Syntactic categories and types: Ajdukiewicz and modern categorial grammars

Wojciech Buszkowski
Faculty of Mathematics and Computer Science
Adam Mickiewicz University in Poznań

Abstract

This essay confronts Ajdukiewicz's approach to syntactic categories with modern categorial grammars, mainly AB-grammars (basic categorial grammars) and Lambek grammars, i.e. categorial grammars based on different versions of the Lambek calculus.

1 Introduction and preliminaries

[2] (1935) is the most often cited paper of Kazimierz Ajdukiewicz[1]. This paper proposes an algorithm for checking the grammatical correctness (syntactic connexion) of expressions, which is based on a reduction of indices (types) assigned to single words. Types indicate semantic categories of expressions: basic categories correspond to atomic types and functor categories to functor (functional) types. The theory of semantic categories was elaborated by S. Leśniewski within his systems of the foundations of mathematics [35]. The decomposition of a (meaningful) compound expression in the functor (an incomplete expression) and its arguments can be traced back to G. Frege and the concept of a semantic category to Husserl [25].

In [2] the algorithm is presented in a form appropriate for languages written in Polish notation. In fact, one of Ajdukiewicz's goals was to extend Polish notation (introduced by J. Łukasiewicz for propositional logics) to richer languages.

Ajdukiewicz's approach is seen nowadays as an anticipation of categorial grammars (type grammars); i.e. formal grammars based on type theories. Types were earlier used by the fathers of modern logic, implicitly by Frege and explicitly by B. Russell. Russell's types are *relational* (i.e. types of relations), whereas Ajdukiewicz's types are *functional* (i.e. types of functions).

Ajdukiewicz's type reduction procedure can be treated as a parsing algorithm which checks whether an arbitrary string of symbols (words) is well-formed (syntactically connected) according to the rules of type-theoretic syntax.

[1] We refer to Ajdukiewicz's papers originally published in German and Polish. The quotations, however, are based on their English translations, collected in [4]

Ajdukiewicz emphasizes the universality of the method: it can be applied to arbitrary languages, not only to particular logical formalisms. 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, which makes it possible to formally define and examine the syntactic connexion of a word pattern."

The term 'semantical category' was used in [2] (after Husserl and Leśniewski) in a sense better expressed by 'syntactic category', since the categories were defined with no explicit reference to semantics. In [3] (1960) Ajdukiewicz comments: "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, his categories from the syntactical viewpoint."

The notion of a syntactic category is the central notion discussed in the present paper. Some authors use the term 'category' in the sense of our 'type', but we prefer to discriminate between them: a category is a set of expressions, and a type is a formal expression (formula). In type logics, i.e. formal logics underlying type grammars, types play the role of formulae.

Now we will briefly recall the main ideas and notions of the theory of formal grammars, especially categorial grammars.

Ajdukiewicz's types are either atomic types, i.e. s (sentence), n (name), or functor types, i.e. fractions $\frac{\alpha}{\beta_1 \ldots \beta_n}$; the denominator shows the types of arguments of the functor (an incomplete expression), and the numerator shows the type of the complex expression formed out of the functor and the arguments. They are intended to denote the basic categories and the functor categories, respectively. In propositional logic every well-formed propositional formula is of type s, the negation connective of type $\frac{s}{s}$, and each binary connective of type $\frac{s}{ss}$. In first-order logic, every formula is of type s, every term of type n, every unary predicate symbol of type $\frac{s}{n}$, every binary predicate symbol of type $\frac{s}{nn}$, every unary function symbol of type $\frac{n}{n}$, quantifiers $\forall, \exists$ of type $\frac{s}{ns}$ (in languages without individual constants and function symbols), and so on. In English, proper nouns are of type n, verb phrases of type $\frac{s}{n}$, and transitive verb phrases of type $\frac{s}{nn}$. For the phrase 'John likes Mary', one assigns n to 'John' and 'Mary' and $\frac{s}{nn}$ to 'likes'. In Polish notation, the functor 'likes' precedes the arguments, and the phrase is rewritten as 'likes John Mary'. The corresponding sequence of types $\frac{s}{nn}, n, n$ reduces to s by applying (an instance of) the reduction rule

$$\text{(RED)} \frac{\alpha}{\beta_1 \ldots \beta_n}, \beta_1, \ldots, \beta_n \Rightarrow \alpha \, .$$

The Ajdukiewicz algorithm recognizes 'John likes Mary' as a well-formed sentence. In general, the reduction procedure may involve several applications of this reduction rule. An expression (i.e. a sequence of words) is *syntactically connected*, if the corresponding sequence of types reduces to a single type.

It might be said that this reduction procedure was, historically, the first parsing algorithm, an important method in mathematical linguistics. It is noteworthy that mathematical linguistics, originated by Noam Chomsky in 1956 (21 years after the publication of [2]) as a formal theory of natural language, was extensively developed in computer science for applications in programming languages. Parsing algorithms play a key role in both disciplines.

Bar-Hillel [5] (1953) modified this method towards a direct parsing of expressions which are not written in prefix notation. The functor types are of the form $\frac{\alpha}{\beta_1\ldots\beta_m;\gamma_1\ldots\gamma_n}$; here $\beta_1,\ldots,\beta_n$ correspond to the left and $\gamma_1,\ldots,\gamma_n$ to the right arguments of the functor. The reduction rule takes the form:

$$\text{(RED')}\ \beta_1,\ldots,\beta_m,\frac{\alpha}{\beta_1\ldots\beta_m;\gamma_1\ldots\gamma_n},\gamma_1,\ldots,\gamma_n \Rightarrow \alpha\,.$$

For instance, 'likes' is assigned type $\frac{s}{n;n}$, hence 'John likes Mary' yields the sequence $n,\frac{s}{n;n},n$, which reduces to s by a single application of (RED').

