Categorial Grammars and Their Logics

Wojciech Buszkowski
Adam Mickiewicz University in Poznań

Published in: A. Garrido and U. Wybraniec-Skardowska (eds.), The Lvov-Warsaw School. Past and Present. Studies in Universal Logic, Birkhäuser, 2018, pp. 91–115.

Abstract

This paper surveys the development of categorial grammars (also called type grammars) with emphasis on type logics (i.e. logical calculi underlying these grammars) and their relation to the origin in Ajdukiewicz [4].

1 Introduction

In the modern literature Kazimierz Ajdukiewicz is commonly accepted as the father of categorial grammars: formal grammars assigning logical types to expressions. His seminal paper [4] provided a clear idea of these grammars. Ajdukiewicz acknowledged an impact of E. Husserl and S. Leśniewski. From Husserl [31] he took the idea of semantical categories which can be defined in terms of mutual substitution of expressions in meaningful or sentential contexts. From Leśniewski he took a classification of categories in basic categories and functor categories. Here I cannot cite any single publication of S. Leśniewski, since in his works he never wrote a longer passage on this matter. Ajdukiewicz and other authors cite [45], but this paper on new foundations of mathematics merely contains short notes on 'the theory of semantical categories'; different logical symbols are characterized as functors of a particular category (no special symbols for categories are introduced). It seems that more elaborated considerations only appeared in Leśniewski's oral lectures.

Ajdukiewicz introduced a system of indices for categories. This system is employed in his procedure for verifying the 'syntactic connexion' of expressions. In [4] he writes: "We shall base our work here on the relevant results of Leśniewski, adding on our part a symbolism, in principle applicable to almost all languages [and enabling us to build a calculus], which makes it possible to formally define and examine the syntactic connexion of a word pattern."[1] (The passage in brackets has been omitted in the English translation in [6]; I add it, since the word 'calculus' is quite important.) In fact, the indices for categories

[1]All citations from Ajdukiewicz are based on the English translations of Ajdukiewicz's original papers, collected in [6].

were introduced in Ajdukiewicz's earlier paper [3], where they were used in a semantical analysis of the problem of universals.

Let me briefly comment on terminology. The indices for categories will be called *types*, according to modern standards in logic. Categories can be understood as some sets of expressions or sets of ontological objects (having a common type). Although Leśniewski and Ajdukiewicz use the term 'semantical category' (after Husserl), the term 'syntactic(al) category' is more appropriate. This was noticed by Ajdukiewicz [5]: "The concept of semantical categories must be clearly distinguished from the concept of syntactical categories. The term 'semantical category' was introduced for the first time by Husserl; however, the concept he associated with it would correspond better to the term 'syntactical category'. For Husserl pointed out that the expressions of a language may be classified according to the role they can play within a sentence. He defined, therefore, the categories from the syntactical viewpoint." Ajdukiewicz [5] outlined a theory of semantical categories: the type of an expression is determined by the ontological type of the denotation of this expression. The first semantical interpretation of [4] is due to Bocheński [13].

The connections of syntactic (semantic) types with categories defined by mutual substitution are by no way obvious, nor simple. They are quite tight for deterministic (or: rigid) grammars which assign at most one type to one expression, but become less regular for categorially ambiguous grammars. A thorough discussion of this topic can be found in [19] .

The present paper focuses on categorial grammars: how they developed from the origin in [4] to their modern forms. Categorial grammars are also called 'type grammars', and the latter term seems better. The classification of expressions in categories appears in different grammar formalisms (e.g. phrase structure grammars), whereas logical types assigned to expressions are characteristic of the grammars considered here. To emphasize this Morrill [51] and others use an even more explicit term 'type logical grammar'. In the present paper both terms are used: the former in traditional names of grammars, the latter in general considerations.

The impact of [4] can be seen in several areas of formal linguistics and logical philosophy of language. Type grammars belong to formal linguistics, since their main intention is to describe natural language. They are closely related to type-theoretic semantics of natural language, initiated by Monatgue [46], with an explicit reference to [4], and extensively studied by many authors as Montague Grammar. Some other works may be counted to the logical turn: they study types and categories in formal languages of logic and mathematics. This direction was represented in Poland by Suszko [62, 63], Wybraniec-Skardowska [67] and others. Suszko elaborated a formal framework for syntax and semantics of higher-order languages. Wybraniec-Skardowska presented a general theory of 'categorial languages' with a distinction between expression-tokens and abstract expressions (this theory, however, does not directly address natural language). Tałasiewicz [64] provided a philosophical analysis of Ajdukiewicz's approach, applied to natural language, with an interpretation in terms of situation semantics, It is a bit surprising that just the linguistic turn leads to new logical calculi (Lambek logics) and models (residuated algebras), whereas the logical turn usually focuses on some standard logical systems (higher-order logics, type theories). To keep this survey in a reasonable size I will mainly write on the new logics elaborated for type grammars and only briefly note some links with other developments. Montague Grammar and its descendants cannot be discussed in detail; the reader is referred to [12] and the items cited there.

This survey is addressed to a wide community, not necessarily experts in type grammars. Therefore I omit mathematical subtleties. I do not even discuss all important mathematical results in this area; I only briefly note some of them to clarify the main ideas. Nonetheless an acquaintance with general logic and formal linguistics may help the reader to follow the text. I provide a couple of linguistic examples, but all are quite simple. The reader is referred to [50, 49, 51, 52, 44] for a more advanced linguistic material.

Section 2 is concerned with basic categorial grammars, a framework directly related to Ajdukiewicz's proposal (modified in [8]). Section 3 discusses the Lambek calculus and several analogous systems with a particular emphasis on their role in type grammars. At the end, I defend the view that Lambek logics are important, general logics of syntactic and semantic types (besides other applications), but not as good for efficient parsing. An optimal strategy seems the following: (1) to apply Lambek logics in metatheory and on the lexical level, (2) to preserve the Ajdukiewicz system as a parsing procedure for compound expressions.

2 Basic categorial grammars

Ajdukiewicz (following Leśniewski) distinguishes two basic categories: sentence (type s) and name (type n), but stipulates that in general "nothing could be decided about the number and kind of basic semantic [categories] and functor categories, since these may vary in different languages." The types of functor categories have the form of fractions:

$$\frac{\alpha}{\beta_1 \ldots \beta_n}\,;$$

an expression of this type with arguments of type $\beta_1, \ldots, \beta_n$ forms a compound expression of type α. For example, an intransitive verb is of type $\frac{s}{n}$, a transitive verb of type $\frac{s}{n\,n}$, a sentential connective of type $\frac{s}{s\,s}$, an adverb of type $\frac{\alpha}{\alpha}$ for $\alpha = \frac{s}{n}$, and so on.

The procedure of checking the 'syntactic connexion' of a compound expression is designed as follows. First, the expression is rewritten in prefix notation: each functor directly precedes its arguments. So one writes `likes John wine` instead of `John likes wine` and `hardly works John` instead of `John works hardly` (my examples). Second, one considers the sequence of types corresponding to the words of the rearranged expression. For these two examples one obtains the sequences:

$$\frac{s}{n\,n}, n, n \text{ and } \frac{\frac{s}{n}}{\frac{s}{n}}, \frac{s}{n}, n.$$

Third, one reduces a block of adjacent types:

$$\frac{\alpha}{\beta_1 \ldots \beta_n}, \beta_1, \ldots, \beta_n$$

to α and repeats this step as many times, as possible. If this reduction ends in a single type, the expression is qualified to be 'syntactically connected' and assigned the resulting type. The Ajdukiewicz reduction procedure applied to our examples yields:

$$\frac{s}{n\,n}, n, n \Rightarrow s \text{ in one step,}$$

$$\frac{\frac{s}{n}}{\frac{s}{n}}, \frac{s}{n}, n \Rightarrow \frac{s}{n}, n \Rightarrow s \text{ in two steps.}$$

So both expressions are syntactically connected (of type s). In fact, Ajdukiewicz's original procedure was more restrictive: at each step one reduces the left-most occurrence of a reducible pattern, but this constraint narrows its applications [17].

This approach reveals two characteristic components of modern type grammars: (1) *the type lexicon*, i.e. an assignment of types to words, (2) *the type processing machinery*, i.e. a procedure of checking the grammatical correctness of arbitrary expressions and at the same time deriving types of them. In terms of contemporary computational linguistics, (2) is *a parsing procedure.* Ajdukiewicz was the first who clearly formulated the problem of parsing and proposed a parsing algorithm (twenty years before mathematical linguistics was founded by Noam Chomsky).

The Ajdukiewicz procedure requires the rewriting of the parsed expression in prefix notation. In practice this restricts its applications to some formal languages. In fact Ajdukiewicz acknowledged that one of his goals was a generalization of the parenthesis-free notation, elaborated by J. Łukasiewicz for propositional logics, toward richer formal languages. On the other hand, his examples came from natural languages, and he expected a wide applicability of his method. Probably he admitted various modifications of the original procedure, when applied in practice.

Bar-Hillel [7] adjusted this approach to natural language. He introduced *directional* types of the form:

$$\frac{\alpha}{\beta_1 \ldots \beta_m; \gamma_1 \ldots \gamma_n}$$

and the reduction procedure based on the rule:

$$\beta_1, \ldots, \beta_m, \frac{\alpha}{\beta_1 \ldots \beta_m; \gamma_1 \ldots \gamma_n}, \gamma_1, \ldots, \gamma_n \Rightarrow \alpha.$$

Now transitive verbs are assigned type $\frac{s}{n;n}$, and `John likes Mary` is parsed as:

$$n,\ \frac{s}{n;n},\ n \Rightarrow s \text{ in one step.}$$

In [8], this approach was modified. After Lambek [40], functor types were restricted to $\alpha\backslash\beta$ and α/β. An expression of type $\alpha\backslash\beta$ (resp. β/α) with an argument of type α on the left (resp. on the right) forms a compound expression of type β. So $\alpha\backslash\beta$ corresponds to $\frac{\beta}{\alpha;}$ in the former notation, α/β to $\frac{\alpha}{;\beta}$, and the fraction $\frac{\alpha}{\beta;\gamma}$ is represented as $\beta\backslash(\alpha/\gamma)$ or $(\beta\backslash\alpha)/\gamma$. The representation of many-argument types by (nested) one-argument types is closely related to 'currying', i.e. the representation of many-argument functions by one-argument functions of higher order, a routine in modern type theories.

