1$
gone $p_2 j^l i, p_2$
good a
green $a a^r n_2 n_2^l$

had $\pi^r s_2 i^l i p_2^l$
has $\pi_3^r s_1 j^l i o^l, \pi_3^r s_1 p_2^l, q_1 p_2^l \pi_3^l, \pi_3^r s_1 o^l$
have $i, io^l, \pi_1^r s_1 j^l i o^l, \pi_2^r s_1 j^l i o^l, j p_2^l, \pi_1^r s_1 p_2^l, \pi_2^r s_1 p_2^l, q_1 p_2^l \pi_1^l, q_1 p_2^l \pi_2^l, \pi_1^r s_1 o^l, \pi_2^r s_1 o^l$
he π_3
her o
him o
house n_1
houses n_2

I π_1
idea n_1
ideas n_2
if $t s^l$
in $i^r i o^l$
intentionally $j j^l, p_2 p_2^l, i i^r$
into $i^r i o^l$
is $\pi_3^r s_1 p_1^l, \overline{a}^l \pi_3^r s_1 p_1^l, \pi_3^r s_1 o^{ll} p_2^l, q_1 p_1^l \pi_3^l, q_1 o^{ll} p_2^l \pi_3^l, \pi_3^r s_1 \overline{a}^l, q_1 \hat{\pi}_3^l p_1^l$
it π_3

kill i, io^l
king n_1
kings n_2
knew $\pi^r s_2 \overline{s}^l$
know $io^l, it^l, \pi_1^r s_1 t^l$

knows $\pi_3^r s_1 t^l$

letter n_1
letters n_2
like io^l
likes $\pi_3^r s_1 o^l$
love io^l
loves $\pi_3^r s_1 j^l io^l, \pi_3^r s_1 o^l$
loved $\pi^r s_2 o^l$

mail $io^l, \pi_1^r s_1 j^l io^l op^l, \pi_2^r s_1 j^l io^l op^l, \pi_1^r s_1 o^l op^l, \pi_2^r s_1 o^l op^l$
mailed $\pi^r s_2 j^l io^l op^l, \pi^r s_2 o^l op^l, , q_2 \hat{\pi}^l j^l io^l$
man n_1
many $\overline{n_2} n_2^l$
may $\pi^r s_1 j^l, q_1 j^l \pi_1^l, q_1 j^l \pi_2^l, q_1 j^l \pi_3^l$
me o
meet io^l
met $\pi^r s_2 o^l, \pi^r s_2 j^l i, p_2 o^l$

not $ii^l, jj^l, p_1 p_1^l, p_2 p_2^l, \overline{aa}^l$

one π_3, o, nn_1^l

person n_1
pig n_1
pigs n_2
please io^l
poor $a, n_0 n_0^l, n_1 n_1^l, n_2 n_2^l, aa^r n_0 n_0^l, aa^r n_1 n_1^l, aa^r n_2 n_2^l$
probably $ss^l, \pi^r ss^l \pi, jj^l, p_2 p_2^l, s^r s$
promised $\pi^r s_2 j^l i\overline{j}^l$

rain $n_0, i, \pi_1^r s_1, \pi_2^r s_1$
rained $\pi^r s_2$
rains $\pi_3^r s_1$
read $io^l, p_2 o^l$

saw $\pi^r s_2 o^l$
say $is^l, \pi_1^r s_1 j^l is^l, \pi_2^r s_1 j^l is^l, i\overline{s}^l, \pi_1^r s_1 j^l i\overline{s}^l, \pi_2^r s_1 j^l i\overline{s}^l, i\overline{q}^l, \pi_1^r s_1 j^l i\overline{q}^l, \pi_2^r s_1 j^l i\overline{q}^l, \quad \pi_1^r s_1 s^l,$
$\pi_2^r s_1 s^l, \pi_1^r s_1 \overline{s}^l, \pi_2^r s_1 \overline{s}^l, \pi_1^r s_1 \overline{q}^l, \pi_2^r s_1 \overline{q}^l$
see $io^l, is^l, \pi_1^r s_1 j^l io^l, \pi_2^r s_1 j^l io^l, \pi_1^r s_1 o^l, \pi_2^r s_1 o^l, \pi_1^r s_1 s^l$
seeing $p_1 j^l io^l, p_1 o^l$
seen $p_2 o^l, p_2 j^{\prime l} io^l$
sent $\pi^r s_2 j^l io^l op^l, \pi^r s_2 o^l op^l, q_2 \hat{\pi}^l j^l io^l$

