Logic and **L**ogical **P**hilosophy
Volume 21 (2012), 229–252
DOI: 10.12775/LLP.2012.012

Aleksandra Kiślak-Malinowska

EXTENDED PREGROUP GRAMMARS APPLIED TO NATURAL LANGUAGES*

Abstract. Pregroups and pregroup grammars were introduced by Lambek in 1999 [14] as an algebraic tool for the syntactic analysis of natural languages. The main focus in that paper was on certain extended pregroup grammars such as pregroups with modalities, product pregroup grammars and tupled pregroup grammars. Their applications to different syntactic structures of natural languages, mainly Polish, are explored/shown here.

Keywords: Pregroups, pregroup grammars, product pregroup grammars, tuple pregroup grammars.

1. Introduction

Pregroups and pregroup grammars were introduced in 1999 [14]. Lambek proposed them as a new tool for the syntactic analysis of natural languages. The formalism of pregroups belongs to the tradition of categorial grammars. In general they are part of the wide field of mathematical linguistics, i.e., the theory of formal grammars and automata with applications in computer science.

They are particularly useful in natural language processing. Many grammatical aspects of natural languages have been described in terms of pregroup grammars. The problems tackled include verbs, nouns, adjectives, adverbs, noun phrases, negative sentences, *why*-questions, *yes-or-no* questions, relative clauses, islands, and unbounded dependencies.

* The first version of this work was presented during The Third Conference: *Non-Classical Logic. Theory and Applications*, NCU, Toruń, September 16–18, 2010.

Received December 14, 2010; *Revised* May 23, 2011

ISSN: 1425-3305

Among the languages to which pregroup grammars have been applied are English [17], German, French [18], Italian [4, 5], Japanese, Chinese and Polish [8, 10, 11].

The idea of applying pregroup grammars to natural languages is as follows. We start from the lexicon — to each word of the natural language one or more types are assigned. Calculations are done on types and when they give a simple s type as a result (the type of the sentence) the sentencehood of the string is established. The formalism of pregroups has two main advantages in comparison with other approaches. Firstly, the calculations are quite easy, as for linguistic application only contractions and induced steps are needed. What is more, they have the very nice computational property of polynomial decidability. Pregroup grammars are weakly equivalent to context-free grammars (without empty strings) [1]. A dynamic polynomial parsing algorithm for pregroup grammars with the proof of its correctness was given in [19].

Pregroup grammars have been under intense study since they were introduced by logicians, mathematicians, linguists and computer scientists. They have looked at axiomatization, the construction of algorithms, and their mathematical and computational properties and their application to natural language processing. A number of authors have tried to modify the pure, initial calculus of pregroups in order to obtain further desirable properties. To prevent unwanted derivations pregroups with modalities have been considered [7, 9]. For feature agreement, especially in Romance languages, product pregroups have been introduced [12, 18]. To tackle certain complicated aspects of grammar tupled pregroups have been employed [20].

In this paper the focus is mainly on the application of the formalism to natural languages. The paper is organized as follows. In the second section the notion of a pregroup and basic properties of pure calculus of pregroups needed for processing the language are introduced. The section ends with an illustration of the techniques. The third section deals with pregroups with modalities which are pregroups with some additional operators. The calculus of pregroups with modalities enjoys properties similar to the pure calculus, the normalization theorem and the cut elimination theorem for which have already been established (see [7, 9]). Another version of the calculus of pregroups — product pregroups — is presented in the fourth section. On this approach it is possible to use more free pregroups (product of pregroups instead of one) in order to deal with different aspects of a given grammar. Some authors

use just two, others more. A grammatical feature of Italian concerning personal pronouns in the accusative case in constructions with verbs in different tenses (here: present and present perfect) is examined. The fifth and final section deals with tupled pregroup grammars, introduced in [20] which allow for the treatment of more sophisticated grammatical phenomena. The prepositional phrases of Polish have been chosen as an illustration. It is worth mentioning that as the two last approaches seem more powerful and cover more grammatical features, it is not surprising that the languages they generate are not necessarily context-free: product grammars can yield intersections of context-free languages which are not always context-free, while languages generated by tupled pregroup grammars are multiple context-free languages. They are very interesting alternatives to the pure calculus of pregroups.

2. Pregroups and free pregroups

The notion of a pregroup was introduced by Lambek [14] as a modification of residuated monoids being algebraic models of the Lambek syntactic calculus [13]. The formal calculus of pregroups is more easily treatable and seems to have greater linguistic flexibility. *A pregroup* is a structure $(G, \leq, \cdot, ^l, ^r, 1)$ such that $(G, \leq, \cdot, 1)$ is a partially ordered monoid, and ^{l} and ^{r} are unary operations satisfying for all $a \in G$ the following inequalities:

(PRE) $\quad a^l a \leq 1 \leq a a^l$ and $a a^r \leq 1 \leq a^r a$.

a^l (resp. a^r) is called *the left* (resp. *the right*) *adjoint to* a.

It is easily shown that the following conditions must hold in every pregroup:

(1) $\quad 1^l = 1^r = 1$, $(a^l)^r = (a^r)^l = a$, $(ab)^l = b^l a^l$, $(ab)^r = b^r a^r$,
(2) $\quad$ if $a \leq b$ then $b^l \leq a^l$ and $b^r \leq a^r$.

Pregroups have been studied in e.g. [1, 2, 8, 15, 16].

For any element a of a pregroup, we define an element $a^{(n)}$, for $n \in \mathbf{Z}$:

- $a^{(0)} = a$,
- $a^{(n+1)} = (a^n)^r$,for $n \geqslant 0$,
- $a^{(n-1)} = (a^n)^l$, for $n \leqslant 0$.

Then the equalities $a^{(n+1)} = (a^n)^r$ and $a^{(n-1)} = (a^n)^l$ hold true, for all $n \in \mathbf{Z}$.

As a consequence of (PRE) and (2), we obtain:

(3) $a^{(n)}a^{(n+1)} \leq 1 \leq a^{(n+1)}a^{(n)}$, for all $n \in \mathbf{Z}$.

(4) If $a \leq b$, then $a^{(2n)} \leq b^{(2n)}$ and $b^{(2n+1)} \leq a^{(2n+1)}$, for all $n \in \mathbf{Z}$.

Quasipregroups are defined as pregroups except that the relation $\rightarrow$ need not be antisymmetrical. If $\rightarrow$ is a quasiordering on G, then one defines $x \simeq y$ iff $x \rightarrow y$ and $y \rightarrow x$, for $x, y \in G$. Then, $\simeq$ is a congruence on G. $\simeq$ is an equivalence relation and the quotient relation $[x] \rightarrow [y]$ iff $x \rightarrow y$ is a partial ordering on the quotient set $G/\sim$. We can construct the quotient structure on the universe $G/\sim$, by setting: $[x] \cdot [y] = [xy]$, $[x]^l = [x^l]$, $[x]^r = [x^r]$, for any $x, y \in G$. This quotient structure is a pregroup whose unit element equals $[\mathbf{1}]$.