The reduction procedure is based on two rules:

$$\text{(RED.1) } \alpha, \alpha\backslash\beta \Rightarrow \beta,\ \text{(RED.2) } \alpha/\beta, \beta \Rightarrow \alpha\,.$$

In [8], *a categorial grammar* is formally defined as a triple $G = (\Sigma, I, s)$ such that Σ is a nonempty finite set, I is a finite relation between elements of Σ and types, and s is an atomic type. The elements of Σ are interpreted as the words of a natural language (then Σ is referred to as *the lexicon*) or symbols of a formal language (then Σ is referred to as *the alphabet*). Nowadays I is called *the type lexicon* or *the initial type assignment.* Often I is represented as a map which assigns finite sets of types to elements of Σ. In examples we write $v : \alpha$ for $\alpha \in I(v)$. One refers to s as *the designated type.* One admits an arbitrary finite set of atomic types.

Finite sequences of elements of Σ are called *strings* (on Σ). The empty string is denoted by ϵ. The string $(v_1, \ldots, v_n)$ is usually written as $v_1 \ldots v_n$. One says that G *assigns* type α to the string $v_1 \ldots v_n$, if there exist types $\alpha_1, \ldots, \alpha_n$, belonging to $I(v_1), \ldots, I(v_n)$, respectively, such that the sequence $\alpha_1, \ldots, \alpha_n$ reduces to α by finitely many applications of rules (RED.1), (RED.2). *The language of* G consists of all strings on Σ which are assigned type s by G.

In the modern literature, categorial grammars in the sense of [8] are called *basic categorial grammars* (BCGs) or: classical categorial grammars, AB-grammars (a credit to Ajdukiewicz and Bar-Hillel).

The main mathematical theorem of [8] establishes the weak equivalence of BCGs and Chomsky's ($\epsilon-$free) context-free grammars (CFGs). Recall that a CFG is defined as a quadruple $G = (\Sigma, N, s, P)$ such that Σ and N are disjoint finite sets (whose elements are treated as simple symbols), $s \in N$, and P is a finite set of pairs (a, x), where $a \in N$ and x is a string on $\Sigma \cup N$. The elements of Σ (resp. N) are called *terminal symbols* (resp. *nonterminal symbols* or *variables*), and s is called *the start symbol.* The pairs in P are called *production rules.* one writes $a \mapsto x$ for (a, x) and interprets it as a rewriting rule: the string yaz can be rewritten as yxz according to this rule (by xy one denotes the concatenation of x and y). *The language of* G (or: generated by G) consists of all strings on Σ which can be derived from s by finitely many applications of the production rules. A CFG is $\epsilon-$*free*, if it contains no nullary rule of the form

$a \mapsto \epsilon$. The equivalence theorem states that BCGs and $\epsilon-$free CFGs generate the same class of languages. More precisely, for any BCG G there exists an $\epsilon-$free CFG G' such that $L(G) = L(G')$, and conversely.

The first part of this theorem can easily be proved: a BCG $G = (\Sigma, I, s)$ generates the same language as the CFG with the terminal alphabet Σ, the nonterminal alphabet consisting of all types involved in G and their subtypes (i.e. subterms), the start symbol s, and the production rules reversing (RED.1), (RED.2) (restricted to the types in the nonterminal alphabet) plus the lexical rules $\alpha \mapsto v$, for $\alpha \in I(v)$. The second part is more difficult; the proof in [8] yields, in fact, the Greibach normal form theorem for CFGs (independently proved a few years later).

In opposition to CFGs, BCGs are *lexical*: the whole information on the described language is contained in the type lexicon, whereas the parsing procedure is independent of this particular language (it employs the language-independent rules (RED.1), (RED.2)). This is not the case for CFGs. For instance, a standard non-lexical rule for English is $s \mapsto np, vp$ (a sentence consists of a noun phrase and a verb phrase). The lexicality is a characteristic feature of all type grammars, considered nowadays. Sometimes it is convenient to admit certain simple non-lexical rules, e.g. $pn \Rightarrow np$ (a proper noun is a noun phrase), but one tends to eliminate them, whenever possible.

The nonterminal symbols of a CFG can be interpreted as names of syntactic categories, like types in a BCG. Types, however, can be compound terms, not just simple symbols. This is significant for lexicality and makes it possible to study logics of types, expressing deeper relations between types.

Although CFGs are weakly equivalent to BCGs, the strong equivalence does not hold; this means that the structured languages differ for the two classes of grammars. For a CFG, each derivation of a string from a nonterminal symbol determines a unique *phrase structure* of this string. For instance, the grammar with production rules:

$$s \mapsto np, vp \quad vp \mapsto tv, np$$

$$np \mapsto \text{ John } \; np \mapsto \text{ tee } \; tv \mapsto \text{ drinks}$$

admits the derivation:

$$s \Rightarrow np, vp \Rightarrow np, tv, np \Rightarrow \cdots \Rightarrow \text{ John, drinks, tee}$$

which yields the phrase structure (John (drinks tee)) or, more explicitly, (John$_{np}$ (drinks$_{tv}$ tee$_{np}$)$_{vp}$)$_s$. These phrase structures can be depicted as binary trees; see Figure 1.

Similarly, each reduction in a BCG gives rise to a unique phrase structure of the input string. With the type lexicon:

$$\text{John} : np \;\; \text{drinks} : (np\backslash s)/np \;\; \text{tee} : np$$

one obtains the reduction:

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

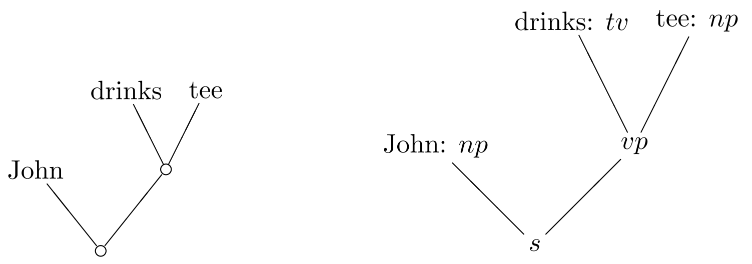

Figure 1: Phrase structures as binary trees

which yields the same phrase structure (John (drinks tee)). We need an auxiliary notion. *The degree* of type α, denoted by $d(\alpha)$, is defined as follows: $d(\alpha) = 0$ if α is atomic, $d(\alpha\backslash\beta) = d(\beta/\alpha) = d(\beta) + 1$. For any phrase structure generated by a BCG G, depicted as a tree, and for any node of this tree, the length of shortest paths from this node to a leaf is not greater than the maximal degree of types involved in G. Therefore a BCG cannot generate languages of phrase structures with arbitrarily long shortest paths from a node to a leaf. On the contrary, a CFG can generate such languages.

For instance, the CFG with rules $s \mapsto s, s$ and $s \mapsto 0$ generates all possible phrase structures on the alphabet $\{0\}$. The BCG with the type lexicon $0 : s/s$, $0 : s$ generates the same language of strings, which consists of all nonempty strings on $\{0\}$, but not the same language of phrase structures. One only gets the phrase structures: 0, (00), (0(00)), (0(0(00))), and so on, but not ((00)0).

For BCGs, one also considers *functor-argument structures* (fa-structures), i.e. phrase structures augmented with functor markers. For the BCG considered above, the phrase structure (John (drinks tee)) can be refined to the fa-structure (John (drinks tee)$_1$)$_2$, which means that (drinks tee) is the functor in the whole structure and `drinks` is the functor in (drinks tee). Every reduction in a BCG determines a unique fa-structure of the recognized string. The languages of fa-structures and phrase structures take an essential part in the theory of BCGs; see [15, 19]. In particular, syntactic categories can be defined as certain sets of fa-structures rather than strings, which results in a more elegant theory.

The type lexicon of a BCG can assign several types to one word. This reflects *the syntactic ambiguity* of words in natural language. For instance, `and` appears as a sentential connective, but also as a noun connective, verb connective, adverb connective, and others. As a rule, in logical and mathematical formalisms one symbol can be assigned a unique type, which completely characterizes the syntactic role of this symbol. These languages can be described by *rigid* (or: deterministic) BCGs (I is a function from Σ to the set of types).

Worthy of noting, not all languages of formal logic can be described by rigid BCGs. The standard example is the language of (type-free) lambda calculus. Also in the language of first-order logic, a unary function symbol f requires two types t/iv, t/t, where iv is the type of individual variables and t of terms (quantifiers are typed $(s/s)/iv$, where s is the type of formulas). Alternatively, one can assign only t/t to f and admit a non-lexical rule $iv \Rightarrow t$.

The type of quantifiers, given above, adequately characterizes their role in the syntax of first-order logic (in modern setting): the quantifier followed by a variable, next by a formula, yields a formula. It is also fully compatible with Tarskian semantics for this logic. It, however, does not express the variable-binding role of quantifiers. The final part of [4] is devoted to the special status of variable-binding operators, and several authors continue this issue; see [62, 63, 53, 67]. I do not discuss this matter here, since it goes too far from the main topics of this paper.

One of the leitmotives of type grammars is a close relationship between syntax and semantics (the dictum *syntax mirrors ontology*). I have already noted that Bocheński [13] proposed the semantical interpretation of the theory of Leśniewski and Ajdukiewicz, and this turn was adopted by Ajdukiewicz [5]. According to the latter, the basic types are i (individual) and w (truth value; 'value' corresponds to Polish 'wartość' and German 'Wert'). Intransitive verbs are typed $\frac{w}{i}$, as they denote functions from the set of individuals to the set of truth values, transitive verbs $\frac{w}{i\,i}$, as they denote two-argument functions of this kind, (binary) sentential connectives $\frac{w}{w\,w}$, as they denote binary truth-value functions, and so on. [5] brings a radical idea of a *purely flectional* language: the types of words only account for semantical categories of these words (i.e. the ontological status of their denotations), whereas their syntactic roles are described by certain new indices, indicating the position of these words in syntactic trees (some representations of fa-structures). This idea seems very interesting, but the symbolism, proposed in [5], has a limited value, since the new indices show the positions of words in one particular tree, not in any well-formed syntactic tree, containing the given word.

On the other hand, the semantical interpretation is quite fundamental and - modulo terminology and notation - has been commonly adopted in modern type-theoretic semantics. Syntactic types are translated into *semantic types*: atomic types and compound types $\alpha \to \beta$, where α, β are simpler types. Each atomic type p corresponds to a semantic domain (or: ontological category) D_p. One defines $D_{\alpha\to\beta}$ as the set of all functions from D_α to D_β. For instance, s is translated into t (the type of truth values) and n into e (the type of entities). Let $\alpha^\bullet$ denote the translation of α. One recursively defines:

$$(\alpha\backslash\beta)^\bullet = (\beta/\alpha)^\bullet = \alpha^\bullet \to \beta^\bullet.$$

So $n\backslash s$ is translated into $e \to t$ (the type of sets of entities, identified with their characteristic functions), $s/(n\backslash s)$ into $(e \to t) \to t$ (the type of families of sets of entities), and so on. The latter agrees with the interpretation of complete noun phrases as generalized quantifiers; see van Benthem [10].