she π_3
sleep $i, \pi_2^r s_1 j^l i$
sleeping $p_1 p_1^r n_2 n_2^l$
slept $\pi_3^r s_2$
speak i

taught $p_2 j^l i o^l o p^l, p_2 o^l o p^l, \pi^r s_2 j^l i o^l o p^l, \pi_1^r s_2 o^l o p^l$
teach $i o^l, \pi_1^r s_1 j^l i o^l o p^l, \pi_1^r s_1 o^l o p^l, \pi_2^r s_1 j^l i o^l o p^l, \pi_2^r s_1 o^l o p^l$
that $\overline{s} s^l, n_0^r n_0 s^l \pi_3, n_1^r n_1 s^l \pi_3, n_2^r n_2 s^l \pi_2, n_1^r n_1 \hat{o}^{ll} s^l$
the $\overline{n_1} n_1^l$
they π_2
to $i^r i o^l, \overline{j} j^l, j i^l$
today $i^r i$
told $p_2 j^l i s^l o p^l, \pi^r s_2 j^l i s^l o p^l, p_2 s^l o p^l, \pi^r s_2 s^l o p^l, p_2 \hat{o}^l \overline{j}^l$
tomorrow $i^r i$
too $\overline{a} \overline{j}^l a^l, \overline{a} \hat{o}^{ll} \overline{j}^l a^l, \overline{a a}^l$

under $i^r i o^l$

very $a a^l$

water n_0
was $\pi_1^r s_2 j^l i \hat{o}^{ll} p_2^l, \pi_3^r s_2 o^{ll} p_2^l, \pi_3^r s_2 \hat{o}^{ll} p_2^l, q_2 p_1^l \pi_1^l, q_2 p_1^l \pi_3^l, q_2 o^{ll} p_2^l \pi_1^l, q_2 o^{ll} p_2^l \pi_3^l, q_2 \hat{\pi}_3^l \hat{o}^{ll} p_2^l, q_2 \hat{\pi}_3^l p_1^l$
we π_2
went $\pi^r s_2$
what $o s^r \overline{q}, \overline{q} \hat{o}^{ll} q^l, \overline{q} s^l \pi_3, \overline{q} \hat{\pi}_3^{ll} q^l, t j^l$
when $i^r i s^l, n_1^r n_1 s^l, t j^l$
where $s^r \overline{q}, \overline{q} q^l, t j^l$
whether $t s^l$
which $\overline{q} \hat{\pi}_3^{ll} q^l n_0^l, t \hat{o}^{ll} \overline{j}^l n_2^l, t \hat{o}^{ll} \overline{j}^l n_1^l, n_0^r n_0 s^l \pi_3, n_1^r n_1 s^l \pi_3, n_2^r n_2 s^l \pi_2, n_1^r n_1 \hat{o}^{ll} s^l$
who $\overline{q} \hat{\pi}^{ll} q^l, n_0^r n_0 s^l \pi_3, n_0^r n_0 s^l \pi_3, n_1^r n_1 s^l \pi_3, n_2^r n_2 s^l \pi_2, t \hat{\pi}_3^{ll} q^l, t j^l$
whom $\overline{q} \hat{o}^{ll} q^l, o s^r \overline{q}, t \hat{o}^{ll} \overline{j}^l, n_1^r n_1 \hat{o}^{ll} s^l, n_1^r n_1 \hat{o}^{ll} \overline{j}^l, t j^l$
whose $\overline{q} \hat{o}^{ll} q^l n_1^l, \overline{q} \hat{o}^{ll} q^l n_2^l, n_1^r n_1 s^l \pi_2 n_2^l$
will $\pi^r s_1 j^l, q_1 j^l \pi^l, q_1 \hat{\pi}_3^l j^l$
wine n_0
with $i^r i o^l, i^r i \hat{o}^l i^l i, n_1^r n_1 \hat{o}^{lll} n_1^l n_1, \overline{q} \hat{o}^{lll} \overline{q}^l$
wonder $\pi_1^r s_1 j^l i t^l$
would $\pi^r s_2 j^l$