Given a poset P, we may construct the *free* pregroup $F(P)$ generated by P. Let $(P, \leq)$ be a poset. Elements of P are treated as constant symbols and are called basic types. Terms are expressions $a^{(n)}$, for $a \in P$, $n \in \mathbf{Z}$ and we call them simple types; $a^{(0)}$ is identified with a. Types are finite strings of terms, called types.

First, we define a quasipregroup whose elements are types. For types x, y, $x \cdot y$ is the concatenation of strings x, y, and 1 is the empty string. The adjoints are defined as follows:

(5a) $(a_1^{(n_1)}a_2^{(n_2)} \dots a_k^{(n_k)})^l = a_k^{(n_k-1)} \dots a_2^{(n_2-1)}a_1^{(n_1-1)}$,

(5b) $(a_1^{(n_1)}a_2^{(n_2)} \dots a_k^{(n_k)})^r = a_k^{(n_k+1)} \dots a_2^{(n_2+1)}a_1^{(n_1+1)}$.

It follows that $(t_1 \dots t_k)^l = t_k^l \dots t_1^l$ and $(t_1 \dots t_k)^r = t_k^r \dots t_1^r$ for any terms $t_1, \dots, t_k$.

Let $\rightarrow$ be the least reflexive and transitive relation defined by the following cases for all types x, y; $n \in \mathbf{Z}$ and $a, b \in P$:

contractions	$xa^{(n)}a^{(n+1)}y \rightarrow xy$,	
expansions	$xy \rightarrow xa^{(n+1)}a^{(n)}y$,	
induced steps	$xa^{(n)}y \rightarrow xb^{(n)}y$,	if $a \leq b$ and n is even,
		if $b \leq a$ and n is odd.

Contractions, expansions and induced steps 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 the rules. This procedure may be also called a derivation of a string. This rewriting system is Lambek's original form of the logic of pregroups and is also called Compact Bilinear Logic.

It is easy to check that this structure is a quasipregroup. It is not a pregroup, since $aa^{(l)}a \simeq a$, but $aa^{(l)}a \neq a$. The quotient structure

employing the quasiordering defined above is a pregroup. It is called *the free pregroup* generated by $(P, \leq)$ and is denoted $F(P)$.

A generalized contraction combines a contraction with an induced step and has the form:

(6) $xa^{(n)}b^{(n+1)}y \to xy$, if $a \leq b$ and n is even or $b \leq a$ and n is odd.

A generalized expansion combines an expansion with an induced step and has the form:

(7) $xy \to xa^{(n+1)}b^{(n)}y$, if $a \leq b$ and n is even or $b \leq a$ and n is odd.

We recall the Lambek switching lemma from [14].

LEMMA. *If $x \to y$ in $F(P)$, then there exist types z and z' such that: $x \to z \to z' \to y$, where $x \to z$ by generalized contractions only, $z \to z'$ by induced steps only and $z' \to y$ by generalized expansions only.*

This lemma states that if we derive type y from the type x ($x \to y$) using generalized contractions, generalized expansions and induced steps the order of executing those operations in the derivation can be changed (if necessary) starting from all contractions and ending with all expansions. It is of vital importance for the application to natural language as in particular for $y = s$ no expansions need to be involved.

This lemma is closely related to the cut elimination theorem for Compact Bilinear Logic. It also yields the polynomial time decidablility of this logic [1].

A pregroup grammar is a quintuple $G = (\Sigma, P, \leq, s, I)$ such that: Σ is a nonempty, finite alphabet, $(P, \leq)$ a finite poset, $s \in P$ and I the relation between symbols for Σ and nonempty types on P. The relation I will be called *lexicon* in our further considerations.

Let $x \in \Sigma^+$, $x = a_1 a_2 \dots a_n$, $(a_i \in \Sigma)$. We say that the grammar G assigns type y to x, if there exist $x_i \in I(a_i)$, $1 \leqslant i \leqslant n$, such that $x_1, \dots, x_n \to y$ in Compact Bilinear Logic; (we write $x \to_G y$). *The language of the pregroup grammar* G is the set $L(G) = \{x \in \Sigma^+ : x \to_G s\}$.

3. Illustration

The idea of applying pregroups for a natural language can be illustrated as follows. We start from the lexicon — to each word of the natural language one or more types are assigned (terms are of the type $a^{(n)}$,

types are finite strings of terms; the lexicon for some chosen types for Polish is given in the appendix). The types used in the lexicon are assigned in such a way that they reflect the syntactic role of a word in natural language. Given a string of lexical entries we use rewriting rules on types associated with words. The sentencehood of a given string may be verified by means of those calculations. When it ends with type s (sentence type) we say the string of words is a well-formed sentence.

To start with, let us consider the following types:
s_1 – declarative sentence in the present,
s_2 – declarative sentence in the past,
π_1 – first person subject, singular
π_3 – third person subject, singular, masculine
π_4 – third person subject, singular, feminine
π_8 – third person plural subject, masculine.

In what follows we will ignore capital letters at the beginning of the sentence. For example the sentence *Ja jestem* [I . am] will be written *ja jestem*, as we are only interested in the sentencehood of a given string of words. Taking from lexicon the type π_1 to the first person subject *ja* [I], and the type $\pi_1^r s_1$ to the Polish word *jestem*, we can get:

$$\begin{array}{ll} ja & jestem \\ \pi_1 & \pi_1^r s_1 \qquad = (\pi_1 \pi_1^r) s_1 \to s_1. \end{array}$$

Using contraction $(\pi_1 \pi_1^r)$ (parentheses play only auxiliary role to show where the contraction occurs) we get the type s_1 for the string *ja jestem*, which means that this is a properly built present tense sentence in Polish.

However, we cannot obtain: *oni był* [they . was]. Typing this string of words one gets:

$$\begin{array}{ll} oni & był \\ \pi_8 & \pi_3^r s_2 \quad \not\to s_2. \end{array}$$

Here a contraction rule cannot be used, as types π_8 and π_3^r do not contract. This means that the string *oni był* is not a sentence in Polish.

To type Polish nouns we use n_{ij} or $\hat{n}_{ij}$. n and $\hat{n}$ stand for a singular and a plural noun, respectively. The first index stands for the gender (masculine, feminine or neuter), whereas the second index is for the case (in Polish nouns are inflected for case). For typing adjectives $n_{ij} n_{ij}^l$ and $\hat{n}_{ij} \hat{n}_{ij}^l$ are used. The motivation behind this is that adjectives can qualify nouns as long as they agree in number and case. We make use of a partial order concerning types. $n_{m1} \leq \pi_3$ means that a masculine

singular Polish noun in the nominative case in the sentence may also play the role of a third person subject, whereas $n_{n4} \leq o_4$ means that a neuter singular noun in the accusative case may be seen as an accusative object in the sentence.

An example here may be:

mały chłopiec lubi kino [(a) small . boy . likes . (a/the) cinema]

Taking types for words from the lexicon in this string one gets:

mały	*chłopiec*	*lubi*	*kino*		
$n_{m1}n_{m1}^l$	n_{m1}	$\pi_3^r s_1 o_4^l$	n_{n4}	$\to$	(contraction $n_{m1}^l n_{m1}$)
n_{m1}		$\pi_3^r s_1 o_4^l$	n_{n4}	$\to$	(partial order $n_{m1} \leq \pi_3$)
π_3		$\pi_3^r s_1 o_4^l$	n_{n4}	$\to$	(contraction $\pi_3^l \pi_3$)
		$s_1 o_4^l$	n_{n4}	$\to$	(partial order $n_{n4} \leq o_4$)
		$s_1 o_4^l$	o_4	$\to$	(contraction $o_4^l o_4$)
		s_1			

The steps on the right illustrate the detailed verification the sentencehood of *mały chłopiec lubi kino*.