In semantics, the reduction rules (RED.1), (RED.2) can be interpreted as the application of a function $f \in D_{\alpha^\bullet\to\beta^\bullet}$ to an argument $a \in D_{\alpha^\bullet}$, which yields $f(a) \in D_{\beta^\bullet}$. Thus, given some fixed denotations of all words of the parsed expression, whose semantic types correspond to their syntactic types, as above, one can determine the denotation of this expression by the (iterated) application of functions to their arguments, following the syntactic reduction procedure. This fully agrees with *the principle of compositionality*, a central idea of logical semantics.

3 Lambek Calculus

3.1 Basic systems

An essential refinement of BCGs is due to Lambek [40]. His Syntactic Calculus, nowadays called Lambek Calculus and denoted by **L**, is regarded as a basic type logic. Lambek presented his system as an improvement of BCGs: "[...] this paper is concerned with a development of the technique of Ajdukiewicz and Bar-Hillel in a mathematical direction. We introduce a calculus of types, which is related to the well-known calculus of residuals. The decision procedure for this system is solved affirmatively, following a procedure first proposed by Gentzen for the intuitionistic propositional calculus."

Types are built from atomic types by $\backslash, /$ and $\cdot$ (product; some authors write $\otimes$). An axiomatization of **L** employs *simple sequents* of the form $\alpha \Rightarrow \beta$, where α, β are types. **L** admits the following axioms and inference rules.

$$\text{(Id) } \alpha \Rightarrow \alpha$$

$$\text{(A.1) } (\alpha \cdot \beta) \cdot \gamma \Rightarrow \alpha \cdot (\beta \cdot \gamma) \quad \text{(A.2) } \alpha \cdot (\beta \cdot \gamma) \Rightarrow (\alpha \cdot \beta) \cdot \gamma$$

$$\text{(Res.1) } \frac{\alpha \cdot \beta \Rightarrow \gamma}{\beta \Rightarrow \alpha\backslash\gamma} \quad \text{(Res.2) } \frac{\alpha \cdot \beta \Rightarrow \gamma}{\alpha \Rightarrow \gamma/\beta}$$

$$\text{(Cut.1) } \frac{\alpha \Rightarrow \beta \quad \beta \Rightarrow \gamma}{\alpha \Rightarrow \gamma}$$

The double line in (Res.1), (Res.2) means that these rules can be used in both directions: top-down and bottom-up.

By dropping the associativity axioms (A.1), (A.2), one obtains Nonassociative Lambek Calculus (**NL**), due to Lambek [41]. The counterparts of (RED.1), (RED.2):

$$\text{(Red.1) } \alpha \cdot (\alpha\backslash\beta) \Rightarrow \beta\,, \quad \text{(Red.2) } (\alpha/\beta) \cdot \beta \Rightarrow \alpha$$

are provable in **NL**, using (Id), (Res.1), (Res.2). We, however, obtain (infinitely) many other laws. Here are some examples.

(L1) $\alpha \Rightarrow (\beta/\alpha)\backslash\beta$ and $\alpha \Rightarrow \beta/(\alpha\backslash\beta)$,

(L2) $\alpha \Rightarrow \beta\backslash(\beta \cdot \alpha)$ and $\alpha \Rightarrow (\alpha \cdot \beta)/\beta$,

(L3) $(\alpha\backslash\beta) \cdot (\beta\backslash\gamma) \Rightarrow \alpha\backslash\gamma$ and $(\alpha/\beta) \cdot (\beta/\gamma) \Rightarrow \alpha/\gamma$,

(L4) $\alpha\backslash\beta \Rightarrow (\gamma\backslash\beta)\backslash(\gamma\backslash\alpha)$ and $\alpha/\beta \Rightarrow (\alpha/\gamma)/(\beta/\gamma)$,

(L5) $(\alpha\backslash\beta)/\gamma \Leftrightarrow \alpha\backslash(\beta/\gamma)$ ($\Leftrightarrow$ stands for both $\Rightarrow$ and $\Leftarrow$).

(L1), (L2) are provable in **NL**, but (L3), (L4), (L5) in **L** only. Other laws can be obtained, by using the monotonicity rules: from $\alpha \Rightarrow \beta$ infer $\gamma \cdot \alpha \Rightarrow \gamma \cdot \beta$, $\alpha \cdot \gamma \Rightarrow \beta \cdot \gamma$, $\gamma \backslash \alpha \Rightarrow \gamma \backslash \beta$, $\beta \backslash \gamma \Rightarrow \alpha \backslash \gamma$, $\alpha / \gamma \Rightarrow \beta / \gamma$, $\gamma / \beta \Rightarrow \gamma / \alpha$, which are derivable in both systems.

The most general algebraic models of **NL** are *residuated groupoids*, i.e. ordered algebras $(A, \cdot, \backslash, /, \leq)$ such that $(A, \leq)$ is a partially ordered set, and $\cdot, \backslash, /$ are binary operations on A, satisfying *the residuation laws*:

$$\text{(RES) } a \cdot b \leq c \text{ iff } b \leq a \backslash c \text{ iff } a \leq c/b, \text{ for all } a, b, c \in A\,.$$

The operations $\backslash, /$ are called *the residual operations* for product. *Residuated semigroups* are residuated groupoids such that $\cdot$ is associative; they are models for **L**. Both systems are *strongly complete* with respect to the corresponding models: the sequents provable in the system from a set of nonlogical hypotheses are precisely those sequents which are true in all models, for all valuations μ, satisfying the hypotheses. $\alpha \Rightarrow \beta$ is *true* for μ, if $\mu(\alpha) \leq \mu(\beta)$.

According to Lambek [40], the intended models for **L** are *language models*, i.e. some algebras of languages (by a language one means a set of strings). By Σ^+ we denote the set of all nonempty strings on Σ. For $L_1, L_2 \subseteq \Sigma^+$, one defines:

$$L_1 \cdot L_2 = \{xy : x \in L_1,\, y \in L_2\}\,,$$
$$L_1 \backslash L_2 = \{y \in \Sigma^+ : xy \in L_2 \text{ for any } x \in L_1\},$$
$$L_1 / L_2 = \{x \in \Sigma^+ : xy \in L_1 \text{ for any } y \in L_2\},$$

where xy denotes the concatenation of strings x and y. It is easy to show that the powerset of Σ^+ with $\cdot, \backslash, /$ defined as above and inclusion as the order, is a residuated semigroup. $\alpha \Rightarrow \beta$ is true for μ in this model if and only if $\mu(\alpha) \subseteq \mu(\beta)$ (equivalently: every string of type α is of type β). The term 'language model' is due to Pentus [56]; this paper shows *the weak completeness* of **L** with respect to language models (the sequents provable in **L** are precisely those which are valid in all language models). The strong completeness does not hold, but it holds for the product-free **L** [14].

Analogously, the intended models for **NL** are algebras of languages consisting of phrase structures. Let Σ^P denote the set of all phrase structures on Σ. On the powerset of Σ^P one defines $\cdot, \backslash, /$ as above except that Σ^+ is replaced by Σ^P and xy by (x, y). The weak and the strong completeness (with respect to these models) hold for the product-free fragment of **NL** only [25, 36].

The intended models exhibit Lambek's interpretation of categories, which is not the same as in BCGs. For a BCG, the category of type α consists of all (structured) expressions which are assigned this type by the grammar. According to Lambek, the basic categories, i.e. those which are assigned atomic types, generate all other categories by operations $\cdot, \backslash, /$, interpreted in the algebra of languages. In particular, if y is of type $\alpha \backslash \beta$ (resp. β / α), then, for any x of type α, xy (resp. yx) is of type β in a BCG. Lambek replaces 'if ... then' by 'if and only if'. This is an essential difference; it leads to new reduction patterns, like (L1)-(L5), not admitted in BCGs.

This novel understanding of types caused, probably, a relatively small impact of Lambek's approach on his contemporaries. Only in the 1980-ties there began more systematic studies in Lambek calculi and their role in type grammars and type-theoretic semantics, initiated by W. Zielonka and the present author in Poznań and J. van Benthem and his students (especially M. Moortgat) in Amsterdam. This research was reported in two collection volumes [54, 21]; the second one also contains reprints of some earlier papers. The books [15, 47] elaborate on logical and algebraic properties of Lambek calculi and grammars.

Lambek grammars are defined like BCGs except that the reduction procedure is replaced with the provability in **L**, **NL** or a related system. One employs sequents of the form $\alpha_1, \ldots, \alpha_n \Rightarrow \beta$; in algebras, each comma is interpreted as product. For nonassociative systems, the antecedents of sequents take the form of bracketed sequences, e.g. $(\alpha, (\beta, \gamma))$, which is different from $((\alpha, \beta), \gamma)$. So (Red.1), (Red.2) can be written as (RED.1), (RED.2), and similarly for other laws. WARNING: (RED.1), (RED.2) have been called reduction rules in Section 2, but now the term 'rule' is reserved for inference rules of type logics, e.g. (Res.1), (Res.2), (Cut.1), whereas the provable sequents are referred to as laws.

Both **L** and **NL** can be presented as sequent systems [40, 41]. For **L**, the axioms are (Id) and the inference rules are as follows (Γ and Δ stand for finite, possibly empty, sequences of types).

$$(\cdot \Rightarrow)\ \frac{\Gamma, \alpha, \beta, \Gamma' \Rightarrow \gamma}{\Gamma, \alpha \cdot \beta, \Gamma' \Rightarrow \gamma} \quad (\Rightarrow \cdot)\ \frac{\Gamma \Rightarrow \alpha \quad \Delta \Rightarrow \beta}{\Gamma, \Delta \Rightarrow \alpha \cdot \beta}$$

$$(\backslash \Rightarrow)\ \frac{\Gamma, \beta, \Gamma' \Rightarrow \gamma \quad \Delta \Rightarrow \alpha}{\Gamma, \Delta, \alpha\backslash\beta, \Gamma' \Rightarrow \gamma} \quad (\Rightarrow \backslash)\ \frac{\alpha, \Gamma \Rightarrow \beta}{\Gamma \Rightarrow \alpha\backslash\beta}$$

$$(/ \Rightarrow)\ \frac{\Gamma, \alpha, \Gamma' \Rightarrow \gamma \quad \Delta \Rightarrow \beta}{\Gamma, \alpha/\beta, \Delta, \Gamma' \Rightarrow \gamma} \quad (\Rightarrow /)\ \frac{\Gamma, \beta \Rightarrow \alpha}{\Gamma \Rightarrow \alpha/\beta}$$

$$(\mathrm{Cut})\ \frac{\Gamma, \alpha, \Gamma' \Rightarrow \beta \quad \Delta \Rightarrow \alpha}{\Gamma, \Delta, \Gamma' \Rightarrow \beta}$$

One assumes that Γ is nonempty in $(\Rightarrow \backslash)$, $(\Rightarrow /)$. In sequents, one omits outer parentheses of antecedent sequences and writes Γ, Δ for the concatenation of Γ and Δ.