year n_1
yesterday $i^r i$
you π_2, o

Appendix B

Pregroup Grammars and Natural Languages

For more inflected languages than English, like Italian or Polish, the number of primitive types is much bigger, and so is an average number of types assigned to one word in the lexicon. To keep the lexicon smaller two tools were introduced: inflectors and metarules (see [Lam01, CL01]). They are different rules applied to the lexicon before parsing pregroup grammar. Metarules are dictionary rewriting rules. They say that if a word is of type X, then it can also have type Y. Inflectors are specially useful for inflected forms of verbs. Therefore, an inflected form can be assigned in the lexicon just appropriate inflector and infinitive, instead of all possible types that should be placed. These devices are a technical simplification of the lexicon. Therefore, while parsing is concerned, we still have only lexicon of the given language (with its metarules) and pregroup rules. More precisely, when parser looks for types assigned to the given word it takes into consideration not only the types listed directly in the lexicon, but also those obtained by application of metarules and inflector rules, to the appropriate types in the lexicon.

As an example of inflectors let us take a *declarative inflector* C_{jk} which is a conjugation matrix where j denotes the tense or mood of the verb and k - person. In Italian (Italian examples are from [Cas07]) the inflector C_{jk} is of the type $(\pi_k^r)s_j\overline{\overline{i}}^l$, where the round brackets mean that the subject can be omitted and no π_k argument is required. In Italian the inflector is defined for $j = 1, \ldots, 7$ (4 tenses: *present, imperfect, absolute past, future* and 3 moods: *conditional, subjunctive present, subjunctive imperfect*) and $k = 1, \ldots, 6$ (3 singular *"io", "tu", "lui"/"lei"* and 3 plural *"noi", "voi", "loro"*). Basic types used for Italian are listed below.

s_i	declarative sentence in $i-th$ tense,
$\tilde{s}$	sentential complement,
$i, \overline{i}, \overline{\overline{i}}$	infinitives,
π_k	$k-th$ person subject,
o	direct object,
λ	locative phrase.

If V is the infinitive of a verb then $V_{jk} = C_{jk}(V)$ is the type of that verb in $j-th$ tense and $k-th$ person. E.g. let us assume *correre (to run)* is of the type $i\lambda^l$ then

$$C_{13}(correre) = (\pi_k^r)\ s_j\ \overline{\overline{i}}^l\ i\ \lambda^l \leqslant (\pi_k^r)\ s_j\ \lambda^l$$

since in definition of partial order there is $i \leqslant \overline{i} \leqslant \overline{\overline{i}}$.

Example B.1 (Italian sentence with complements)**.**

Oggi	*Maria*	*corre*	*allo stadio.*
(*Today*	*Mary*	*is running*	*at the stadium.*)
Oggi	Maria	C_{13}(correre)	(allo stadio).
$(\tilde{s}\ s^l)$	π_3	$(\pi_3^r\ s_1\ \lambda^l)$	(λ)

since $s_1 \leqslant s$.

To illustrate the use of metarules let us consider questions. In Italian some direct questions can be obtained from declarative sentences just by adding a question mark at the end of the sentence and changing the intonation. Therefore a possible solution is to add a new inflector $C?_{jk}$ with the following metarule.

> **Interrogative Form Metarule I**
> If $C_{jk}(V)$ has type $(\pi_k^r)s_j\overline{\overline{i}}^l$, then $C?_{jk}(V)$ has type $(\pi_k^r)q_j\overline{\overline{i}}^l$, where the assignment of type q to a string of words W results in adding the punctuation mark "?" at the end of W; $W?$ is the interrogative form of W, with type q.

An example follows.

Example B.2 (Italian question)**.**

A statement:

Piero	*ama*	*Maria*
(*Peter*	*loves*	*Mary.*)
Piero	C_{13}(amare)	Maria.
π_3	$(\pi_3^r\ s_1\ o^l)$	(o)

The statement is of the type s_1, so it is a grammatically correct declarative sentence in the present tense. And the question:

	Piero	*ama*	*Maria?*
(*Does*	*Peter*	*love*	*Mary?*)
	Piero	$C?_{13}$(amare)	Maria?
	π_3	$(\pi_3^r\ q_1\ o^l)$	(o)

It is of the type q_1, so it is a grammatically correct question in the present tense.

Polish is even more inflected language than Italian. The conjugation matrix C_{jk} is smaller as $j = 1, \ldots, 4$ and $k = 1, \ldots, 9$. There are 3 tenses (*present, past, future*) plus 1 mood (*conditional*) and 9 persons(5 singular *"ja", "ty", "on", "ona", "ono"* and 4 plural *"my", "wy", "oni", "one"*). But there is also a declination of nouns: 7 cases, 3 genders and 2 numbers are distinguished. All of these phenomena can be described by use of pregroup types and sub-types, see [Kiś02, KM08]. The examples come from the first paper.