These examples convey the idea of checking whether a given string of words is a well-formed declarative sentence or not. The calculus of pregroups seems to be more easily treatable in comparison with Lambek's former approach [13] — the syntactic calculus. The aim was to obtain an effective rule for distinguishing sentences from nonsentences which could work both for formal and natural languages. Lambek introduced the calculus of types related to the well-known calculus of residuals.

4. Pregroups with modalities

In some cases one may need particular constraints in order to block undesirable derivations. Examples of such situations and proposals how to overcome the problem using modalities can be found in [7] or [9].

The definition of a pregroup with β-operator was proposed by Fadda [7]. The motivation to introduce modal operators was the need to limit (locally) associativity in the calculus.

A pregroup with β-operator is a pregroup G enriched additionally with a monotone mapping $\beta\colon G \to G$. A β-pregroup is a pregroup with β-operator such that β-operator has the right adjoint $\hat{\beta}$ ($\hat{\beta}$-operator), i.e., there exists a monotone mapping $\hat{\beta}\colon G \to G$ with the property that for all a and b in G, $\beta(a) \leq b$ if and only if $a \leq \hat{\beta}(b)$. It is easy to

show that $\hat{\beta}$, if it exists, is uniquely defined and connected to β via the following rules of expansion and contraction, for any $a \in G$:

$$a \leq \hat{\beta}(\beta(a)) \qquad \text{and} \qquad \beta(\hat{\beta}(a)) \leq a\,.$$

The construction of a free β-pregroup is similar to the free pregroup — the reader may find details in [9]. Below we will give an illustration how β-pregroups work. Consider the following example first (with $n_{m4} \leq o_4$):

[(I) visited.	the boy .	whom .	(I) help]	
odwiedziłam	*chłopca,*	*któremu*	*pomagam*	
$s_2 o_4^l$	n_{m4}	$n_{m4}^r n_{m4} o_3^{ll} s_1^l$	$s_1 o_3^l$	$\to s_2$

It is worth noticing that the sentence has to be parsed in a specified order (to obtain the derivation ending in s_2), reflecting the semantic role of relative pronouns. One may observe that there are two possibilities of contracting $n_{m4} n_{m4}^r$ to 1 ($o_4^l o_4$ to 1 — while making use of our assumptions).

The first one is:

$s_2 o_4^l$	n_{m4}	$n_{m4}^r n_{m4} o_3^{ll} s_1^l$	$s_1 o_3^l$	$\to$
$s_2 o_4^l$		$n_{m4} o_3^{ll} s_1^l$	$s_1 o_3^l$	$\to$
$s_2 o_4^l$		$n_{m4} o_3^{ll}$	o_3^l	$\to$
$s_2 o_4^l$		n_{m4}		$\to$
$s_2 o_4^l$		o_4		$\to$
s_2				

The second one is:

$s_2 o_4^l$	n_{m4}	$n_{m4}^r n_{m4} o_3^{ll} s_1^l$	$s_1 o_3^l$	$\to$
$s_2 o_4^l$	o_4	$n_{m4}^r n_{m4} o_3^{ll} s_1^l$	$s_1 o_3^l$	$\to$
s_2		$n_{m4}^r n_{m4} o_3^{ll} s_1^l$	$s_1 o_3^l$	$\to$
s_2		$n_{m4}^r n_{m4} o_3^{ll}$	o_3^l	$\to$
s_2		$n_{m4}^r n_{m4}$		$\not\to s_2$

Of the two derivations above only one proved successful. Some kind of indeterminism can be observed here. It is clear from the example that the contraction of type $s_2 o_4^l$ of the verb in the main clause should be postponed until the type n_{m4} of the noun *chłopca* is contracted with the relative clause describing this noun. It reflects the semantic role of the relative clause as describing and adding information about the noun.

On the former approach the types from lexicon allowed two derivations — one of them undesirable. In order to block that problematic derivation one can use types enriched with β-s. Enriching our lexicon with types with modalities, one gets:

[(I) visited .	the boy.	whom .	(I) help]	
odwiedziłam	*chłopca,*	*któremu*	*pomagam.*	
$s_2 o_4^l$	$\beta(n_{m4})$	$[\beta(n_{m4})]^r n_{m4} o_3^{ll} s_1^l$	$s_1 o_3^l$	$\to s_2$

and the derivation is then as follows:

$s_2 o_4^l$	$\beta(n_{m4})$	$[\beta(n_{m4})]^r n_{m4} o_3^{ll} s_1^l$	$s_1 o_3^l$	$\to$
$s_2 o_4^l$		$n_{m4} o_3^{ll} s_1^l$	$s_1 o_3^l$	$\to$
$s_2 o_4^l$		$n_{m4} o_3^{ll}$	o_3^l	$\to$
$s_2 o_4^l$		n_{m4}		$\to$
$s_2 o_4^l$		o_4		$\to$
s_2				

The undesired derivation has been blocked by the β-operator, which solves the problem of nondeterministic derivations.

5. Product pregroup grammars

Lambek [17] proposed an analysis of English verbs which consisted in two parts: the first, called *inflector*, the second one, called *infinitive*. The verb is described by these two parts. The inflector is responsible for the conjugated form of the verb, whereas the infinitive contains additional information concerning specific verb features like transitivity, possible complementizers, etc. It means that the inflector modifies the infinitive and as a result one gets a type of the verb in a certain grammar tense, person and form.

An interesting approach was elaborated by Kusalik [12], who tried to separate the two aspects of the English verb. The author presented an alternative analysis which replaced Lambek's free pregroup with product pregroups, allowing for a separation between them. This could work better for the cases which were problematic for Lambek's former approach.

The product pregroup grammar is understood as an intersection of two (or more) pregroup grammars. In the lexicon this time one can find types of the first and the second (possibly more) pregroup grammars. The first can be seen as a usual pregroup grammar checking sentencehood on the syntactic level. The second can be responsible for feature checking, etc. The calculations must end successfully on both (more) levels in order to accept the string of words as a sentence. It has been shown, that if G_1 and G_2 are pregroup grammars, the language defined by the pregroup $G_1 \times G_2$ is an intersection of two context-free languages, and there is an algorithm of polynomial complexity for determining whether a

given string of types in $G_1 \times G_2$ reduces to $\mathbf{1}$. ($\mathbf{1}$ is understood as a vector of coordinates such that each coordinate belongs to the given pregroup grammar. It is usually $s \in P_1$ for the first grammar and $1 \in P_i$, $2 \leqslant i \leqslant k$ for the additional grammars.) Given a product of k free pregroups, the language can be expressed as the intersection of k context-free languages. The fact that the finite products of free pregroups are computationally no more complex than free pregroup itself means that they can be used as a model of a grammar structure of a given natural language.