The sequent system for **NL** is similar. The antecedents of sequents are bracketed sequences of types, hence all rules look a bit differently; see [15, 49].

Clearly these systems are certain intuitionistic sequent systems, types play the role of formulas, and atomic types of variables (or nonlogical constants). The rules $(\cdot \Rightarrow)$-$(\Rightarrow /)$ are *the introduction rules* for connectives, and (Cut) is *the cut rule*.

By dropping (Cut), one obtains the cut-free (sequent system for) **L**. Lambek [40] proved *the cut elimination theorems* for **L** (and in [41] for **NL**): every provable sequent is provable in the cut-free system. As a consequence, both systems possess *the subformula property*: every provable sequent possesses a proof such that each formula appearing in this proof is a subformula of a formula occurring in this sequent. Since, additionally, each introduction rule increases the size of sequents, then the provability in either system is decidable. It is easy to extract language-restricted fragments. For instance, the product-free fragment admits product-free formulas only and drops rules $(\cdot \Rightarrow)$, $(\Rightarrow \cdot)$. **L** is a conservative extension of its language-restricted fragments, and similarly for **NL**.

The product-free **L**, restricted to (Id), $(\backslash \Rightarrow)$ and $(/ \Rightarrow)$ ((Cut) is admissible), yields precisely the correct reduction patterns of BCGs. This system is sometimes denoted by **AB**. **L** is much stronger than **AB**, but both systems coincide for sequents of the form $\alpha_1, \ldots, \alpha_n \Rightarrow p$ such that p is an atom and no α_i contains a compound type on the argument place. In other words, the order of each α_i is at most 1. *The order* of α, denoted by $o(\alpha)$, is recursively defined as follows: $o(p) = 0$ for atomic p,

$$o(\alpha\backslash\beta) = o(\beta/\alpha) = \max(o(\beta), o(\alpha) + 1),\ o(\alpha \cdot \beta) = \max(o(\alpha), o(\beta)).$$

For example, $p\backslash q$, $p\backslash(q\backslash r)$, $(p\backslash q)/r$ are of order 1, $p/(q\backslash p)$ is of order 2, and so on (p, q, r are atoms). The product-free **L** is stronger than any extension of **AB** by finitely many new reduction patterns, provable in **L** [69].

[8] shows that every $\epsilon-$free CFG G is equivalent to a BCG G' with all types of order at most 1. By the above, the language of G' does not change, if one replaces **AB** by **L**. Consequently, every $\epsilon-$free CFG is equivalent to a Lambek grammar. The converse holds as well [55]. Analogous results for **NL** were obtained in [15, 37].

Due to new laws, Lambek grammars provide a more flexible description of natural language. We consider atomic types s, n, as above, and n^* for plural nouns. In BCGs we get:

1. John likes Jane. n, $(n\backslash s)/n$, $n \Rightarrow s$.
2. John works here. n, $n\backslash s$, $s\backslash s \Rightarrow s$.
3. John never works. n, $(n\backslash s)/(n\backslash s)$, $n\backslash s \Rightarrow s$.
4. John works for Jane. n, $n\backslash s$, $(s\backslash s)/n$, $n \Rightarrow s$.
5. John works and Jane rests. n, $n\backslash s$, $(s\backslash s)/s$, n, $n\backslash s \Rightarrow s$.
6. men work. n^*, $n^*\backslash s \Rightarrow s$.
7. poor men work. n^*/n^*, n^*, $n^*\backslash s \Rightarrow s$.
8. men works. n^*, $n\backslash s \not\Rightarrow s$.
9. John work. n, $n^*\backslash s \not\Rightarrow s$.

Here $\not\Rightarrow$ means that the sequent is not provable in **AB**. The sequents in 8, 9 are unprovable in **L**, either.

Now assign $s/(n\backslash s)$ to `he` and $(s/n)\backslash s$ to `her`; We abbreviate these types as np_s and np_o, respectively, since they correspond to (singular) noun phrase as subject and noun phrase as object.

10. he likes Jane. $s/(n\backslash s)$, $(n\backslash s)/n$, $n \Rightarrow s$.

11. John likes her. n, $n\backslash(s/n)$, $(s/n)\backslash s \Rightarrow s$.

12. he likes her. $s/(n\backslash s)$, $(n\backslash s)/n$, $(s/n)\backslash s \not\Rightarrow s$.

13. John works for her. n, $n\backslash s$, $(s\backslash s)/n$, $(s/n)\backslash s \not\Rightarrow s$.

The sequent in 12 remains unprovable in **AB**, if even one replaces $(n\backslash s)/n$ by $n\backslash(s/n)$. In **L**, these two types are equivalent, by (L5), and this sequent is provable: use (Red.1) $s/n, (s/n)\backslash s \Rightarrow s$, (L3) (to the first and the second type of 12) and (Cut). Also the sequent in 13 is provable in **L**. Notice that `the student follows the teacher` can be parsed like 12 and `John works for a friend` like 13 (assign n_c to common nouns and np_s/n_c, np_o/n_c to articles).

These examples, similar to those in [40], well illustrate the power of Lambek grammars. In a BCG we need at least two types of `likes` (see 10, 11); they are equivalent in **L**, hence only one of them is sufficient. To parse 12 in a BCG we need additional types of words, e.g. $(s/n)/((n\backslash s)/n)$ of `he`; $s/(n\backslash s) \Rightarrow (s/n)/((n\backslash s)/n)$ is an instance of (L4), hence $s/(n\backslash s)$ is sufficient in a Lambek grammar. Even in **NL** one proves $n \Rightarrow np_s$, $n \Rightarrow np_o$ as instances of (L1). This shows that **L** provides some logical transformations of types and explains certain syntactic ambiguities of expressions. Of course, not all; we still need $n\backslash s$ and $n^*\backslash s$ for `worked`, n/n and n^*/n^* for `poor`.

Only four atomic types appear in these examples. Realistic grammars for a natural language employ much more atoms. Lambek [44] uses 33 atomic types for a fragment of English, described by a pregroup grammar (see 3.2.4). We list some of them.

$\pi =$	subject
$\pi_1 =$	first person singular subject
$\pi_2 =$	second person singular and any plural personal subject
$\pi_3 =$	third person singular subject
$s =$	statement (declarative sentence)
$s_1 =$	statement in present tense
$s_2 =$	statement in past tense
$\bar{q} =$	question
$q =$	yes-or-no question
$q_1 =$	yes-or-no question in present tense
$q_2 =$	yes-or-no question in past tense
$i =$	infinitive of transitive verb
$j =$	infinitive of complete verb phrase
$\bar{j} =$	complete infinitive with `to`
$o =$	direct object
$n =$	name
$n_0 =$	mass noun
$n_1 =$	count noun
$n_2 =$	plural noun
$\bar{n} =$	complete noun phrase

$p_1 =$ present participle
$p_2 =$ past participle

For semantic considerations, however, it is more natural to reduce the number of atomic types. Lambek's $\bar{n}$ can be defined as $s/(n\backslash s)$ or $(s/n)\backslash s$, depending on the role in a sentence (subject or object). Some authors choose np (noun phrase) as an atom and assign $(np\backslash s)/np$ to transitive verbs, instead of $(n\backslash s)/n$ (this neglects tense and number).

Every proof of $\Gamma \Rightarrow \alpha$ in the sequent system of **L** determines a unique bracketing of Γ, and similarly for **NL**. This induces a unique phrase structure of the parsed expression. Due to associativity, **L** is 'structurally omnipotent': every possible bracketing of Γ comes from some proof of $\Gamma \Rightarrow \alpha$ (if (Cut) can be used). Consequently, Lambek grammars based on **L** are not sensitive to phrase structures; they describe languages of strings.

On the contrary, Lambek grammars based on **NL** naturally describe languages of phrase structures. [38] shows the strong equivalence of these grammars and BCGs. Therefore some linguists prefer this weaker logic. It is quite weak, indeed; neither 12, nor 13 can be parsed in **NL**, if the same types are used. Worthy of notice, with **NL** one can interchange the roles of functors and arguments. From $x : \alpha$ and $y : \alpha\backslash\beta$ we infer $(x, y)_2 : \beta$, but, using (L1), we obtain $x : \beta/(\alpha\backslash\beta)$, hence also $(x, y)_1 : \beta$.

3.2 Extensions

3.2.1 Multi-modal systems

To make it more flexible Moortgat [49] and other authors extend **NL** in different ways: admit several products $\otimes_i$ with residuals $\backslash_i, /_i$ and unary modalities $\Diamond_i, \Box_i^{\downarrow}$, which form a residuation pair ($\Diamond_i\alpha \Rightarrow \beta$ and $\alpha \Rightarrow \Box_i^{\downarrow}\beta$ are equivalent in models and derivable from each other in the formal system). From $\Diamond_i\alpha \Rightarrow \Diamond_i\alpha$ we obtain $\alpha \Rightarrow \Box_i^{\downarrow}\Diamond_i\alpha$, and from $\Box_i^{\downarrow}\alpha \Rightarrow \Box_i^{\downarrow}\alpha$ we obtain $\Diamond_i\Box_i^{\downarrow}\alpha \Rightarrow \alpha$.

Let us consider an example from [50]. np can be lifted up to both $\Box_n^{\downarrow}\Diamond_n np$ and $\Box_a^{\downarrow}\Diamond_a np$, where the subscripts abbreviate nominative and accusative. We assign np to `John`, `Mary`, $\Box_n^{\downarrow}\Diamond_n np$ to `he`, `she`, $\Box_a^{\downarrow}\Diamond_a np$ to `him`, `her`, and $(\Box_n^{\downarrow}\Diamond_n np\backslash s)/\Box_a^{\downarrow}\Diamond_a np$ to `likes`. The resulting grammar assigns s to `John likes Mary`, `he likes her`, but not to `her likes Mary`.