Bar-Hillel, Gaifman and Shamir [6] (1960) formulated the first precise definition of a categorial grammar; these categorial grammars are presently called basic categorial grammars or *AB-grammars* (a credit to Ajdukiewicz and Bar-Hillel). Types are either atomic, or functional, the latter being restricted to one-argument types $\alpha\backslash\beta$ (the argument of type α stands to the left of the functor) and β/α (the argument of type α stands to the right of the functor). The reduction rules are as follows:

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

An AB-grammar is formally defined as a triple $G=(\Sigma_G,I_G,s_G)$ such that Σ_G is a nonempty finite set, I_G is a map which assigns a finite set of types to each element of Σ_G, and s_G is a designated atomic type. Σ_G, I_G and s_G are called the lexicon (or: alphabet), the initial (lexical) type assignment and the designated type, respectively, of G. In examples we write $a:\alpha$ for $\alpha\in I_G(a)$. The elements of Σ_G are interpreted as words (lexical units) of a natural language; for a formal language, they are symbols of that language.

Let us note that [6] refers to Lambek [32] (1958) who introduced the slash notation for types and essentially extended the reduction procedure; see below.

The definition of an AB-grammar, given above, has been commonly adopted in the literature (sometimes with minor changes, e.g. s_G need not be atomic). The same definition is applicable for type grammars based on richer type logics. Let us briefly comment on some aspects of the definition.

Ajdukiewicz's two atomic types, s (sentence) and n (name), are replaced by an arbitrary, finite set of atomic types. This fully agrees with Ajdukiewicz's views. He writes in [2]: "If the concept of syntactic connexion were to be defined in strict generality, nothing could be decided about the number and kind of basic semantic and functor categories, since these may vary in different languages. For the sake of simplicity, however, we shall restrict ourselves (like Leśniewski) to languages having only two basic semantic categories, that of sentences and that of names." Lambek [34] presents a type grammar for English, applying

33 atomic types, e.g. π (subject), π_k for $k = 1, 2, 3$ ($k-$th person subject), $\mathbf{s}$ (statement), $\mathbf{s}_1$ (statement in present tense), $\mathbf{s}_2$ (statement in past tense), $\mathbf{n}$ (name), $\mathbf{n}_0$ (mass noun), $\mathbf{n}_1$ (count noun), $\mathbf{n}_2$ (plural noun), $\bar{\mathbf{n}}$ (complete noun phrase), $\bar{\mathbf{n}}_k$ ($k-$th person complete noun phrase), and others.

The map I_G is allowed to assign finitely many types to one word. This reflects the syntactic and semantic ambiguity of many words in natural languages. For instance, 'can' is used as a noun and a modal verb. Such homonyms can be eliminated by introducing different words, say, can_1, can_2, like in lexicons. This solution, however, seems problematic for other ambiguities, where a word can play different syntactic and semantic roles, preserving (essentially) the same meaning. For instance, (1) 'and' is a sentence conjunction in 'Mary sings and Mary dances' and a verb conjunction in 'Mary sings and dances', (2) an adverb can act on intransitive verbs and transitive verbs, (3) adjectives and determiners ('some', 'no', 'the') can act on singular nouns and plural nouns, and so on. Even formal languages of logic often require more than one type of one symbol. The type of quantifiers $(s/s)/n$ (i.e. the one-argument counterpart of $\frac{s}{ns}$) does not work for languages with function symbols, since the first argument of a quantifier must be a variable, not a compound term. We need different atomic types for variables (n) and compound terms (n'); quantifiers are typed $(s/s)/n$, but each unary function symbol has two types: n'/n and n'/n'.

Formal languages of logic usually contain infinitely many symbols, e.g. infinitely many variables. The corresponding AB-grammar has an infinite alphabet. For such grammars, the infinite alphabet Σ_G can be partitioned in finitely many disjoint sets Σ_i such that each symbol from Σ_i is assigned the same finite set of types. This generalization has no essential impact on our further considerations, and we adopt it, while discussing languages of logic. For natural languages and formal languages of mathematical linguistics we assume the finiteness of Σ_G.

Finite sequences of elements of Σ are called *strings* on Σ. Σ^* (resp. Σ^+) denotes the set of all (resp. nonempty) strings on Σ. By ϵ we denote the empty string. By *a language* on Σ we mean an arbitrary set $L \subseteq \Sigma^*$. A language L is said to be $\epsilon-$free, if $\epsilon \notin L$.

Let G be an AB-grammar. One says that G assigns type α to a string $a_1 \ldots a_n$, where $n > 0$ and each a_i belongs to Σ_G, if for any $i = 1, \ldots, n$ there exists $\alpha_i \in I_G(a_i)$ such that the sequence $(\alpha_1, \ldots, \alpha_n)$ reduces to α by a number of applications of (RED\), (RED/); we write: $a_1 \ldots a_n :_G \alpha$. $L(G, \alpha)$ denotes the set of all $u \in \Sigma_G^+$ such that $u :_G \alpha$. The set $L(G, s_G)$ is called the language of G (or: generated by G) and denoted by $L(G)$.

Two grammars are said to be (weakly) equivalent if they generate the same language; two classes of grammars are equivalent if they generate the same class of languages (these notions can be applied for grammars of different kinds provided that $L(G)$ is defined for them). The main mathematical theorem of [6] states *the equivalence of AB-grammars and $\epsilon-$free context-free grammars*. Let us explain the second notion.