Product pregroup grammars were used by Lambek [18] while trying to analyze feature agreement in French. He made use of two pregroup grammars, one for syntactic types and the other for feature types. Kusalik [12] used the product of three pregroup grammars for analyzing English sentences. We will show examples of this formalism having first reminded ourselves of some of the technical details.

According to Lambek [14] with each verb V we may associate a matrix $C_{jk}(V)$, with j referring to the tense and k referring to the person. For example the matrix for the verb *be* is given as follows:

$$C_{jk}\ (be) \rightarrow \begin{pmatrix} am & are & is \\ was & were & was \end{pmatrix}$$

Taking single elements of the matrix we get the following, where j (as well as i, $\bar{i}$, $\bar{j}$ etc.) stand for an infinitive clause and it is assumed from the lexicon that $i \leqslant j$):

$$\begin{array}{lcl} C_{13}(go) & = & goes \\ (\pi_3^r s_1 j^l) i & \rightarrow & \pi_3^r s_1 \end{array}$$

$$\begin{array}{lcl} C_{23}(go) & = & went \\ (\pi_3^r s_2 j^l) i & \rightarrow & \pi_3^r s_2 \end{array}$$

$$\begin{array}{lcl} C_{11}(like) & = & like \\ (\pi_1^r s_1 j^l)(i o^l) & \rightarrow & \pi_1^r s_1 o^l \end{array}$$

As the constructions that need product pregroup grammars in Polish language are quite rare, this approach is illustrated below with some Italian sentences. All types used while assigned to words come from Casadio [6, 5]. Consider first:

vedere	*il ragazzo*	[to see the boy]
vedere	*la ragazza*	[to see the girl]
vedere	*i ragazzi*	[to see the boys]
vedere	*le ragazze*	[to see the girls]
io^l	o	$\rightarrow i$

The inflectors used justify the Italian sentence, *Mario vede la ragazza* [Mario sees a girl], namely:

Mario	*vede*	*la ragazza.*	
Mario	C_{13}*(vedere)*	*la ragazza*	
π_3	$(\pi_3^r s_1 i^l)(io^l)$	o	$\to s_1$

When the accusative objects *il ragazzo, la ragazza, i ragazzi, le ragazze* are changed into personal pronouns in accusative case they become *lo, la, li, le*, respectively. Then, in Italian, their position in the sentence changes, as they need to be preverbal. Thus one gets:

lo	*vedere*	[to see him]
la	*vedere*	[to see her]
li	*vedere*	[to see them (masculine)]
le	*vedere*	[to see them (feminine)]
$jo^{ll}i^l$	io^l	$\to j$

Here j (as well as i, $\bar{i}$, $\bar{j}$ etc.) stand for infinitive clauses. All constraints and partial order concerning them are at the moment irrelevant and we will not bother the reader with the details.

Taking into consideration infinitive clauses in the present perfect tense, we get:

avere	*visto*	*il ragazzo*	[to have seen the boy]
avere	*visto*	*la ragazza*	[to have seen the girl]
avere	*visto*	*i ragazzi*	[to have seen the boys]
avere	*visto*	*le ragazze*	[to have seen the girls]
ip_2^l	$p_2 o^l$	o	$\to i$

Here p_2 stands for the past participle of the verb *vedere* [to see].

Now nouns in the accusative case *il ragazzo, la ragazza, i ragazzi, le ragazze* are changed into personal pronouns in the accusative case *lo, la, li, le* again. And this causes problems. This is due to some grammatical peculiarities of Italian: when using personal pronouns in accusative case together with present perfect tense the endings of past participle according to the pronoun's gender and number must be changed. It should look as follows:

lo	*avere*	*vist**o***	[to have seen him]
la	*avere*	*vist**a***	[to have seen her]
li	*avere*	*vist**i***	[to have seen them (masculine)]
le	*avere*	*vist**e***	[to have seen them (feminine)]
$jo^{ll}i^l$	ip_2^l	$p_2 o^l$	$\to j$

Now on we face over-generation. On syntactic grounds the wrong word order can be blocked. For example *avere visto lo* cannot be accepted in Italian. If personal pronouns are got rid of and replaced by objects in the form of nouns no changes should be made, and *avere visto* will be proper in all cases, irrespective of the gender and the number of the noun. Changing the form of past participle is only needed when using a personal pronoun instead of the noun in the present perfect tense. One can say *lo avere visto*, but *lo avere visti* would not be correct, because there is a lack of feature agreement. A similar problem arises when considering the present perfect tense of intransitive verbs which form present perfect forms with the verb *essere*. Here the endings of past participle must agree with the personal pronoun.

In thse cases product pregroup grammars seem to be applicable. Let us consider feature agreement. We introduce into to the lexicon of the second pregroup grammar four new types: π_{ms}, π_{fs}, π_{mp}, π_{fp}, where m, f stand for masculine and feminine, and s, p stand for singular and plural. Then the string of words from Italian can be typed on both levels (using the product of two pregroup grammars), and the computations are done in a parallel way. If both are successful, the sentence is accepted; otherwise it is rejected (in case one or two are wrong). The first type assignment (syntactic level) should end in a single s_j type (infinitive clause), while the second one (feature agreement) should end in 1. Making use of two free pregroups the following can be obtained:

lo	*avere*	*visto*	
$jo^{ll}i^{l}$	ip_2^l	p_2o^l	$\to j$
π_{ms}	1	π^r_{ms}	$\to 1$

lo	*avere*	*visti*	
$jo^{ll}i^{l}$	ip_2^l	p_2o^l	$\to j$
π_{ms}	1	π^r_{mp}	$\not\to 1$

As can be seen above, the second sentence is rejected on the second level and the string of words *lo avere visti* cannot be accepted. When considering French Lambek [18] made a lot of additional assumptions and tackled more complicated matters, but the above examples are enough to give the reader some ideas about this particular pregroup usage.

6. Tupled pregroup grammars

Another approach was proposed by Stabler in [20]. In his paper he presented the idea of tupled pregroup grammars and showed how to make use of them when treating some aspects of English grammar. According to Stabler the pregroup operations provide a simple feature checking mechanism and the tupling allows additional operations like movement. Languages definable in tupled pregroup grammars are weakly equivalent to multiple context-free grammars with the power to define for example mildly context sensitive languages like $\{xx \mid x \in \{a,b\}^*\}$.

In pregroup grammars the lexicon consists of ordered pairs whose first element is a symbol from the alphabet and whose second element is a type. Elements of the lexicon are also called lexical entries. In *tupled pregroup grammars* we use tuples of those ordered pairs. It can be motivated and explained by the fact that in natural languages certain words tend to occur together in the sentence, as for example prepositions with nouns, etc. Here the lexical entries are of the following form:

$$\begin{pmatrix} t_1 & \dots & t_k \\ s_1 & \dots & s_k \end{pmatrix}$$

or alternatively $t_1, \dots, t_k : s_1, \dots, s_k$. Here $s_1, \dots, s_k$ are elements of the alphabet and $t_1, \dots, t_k$ are types.