Another example comes from [49]. Let r be the type of relative clause, e.g. `that Kazimierz wrote` (in `the book that Kazimierz wrote`). With **L** we can assign $r/(s/np)$ to `that`, which yields `that Kazimierz wrote`: r, since

$$r/(s/np),\ np,\ (np\backslash s)/np \Rightarrow r$$

is provable. This sequent, however, is not provable in **NL** (with any bracketing). We assign $r/(s/\Diamond_a\Box_a^{\downarrow}np)$ to `that` and admit the weak associativity axiom:

$$(\alpha \cdot \beta) \cdot \Diamond_a\gamma \Rightarrow \alpha \cdot (\beta \cdot \Diamond_a\gamma).$$

In **NL** we prove $(np, ((np\backslash s)/np, np)) \Rightarrow s$, and consequently,

$$(np, ((np\backslash s)/np, \Diamond_a \Box_a^{\downarrow} np)) \Rightarrow s.$$

This yields $((np, (np\backslash s)/np), \Diamond_a \Box_a^{\downarrow} np) \Rightarrow s$, by the new axiom, hence

$$(np, (np\backslash s)/np) \Rightarrow s/\Diamond_a \Box_a^{\downarrow} np,$$

by $(\Rightarrow /)$. Thus, `Kazimierz wrote`: $s/\Diamond_a \Box_a^{\downarrow} np$ and `that Kazimierz wrote`: r.

These examples show the spirit of multi-modal Lambek grammars, extensively studied by a group of contemporary linguists. The unary modalities are used to construct subtypes and super-types of some types and to restrict associativity and commutativity to some special cases. (By a subtype of α we mean a type β such that $\beta \Rightarrow \alpha$ is true, not a subformula of α.) This resembles the usage of exponentials !, ? in linear logic, where structural rules (weakening, contraction) are limited to formulas $!\alpha$, $?\alpha$. More information on the multi-modal framework can be found in [49, 51, 50].

Morrill [52] elaborated Discontinuous Lambek Calculus, a special multimodal and multi-sorted logic, intended to process types of discontinuous expressions. In language models, one admits strings with some occurrences of $|$ (separator); $\alpha \otimes_i \beta$ denotes the substitution of β for the $i-$the separator in α. The interpretation of product as substitution also appeared in [15].

3.2.2 Substructural logics

One can add lattice connectives $\wedge, \vee$, satisfying the lattice laws. It suffices to add:

$$\alpha \wedge \beta \Rightarrow \alpha \quad \alpha \wedge \beta \Rightarrow \beta \quad \frac{\alpha \Rightarrow \beta \quad \alpha \Rightarrow \gamma}{\alpha \Rightarrow \beta \wedge \gamma}$$

$$\alpha \Rightarrow \alpha \vee \beta \quad \beta \Rightarrow \alpha \vee \beta \quad \frac{\alpha \Rightarrow \gamma \quad \beta \Rightarrow \gamma}{\alpha \vee \beta \Rightarrow \gamma}$$

to the first axiomatization of **L** or **NL**. The corresponding sequent systems (we skip details) admit cut elimination, hence both logics are decidable. Here are the distributive laws, provable in **NL** with $\wedge, \vee$.

$$\alpha \cdot (\beta \vee \gamma) \Leftrightarrow (\alpha \cdot \beta) \vee (\alpha \cdot \gamma) \quad (\alpha \vee \beta) \cdot \gamma \Leftrightarrow (\alpha \cdot \gamma) \vee (\beta \cdot \gamma)$$

$$\alpha\backslash(\beta \wedge \gamma) \Leftrightarrow (\alpha\backslash\beta) \wedge (\alpha\backslash\gamma) \quad (\alpha \wedge \beta)/\gamma \Leftrightarrow (\alpha/\gamma) \wedge (\beta/\gamma)$$

$$(\alpha \vee \beta)\backslash\gamma \Leftrightarrow (\alpha\backslash\gamma) \wedge (\beta\backslash\gamma) \quad \alpha/(\beta \vee \gamma) \Leftrightarrow (\alpha/\beta) \wedge (\alpha/\gamma)$$

Let us note some simple applications of types with $\wedge, \vee$ in type grammars. Lambek [41] noticed that a type assignment $I(v) = \{\alpha_1, \ldots, \alpha_n\}$ could be replaced with the rigid type assignment $I(v) = \alpha_1 \wedge \cdots \wedge \alpha_n$. Another application concerns subtypes. Lambek [44] needs nonlogical assumptions $\pi_i \Rightarrow \pi$, $s_j \Rightarrow s$, for $i = 1, 2, 3$, $j = 1, 2$. Instead one can define $s = s_1 \vee s_2$, $\pi = \pi_1 \vee \pi_2 \vee \pi_3$ and apply a pure logic with $\vee$ but no nonlogical assumptions (according to the paradigm of lexicality). Kanazawa [35] proposed feature-decomposition types: `works` is of type $(np \wedge sg)\backslash s$, `work` of type $(np \wedge pl)\backslash s$, `worked` of type $np\backslash s$, and `became` of type $((np\backslash s)/(np \vee ad)$, where np, sg, pl, ad are types of noun phrase, singular, plural, and adjective, respectively.

By **L1** we denote **L** with constant 1 and the axioms:

$$1 \cdot \alpha \Leftrightarrow \alpha \quad \alpha \cdot 1 \Leftrightarrow \alpha\,.$$

L1 is strongly complete with respect to *residuated monoids*, i.e. residuated semigroups with an element 1 (the unit for product). The sequent system is obtained from that for **L** by admitting sequents $\Rightarrow \alpha$ (interpreted in algebras as $1 \leq \mu(\alpha)$), allowing Γ to be empty in rules $(\Rightarrow \backslash)$, $(\Rightarrow /)$, and adding:

$$(1 \Rightarrow)\ \frac{\Gamma, \Gamma' \Rightarrow \alpha}{\Gamma, 1, \Gamma' \Rightarrow \alpha} \quad (\Rightarrow 1) \quad \Rightarrow 1\,.$$

NL1 can be presented in a similar way. Notice that in **L1** one proves new laws, not containing 1, nor the empty antecedent, e.g. $\alpha/(\alpha\backslash\alpha) \Rightarrow \alpha$; $\Leftarrow$ is provable in **L**. To prove the former, from $\alpha \Rightarrow \alpha$ infer $\Rightarrow \alpha\backslash\alpha$, by $(\Rightarrow \backslash)$, then apply $(/ \Rightarrow)$. This proof works in **NL1** as well.

The language models for **L1** are algebras of subsets of $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$; the operations $\cdot, \backslash, /$ for languages are defined as above except that Σ^+ is replaced with Σ^*. The language $\{\epsilon\}$ is the unit for product. The intended models for **NL1** employ languages of phrase structures, now enriched with the empty structure ϵ such that $(\epsilon, x) = (x, \epsilon) = x$ for any phrase structure x.

Some linguists object the suitability of **L1** as a logic for type grammars. $\alpha/\alpha \Leftrightarrow (\alpha/\alpha)/(\alpha/\alpha)$ is provable in **L1**, hence n_c/n_c and $(n_c/n_c)/(n_c/n_c)$ are equivalent, but the former is a natural type of adjectives and the latter of adverbs.

On the other hand, logicians prefer **L1** and its extensions. In these systems, some formulas are provable (a formula α is said to be *provable*, if $\Rightarrow \alpha$ is provable); for example, $\alpha\backslash\alpha$ in **NL1** and $(\alpha\backslash\beta)\backslash((\gamma\backslash\alpha)\backslash(\gamma\backslash\beta))$ in **L1**. Furthermore, every sequent is deductively equivalent to a formula, e.g. $\alpha, \beta \Rightarrow \gamma$ to $\beta\backslash(\alpha\backslash\gamma)$. Accordingly, these logics can be presented in the form of Hilbert style systems and more easily compared with other nonclassical logics. For example, the product-free **L1** can be axiomatized as a Hilbert style system with the following axioms and rules.

$$\text{(a.1) } 1 \quad \text{(a.2) } 1\backslash(\alpha\backslash\alpha) \quad \text{(a.3) } ((\alpha\backslash\beta)/\gamma)\backslash(\alpha\backslash(\beta/\gamma)) \quad \text{(a.4) } (\alpha\backslash(\beta/\gamma))\backslash((\alpha\backslash\beta)/\gamma)$$

$$\text{(a.5) } (\alpha\backslash\beta)\backslash((\gamma\backslash\alpha)\backslash(\gamma\backslash\beta)) \quad \text{(a.6) } ((\alpha/\gamma)/(\beta/\gamma))/(\alpha/\beta)$$

$$(\text{mp}\backslash)\ \frac{\alpha \quad \alpha\backslash\beta}{\beta} \quad (\backslash\text{-}/)\ \frac{\alpha\backslash\beta}{\overline{\beta/\alpha}}$$

For the 1-free fragment, (a.1), (a.2) are replaced by (id) $\alpha\backslash\alpha$. Other axiom systems can be found in [70] and for richer logics in [26].

L1 with $\wedge, \vee$ is called Full Lambek Calculus (**FL**) and regarded as a basic substructural logic [26]. Substructural logics can be defined as axiom and rule extensions of **FL**. They correspond to some classes (usually varieties or quasi-varieties) of *residuated lattices*, i.e. lattice-ordered residuated monoids. One often adds a new constant 0 and defines *negations*: $\sim\alpha = \alpha\backslash 0$, $-\alpha = 0/\alpha$ (0 is interpreted as an arbitrary element of the residuated lattice).

The term 'substructural logics' refers to the fact that sequent systems for these logics lack some structural rules, characteristic of the Gentzen system for intutionistic logic: exchange (e), contraction (c), left weakening or integrality (i), right weakening (o). The first three rules have the following forms.

$$(e)\ \frac{\Gamma, \alpha, \beta, \Gamma' \Rightarrow \gamma}{\Gamma, \beta, \alpha, \Gamma' \Rightarrow \gamma} \quad (c)\ \frac{\Gamma, \Delta, \Delta, \Gamma' \Rightarrow \alpha}{\Gamma, \Delta, \Gamma' \Rightarrow \alpha}$$

$$(i)\ \frac{\Gamma, \Gamma' \Rightarrow \beta}{\Gamma, \alpha, \Gamma' \Rightarrow \beta}$$

They express some algebraic properties of product: (e) $a \cdot b = b \cdot a$, (c) $a \leq a \cdot a$, (i) $a \cdot b \leq a$, $a \cdot b \leq b$ (with 1 this amounts to $a \leq 1$). (o) involves sequents of the form $\Gamma \Rightarrow$ (interpreted as $\mu(\Gamma) \leq 0$). They can be eliminated, and (o) can be replaced by the axiom $\Gamma, 0, \Gamma' \Rightarrow \alpha$.