Context-free grammars (CFGs) form one of the four classes of the Chomsky hierarchy of formal grammars (the others are regular grammars, context-

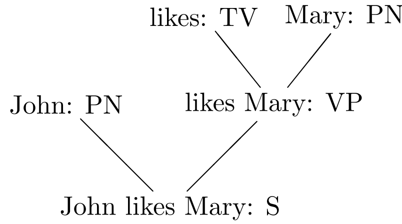

Figure 1: A parse tree in a context-free grammar.

sensitive grammars and general production grammars). A CFG G consists of a terminal alphabet Σ_G, a nonterminal alphabet V_G, an initial symbol $S_G \in V_G$, and a set of production rules $P_G \subset V_G \times (\Sigma_G \cup V_G)^*$; the sets Σ_G, V_G, P_G are finite and $\Sigma_G \cap V_G = \emptyset$. A production rule (A, u) is written $A \mapsto u$ and interpreted as a rewriting rule: rewrite A as u. The language of G, denoted by $L(G)$, is defined as the set of all strings on Σ_G which can be derived from S_G by a finite number of applications of production rules from P_G.

A production rule of the form $A \mapsto \epsilon$ is called a nullary rule. A CFG G is said to be $\epsilon-$free, if P_G contains no nullary rule (then, $L(G)$ is certainly $\epsilon-$free). A language is said to be context-free, if it is generated by a CFG. Every $\epsilon-$free context-free language is generated by some $\epsilon-$free CFG.

Every $\epsilon-$free CFG can be transformed into an equivalent CFG in some normal form. The basic normal form admits only production rules $A \mapsto B_1 \ldots B_n$, $(n > 0)$, or $A \mapsto a$ (the lexical rules); hereafter the upper-case letters represent nonterminal symbols and the lower-case letters represent terminal symbols. The Chomsky normal form is the basic normal form such that $n = 2$ in each nonlexical rule. The Greibach normal form (precisely: 2-normal form) admits only rules of the form $A \mapsto a$, $A \mapsto aB$, and $A \mapsto aBC$.

In a CFG the derivation of a string x from a nonterminal A can be written in a linear form $A \Rightarrow x_1 \Rightarrow \cdots \Rightarrow x_n$, where $n \geq 0$, $x_n = x$, and each $\Rightarrow$ represents one application of a production rule. For instance, with the rules S$\mapsto$ PN VP, VP$\mapsto$ TV PN, TV$\mapsto$ 'likes', PN$\mapsto$ 'John', PN$\mapsto$ 'Mary' one derives: S $\Rightarrow$ PN VP $\Rightarrow$ PN TV PN $\Rightarrow$ 'John' TV PN $\Rightarrow$ 'John likes' PN $\Rightarrow$ 'John likes Mary'. (Here PN, VP and TV are nonterminals representing proper noun, verb phrase and transitive verb)

The derivation tree corresponding to this linear derivation is depicted in Figure 1. More precisely, this is a parse tree; the derivation tree omits the terminal strings in the internal nodes. This tree determines a unique *phrase structure* (John (likes Mary)), i.e. a decomposition of the string of words in constituents, corresponding to subtrees of the tree. One defines *the ps-language* of G as the set $L^P(G)$ which consists of all phrase structures determined by derivation trees of strings from $L(G)$. The full parse tree can be encoded by adding nonterminals to the phrase structure: here $(\mathrm{John}_{PN}(\mathrm{likes}_{TV}\mathrm{Mary}_{PN})_{VP})_S$.

Every AB-grammar G can easily be transformed into an equivalent CFG G'

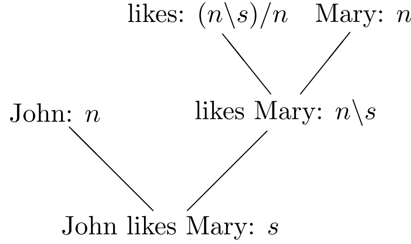

Figure 2: A parse tree in an AB-grammar.

in Chomsky normal form. We define: $\Sigma_{G'} = \Sigma_G$, $V_{G'}$ consists of all subtypes of the types appearing in G, $S_{G'} = s_G$, and $P_{G'}$ consists of all rules $\beta \mapsto \alpha(\alpha\backslash\beta)$, for $\alpha\backslash\beta \in V_{G'}$, $\beta \mapsto (\beta/\alpha)\alpha$, for $\beta/\alpha \in V_{G'}$, and all lexical rules $\alpha \mapsto a$, for $\alpha \in I_G(a)$. Consequently, every AB-grammar generates some $\epsilon-$free context-free language. The converse direction of the equivalence theorem (every $\epsilon-$free context-free language is generated by some AB-grammar) is nontrivial. It is equivalent to the Greibach normal form theorem for CFGs: every $\epsilon-$free CFG is equivalent to some CFG in Greibach normal form. The latter theorem has independently been proved by S. Greibach in 1965.

The parse tree of 'John likes Mary' in an AB-grammar is shown in Figure 2. As above, the tree determines the phrase structure (John (likes Mary)). For AB-grammars, however, it is natural to distinguish in every compound substructure the functor and the argument, which yields a *functor-argument structure* (fa-structure). The fa-structure for this example is (John (likes Mary)$_1$)$_2$; this means that (likes Mary) is the functor in (John (likes Mary)), and 'likes' is the functor in (likes Mary).

In general, a compound fa-structure is of the form $(X_1X_2)_i$, where $i = 1$ or $i = 2$ indicates X_i as the functor, and a compound phrase structure is of the form (X_1X_2). The precise definition is recursive: (i) all elements of Σ are fa-structures on Σ, (ii) if X, Y are fa-structures on Σ, then $(XY)_1$ and $(XY)_2$ are fa-structures on Σ. One can represent fa-structures and phrase structures as trees; see Figure 3.