A merge operation applying to any pair of tuples is defined as follows:

$$\begin{pmatrix} t_1 & \dots & t_i \\ s_1 & \dots & s_i \end{pmatrix} \bullet \begin{pmatrix} t_{i+1} & \dots & t_k \\ s_{i+1} & \dots & s_k \end{pmatrix} = \begin{pmatrix} t_1 & \dots & t_k \\ s_1 & \dots & s_k \end{pmatrix}$$

An operation of deleting i-th coordinate, for any k-tuple $k > 0$ and any $1 \leqslant i \leqslant k$ is defined as follows:

$$\begin{pmatrix} t_1 & \dots & t_k \\ s_1 & \dots & s_k \end{pmatrix}_{-i} = \begin{pmatrix} t_1 & \dots & t_{i-1} & t_{i+1} & \dots & t_k \\ s_1 & \dots & s_{i-1} & s_{i+1} & \dots & s_k \end{pmatrix}$$

Let us define a binary relation on tupled pregroup grammar expressions, denoted by $\to_{TPG}$ that holds in the following cases, for any tuples e_1, e_2 and sequence of tuples α, β:

(Mrg) $\alpha \ e_1 \ e_2 \ \beta \Rightarrow_{TPG} \alpha \ e_1 \bullet e_2 \ \beta$

(Move) $\alpha \begin{pmatrix} t_1 & \dots & t_k \\ s_1 & \dots & s_k \end{pmatrix} \beta \Rightarrow_{TPG} \alpha \begin{pmatrix} t_i t_j \\ s_i s_j \end{pmatrix} \bullet \begin{pmatrix} t_1 & \dots & t_k \\ s_1 & \dots & s_k \end{pmatrix}_{-i-j} \beta$

(Move) applies to any k-tuple ($k > 1$), for any $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant k$.

The type in any coordinate can be contracted or expanded, for any a, b such that either $a \leq b$ and n is even or $b \leq a$ and n is odd.

$$\text{(GCon)} \quad \alpha \begin{pmatrix} \dots & \begin{matrix} xa^{(n)}b^{(n+1)}y \\ s \end{matrix} & \dots \end{pmatrix} \beta \Rightarrow_{TPG} \alpha \begin{pmatrix} \dots & \begin{matrix} xy \\ s \end{matrix} & \dots \end{pmatrix} \beta$$

$$\text{(GExp)} \quad \alpha \begin{pmatrix} \dots & \begin{matrix} xy \\ s \end{matrix} & \dots \end{pmatrix} \beta \Rightarrow_{TPG} \alpha \begin{pmatrix} \dots & \begin{matrix} xa^{(n+1)}b^{(n)}y \\ s \end{matrix} & \dots \end{pmatrix} \beta$$

Before going into details, an example will be presented in order to show how tupled pregroup grammars work. Let our lexicon be as follows:

$$I = \left\{ \begin{pmatrix} n_{m1} \\ profesor \end{pmatrix} \begin{pmatrix} n_{m1}n^l_{m1} \\ znany \end{pmatrix} \begin{pmatrix} n_{f2}\, n^l_{f2} \\ nowej \end{pmatrix} \begin{pmatrix} a_{f1} \\ ogromna \end{pmatrix} \begin{pmatrix} \pi^r_3 s_1 \lambda^l_{(1)} \\ jedzie \end{pmatrix} \right.$$
$$\begin{pmatrix} \alpha_{(1)} & n_{f2} \\ do & szkoły \end{pmatrix} \begin{pmatrix} \alpha_{(1)} & n_{f4} \\ na & konferencję \end{pmatrix} \begin{pmatrix} \pi^r_4 s_1 a^l_{f1} \\ jest \end{pmatrix} \begin{pmatrix} w & \pi_4 \\ która & \epsilon \end{pmatrix}$$
$$\left. \begin{pmatrix} w^r n^r_{f2} n_{f2} s^l_1 \\ \epsilon \end{pmatrix} \begin{pmatrix} n^r_{f2} \alpha^r_{(1)} \lambda_{(1)} \\ \epsilon \end{pmatrix} \begin{pmatrix} n^r_{f4} \alpha^r_{(1)} \lambda_{(1)} \\ \epsilon \end{pmatrix} \right\}$$

with: $n_{m1} \leq \pi_3$.

Word to word translations of the words are: *profesor* [*professor*], *znany* [*known*], *nowej* [*new* — second case (genitive) feminine singular], *ogromna* [*huge*, *very big* — first case (nominative) feminine singular], *jedzie* [*goes*], *do szkoły* [*to school*], *jest* [*is*], *która* [*which* — first case (nominative) feminine singular]. Using this formalism we can check the sentencehood of *profesor jedzie do szkoły* [a professor . goes . to . school]. The verification can be presented in two ways. The first is done using rewriting rules, the second as a derivation tree. They are as follows, with detailed comments right below each line:

$$\begin{pmatrix} n_{m1} \\ profesor \end{pmatrix} \begin{pmatrix} \pi^r_3 s_1 \lambda^l_{(1)} \\ jedzie \end{pmatrix} \begin{pmatrix} \alpha_{(1)} & n_{f2} \\ do & szkoły \end{pmatrix} \begin{pmatrix} n^r_{f2} \alpha^r_{(1)} \lambda_{(1)} \\ \epsilon \end{pmatrix} \Rightarrow$$

Applying (Mrg) to the third and the fourth tuple we get:

$$\begin{pmatrix} n_{m1} \\ profesor \end{pmatrix} \begin{pmatrix} \pi^r_3 s_1 \lambda^l_{(1)} \\ jedzie \end{pmatrix} \begin{pmatrix} \alpha_{(1)} & n_{f2} & n^r_{f2} \alpha^r_{(1)} \lambda_{(1)} \\ do & szkoły & \epsilon \end{pmatrix} \Rightarrow$$

We apply (Move) to the third tuple. First we pick the third and the second coordinate and then merge them with the rest of the tuple — in this case the rest means the first coordinate:

$$\begin{pmatrix} n_{m1} \\ profesor \end{pmatrix} \begin{pmatrix} \pi_3^r s_1 \lambda_{(1)}^l \\ jedzie \end{pmatrix} \begin{pmatrix} n_{f2} n_{f2}^r \alpha_{(1)}^r \lambda_{(1)} & \alpha_{(1)} \\ szkoły & do \end{pmatrix} \Rightarrow$$

We execute (GCon) within the third tuple and get:

$$\begin{pmatrix} n_{m1} \\ profesor \end{pmatrix} \begin{pmatrix} \pi_3^r s_1 \lambda_{(1)}^l \\ jedzie \end{pmatrix} \begin{pmatrix} \alpha_{(1)}^r \lambda_{(1)} & \alpha_{(1)} \\ szkoły & do \end{pmatrix} \Rightarrow$$

Again we apply (Move) to the third tuple. This time we pick the second and the first coordinate:

$$\begin{pmatrix} n_{m1} \\ profesor \end{pmatrix} \begin{pmatrix} \pi_3^r s_1 \lambda_{(1)}^l \\ jedzie \end{pmatrix} \begin{pmatrix} \alpha_{(1)} \alpha_{(1)}^r \lambda_{(1)} \\ doszkoły \end{pmatrix} \Rightarrow$$