For logics with (e), corresponding to commutative algebras, $\alpha\backslash\beta$ is equivalent to β/α (in algebras $a\backslash b = b/a$), and one writes $\alpha \to \beta$ for both. The sequent systems are simpler (we omit details). Also $\sim\alpha$ is equivalent to $-\alpha$, and one writes $\neg\alpha$ for both.

From the algebraic point of view, substructural logics treat implication(s) as residual(s) of the product operation; the latter usually differs from the lattice meet. Many well-known nonclassical logics belong to this family: relevant logics, many-valued logics, fuzzy logics, and intuitionistic and classical logics as the limit cases. For instance, intuitionistic logic amounts to **FL** with (e), (c), (i), (o) (in fact, (e) is derivable with (c), (i)), and Łukasiewicz infinitely valued logic to **FL** with (e), (i), (o) and the axiom $\alpha \vee \beta \Leftrightarrow (\alpha \to \beta) \to \beta$ [26].

Linear logic of Girard [28] can be presented as **FL** with 0, (e) and the double negation axiom $\neg\neg\alpha \Rightarrow \alpha$ ($\Leftarrow$ is provable); we neglect exponentials !, ?. Noncommutative versions are due to Yetter [68] and Abrusci [1]. The former can be presented as **FL** with 0 and the axioms $\alpha\backslash 0 \Leftrightarrow 0/\alpha$ (hence $\sim\alpha$ and $-\alpha$ collapse in $\neg\alpha$) and $\neg\neg\alpha \Rightarrow \alpha$; the latter as **FL** with 0 and the axioms $\sim -\alpha \Rightarrow \alpha$, $- \sim \alpha \Rightarrow \alpha$ (again $\Leftarrow$ are provable). Both logics are conservative extensions of **FL** without 0 [2]. In [26] they are called Cyclic Involutive Full Lambek Calculus (**CyInFL**) and Involutive Full Lambek Calculus (**InFL**), respectively.

Cut-free sequent systems of linear logics look differently. Formulas are built from atoms by $\otimes, \oplus$ and negation(s), where $\otimes$ stands for product and $\oplus$ for the dual product ('par'). In algebras, $a \oplus b = \neg(\neg b \otimes \neg a)$ for Girard's logic and **CyInFL** and $a \oplus b = \sim(-b \otimes -a)$ for **InFL**. One employs classical sequents $\Gamma \Rightarrow \Delta$ or one-sided sequents only: either $\Rightarrow \Delta$ (Schütte style), or $\Gamma \Rightarrow$ (dual Schütte style). Each comma Γ is interpreted as product and in Δ as dual product. In **InFL**, presented in this way, one can define $\backslash, /$ as follows: $\alpha\backslash\beta = \sim\alpha \oplus \beta$, $\alpha/\beta = \alpha \oplus -\beta$.

In the literature on linear logics, $\otimes, \oplus, \backslash, 1, 0$ and negation(s) are referred to as *multiplicatives* and $\wedge, \vee$ (also constants $\top, \bot$, interpreted as the greatest and the least element) as *additives*. According to a different tradition, they are intensional and extensional connectives and constants, respectively. **L1** is often characterized as the intuitionistic fragment of multiplicative linear logic.

Type grammars usually employ basic intuitionistic substructural logics, often not admitting empty antecedents of sequents and being restricted to multiplicative connectives (also multi-modal). Nonetheless the impact of linear logics (which are 'classical') can be seen in current developments. I have already noted an analogy between modalities in type grammars and exponentials of Girard [28]. Also *proof nets*, i.e. a representation of proofs in multiplicative linear logics by means of some graphs of links between formulas, are used as representations of syntactic structures in type grammars, either directly, or in a form suitable for intuitionistic fragments. We cannot discuss this matter here; the reader is referred to [50].

In language models, $\wedge, \vee$ can be interpreted as the set theoretic intersection and union of languages. Then, we obtain a distributive lattice. The distributive laws for $\wedge, \vee$ are not provable in **FL**, nor other logics, discussed above. One can add them as new axioms; it suffices to add:

$$\text{(D) } \alpha \wedge (\beta \vee \gamma) \Rightarrow (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)\,.$$

Nevertheless, some interesting linguistic interpretations of logics without (D) are possible. Clark [22] introduced *syntactic concept lattices* as a special kind of concept lattices from lattice theory. Let $L_0 \subseteq \Sigma^*$ be a fixed language. Pairs (x, y), for $x, y \in \Sigma^*$, are called *contexts*. For a set of contexts X, one defines $X^{\triangleleft}$ as the set of all $z \in \Sigma^*$ such that $xzy \in L_0$, for all $(x, y) \in X$. The sets of the form $X^{\triangleleft}$ are called *syntactic concepts* for L_0. They can be interpreted as the syntactic categories determined by L_0, a reasonable generalization of Husserl's idea, followed by Ajdukiewicz. Since **L** provides nontrivial laws $\alpha \Rightarrow \beta$, syntactic categories in Lambek grammars cannot be equivalence classes (substitution classes). The family of syntactic concepts for L_0 is a complete residuated lattice with operations: $X \wedge Y = X \cap Y$, $X \vee Y$ (resp. $X \otimes Y$, 1) equal to the smallest concept containing $X \cup Y$ (resp. $X \cdot Y$, $\{\epsilon\}$), and $\backslash, /$ defined as for languages.

3.2.3 Semantic types

The product-free **L** with (e) was studied by van Benthem [9, 10] as a logic of semantic types; we call this logic the Lambek-van Benthem calculus (**LB**). Proofs in a natural deduction system (ND-system) for **LB** can be encoded by some terms of typed lambda calculus, namely linear terms (i.e. every λ binds exactly one occurrence of a variable), satisfying the additional constraint: no subterm is closed. This is an adaptation of the 'Curry-Howard isomorphism' between ND-proofs and lambda terms [59]. Since every ND-proof in **L** can be translated into an ND-proof in **LB**, the former determines a unique lambda term; this lambda term, interpreted in a standard type-theoretic model (see Section 2), denotes a semantic transformation corresponding to the syntactic parsing in the grammar. Size limits do not allow us to discuss this framework in detail. Let us consider one example. Recall that the characteristic inference rules of ND-systems are the introduction rules and the elimination rules for connectives.

From $n \Rightarrow n$ and $n\backslash s \Rightarrow n\backslash s$ we get $n, n\backslash s \Rightarrow s$, by the $\backslash-$elimination rule: from $\Gamma \Rightarrow \alpha$ and $\Delta \Rightarrow \alpha\backslash\beta$ infer $\Gamma, \Delta \Rightarrow \beta$ (in an ND-system for **L**). This is translated in **LB** as: from $e \to t \Rightarrow e \to t$ and $e \Rightarrow e$ infer $e \to t, e \Rightarrow t$, by the $\to -$elimination rule (in an ND-system for **LB**). In **L** we obtain $n \Rightarrow s/(n\backslash s)$, by the $/-$introduction rule, which is translated into $e \Rightarrow (e \to t) \to t$ in **LB**. The ND-proof in **LB** is encoded by the term:

$$\lambda y^{e\to t}.y^{e\to t}x^{e}.$$

This is a linear term, satisfying the additional constraint. In a type-theoretic model, if x^e is valuated as $a \in D_e$ (an individual), then this term denotes the family of all (characteristic functions of) $X \subseteq D_e$ such that $a \in X$. Thus, the syntactic law $n \Rightarrow s/(n\backslash s)$ corresponds to the semantic transformation which sends an individual into the family of all properties (interpreted extensionally) of this individual. In particular, any proper noun (denoting an individual) can be treated as a noun phrase (denoting a generalized quantifier, i.e. a family of sets of individuals).

We have explained a semantic interpretation of (L1). Similarly, (L3) correspond to the composition of functions, (L5) to the interchange of arguments, and so on. An interesting theorem of [9] shows that every provable sequent of **LB** admits only finitely many different proofs up to the equality in lambda calculus; so every expression possesses only finitely many 'semantic readings'. Further studies on this topic can be found in [47, 49].

Analogous correspondences were elaborated for richer logics [66]. Lambek [42] studied category-theoretic interpretations of **L** and its extensions. Abstract Categorial Grammars, introduced by de Groote [29], employ linear lambda-terms as representations of both syntactic structures and semantic structures (logical forms) of expressions in natural language with certain homomorphisms between them.

3.2.4 Pregroup grammars

Lambek [43] proposed another extension of **L1**, called compact bilinear logic (**CBL**). It corresponds to *pregroups*, i.e. ordered algebras $(A, \cdot, ^r, ^l, 1, \leq)$ such that $(A, \cdot, 1, \leq)$ is a partially ordered monoid and $^r, ^l$ are unary operations, satisfying *the adjoint laws*:

$$a^l \cdot a \leq 1 \leq a \cdot a^l \text{ and } a \cdot a^r \leq 1 \leq a^r \cdot a\,,$$

for any $a \in A$. a^r (resp. a^l) is called the *right* (resp. *left*) *adjoint* of a. (This terminology is transferred from category theory.) Pregroups coincide with the algebras for the multiplicative fragment of noncommutative linear logic of Abrusci [1] such that $\otimes$ equals $\oplus$ and $1 = 0$. The residuals of product are defined by: $a\backslash b = a^r \cdot b$, $a/b = a \cdot b^l$. We have: $a^{rl} = a^{lr} = a$, $(a \cdot b)^r = b^r \cdot a^r$, and similarly for ^{l}. Adjoints reverse the ordering: if $a \leq b$ then $b^r \leq a^r$ and $b^l \leq a^l$.

CBL is a logic of free pregroups. From atoms $p, q, r, \ldots$ one builds *simple types* $p^{(n)}$, where n is an integer. $p^{(0)}$ is interpreted as p, $p^{(n)}$, $n > 0$, as $p^{r \cdots r}$ (n times), and $p^{(n)}$, $n < 0$, as $p^{l \cdots l}$ ($|n|$ times). *Pregroup types* are finite strings of simple types.

One also assumes that the set of atoms is partially ordered by a relation $\preceq$. The relation $\Rightarrow$, between pregroup types, is defined by the following rewriting rules:

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

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

(Induced Step) $X, p^{(n)}, Y \Rightarrow X, q^{(n)}, Y$, if either $p \preceq q$ and n is even, or $q \preceq p$ and n is odd.