Here we confine ourselves to one-argument types, but it is not very essential. Many-argument types of the form, say, $\beta_1 \dots \beta_m\backslash\alpha/\gamma_1 \dots \gamma_n$ (a flattened version of Bar-Hillel's fractions) might be admitted as well, and the corresponding fa-structures would be of the form $(X_1 \dots X_k)_i$, where $1 \leq i \leq k$; this means that X_i is the functor, $X_1, \dots, X_{i-1}$ are the left arguments, and $X_{i+1}, \dots, X_k$ are the right arguments. This approach seems even more natural in many situations. For instance, sentential connectives 'and' ,'or' can be assigned type $s\backslash s/s$, whereas the one-argument format enforces $(s\backslash s)/s$ and/or $s\backslash(s/s)$, both artificial. Categorial grammars with many-argument types were considered in [11, 13, 14]. In fact, these papers admit more general structures and types, which indicate a syntactic operation acting on the surface level; besides concatenation

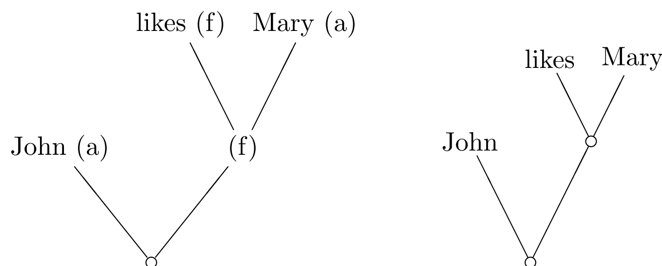

Figure 3: (John (likes Mary)$_1$)$_2$ and (John (likes Mary)) depicted as trees.

one may regard other operations, e.g. substitution is useful for discontinuous expressions.

The representation of expressions as fa-structures quite naturally reflects the fundamental idea of categorial grammars: each compound expression can be constructed by application of a functor to its argument(s). The particular form of fa-structures, admitted here, may look strange even for logicians. In formal languages of logic, functors are usually simple symbols, and their role is determined by their meaning, e.g. connectives, function symbols, relation symbols etc. are functors, and variables, individual constants, etc. are not functors. A standard tree representation of the propositional formula $NCpq$ has the root N, its daughter C, and p and q as the daughters of C. This representation, however, is inappropriate for languages with compound functors, and natural languages belong to this class; e.g. (likes Mary) can be treated as the compound functor in (John (likes Mary)), (no student) as the compound functor in ((no student) came), and so on. Our notation for fa-structures seems clear and economical; other authors use some variants of it, e.g. Kanazawa [27] writes FA(X, Y) (forward application) for our $(XY)_1$ and BA(X, Y) (backward application) for our $(XY)_2$.

Y. Bar-Hillel argued that AB-grammars are an 'analytic' counterpart of CFGs; while the latter generate the terminal string of words from the initial symbol, the former start from a string of words and reduce the corresponding sequence of types to the designated type. Currently this difference seems less important; parsing algorithms for both kinds of grammars can be designed in the bottom-up style and the top-down style; see [24]. The most characteristic feature distinguishing categorial grammars from production grammars is *lexicality*: the grammatical information on the particular language is totally encoded in the lexical type-assignment, and the processing of compound expressions is based on some universal rules, independent of the particular language (e.g. (RED\), (RED/)). This is not the case for production grammars. Typical context-free production rules are of the form $A \mapsto B_1 \dots B_n$, e.g. $S \mapsto$NP VP (a sentence consists of a noun phrase and a verb phrase). A lexicalization of CFGs can be done through their reduction to the Greibach normal form; the production

rule $A \mapsto aB_1 \dots B_n$ provides information on the syntactic role of a, which can be expressed by $a : (\dots((A/B_n)/B_{n-1})/\cdots)/B_1$ in an AB-grammar. AB-grammars may be regarded as a lexical counterpart of CFGs, more refined and logically oriented than the Greibach normal form.

The reduction procedure based on (RED\), (RED/) can be replaced by a richer type logic. The seminal paper in this direction is Lambek [32] (1958), introducing Syntactic Calculus, nowadays called Lambek Calculus and denoted by L. It is usually presented as a sequent system; *sequents* are formal expressions of the form $\alpha_1, \dots, \alpha_n \Rightarrow \alpha$, whose intended meaning is: if $u_1, \dots, u_n$ are strings of type $\alpha_1, \dots, \alpha_n$, respectively, then the concatenation $u_1 \cdots u_n$ is of type α. In L, types are formed out of atomic types (variables) by means of three operators (connectives): $\cdot$ (product, fusion, multiplicative conjunction), \ (right multiplicative implication), and / (left multiplicative implication). Some authors write $\rightarrow$ for \ and $\leftarrow$ for /. Besides the laws equal to (RED\), (RED/), there are many other laws provable in L, for instance: $\alpha\backslash\beta, \beta\backslash\gamma \Rightarrow \alpha\backslash\gamma$ and $\alpha/\beta, \beta/\gamma \Rightarrow \alpha/\gamma$ (composition laws), $\alpha \Rightarrow (\beta/\alpha)\backslash\beta$ and $\alpha \Rightarrow \beta/(\alpha\backslash\beta)$ (type-raising laws), $\alpha \Rightarrow \beta\backslash(\beta \cdot \alpha)$ and $\alpha \Rightarrow (\alpha \cdot \beta)/\beta$ (expansion laws).