We execute (GCon) within the third tuple and get:

$$\begin{pmatrix} n_{m1} \\ profesor \end{pmatrix} \begin{pmatrix} \pi_3^r s_1 \lambda_{(1)}^l \\ jedzie \end{pmatrix} \begin{pmatrix} \lambda_{(1)} \\ doszkoły \end{pmatrix} \Rightarrow$$

Applying (Mrg) to the second and the third tuple we get:

$$\begin{pmatrix} n_{m1} \\ profesor \end{pmatrix} \begin{pmatrix} \pi_3^r s_1 \lambda_{(1)}^l & \lambda_{(1)} \\ jedzie & doszkoły \end{pmatrix} \Rightarrow$$

Again we apply (Move) to the second tuple. This time we pick the first and the second coordinate:

$$\begin{pmatrix} n_{m1} \\ profesor \end{pmatrix} \begin{pmatrix} \pi_3^r s_1 \lambda_{(1)}^l \lambda_{(1)} \\ jedzie\ do\ szkoły \end{pmatrix} \Rightarrow$$

We execute (GCon) within the second tuple and get:

$$\begin{pmatrix} n_{m1} \\ profesor \end{pmatrix} \begin{pmatrix} \pi_3^r s_1 \\ jedzie\ do\ szkoły \end{pmatrix} \Rightarrow$$

Then we merge (Mrg) two remaining tuples:

$$\begin{pmatrix} n_{m1} & \pi_3^r s_1 \\ profesor & jedzie\ do\ szkoły \end{pmatrix} \Rightarrow$$

Then we (Move) the first and the second coordinate in the tuple:

$$\begin{pmatrix} n_{m1}\pi_3^r s_1 \\ \textit{profesor jedzie do szkoły} \end{pmatrix} \Rightarrow$$

Executing (GCon) within the tuple — making use of $n_{m1} \leq \pi_3$ — we get the sentence:

$$\begin{pmatrix} s_1 \\ \textit{profesor jedzie do szkoły} \end{pmatrix}$$

It is often more convenient to view the derivation of the sentence in the form of a derivation tree. The second way of presenting the sentence by means of tupled pregroup grammars is shown below. We do not give a line by line description here, only a general explanation. In what follows underbracing $\smile$ means making use of (Mrg), $\downarrow$ means applying (Move) and ($\Downarrow$) is used for (GCon).

$$\underbrace{\alpha_{(1)}, n_{f2} : \textit{do, szkoły} \qquad n_{f2}^r \alpha_{(1)}^r \lambda_{(1)} : \epsilon}$$

$$\alpha_{(1)}, n_{f2}, n_{f2}^r \alpha_{(1)}^r \lambda_{(1)} : \textit{do, szkoły}, \epsilon$$

$$\downarrow$$

$$n_{f2} n_{f2}^r \alpha_{(1)}^r \lambda_{(1)}, \alpha_{(1)} : \textit{szkoły, do}$$

$$\Downarrow$$

$$\alpha_{(1)}^r \lambda_{(1)}, \alpha_{(1)} : \textit{szkoły, do}$$

$$\downarrow$$

$$\alpha_{(1)} \alpha_{(1)}^r \lambda_{(1)} : \textit{do szkoły}$$

$$\Downarrow$$

$$\underbrace{\lambda_{(1)} : \textit{do szkoły} \qquad \pi_3^r s_1 \lambda_{(1)}^l : \textit{jedzie}}$$

$$\lambda_{(1)}, \pi_3^r s_1 \lambda_{(1)}^l : \textit{do szkoły, jedzie}$$

$$\downarrow$$

$$\pi_3^r s_1 \lambda_{(1)}^l \lambda_{(1)} : \textit{jedzie do szkoły}$$

$$\Downarrow$$

$$\underbrace{\pi_3^r s_1 : \textit{jedzie do szkoły} \qquad n_{m1} : \textit{profesor}}$$

$$\pi_3^r s_1, n_{m1} : \textit{jedzie do szkoly, profesor}$$

$$\downarrow$$

$$n_{m1} \pi_3^r s_1 : \textit{profesor jedzie do szkoły}$$

$$\Downarrow$$

$$s_1 : \textit{profesor jedzie do szkoły}$$

The problem of relative pronouns in Polish in terms of tupled pregroup grammars was presented in [11]. Another tough problem to solve by means of traditional pregroup grammars is the problem of prepositional phrases or adverbs. In earlier approaches every adverb or prepositional phrase was just given in the lexicon the type α, without making any distinction between different types. In English phrases like *to school, today, later, to school soon, fast* had type α in the lexicon, with the assumption that $\alpha\alpha = \alpha$. It can be easily seen that this led to overgeneralization. In [17] Lambek encountered the problem too. One can say *seen with her, seen by her, etc.* whereas *seen for her* would be doubtful and questionable.

When treating Polish sentences, the phrase *do szkoły* [to school] was simply given the type α. Some attempts to introduce to the lexicon more specified types for prepositions, like $\lambda_{(1)}n^l_{m2}$, $\lambda_{(1)}n^l_{f2}$, $\lambda_{(1)}n^l_{n2}$, etc. . . for the preposition *do*, were not fully successful. For example in the sentence *profesor jedzie do szkoły* [(a) professor . goes . to . school]:

profesor	*jedzie*	*do*	*szkoły*	
n_{m1}	$\pi^r_3 s_1 \lambda^l_{(1)}$	$\lambda_{(1)}n^l_{f2}$	n_{f2}	$\to s_1$

this works perfectly. Also for many other sentences like *profesor jedzie do domu* [(a) professor . goes . . home]. But in the sentence *profesor jedzie do konferencji* [(a) professor . goes . on . conference] it violates the rules concerning the use of the the noun *konferencja* in combination with prepositions. In Polish it is not correct — one can *jechać na konferencję*, *uczestniczyć w konferencji*, *być na konferencji*, etc., but not ~~*jechać do konferencji*~~. Similarly, trying to introduce types for the preposition *na* in the lexicon as follows: $\lambda_{(1)}n^l_{m4}$, $\lambda_{(1)}n^l_{f4}$, $\lambda_{(1)}n^l_{n4}$, etc., allows one to parse the sentences *profesor jedzie na konferencję*, but also ~~*profesor jedzie na dom*~~, which cannot be said in Polish.