$U \Rightarrow V$ holds, if U can be transformed into V by finitely many applications of these rules.

Pregroup grammars are defined as Lambek grammars except that **L** is replaced by **CBL** (and types of **L** by pregroup types). Lambek [43] shows that (Expansion) can be eliminated from proofs of $X \Rightarrow p$, where p is an atom. Furthermore, (Induced Step) and (Contraction) can be collapsed with one rule of generalized contraction:

(GCON) $X, p^{(n)}, q^{(n+1)}, Y \Rightarrow X, Y$, with the same condition as in (Induced Step).

If $\alpha_1, \ldots, \alpha_n$ are assigned to $v_1, \ldots, v_n$, respectively, then the grammar assigns p to $v_1 \ldots v_n$, if the concatenation $\alpha_1 \cdots \alpha_n$ reduces to p by a finite number of applications of (GCON) and, possibly, (Induced Step) at the end of the reduction. Such derivations can be presented by means of links, joining the reduced types of (GCON).

For example, we assume `she`: π_3, `will`: $\pi^r s_1 j^l$, `see`: io^l and `him`: o, where i, j are types of infinitive of intransitive verb and infinitive of any complex verb phrase, and π, π_3, o are understood as above. We also assume $\pi_3 \preceq \pi$, $i \preceq j$. Then, `she will see him` is assigned s_1, since:

$$\pi_3,\ \pi^r s_1 j^l,\ io^l,\ o \Rightarrow s_1\,.$$

The reduction can be depicted as follows:

$$\underbrace{\pi_3,\ \pi^r}\, s_1\, \underbrace{j^l,\ i}\, \underbrace{o^l,\ o}$$

where each link corresponds to one application of (GCON).

With `man`: n_1 (count noun), `whom`: $n_1^r n_1 o^{ll} s^l$, `saw`: $\pi^r s_2 o^l$ and $s_2 \preceq s$, one assigns n_1 to `man whom she saw`, by the reduction:

$$\underbrace{n_1,\ n_1^r}\, n_1\, \underbrace{o^{ll}\ \underbrace{s^l,\ \underbrace{\pi_3,\ \pi^r}\, s^2}\ o^l}\ .$$

These examples come from [44] (up to minor changes), where Lambek analyzed many basic grammatical constructions of English within the pregroup framework. In other publications he and his collaborators applied this approach to several languages: German, French, Italian, Polish and some non-European languages; see [44] for references.

Parsing by pregroups is computationally simple; it runs in polynomial time [18], whereas **L** is NP-complete [57]. **CBL** is stronger than **L1**: $(p/((p/p)/p))/p \Rightarrow p$ is provable in **CBL** (define / as above), but not in **L1**. The logical meaning of the new laws is not clear; the latter does not hold even in classical logic (interpret / as implication with the antecedent on the right). No type-theoretic semantics for pregroup grammars is known. It seems that **CBL** is an algebraic calculus rather than a genuine logic. This opinion is confirmed by the fact that bounded pregroups are trivial (one-element) algebras, hence **CBL** with $\top$ is inconsistent [18]. (The latter paper shows that pregroup grammars are equivalent to CFGs.)

On the other hand, all linguistic examples, analyzed by Lambek and other authors by means of pregroups, can easily be parsed with **L**. We return to `man whom she saw`. The pregroup types, given above, are translations of **L**-types; e.g. `whom`: $(n_1\backslash n_1)/(s_2/o)$, `saw`: $(\pi\backslash s_2)/o$. The sequent:

$$n_1,\ (n_1\backslash n_1)/(s/o),\ \pi_3,\ (\pi\backslash s_2)/o \Rightarrow n_1$$

is provable in **L** augmented with $s_2 \Rightarrow s$, $\pi_3 \Rightarrow \pi$. Therefore, the semantics for these examples can be transferred from **L**.

3.2.5 Modal logics

At the end, we consider other modal logics, extending **L** and **NL**. [20] studied **NL** with $\wedge, \vee$, which satisfy the laws of a distributive lattice, and its extensions with either classical negation (**BFNL**), or intuitionistic implication and $\top, \bot$ (**HFNL**); these logics were presented as sequent systems with cut. Hilbert-style systems for the latter logics, denoted by **NLC** and **NLI**, were studied in [33, 34]. The connectives are $\wedge, \vee, \Rightarrow, \neg$ (now $\Rightarrow$ stands for the classical or intuitionistic implication, and $\neg$ for the classical or intuitionistic negation) and Lambek connectives $\cdot, \backslash, /$. Lambek's sequents $\alpha \Rightarrow \beta$ are treated as conditionals. **NLC** (resp. **NLI**) can be axiomatized by all tautologies of classical (resp. intuitionistic) propositional logic in the extended language and the rules: *modus ponens* for $\Rightarrow$, (Res.1), (Res.2). In the associative versions **LC**, **LI** one adds axioms (A.1), (A.2). The following formulas, similar to the modal axiom (K), are provable in **NLI**, hence also in **NLC**, **LI**, **LC** (we assume that $\backslash, /$ bind stronger than $\Rightarrow$).

$$\gamma\backslash(\alpha \Rightarrow \beta) \Rightarrow (\gamma\backslash\alpha \Rightarrow \gamma\backslash\beta) \quad (\alpha \Rightarrow \beta)/\gamma \Rightarrow (\alpha/\gamma \Rightarrow \beta/\gamma)$$

It should be emphasized that the theorems (i.e. provable formulas) of these systems are $\top-$theorems: they satisfy $\mu(\alpha) = \top$ in algebras. In substructural logics one usually considers 1-theorems ($1 \leq \mu(\alpha)$ in algebras). Both notions

$$\begin{array}{ccccc}
v_1 & \dots & v_n & & \\
\alpha_1 & \dots & \alpha_n & \Rightarrow & s \\
\Downarrow & \dots & \Downarrow & & \\
\beta_1 & \dots & \beta_n & \overset{\mathbf{AB}}{\Rightarrow} & s
\end{array}$$

Figure 2: A parsing scheme

collapse for substructural logics with (i). **LC** is a conservative extension of **L**, and **NLC** of **NL**.

NLC, **LC** and **NLI**, **LI** are, in fact, some classical and intuitionistic multi-modal logics; product and its residuals are binary modalities. This perspective was already admitted in Arrow Logic of van Benthem [11] and multi-modal versions of Lambek calculi. [33, 34] study relational frames for **NLC**, **LC**, **NLI**, **LI**, proving some completeness and decidability results. Interestingly, the undecidability of **LC** follows from some results of [39], whereas **LI** is decidable [34]. For **NLC**, **NLI** even the consequence relations are decidable [20].

3.3 Lambek versus Ajdukiewicz

Although Lambek logics are much stronger than **AB**, the parsing procedure in Lambek grammars can be carried out in a similar way as in BCGs. The action of **L** and related systems can be reduced to the lexical level: the type lexicon is extended by new types, derivable from the initial types in the system.

For example, if $\alpha, \beta, \gamma \Rightarrow \delta$ is provable in **L**, then $\alpha \Rightarrow (\delta/\gamma)/\beta$ is provable, by $(\Rightarrow /)$, and the sequent:

$$(\delta/\gamma)/\beta, \beta, \gamma \Rightarrow \delta$$

is provable in **AB**. For **NL**, if $\alpha, (\beta, \gamma) \Rightarrow \delta$ is provable, then $\alpha \Rightarrow \delta/(\beta \cdot \gamma)$ is provable, and $\delta/(\beta \cdot \gamma), (\beta, \gamma) \Rightarrow \delta$ is provable in **AB** (with product; see [38]). To eliminate product, one can use $\beta \Rightarrow (\alpha\backslash\delta)/\gamma$ and prove in **AB**:

$$\alpha, (\alpha\backslash\delta)/\gamma, \gamma \Rightarrow \delta\,.$$

For $v_i : \alpha_i$, $i = 1, \dots, n$, a successful parsing can be arranged as in Figure 2 ($\alpha_i \Rightarrow \beta_i$ is provable in a Lambek logic).

In the same way one can arrange semantic derivations: the semantic transformations, definable in (a fragment of) lambda calculus, can be performed on the initial denotations of words in a type-theoretic model, and the denotations of compound expressions are obtained by the (iterated) application of functions to their arguments.

Such laws as (L1), (L2), (L4) produce infinitely many types β derivable from a single type α. For instance, starting from n, one derives:

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

Nonetheless only finitely many of them are really needed to parse any expression in a particular grammar. [16] shows that every type grammar G, based on **L**, is equivalent to a BCG G' whose type lexicon extends that of G by finitely many new types, derivable in **L** from those in the type lexicon of G. The same was earlier shown for **NL** in [38].

These results seem to support the opinion that Lambek logics can be regarded as general logics of syntactic or semantic types rather than type processing systems in type grammars. The former explain deeper reasons for syntactic ambiguities of expressions and guide our choice of lexical types. On the other hand, parsing can be based on the classical type reduction procedure, proposed by Ajdukiewicz, with necessary modifications.

This opinion is non-orthodox. Many authors maintain the priority of Lambek logics, directly applied in grammars, according to the general paradigm of *parsing as deduction*. They, however, usually ignore the problems of efficiency. Parsers for BCGs can be designed like for CFGs; they run in cubic time in the length of the parsed expression. This is impossible for type grammars based on **L**, which is NP-complete [57]. Type grammars with **NL** remain polynomial [30], but parsers are not as simple as for BCGs.

At the end of this subsection, let me mention some developments in type grammars, which are closer to Ajdukiewicz.

Combinatory Categorial Grammars (CCGs), developed by M. Steedman, A. Szabolcsi and others, enrich **AB** with finitely many new reduction patterns, semantically corresponding to some combinators, i.e. closed lambda-terms; see [61] for an overview. This direction continues certain ideas of Curry [23] and Shaumyan [58]. Some of the new patterns are provable in **L**, but others require a stronger logic (some instances of exchange and contraction). The Ajdukiewicz procedure enriched with composition laws (similar to (L3), (L4)) was earlier proposed by Geach [27].

Categorial Unification Grammars (CUGs), studied by Uszkoreit [65], admit polymorphic types, containing variables, which range over a family of types. The simplest example is $(x\backslash x)/x$ as the type of `and`. In the course of parsing, one applies the reduction rules of BCGs and a unification algorithm. For instance, $\alpha, \beta\backslash\gamma \Rightarrow \sigma(\gamma)$, where σ is a substitution such that $\sigma(\alpha) = \sigma(\beta)$.