Many extensions and variants of L play the role of type logics in the modern literature on categorial grammars. In the logical community these logics are called *substructural logics*; their sequent systems lack some structural rules, appearing in Gentzen-style sequent systems for intuitionistic and classical logic (exchange, contraction, left and right weakening). Given a type logic $\mathcal{L}$, the categorial grammars based on $\mathcal{L}$ are referred to as $\mathcal{L}-$grammars. They are defined like AB-grammars except that one may admit a larger set of types and the provability in $\mathcal{L}$ replaces the former reduction procedure. In fact, this procedure is equivalent to a subsystem of L, denoted by AB. (So AB-grammars are the type grammars based on AB.)

Substructural logics enjoy a still growing attention of contemporary logicians, since they seem to play a fundamental role in the family of nonclassical logics. From the algebraic point of view, they are logics of residuation: implication(s) is (are) treated as residual operation(s) for product. Many important nonclassical logics, e.g. Łukasiewicz's many-valued logics, Hajek's fuzzy logics, some relevant logics, intuitionistic logic, linear logics, can be presented as axiomatic extensions of Full Lambek Calculus (FL), i.e. Lambek Calculus with lattice connectives $\wedge, \vee$ and constants $1, 0$. Type logics of categorial grammars are usually restricted to the basic logics of this family: AB, L, FL and their variants.

In the present paper we cannot present many highly interesting aspects of type logics, such as their algebras, different proof systems, relations to other nonclassical logics, complexity, and others. The reader is referred to [23] (a recent monograph on substructural logics), and the author's survey papers [16, 18]; also see Moortgat [38, 39].

Another characteristic feature of categorial grammars is their close relation to type-theoretic semantics. The semantical interpretation of Ajdukiewicz's approach was proposed by Bocheński [9], and Ajdukiewicz adopted it in [3]. The reduction procedure of AB-grammars can be reflected in semantics as the com-

putation of meanings (denotations) by function application as the only computation rule; richer type logics involve lambda abstraction. The Curry-Howard isomorphism provides a correspondence between logical proofs in the natural deduction format and logical forms of expressions, usually represented by typed lambda-terms, which determine the denotations of these expressions, if interpreted in a particular model.

In this short paper we cannot thoroughly discuss type-theoretic semantics, which is today a large, advanced discipline, developed in different directions. We focus on syntax and only briefly outline some interrelations with semantics. For more information on type-theoretic semantics the reader is referred to Montague [37] and further developments in e.g. [30, 7, 42].

The further contents are divided into two sections. In Section 2 we consider AB-grammars: the relation of typed categories to categories as substitution classes, some basic properties of structure languages generated by these grammars, and others. Section 3 is concerned with syntactic categories in Lambek categorial grammars; we emphasize essential differences in comparison with the AB-framework. Although we focus on basic philosophical ideas, we employ some mathematical notions, needed for explication of these ideas. For AB-grammars, these are mainly free algebras of fa-structures and some congruences in them. Lambek grammars require some elements of proof theory and residuated algebras. All linguistic examples are simple; they merely illustrate general principles. Fine linguistic applications are provided in [40, 39, 34].

This essay often refers to earlier publications of the author and others. Especially the author's book [14] (1989), in Polish, extensively discussed the foundations of AB-grammars and L-grammars. A formal reconstruction of Ajdukiewicz's ideas can be found in the book [50] of U. Wybraniec-Skardowska. Our approach is more general ([50] is confined to one-valued grammars for languages in prefix notation) and regards more concepts, central for the subject-matter; on the other hand, we do not take into account the distinction between expressions (types) as tokens and expressions (types) as types, elaborated in [50]. A recent philosophical discussion of the basic properties of categorial grammars, mainly AB-grammars, can be found in [46]. In this paper we briefly recall some earlier results (without proofs), but our main concerns are different: we confront Ajdukiewicz's approach to syntactic categories with later developments and try to point out main similarities and differences.

Type-theoretic syntax and semantics were studied by many other authors (in Poland, by R. Suszko [45], A. Nowaczyk [41], and others). These works were primarily concerned with formal languages; especially the construction of higher-order logical languages and their models. Here we focus on categorial grammars appropriate for the description of natural languages.

Not all problems posed by Ajdukiewicz in [2, 3] are touched on here. For example, we do not discuss the distinction between functors and (variable binding) operators, though we apply Ajdukiewicz's types with | in a different context.

2 Syntactic categories in AB-grammars

One of the basic intuitions of the theory syntactic categories is *the principle of substitution*: two expressions belong to the same category, if they can be substituted for each other in sentential contexts. This principle, admitted by Ajdukiewicz after Husserl (who wrote 'meaningful' for 'sentential') and Leśniewski, can be explicated in different ways.

In [1], Ajdukiewicz writes: "expressions A and B, taken in sense a and b respectively, belong to the same semantical category if and only if every sentence Z_A containing expression A in sense a upon replacement of A by B taken in sense b (the meaning of all other expressions and their interconnections remaining unaltered) is transformed into an expression which is also a sentence, and if vice versa: every sentence Z_B upon replacement of B by A (with analogous qualifications) is also transformed into a sentence."

A similar, but not identical formulation appears in [2]. "The word or expression A, taken in sense x, and the word or expression B, taken in sense y, belong to the same semantic category if and only if there is a sentence (or sentential function) S_A, in which A occurs with meaning x, and which has the property that if S_A is transformed into S_B upon replacing A by B (with meaning y), then S_B is also a sentence (or sentential function). (It is understood that in this process the other words and the structure of S_A remain the same.)"

Our further discussion will focus on these two versions of the principle of substitution. The striking difference between them is the universal quantification 'every sentence Z_A' in the former and the existential quantification 'there is a sentence' in the latter. Furthermore, it is not obvious that the equivalence classes of the relation defined as in the first version coincide with the typed categories, corresponding to different types. We show that both formulations are compatible with the theory of AB-grammars, if interpreted in a proper way.