The following basic types are used.

s_i	declarative sentence in i-th tense,
π_k	$k-th$ person subject,
o_i	objects,
n_{ij}	singular noun, where $i \in \{m, f, n\}$ for masculine, feminine or neuter gender, $j = 1, \cdots 7$ for j-th case,
$\hat{n}_{ij}$	plural noun with i and j as above

Example B.3 (Polish nouns)**.**

Dziecko	*daje*	*prezenty*	*mamie.*
(*A child*	*is giving* or *gives*	*presents*	*to (his) mommy.*)
n_{n1}	$\pi_5^r\ s_1\ o_3^l\ o_4^l$	$\hat{n}_{m4}$	n_{f3}
$\underbrace{\pi_5}$	$\underbrace{(\pi_5^r}\ s_1\ \underbrace{o_3^l\ o_4^l)}$	$\underbrace{(o_4)}$	$\underbrace{(o_3)}$

since in definition of partial order there is $n_{n1} \leqslant \pi_5$ (nouns in nominative cases may play the role of personal pronouns), $n_{ij} \leqslant o_j$ and $\hat{n}_{ij} \leqslant o_j$, both for $j = 2, \cdots, 6$. The string of words is of the type s_1, so it is a grammatically correct sentence in the present tense.

Polish allows quite a free order of words in a sentence, so verbs need a few types to accept subject and objects in different positions. Kiślak [Kiś02] proposed a solution of universal types, for example for the verb *być* (to be): $x^r\ s_i\ y^r$ where either

$$x = (\alpha)(\pi_k)(\alpha) \text{ and } y = (\alpha)$$

or

$$x = (\alpha) \text{ and } y = (\alpha)(\pi_k)(\alpha)$$

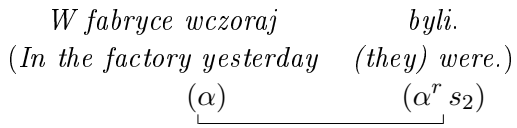
The parentheses mean, as before, optional occurrence of the type in between them. α is a general type of a prepositional phrase. Below we give an example of a sentence in different order (all of them are correct in Polish).

Example B.4 (Polish word order)**.**

Byli	*wczoraj w fabryce.*
(*(They) were*	*yesterday in the factory.*)
$(s_2\ \underbrace{\alpha^l}$	$\underbrace{(\alpha)}$

Wczoraj	*byli*	*w fabryce.*
(*Yesterday*	*(they) were*	*in the factory.*)
$\underbrace{(\alpha)}$	$\underbrace{(\alpha^r}\ s_2\ \underbrace{\alpha^l)}$	$\underbrace{(\alpha)}$

Oni	*byli*	*wczoraj w fabryce.*
(*They*	*were*	*yesterday in the factory.*)
$\underbrace{(\pi_8)}$	$\underbrace{(\pi_8^r}\ s_2\ \underbrace{\alpha^l)}$	$\underbrace{(\alpha)}$


W fabryce wczoraj
byli.
(In the factory yesterday
(they) were.)
(α)
(α^r s_2)

Appendix C

Computations in different calculi

Kiślak-Malinowska [KM05] observed that all parsing examples given by Lambek in [Lam99] can also be parsed using **L**-grammars or even **CCG**s.

Recall that $(\backslash, /)$- types have to be translated into pregroup types using the rules:

$$a\backslash b = a^r b \text{ and } a/b = ab^l,$$

and the equalities

$$(ab)^l = b^l a^l \text{ and } (ab)^r = b^r a^r,\ (a^r)^l = a = (a^l)^r,$$

which are valid in pregroups. Moreover, in some cases, one must add to **L** the same postulates as in pregroups: $a \Rightarrow b$, for some atomic a, b (order of types). We assume that the list of basic types is the same for all calculi (**AB**, **L** and **CBL**).

In the example below, there are a few sentences for which the type s is derivable in **AB**, **L** and **CBL** (with assumptions on types and postulates as described above).

Example C.1.