L with $\wedge, \vee$ can generate some non-context-free languages, e.g. the intersection of two context-free languages [35]. This also holds for grammars based on **AB** with $\wedge, \vee$. Other frameworks going beyond the context-free world are Tupled Pregroup Grammars [60] and Categorial Dependency Grammars [24]. Both approaches employ very restricted types only; the resulting grammars might be presented as BCGs with all types of order at most 1 and certain constraints imposed on reductions.

References

[1] Abrusci, V.M.: Phase semantics and sequent calculus for pure noncommutative classical linear logic. Journal of Symbolic Logic **56**, 1403–1451

(1991)

[2] Abrusci, V.M.: Classical Conservative Extensions of Lambek Calculus. Studia Logica **71**, 277–314 (2002)

[3] Ajdukiewicz, K.: W sprawie 'uniwersaliów' (On the problem of 'universals'). Przegląd Filozoficzny **37**, 219–234 (1934)

[4] Ajdukiewicz, K.: Die syntaktische Konnexität (Syntactic connexion). Studia Philosophica **1**, 1–27 (1935)

[5] Ajdukiewicz, K.: Związki składniowe między członami zdań oznajmujących (Syntactic connections between constituents of declarative sentences). Studia Filozoficzne **6.21**, 73–86 (1960)

[6] Ajdukiewicz, K.: The Scientific World-Perspective and Other Essays, 1931-1963. J. Giedymin (ed.), D. Reidel (1978)

[7] Bar-Hillel, Y.: A quasi-arithmetical notation for syntactic description. Language **29**, 47–58 (1953)

[8] Bar-Hillel, Y., Gaifman, C., Shamir, E.: On categorial and phrase structure grammars. Bull. Res. Council Israel **F9**, 155–166 (1960)

[9] Benthem, J. van: The semantics of variety in categorial grammar. In: [21], pp. 37–55.

[10] Benthem, J. van: Essays in Logical Semantics. D. Reidel (1986)

[11] Benthem, J. van: Language in Action. Categories, Lambdas and Dynamic Logic. North-Holland (1991)

[12] Benthem, J. van, Meulen. A. ter (eds.): Handbook of Logic and Language. Elsevier, The MIT Press (1997)

[13] Bocheński, J.: On the syntactical categories. New Scholasticism **23**, 257–280 (1949)

[14] Buszkowski, W.: Some decision problems in the theory of syntactic categories. Zeitschrift f. mathematische Logik und Grundlagen der Math. **28**, 539–548 (1982)

[15] Buszkowski, W.: Logiczne podstawy gramatyk kategorialnych Ajdukiewicza-Lambeka (Logical foundations of Ajdukiewicz-Lambek categorial grammars). Państwowe Wydawnictwo Naukowe (1989)

[16] Buszkowski, W.: Extending Lambek Grammars to Basic Categorial Grammars. Journal of Logic, Language, and Information **5**, 279–295 (1996)

[17] Buszkowski, W.: The Ajdukiewicz calculus, Polish notation and Hilbert-style proofs. In: Woleński, J. (ed.), The Lvov-Warsaw School and Contemporary Philosophy, pp. 241–252 (1998)

[18] Buszkowski, W.: Lambek grammars based on pregroups. In: Logical Aspects of Computational Linguistics. LNCS 2099, pp. 95–109. Springer (2001)

[19] Buszkowski, W.: Syntactic Categories and Types: Ajdukiewicz and Modern Categorial Grammars. In: Brożek, A., et al. (eds.) Tradition of the Lvov-Warsaw School: Ideas and Continuations, pp. 35–71. Brill - Rodopi (2016)

[20] Buszkowski, W., Farulewski, M.: Nonassociative Lambek Calculus with Additives and Context-Free Languages. In: Languages: From Formal to Natural. LNCS 5533, pp. 45–58. Springer (2008)

[21] Buszkowski, W., Marciszewski, W., Benthem, J. van (eds.): Categorial Grammar. J. Benjamins (1988)

[22] Clark, A.: A learnable representation of syntax using residuated lattices. In: Formal Grammar. LNCS 5591, pp. 183–198. Springer (2011)

[23] Curry, H.B.: Some logical aspects of grammatical structure. In: [32], pp. 56–68

[24] Dekhtyar, M., Dikovsky, A., Karlov, B.: Categorial dependency grammars. Theoretical Computer Science **579**, 33–63 (2015)

[25] Došen, K.: A brief survey of frames for the Lambek calculus. Zeitschrift f. mathematische Logik und Grundlagen der Math. **38**, 179–187 (1992)

[26] Galatos, N., Jipsen, P., Kowalski, T., Ono, H.: Residuated Lattices: An Algebraic Glimpse at Substructural Logics. Elsevier (2007)

[27] Geach, P.T.: A program for syntax. Synthese **22**, 3–17 (1971)

[28] Girard, J.-Y.: Linear logic. Theoretical Computer Science **50**, 1–102 (1987)

[29] Groote, Ph. de: Towards abstract categorial grammars. In: Proc. 39th Annual Meeting and 10th Conference of the European Chapter of ACL. Association of Computational Linguistics, pp. 148–155 (2001)

[30] Groote, Ph. de, Lamarche, F.: Classical Non-Associative Lambek Calculus. Studia Logica **71**, 355–388 (2002)

[31] Husserl, E.: Logische Untersuchungen (1900–1901)

[32] Jakobson, R. (ed.): Structure of Language and Its Mathematical Aspects. Proc. Symposium in Applied Mathematics. American Mathematical Society (1961)

[33] Kaminski, M., Francez, N.: Relational semantics of the Lambek calculus extended with classical propositional logic. Studia Logica **102**, 479–497 (2014)

[34] Kaminski, M., Francez, N.: The Lambek Calculus Extended with Intuitionistic Propositional Logic. Studia Logica **104**, 1051–1082 (2016)

[35] Kanazawa, M.: The Lambek Calculus Enriched with Additional Connectives. Journal of Logic, Language, and Information **1**, 141–171 (1992)

[36] Kandulski, M.: The non-associative Lambek calculus. In: [21], pp. 141–151

[37] Kandulski, M.: The equivalence of nonassociative Lambek categorial grammars and context-free grammars. Zeitschrift f. mathematische Logik und Grundlagen der Math. **34**, 41–52 (1988)

[38] Kandulski, M.: Phrase-structure languages generated by categorial grammars with product. Zeitschrift f. mathematische Logik und Grundlagen der Math. **34**, 373–383 (1988)

[39] Kurucz, A., Németi, I., Sain, I., Simon, A.: Decidable and Undecidable Logics with a Binary Modality. Journal of Logic, Language, and Information **4**, 191–206 (1995)

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

[41] Lambek, J.: On the calculus of syntactic types. In: [32], pp. 166-178

[42] Lambek, J.: Deductive systems and categories I. Journal of Mathematical Systems Theory **2**, 287–318 (1968)

[43] Lambek, J.: Type grammars revisited. In: Logical Aspects of Computational Linguistics. LNCS 1582, pp. 1–27. Springer (1999)

[44] Lambek, J.: From Word to Sentence: a computational algebraic approach to grammar. Polimetrica (2008)

[45] Leśniewski, S.: Grundzüge eines neuen System der Grundlagen der Mathematik. Fundamenta Mathematicae **14**, 1–81 (1929)

[46] Montague, R.: English as a Formal Language. In: Thomason, R. (ed.) Formal Philosophy: Selected Papers of Richard Montague, pp. 188-221. Yale University Press (1974)

[47] Moortgat, M.: Categorial Investigations. Logical and Linguistic Aspects of the Lambek Calculus. Foris (1988)

[48] Moortgat, M.: Multimodal linguistic inference. Journal of Logic, Language, and Information **5**, 349–385 (1996)

[49] Moortgat, M.: Categorial Type Logics. In: [12], pp. 93–177

[50] Moot, R., Retoré, C.: The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics. LNCS 6850. Springer (2012)

[51] Morrill, G.: Type-Logical Grammar: Categorial Logic of Signs. Kluwer (1994)

[52] Morrill, G.: Categorial Grammar: Logical Syntax, Semantics, and Processing. Oxford University Press (2011)

[53] Nowaczyk, A.: Categorial languages and variable-binding operators. Studia Logica **37**, 27–39 (1978)

[54] Oehrle, R.T., Bach, E., Wheeler, D. (eds.): Categorial Grammars and Natural Language Structures. D. Reidel (1988)

[55] Pentus, M.: Lambek grammars are context-free. In: Proc. of 8th IEEE Symposium on Logic in Computer Science, pp. 429–433 (1993)

[56] Pentus, M.: Models for the Lambek calculus. Annals of Pure and Applied Logic **75**, 179–213 (1995)

[57] Pentus, M.: Lambek calculus is NP-complete. Theoretical Computer Science **357**, 186–201 (2006)

[58] Shaumyan, S.: Applicational grammar as a semantic theory of natural language. Edinburgh University Press (1977)

[59] Sørensen, M.H., Urzyczyn, P.: Lectures on the Curry-Howard Isomorphism. Elsevier (2006)

[60] Stabler, E.: Tupled pregroup grammars. In: Casadio, C., Lambek, J. (eds.) Computational Algebraic Approaches to Natural Language, pp. 23–52. Polimetrica (2008)

[61] Steedman, M.: Categorial Grammar. Lingua **90**, 221–258 (1993)

[62] Suszko, R.: Syntactic Structure and Semantical Reference I. Studia Logica **8**, 213–244 (1958)

[63] Suszko, R.: Syntactic Structure and Semantical Reference II. Studia Logica **9**, 63–91 (1960)

[64] Tałasiewicz, M.: Philosophy of Syntax. Foundational Topics. Trends in Logic 29. Springer (2010)

[65] Uszkoreit, H.: Categorial unification grammar. In: Proc. 11th International Conference on Computational Linguistics, pp. 187–194 (1986)

[66] Wansing, H.: The Logic of Information Structures. Ph.D. Thesis, University of Amsterdam (1992)

[67] Wybraniec-Skardowska, U.: Theory of Language Syntax. Kluwer (1991)

[68] Yetter, D.N.: Quantales and (non-commutative) linear logic. Journal of Symbolic Logic **55**, 41–64 (1990)

[69] Zielonka, W.: Axiomatizability of Ajdukiewicz-Lambek calculus by means of cancellation schemes. Zeitschrift f. mathematische Logik und Grundlagen der Math. **27**, 215–224 (1981)

[70] Zielonka, W.: Weak implicational logics related to the Lambek calculus - Gentzen versus Hilbert formalisms. In: Towards mathematical philosophy. Trends in Logic 28, pp. 201–212. Springer (2009)