In both versions A and B belong to the same category if and only if they are mutually substitutable in sentences (sentential functions). It follows that the categories are determined by some set L of sentences; for languages of logic, sentences should be replaced by formulae (i.e. propositional functions).

Both versions assume that the replacement of A by B preserves the sentence structure (in the first version: the interconnections of all other expressions). It follows that sentences are treated as structured expressions, for instance, phrase structures or fa-structures rather than strings of words. Let us discuss the matter more formally.

In mathematical linguistics, for a language $L \subseteq \Sigma^*$ one defines *the syntactic congruence* determined by L as follows:

$$u \equiv_L v \Leftrightarrow \forall_{w_1, w_2 \in \Sigma^*} (w_1 u w_2 \in L \Leftrightarrow w_1 v w_2 \in L)\,,$$

for $u, v \in \Sigma^*$. The relation $\equiv_L$ is a congruence in the free monoid Σ^* (the monoid operation is concatenation, and ϵ is the unit); furthermore, $\equiv_L$ is the largest congruence in Σ^* *compatible* with L (this means: L is the union of some family of equivalence classes). Clearly $u \equiv_L v$ explicates the idea of mutual substitutability of u and v in the strings from L.

By the index of an equivalence relation $\equiv$ one means the cardinality of the family of the equivalence classes of $\equiv$. It is well known that $\equiv_L$ is of finite index if and only if L is a regular language (i.e. accepted by a finite-state automaton); see [24]. The regular languages are a poor family of formal languages. The standard languages of logic are not regular. For instance, the language L_{PL} of propositional logic in Polish notation is not regular; the strings C, CC, CCC etc. belong to different equivalence classes of $\equiv_{L_{PL}}$, since e.g. $Cpp \in L_{PL}$, $CCpp \notin L_{PL}$. On the other hand, one can distinguish only three natural syntactic categories in L_{PL}: well-formed formulae, negation, binary connectives (one can add the fourth category: not well-formed expressions). Consequently even for simple formal languages L, syntactic categories cannot be defined as the equivalence classes of $\equiv_L$.

Following Ajdukiewicz's suggestion, we consider analogous relations on the level of structures. By Σ^F (resp. Σ^P) we denote the set of all fa-structures (resp. phrase structures) on Σ. Any set $L \subseteq \Sigma^F$ (resp. $L \subseteq \Sigma^P$) is called a fa-language (resp. a ps-language). *Contexts* are structures containing one occurrence of a special atom $_$ (a place for substitution). If X is a context, then $X[Y]$ denotes the substitution of Y for $_$ in X. $\Sigma^{F[]}$ (resp. $\Sigma^{P[]}$) denotes the set of all fa-contexts (resp. ps-contexts) on Σ.

Let $L \subseteq \Sigma^F$. The relation $\equiv_L$ on Σ^F is defined as follows:

$$X \equiv_L Y \Leftrightarrow (\forall Z \in \Sigma^{F[]})(Z[X] \in L \Leftrightarrow Z[Y] \in L)\,,$$

for $X, Y \in \Sigma^F$. (For $L \subseteq \Sigma^P$, the relation $\equiv_L$ on Σ^P is defined in a similar way.) $\equiv_L$ is a congruence in the free algebra Σ^F (the operations are $(\ ,\ )_1$ and $(\ ,\ )_2$); furthermore, $\equiv_L$ is the largest congruence in Σ^F compatible with L. The analogous facts hold for the ps-version. The equivalence classes of $\equiv_L$ are called *the substitution classes* of the fa-language L, and similarly for ps-languages.

It is known that that the ps-languages of CFGs are precisely the ps-languages L such that $\equiv_L$ is of finite index (i.e. the regular ps-languages). WARNING: for unrestricted CFGs, one must admit phrase structures of the form $(X_1 \ldots X_n)$, for $n \geq 1$; our 'binary' phrase structures are appropriate for CFGs in Chomsky normal form.

For AB-grammars the most natural representations of expressions are fa-structures. They are uniquely determined by proofs in AB, presented as a natural deduction system (ND-system).

The axioms are:

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

and the inference rules are the elimination rules for $\backslash, /$:

$$\text{(E}\backslash\text{)}\ \frac{\Gamma \Rightarrow \alpha\,;\ \Delta \Rightarrow \alpha\backslash\beta}{\Gamma, \Delta \Rightarrow \beta}\,,\ \text{(E/)}\ \frac{\Gamma \Rightarrow \beta/\alpha\,;\ \Delta \Rightarrow \alpha}{\Gamma, \Delta \Rightarrow \beta}\,.$$

Here Γ and Δ stand for finite sequences of formulae and 'Γ, Δ' denotes the concatenation of Γ and Δ. (This notation is commonly used in sequent systems.)

Proofs in this system, presented as proof trees, determine fa-structures on the antecedents of sequents. These are fa-structures on the set of formulae

(types); we call them formula structures. They can be made explicit, if one replaces the conclusion of (E\) by $(\Gamma, \Delta)_2 \Rightarrow \beta$ and the conclusion of (E/) by $(\Gamma, \Delta)_1 \Rightarrow \beta$. WARNING: here Γ and Δ stand for formula structures, and $(\Gamma, \Delta)_i$ is a compound formula structure whose constituents are Γ and Δ (in formula structures we separate the constituents by a comma, just for better readability).