He	*goes.*	
π_3	$\pi_3^r\ s_1$	$= (\pi_3\pi_3^r)s_1 \leqslant s_1$
π_3	$\pi_3\backslash s_1$	$\Rightarrow s_1$

I	*slept.*	
π_1	$\pi_1^r\ s_2$	$= (\pi_1\pi_1^r)s_2 \leqslant s_2$
π_1	$\pi_1\backslash s_2$	$\Rightarrow s_2$

I	*saw*	*her.*	
π_1	$\pi_1^r\ s_2\ o^l$	o	$= (\pi_1\pi_1^r)s_2(o^l o) \leqslant s_2$
π_1	$(\pi_1\backslash s_2)/o$	o	$\Rightarrow \pi_1(\pi_1\backslash s_2) \Rightarrow s_2$

I	*am*	*going.*	
π_1	$\pi_1^r\ s_2\ p_1^l$	p_1	$= (\pi_1\pi_1^r)s_2(p_1^l p_1) \leqslant s_2$
π_1	$(\pi_1\backslash s_2)/p_1$	p_1	$\Rightarrow \pi_1(\pi_1\backslash s_2) \Rightarrow s_2$

I	*should*	*have*	*loved*	*him.*	
π_1	$\pi^r\ s_2\ j^l$	$j\ p_2^l$	$p_2 o^l$	o	$\leqslant (\pi\pi^r)s_2(j^l j)(p_2^l p_2)(o^l o) \leqslant s_2$
π_1	$(\pi\backslash s_2)/j$	j/p_2	p_2/o	o	$\Rightarrow s_2$

He	*was*	*seen.*	
π_3	$\pi_3^r\ s_2\ o^{ll}\ p_2^l$	$p_2\ o^l$	$\leqslant (\pi_3\pi_3^r)s_2(o^{ll}(p_2^lp_2)o^l) \leqslant s_2$
π_3	$(\pi_3\backslash s_2)/(p_2/o)$	p_2/o	$\Rightarrow s_2$

The following example illustrates sentences which cannot be successfully parsed in **AB**.

Example C.2.

He	*was*	*not*	*seen.*	
π_3	$\pi_3^r\ s_2\ o^{ll}\ p_2^l$	$p_2\ p_2^l$	$p_2\ o^l$	$\leqslant (\pi_3\pi_3^r)s_2(o^{ll}(p_2^lp_2)(p_2^lp_2)o^l) \leqslant s_2$
π_3	$(\pi_3\backslash s_2)/(p_2/o)$	p_2/p_2	p_2/o	$\Rightarrow s_2$ (in **L** only)

The following example comes from [CL02].

Whom	*did*	*you*	*see?*	
$\bar{q}\ o^{ll}\ q^l$	$q_2\ i^l\ \pi^l$	π_2	io^l	$\leqslant \bar{q}(o^{ll}(q^lq)(i^l(\pi^l\pi)i)o^l) \leqslant \bar{q}$
$\bar{q}/(q/o)$	$((q_2/i)/\pi)$	π_2	i/o	$\Rightarrow s_2$ (in **L** only)

A question, whether there exist linguistic examples which can be parsed by means of pregroups but not the Lambek Calculus, remains still open.

However, even though, all known linguistic examples can be parsed by means of an **L**-grammar, parsing by use of pregroup grammar has an important advantage. All computations in pregroups can be performed in polynomial time, which is not true for calculations in Lambek Calculus (if the hypothesis $P \neq NP$ holds).

Index

References

[Abr91] V. M. Abrusci. Phase semantics and sequent calculus for pure noncommutative classical linear propositional logic. *The Journal of Symbolic Logic*, 56(4):1403–1451, 1991. 25

[Ajd35] K. Ajdukiewicz. Die syntaktische konnexität. *Studia Philosophica*, 1:1–27, 1935. 2

[Béc07] D. Béchet. Parsing pregroup grammars and Lambek Calculus using partial composition. *Studia Logica*, 87(2-3):199–224, 2007. 7, 35, 37, 38, 66

[BGS60] Y. Bar-Hillel, C. Gaifman, and E. Shamir. On categorial and phrase structure grammars. *Bulletin of the Research Council of Israel*, 9F:155–166, 1960. 3, 20, 35

[BL01] D. Bargelli and J. Lambek. An algebraic approach to french sentence structure. In G. Morrill P. de Groote and C. Rétoré, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *LNAI*, pages 62–78. Springer-Verlag, Berlin, 2001. 5, 25