Tupled pregroup grammars can be very useful for this case. Consider the preposition *do* [to] as an example. In order to create a phrase *do szkoły* [to school], we take two tuples from the lexicon.

$$\begin{pmatrix} \alpha_{(1)} & n_{f2} \\ do & szkoły \end{pmatrix} \begin{pmatrix} n^r_{f2}\alpha^r_{(1)}\lambda_{(1)} \\ \epsilon \end{pmatrix}$$

The type assigned to *do* gives us the information that this preposition may be used in creating a prepositional phrase answering the question *where to?* [superscripts of λ's and α's must be coherent], but it needs a complement in form of a noun of feminine gender in the accusative

singular. Combining those two tuples we get the phrase *do szkoły* with the type $\lambda_{(1)}$, as show below:

$$\begin{pmatrix} \alpha_{(1)} & n_{f2} \\ do & szkoły \end{pmatrix} \begin{pmatrix} n^r_{f2}\alpha^r_{(1)}\lambda_{(1)} \\ \epsilon \end{pmatrix} \Rightarrow$$

$$\begin{pmatrix} \alpha_{(1)} & n_{f2} & n^r_{f2}\alpha^r_{(1)}\lambda_{(1)} \\ do & szkoły & \epsilon \end{pmatrix} \Rightarrow$$

$$\begin{pmatrix} n_{f2}n^r_{f2}\alpha^r_{(1)}\lambda_{(1)} & \alpha_{(1)} \\ szkoły & do \end{pmatrix} \Rightarrow \begin{pmatrix} \alpha^r_{(1)}\lambda_{(1)} & \alpha_{(1)} \\ szkoły & do \end{pmatrix} \Rightarrow$$

$$\begin{pmatrix} \alpha_{(1)}\alpha^r_{(1)}\lambda_{(1)} \\ do\ szkoły \end{pmatrix} \Rightarrow \begin{pmatrix} \lambda_{(1)} \\ do\ szkoły \end{pmatrix}$$

What is more, we can also introduce an adjective between *do* and *szkoły*, in order to build a prepositional phrase *do nowej szkoły* [to . (the) new . school]. The verb and its type must be also involved. So one of the lexical entries for the word *jedzie* [goes] is:

$$\begin{pmatrix} \pi^r_3 s_1 \lambda^l_{(1)} \\ jedzie \end{pmatrix}$$

This approach is very useful as it can block unwanted prepositional phrases as well as their incorrect usage with inappropriate verbs. It can also be used with phrases *do której* [to which – feminine], *do którego* [to which — masculine, neuter] etc. Consider the sentence: *Lubię szkołę, do której jedzie* [(I) like . school . to . which . (he) goes]. As the phrase *do której* cannot be separated in any way (not as in English where one can say I like the school which he goes to), it may be treated in the lexicon as a single lexical entry that may occur in a certain position in the sentence. The lexical entries needed to parse this sentence will be as follows:

$$\begin{pmatrix} n_{f4} \\ szkołę \end{pmatrix} \begin{pmatrix} s_1 o^l_4 \\ lubię \end{pmatrix} \begin{pmatrix} \lambda^r_{(1)} s_1 \\ jedzie \end{pmatrix}$$

$$\begin{pmatrix} w & \lambda_{(1)} \\ do\ której & \epsilon \end{pmatrix} \begin{pmatrix} n^r_{f4} n_{f4} s^l_1 w^l \\ \epsilon \end{pmatrix}$$

Let us conclude this section with a derivation tree for that sentence:

$$\underbrace{w, \lambda_{(1)} : do\ której,\ \ \epsilon \qquad\qquad \lambda^r_{(1)} s_1 : jedzie}$$

$$
\begin{array}{c}
w, \lambda_{(1)}, \lambda^r_{(1)} s_1 : \textit{do której}, \epsilon, \textit{jedzie} \\
\downarrow \\
\lambda_{(1)} \lambda^r_{(1)} s_1, w : \textit{jedzie}, \textit{do której} \\
\Downarrow \\
\underbrace{s_1, w : \textit{jedzie}, \textit{do której} \qquad\qquad n^r_{f4} n_{f4} s^l_1 w^l : \epsilon}_{} \\
s_1, w, n^r_{f4} n_{f4} s^l_1 w^l : \textit{jedzie}, \textit{do której}, \epsilon \\
\downarrow \\
n^r_{f4} n_{f4} s^l_1 w^l w, s_1 : \textit{do której}, \textit{jedzie} \\
\Downarrow \\
n^r_{f4} n_{f4} s^l_1, s_1 : \textit{do której}, \textit{jedzie} \\
\downarrow \\
n^r_{f4} n_{f4} s^l_1 s_1 : \textit{do której jedzie} \\
\Downarrow \\
\underbrace{n_{f4} : \textit{szkołę} \qquad\qquad n^r_{f4} n_{f4} : \textit{do której jedzie}}_{} \\
n_{f4}, n^r_{f4} n_{f4} : \textit{szkołę}, \textit{do której jedzie} \\
\downarrow \\
n_{f4} n^r_{f4} n_{f4} : \textit{szkołę do ktorej jedzie} \\
\Downarrow \\
\underbrace{s_1 o^l_4 : \textit{lubię} \qquad\qquad n_{f4} : \textit{szkołę do której jedzie}}_{} \\
s_1 o^l_4, n_{f4} : \textit{lubię}, \textit{szkołę do której jedzie} \\
\downarrow \\
s_1 o^l_4 n_{f4} : \textit{lubię szkołę do której jedzie} \\
\Downarrow \\
s_1 : \textit{lubię szkołę do której jedzie}
\end{array}
$$

Note. All types w occurring in the tuples should be treated separately for each example. They are just auxiliary variables used in the types. In building the dictionary from lexical items we can use subscripts or different letters.

In all the examples above with tupled pregroup grammars some restrictions were needed. In order not to bother the reader with too many details they were omitted. They are as follows: all types used in tuples must be of the form $t^r_1 t^r_2 \dots t^r_k v w^k_1 w^l_2 \dots w^l_m$, for $k, m \geqslant 0$, (Mrg) can be applied to a pair of tuples only when in one tuple all types are of the form v (without left and right adjoints) and (Move) takes two items of a tuple if one of the types is of the form v.

7. Conclusion

We are currently observing an increasing interest in different kinds of pregroup grammars. Although treating all grammatical aspects and solving all problems may prove to be very hard it is undoubtedly worth trying. The range of grammatical aspects that have been treated by means of them so far is quite large. There is no doubt that pregroup grammars can help with really tricky aspects of natural languages. Tupled pregroup grammars, product pregroup grammars instead of pure calculus of pregroup grammars are an interesting kinds of approach that seem to work better for different linguistic situations.

Appendix

List of basic types in Polish. In analyzing Polish grammar we distinguish a number of types. Below we present the list of all types the reader will find in our examples.

In Polish we distinguish four declarative sentence forms:

- s_1 declarative sentence in the present tense;
- s_2 declarative sentence in the past tense;
- s_3 declarative sentence in the future tense;
- s_4 declarative sentence in the conditional mood.

The basic types of verbs are:

- i verb infinitive;
- $\bar{i}$ infinitive of modal verb.

The basic types of personal pronouns are of the form π_i, where:

- π_1 ja [I];
- π_2 ty [you – singular forms];
- π_3 on [he];
- π_4 ona [she];
- π_5 ono [it];
- π_6 my [we];
- π_7 wy [you – plural forms];
- π_8 oni [they – masculine and neuter forms];
- π_9 one [they – female forms].

The basic types of nouns are of the form n_{ki} or $\hat{n}_{ki}$, where:

n_{ki} noun in singular form
$\hat{n}_{ki}$ noun in plural form

and $k = m, f, n$, whereas $i = 1, 2, \ldots, 7$, where:

$k = m$ masculine noun;
$k = f$ feminine noun;
$k = n$ neuter noun;
$i = 1$ nominative case;
$i = 2$ genitive case;
$i = 3$ dative case;
$i = 4$ accusative case;
$i = 5$ instrumental case;
$i = 6$ locative case;
$i = 7$ vocative case.