Let G be an AB-grammar. The map I_G is extended for all $X \in \Sigma_G^F$ by setting:

$$I_G((XY)_i) = \{(\Gamma, \Delta)_i : \Gamma \in I_G(X), \Delta \in I_G(Y)\}\,.$$

We say that G assigns type α to $X \in \Sigma_G^F$ (write: $X :_G \alpha$), if there is $\Gamma \in I_G(X)$ such that $\Gamma \Rightarrow \alpha$ is provable in AB. This definition is compatible with the procedure of determining fa-structures from parse trees, described in Section 1.

We define $T_G^F(X) = \{\alpha : X :_G \alpha\}$, $L^F(G, \alpha) = \{X \in \Sigma_G^F : \alpha \in T_G^F(X)\}$. We also define $\mathcal{T}(G) = \bigcup\{I_G(a) : a \in \Sigma_G\} \cup \{s_G\}$ and $s(\mathcal{T}(G))$ as the set of all subformulae of the formulae (types) from $\mathcal{T}(G)$.

Since the conclusions of (E\) and (E/) consist of subformulae of the formulae occurring in the premises, then $T_G^F(X) \subseteq s(\mathcal{T}(G))$ for any $X \in \Sigma_G^F$.

The fa-language $L^F(G, s_G)$ is called the fa-language generated by G and denoted by $L^F(G)$. By dropping all functor markers in the structures from $L^F(G)$ one obtains the ps-language of G, denoted by $L^P(G)$, and $L(G)$ is obtained by dropping all structure markers. The set $L^F(G, \alpha)$ can be referred to as *the category of type* α in G; see the discussion below.

EXAMPLE 1. Consider the following AB-grammar G. The lexicon consists of four words: 'Mary', 'is', 'very', 'apt'. I_G is defined by: 'Mary': n, 'is': $(n\backslash s)/a$, 'very': a/a, 'apt': a (so a is the type of adjectives). The designated type is s. This grammar generates an infinite fa-language, whose first elements are:

(Mary (is apt)$_1$)$_2$
(Mary (is (very apt)$_1$)$_1$)$_2$
(Mary (is (very (very apt)$_1$)$_1$)$_1$)$_2$

The ps-language of G contains the phrase structures (Mary (is apt)), (Mary (is (very apt))) and so on, and the language of G the strings 'Mary is apt', 'Mary is very apt' and so on. $L^F(G, a)$ contains the fa-structures 'apt', (very apt)$_1$, (very (very apt)$_1$)$_1$ and so on.

We need some notions which intuitively appeal to the tree representation of fa-structures and phrase structures; see Section 1. We only consider paths going upwards. An f-path is a path for which all nodes, possibly except the first one, are marked by (f). The f-degree of $L \subseteq \Sigma^F$, denoted $d_f(L)$, is the maximal length of f-paths in structures from L, if it exists; we set $d_f(\emptyset) = 0$, and $d_f(L) = \infty$, if the lengths of f-paths are unbounded in L. The following notions are meaningful for fa-structures, phrase structures and the corresponding languages. The outer degree of a structure X is the length of the shortest path in X from the root to a leaf. The degree of X, denoted $d(X)$, is the maximal outer degree of substructures of X. The degree of L, denoted $d(L)$, is the maximal $d(X)$, for $X \in L$, if it exists; we set $d(\emptyset) = 0$ and $d(L) = \infty$, if $\{d(X) : X \in L\}$ is unbounded.

Example 2. In X depicted in Figure 3, the maximal f-path goes from the root to 'likes' and is of length 2, but $d(X) = 1$, since the distance of each node to the closest leaf is at most 1. The f-degree of the fa-language from Example 1 equals 2, but its degree equals 1. $d_f(\Sigma^F) = \infty$, $d(\Sigma^F) = \infty$, $d(\Sigma^P) = \infty$.

The fa-languages of AB-grammars can be characterized as *the fa-languages L such that $\equiv_L$ is of finite index and $d_f(L)$ is finite.* This result has been proved in [11].

We outline the proof in one direction, since it employs some relevant notions. For an AB-grammar G, the relation $\equiv^F_G$ on Σ^F_G is defined as follows:

$$X \equiv^F_G Y \Leftrightarrow T^F_G(X) = T^F_G(Y)\,,$$

for $X, Y \in \Sigma^F_G$. If U, V are sets of types, one defines:

$$U \rhd V = \{\beta : \exists_\alpha(\alpha \in U \wedge (\alpha\backslash\beta) \in V)\}\,, U \lhd V = \{\beta : \exists_\alpha((\beta/\alpha) \in U \wedge \alpha \in V)\}\,.$$

It is easy to show:

$$T^F_G((XY)_1) = T^F_G(X) \lhd T^F_G(Y)\,,\ T^F_G((XY)_2) = T^F_G(X) \rhd T^F_G(Y)\,,$$

for all $X, Y \in \Sigma^F_G$. Consequently $X \equiv^F_G X'$ and $Y \equiv^F_G Y'$ entail $(XY)_i \equiv^F_G (X'Y')_i$, which means that $\equiv^F_G$ is a congruence in Σ^F_G. It is compatible with $L^F(G)$, since $X \in L^F(G)$ if and only if $s_G \in T^F_G(X)$. The index of $\equiv^F_G$ is finite (at most 2^m, where m is the cardinality of $s(\mathcal{T}(G))$). Since $\equiv^F_G \subseteq \equiv_{L^F(G)}$ (the former is a congruence compatible with $L^F(G)$, and the latter is the largest congruence compatible with $L^F(G)$), then every equivalence class of $\equiv_{L^F(G)}$ is the union of a family of equivalence classes of $\equiv^F_G$. Consequently $\equiv_{L^F(G)}$ is of finite index.