[BLZ09] W. Buszkowski and Lin Zhe. Pregroup grammars with letter promotions. *manuscript*, 2009. 8, 69, 70, 71, 72, 76, 82

[BM08] W. Buszkowski and K. Moroz. Pregroup grammars and context-free grammars. In C. Casadio and J. Lambek, editors, *Computational Algebraic Approaches to Natural Language*, pages 1–21. Polimetrica, Monza, Milano, 2008. 8, 37, 38, 39

[Bus78] W. Buszkowski. Undecidability of some logical extensions of Ajdukiewicz-Lambek Calculus. *Studia Logica*, 37(1):59–64, 1978. 3

[Bus82] W. Buszkowski. Some decision problems in the theory of syntactic categories. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 28:539–548, 1982. 3

[Bus85] W. Buszkowski. The equivalence of unidirectional Lambek categorial grammars and context-free grammars. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 31:369–384, 1985. 3

[Bus86a] W. Buszkowski. Completeness results for Lambek Syntactic Calculus. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 32:13–28, 1986. 3, 24

[Bus86b] W. Buszkowski. Generative capacity of nonassociative Lambek Calculus. *BULLETIN of the POLISH ACADEMY of SCIENCES MATHEMATICS*, 34:507–516, 1986. 3

[Bus01] W. Buszkowski. Lambek grammars based on pregroups. In P. de Groote, G. Morrill, and C. Retorè, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *LNAI*, pages 95–109. Springer-Verlag, Berlin, 2001. 4, 7, 28, 31, 35, 37

[Bus03] W. Buszkowski. Sequent systems for compact bilinear logic. *Mathematical Logic Quarterly*, (49):467–474, 2003. 4

[Bus07] W. Buszkowski. Type logics and pregroups. *Studia Logica*, 87(2-3):145–169, 2007. 18, 27, 28, 31

[Car07] Kumi Cardinal. A pregroup analysis of Japanese causatives. In *Proceedings of the 21st Pacific Asia Conference on Language, Information and Computation*, 2007. 5, 25

[Cas07] C. Casadio. Applying pregroups to Italian statements and questions. *Studia Logica*, 87(2-3):253–268, 2007. 117

[Cho57] N. Chosky. *Syntactic Structures*. Mouton, The Hague, 1957. 1

[CL01] C. Casadio and J. Lambek. An algebraic analysis of clitic pronouns in Italian. In G. Morrill P. de Groote and C. Rétoré, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *LNAI*, pages 110–124. Springer-Verlag, Berlin, 2001. 5, 25, 117

[CL02] C. Casadio and J. Lambek. A tale of four grammars. *Studia Logica*, 71:315–329, 2002. 122

[Coh67] J. M.. Cohen. The equivalence of two concepts of categorial grammar. *Information and Control*, 5(5):475–484, 1967. 3

[DP05] S. Degeilh and A. Preller. Efficiency of pregroups and the French noun phrase. *Journal of Logic, Language and Information*, 14:423–444, 2005. 67

[Fre92] G. Frege. Über begriff und gegenstand. *Vierteljahresschrift für wissenschaftliche Philosophie*, 16:192–205, 1892. 1

[Gir87] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987. 4, 25

[HU03] J.E. Hopcroft and J.D. Ullman. *Wprowadzenie do teorii automatów, jezyków i obliczeń*. Wydawnictwo Naukowe PWN, Warszawa, 2003. 9

[JS08] J. Jędrzejowicz and A. Szepietowski. *Języki, automaty, złożoność obliczeniowa*. Wydawnictwo Uniwersytetu Gdańskiego, Gdańsk, 2008. 9

[Kan88a] M. Kandulski. The equivalence of nonassociative Lambek categorial grammars and context-free grammars. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 34:41–52, 1988. 3

[Kan88b] M. Kandulski. Phrase structure languages generated by categorial grammars with product. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 34:373–383, 1988. 3

[Kiś02] A. Kiślak. Pregroups versus English and Polish grammar. In V. M. Abrusci and C. Casadio, editors, *New Perspectives in Logic and Formal Linguistics*, pages 129–154. Bulzoni Editore, Bologna, 2002. 118, 119

[KM05] A. Kiślak-Malinowska. *Systemy logiczne rachunku pregrup i ich zastosowanie w gramatykach formalnych.* PhD thesis, Uniwersytet Warmińsko-Mazurski, Olsztyn, 2005. 121