Note. Polish differs from many other western European languages in how it handles nouns. First, it does not need to use definite and indefinite articles or determiners, whereas this is essential in English or German. Second, Polish nouns have to be inflected for case. In English there are no case inflections. In German one has to pay attention to articles only, and in some cases to the endings of the particular noun. Like German, Polish also syntactically distinguishes the genders of the nouns. There are three: masculine, feminine and neuter.

A noun is a collection of forms which are chosen (one at time) depending on the requirements of a sentence. Such forms may be reduced to seven cases (German has four).

Summing up, there are three quite important things to note about the Polish noun: its gender, its case and whether it is singular or plural. The above mentioned types fulfil the requirements. One can easily determine that *książkę* [book] with the type n_{f4} is a singular feminine noun in the fourth (accusative) case, whereas *dzieciom* [children] with the type $\hat{n}_{n3}$ is a neuter noun in the plural in the third (dative) case. More detailed explanation and examples may be found in [8].

Basic types concerning adjectives are of the type a_{ki} or $\hat{a}_{ki}$, where:

a_{ki} adjective in singular form
$\hat{a}_{ki}$ adjective in plural form

and $k = m, f, n$, whereas $i = 1, 2, \ldots, 7$, where the first index k indicates gender, and the second index i indicates case.

$k = m$ masculine adjective;
$k = f$ feminine adjective;
$k = n$ neuter adjective.

The meaning of i is the same as for nouns. We understand a_{ki} as an abbreviation for $n_{ki}n^l_{ki}$ and $\hat{a}_{ki}$ as an abbreviation for $\hat{n}_{ki}\hat{n}^l_{ki}$.

Note. The function of the declension form of adjectives is to express their agreement in case and number with the nouns which they qualify. In order to build a correct noun phrase the gender, number and case of an adjective must comply with the gender, number and case of the noun. It is therefore clear that the above proposed types comply with these requirements. More detailed explanation and examples may be found in [8].

The basic types concerning objects are of the type o_i, where:

o_2 genitive object; (kogo? czego?);
o_3 dative object; (komu? czemu?);
o_4 accusative object; (kogo? co?);
o_5 instrumental object; (kim? czym?);
o_6 locative object; (o kim? o czym?).

The basic types concerning prepositional phrases are:

$\lambda_{(1)}$ which answers the question *dokad?* [where to?]
$\lambda_{(2)}$ which answers the question *gdzie?* [where?]
$\lambda_{(3)}$ which answers the question *kiedy?* [when?]
$\lambda_{(4)}$ which answers the question *po co?* [what for?]
$\lambda_{(5)}$ which answers the question *dlaczego?* [why?]
...

Partial order. We will make use of the following partial ordering on types: $s_1 \leq s$, $s_2 \leq s$, $s_3 \leq s$, $s_4 \leq s$, if the tense of the sentence is irrelevant. That means that s is a sentence but we are not interested in the tense. And again, it cannot be done the other way round.

In our further work we also make use of the following:

$$n_{ki} \leq o_i,$$
$$\hat{n}_{ki} \leq o_i.$$

for $k = m, f, n$ and $i = 2, 3, \ldots, 6$.

This means that nouns may also play the role of objects in a sentence.

$$n_{m1} \leq \pi_3, \qquad \hat{n}_{m1} \leq \pi_8,$$
$$n_{f1} \leq \pi_4, \quad \text{and} \quad \hat{n}_{f1} \leq \pi_9,$$
$$n_{n1} \leq \pi_5, \qquad \hat{n}_{n1} \leq \pi_9.$$

This accounts for the fact that nouns in nominative cases also play the grammatical role of the subject.

Acknowledgments. I would like to thank an anonymous referee for helpful comments and suggestions on an earlier version of this paper.

References

[1] Buszkowski, W., "Lambek grammars based on pregroups", *Logical Aspects of Computational Linguistics*, LNAI 2099, Springer, 2001, 95–109.

[2] Buszkowski, W., "Sequent systems for compact bilinear logic", *Mathematical Logic Quarterly* 49, 5 (2003): 467–474.

[3] Buszkowski, W., and K. Moroz, "Pregroup grammars and context-free grammars", pages 1–22 in: *Computational Algebraic Approaches to Natural Language*, Polimetrica, 2008.

[4] Casadio, C., and J. Lambek, "An algebraic analysis of clitic pronouns in Italian", pages 110–124 in: *Logical Aspects of Computational Linguistics*, LNAI 2099, Springer, 2001.

[5] Casadio, C., "Applying pregroups to Italian statements and questions", *Studia Logica* 87 (2007).

[6] Casadio, C., "Agreement and cliticization in Italian: A pregroup analysis", pages 166–177 in: *Lecture Notes in Computer Science*, LNCS 6031, Springer, 2010.

[7] Fadda, M., "Toward flexible pregroup grammars", pages 95–112 in: *New Perspectives in Logic and Formal Linguistics*, Bulzoni Editore, Roma, 2002.

[8] Kiślak, A., "Pregroups versus English and Polish grammar", pages 129–154 in: *New Perspectives in Logic and Formal Linguistics*, Bulzoni Editore, Roma, 2002.

[9] Kiślak-Malinowska, A., "On the Logic of β-pregroups", *Studia Logica* 87 (2007): 321–340.

[10] Kiślak-Malinowska, A., "Polish language in terms of pregroups", pages 145–172 in: *Computational Algebraic Approaches to Natural Language*, Polimetrica, 2008.

[11] Kiślak-Malinowska, A., "Some aspects of Polish grammar in terms of tupled pregroups", *Linguistic Analysis* (2010): 93–119.

[12] Kusalik, T., "Product pregroups as an alternative to inflectors", pages 173–190 in: *Computational Algebraic Approaches to Natural Language*, Polimetrica, 2008.

[13] Lambek, J., "The mathematics of sentence structure", *The American Mathematical Monthly* 65 (1958): 154–170.

[14] Lambek, J., "Type grammars revisited", pages 1–27 in: *Logical Aspects of Computational Linguistics*, A. Lecomte, F. Lamarche and G. Perrier (eds.), LNAI 1582, Springer, Berlin, 1999.
[15] Lambek, J., "Type grammars as pregroups", *Grammars* 4 (2001): 21–39.
[16] Lambek, J., "Pregroups: a new algebraic approach to sentence structure", pages 39–54 in: *New Perspectives in Logic and Formal Linguistics*, Bulzoni Editore, Roma, 2002.
[17] Lambek, J., *From word to sentence*, Polimetrica, 2008.
[18] Lambek, J., "Exploring feature agreement in French with parallel pregroup computations", *Journal of Logic, Language and Information* (2009).
[19] Moroz, K., "Algorithmic questions for pregroup grammrs", PhD Thesis, Poznań 2010.
[20] Stabler, E., "Tupled pregroup grammars", pages 23–52 in: *Computational Algebraic Approaches to Natural Language*, Polimetrica, 2008.

ALEKSANDRA KIŚLAK-MALINOWSKA
Faculty of Mathematics and Computer Science
University of Warmia and Mazury
Olsztyn, Poland
akis@uwm.edu.pl