[KM08] A. Kiślak-Malinowska. Polish language in terms of pregroups. In C. Casadio and J. Lambek, editors, *Computational Algebraic Approaches to Natural Language*, pages 145–172. Polimetrica, Monza, Milano, 2008. 5, 25, 118

[Lam58] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:21–39, 1958. 3, 4, 21, 22

[Lam93] J. Lambek. From categorial grammar to bilinear logic. In K. Došen and P. Schroeder-Heister, editors, *Substructural Logics*, pages 207–237. Clarendon Press, Oxford, 1993. 25

[Lam99] J. Lambek. Type grammars revisited. In A. Lecomte, F. Lamarche, and G. Perrier, editors, *Logical Aspects of Computational Linguistics*, volume 1582 of *LNCS*, pages 1–27. Springer-Verlag, Berlin, 1999. 1, 4, 25, 26, 27, 29, 30, 31, 121

[Lam01] J. Lambek. Type grammars as pregroups. *Grammars*, 4(1):21–39, 2001. 26, 117

[Lam08] J. Lambek. *From Word to Sentence: a computational algebraic approach to grammar.* Polimetrica, Monza, Milano, 2008. 5, 25, 32, 48, 62, 101, 111

[Leś29] S. Leśniewski. Grundzüge eines neuen systems der grundlagen der mathematik. *Fundamenta Mathematicae*, 14:1–81, 1929. 2

[LP04] J. Lambek and A. Preller. An algebraic approach to the German sentence. *Linguistic Analysis*, 31:270–290, 2004. 5, 25

[MF05] A. H. Mater and J. D. Fix. Finite presentations of pregroups and the identity problem. In *Proceedings of FG-MoL 2005*, CSLI (electronic), pages 63–72, 2005. 69, 70

[Moo88] M. Moortgat. *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus.* Foris, Dordrecht, 1988. 4

[Mor94] G. Morrill. *Type Logical Grammar: Categorial Logic of Signs*. Kluwer Academic Publishers, Dordrecht, 1994. 4

[Mor09] K. Moroz. Parsing pregroup grammars in polynomial time. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, volume 4, page 257–264, 2009. 8

[Mor10] K. Moroz. A Savateev-style parsing algorithm for pregroup grammars. In *Proceedings of the 14th conference of Formal Grammar*, volume 5591 of *Lecture Notes in Computer Science*. Springer-Verlag, 2010. 8

[Oeh04] R. Oehrle. A parsing algorithm for pregroup grammars. In *Proceedings of Categorial Grammars 2004*, pages 59–75, 2004. 4, 8, 47, 62, 65, 66, 67

[Pen93] M. Pentus. Lambek grammars are context-free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, page 429–433, 1993. 3, 23

[Pen95] M. Pentus. Models for the Lambek Calculus. *Annals of Pure and Applied Logic*, 75(1-2):179–213, 1995. 24

[Pen08] M. Pentus. Lambek Calculus is NP-complete. *Theoretical Computer Science*, 357:186–201, 2008. 5, 22

[Pre07] A. Preller. Linear processing with pregroups. *Studia Logica*, 87(2-3):171–197, 2007. 4, 8, 47, 67, 68

[Sav09] Y. Savateev. Unidirectional Lambek grammars in polynomial time. *Theory of Computing Systems*, online, 2009. 8, 47, 49, 50

[Tar33] A. Tarski. *Pojęcie prawdy w językach nauk dedukcyjnych*. TNW, Warszawa, 1933. 1

[vB86] J. van Benthem. *Essays in logical semantics*. D. Reidel, Dordrecht, 1986. 4

[vB91] J. van Benthem. *Language in Action: categories, lambdas and dynamic logic*, volume 130 of *Studies in Logic and the Foundations of Mathematics*. Amsterdam, 1991. 4

[WR13] A. Whitehead and B. Russell. *Principia Mathematica, 3 vols*. Cambridge University Press, 1910-1913. 2

[Zie78] W. Zielonka. A direct proof of the equivalence of free categorial grammars and simple phrase structure grammars. *Studia Logica*, 37(1):41–57, 1978. 3

[Zie81] W. Zielonka. Axiomatizability of Ajdukiewicz-Lambek Calculus by means of cancellation schemes. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 27:215–224, 1981. 3

[Zie85] W. Zielonka. J. M. Cohen's claim on categorial grammars remains unproved. *Bulletin of the Section of Logic*, 14(4):130–133, 1985. 3