To show that $d_f(L^F(G))$ is finite it is convenient to represent types as fa-structures on the set of variables: write $(\alpha\beta)_2$ for $\alpha\backslash\beta$ and $(\alpha\beta)_1$ for α/β. Although the new notation looks weird, it is quite helpful; e.g. the degree and the f-degree of a set of types can be defined as for fa-languages. It is easy to show that for any AB-grammar G, $d_f(L^F(G)) \leq d_f(\mathcal{T}(G))$, and consequently $d_f(L^F(G))$ is finite.

The ps-languages of AB-grammars can be characterized as *the ps-languages L such that $\equiv_L$ is of finite index and $d(L)$ is finite.* Σ^P is generated by the CFG with rules $S \mapsto S\,S$, $S \mapsto a$, for $a \in \Sigma$, but by no AB-grammar, since $d(\Sigma^P) = \infty$. Consequently the ps-languages of AB-grammars are a proper subclass of the ps-languages of CFGs (in Chomsky normal form). One can say that AB-grammars are not strongly equivalent (precisely: P-equivalent) to CFGs, although the (weak) equivalence holds.

We return to the main issue of this section: the relation of categories as substitution classes to typed categories. For an AB-grammar G, the former can be identified with the substitution classes of $L^F(G)$ and the latter with the sets $L^F(G, \alpha)$. Take note that for $\alpha \notin s(\mathcal{T}(G))$, $L^F(G, \alpha) = \emptyset$. Although both families are finite, they are different in general.

We have shown that the relation $\equiv^F_G$ is always finer than (or equal to) the relation $\equiv_{L^F(G)}$. An AB-grammar G is said to be *well-constructed*, if it satisfies

the following conditions: (WC.1) $\equiv_{L^F(G)} = \equiv_G^F$, (WC.2) for any $\alpha \in s(\mathcal{T}(G))$, $L^F(G,\alpha) \neq \emptyset$.

(WC.1) requires that each substitution class of $L^F(G)$ consists of all structures which are assigned the same set of types in G. (WC.2) additionally requires that G employs no 'void' type, i.e. a type not assigned to any expression.

For $L \subseteq \Sigma^F$, by $s(L)$ we denote the set of all substructures of the structures from L. For any $L \subseteq \Sigma^F$, if $s(L) \neq \Sigma^F$, then $\Sigma^F - s(L)$ is a single substitution class of L. This class consists of not well-formed expressions with respect to L and is denoted by non(L) (the class of nonsense). Notice that for any AB-grammar G, $s(L^F(G)) \neq \Sigma_G^F$, since $d_f(\Sigma_G^F) = \infty$, but $d_f(s(L^F(G)))$ is finite (equals $d_f(L^F(G))$). Consequently non$(L^F(G))$ is defined.

It follows from (WC.1) that X belongs to non$(L^F(G))$ if and only if $T_G^F(X) = \emptyset$. The implication $(\Leftarrow)$ is obvious. For $(\Rightarrow)$, observe that non$(L^F(G))$ must contain a structure Y such that $T_G^F(Y) = \emptyset$, since the f-degree of non$(L^F(G))$ is infinite, but the union of all typed categories in G has a finite f-degree. Thus, for any X in non$(L^F(G))$, we have $X \equiv_{L^F(G)} Y$, hence $X \equiv_G^F Y$, by (WC.1), which yields $T_G^F(X) = \emptyset$.

Example 3. We present some examples of grammars which are not well-constructed. G_1 assigns $s/(p/q)$ to a and p/q to b, $\Sigma_{G_1} = \{a, b\}$, $s_{G_1} = s$. Then, $L^F(G_1) = \{(ab)_1\}$, (WC.1) holds, but (WC.2) fails, since $L^F(G,q) = \emptyset$. G_2 assigns s/p and s/q to a, p to b, and q to c, $\Sigma_{G_2} = \{a,b,c\}$, $s_{G_2} = s$. Then, $L^F(G_2) = \{(ab)_1, (ac)_1\}$, (WC.2) holds, but (WC.1) fails: b and c belong to the same substitution class, but $T_{G_2}^F(b) \neq T_{G_2}^F(c)$.

These examples are artificial. If one designs an AB-grammar for a formal language or (a fragment of) a natural language, then one usually obtains a well-constructed grammar. In [11] it is proved that *every AB-grammar G is F-equivalent to a well-constructed AB-grammar G'*, where the F-equivalence means: $L^F(G) = L^F(G')$ ([11] uses 'adequate' for 'well-constructed'). We skip the proof. We only note that the atomic types in G', different from $s_{G'}$, are in a one-one correspondence with the substitution classes of $L^F(G)$, different from the nonsense class, and $s_{G'}$ is assigned to the structures from $L^F(G)$.

[13] provides an effective construction of G' from G. One uses the powerset algebra $\mathcal{P}(s(\mathcal{T}(G)))$ with operations $\rhd, \lhd$. The subalgebra $A(G)$ of this algebra, generated by the set $\{I_G(a) : a \in \Sigma_G\}$, is isomorphic to the quotient-algebra $\Sigma_G^F / \equiv_G^F$; the isomorphism is defined by $h([X]_\equiv) = T_G^F(X)$. $A(G)$ is a finite algebra, effectively constructed from G. The elements of $A(G)$, i.e. certain sets of types, represent the structures from Σ_G^F up to $\equiv_G^F$.

By this method, G_1 from Example 3 is transformed into the well-constructed grammar $a : s/p$, $b : p$, and G_2 to $a : s/p$, $b : p$, $c : p$.

The well-constructed grammars attain the closest concord between the substitution classes of the generated fa-language and the typed categories of the grammar, which is possible for the general case. Since one expression can be assigned several types, typed categories may overlap, hence they do not partition the universe. Every well-constructed AB-grammar G satisfies the following.

(S-T) Every typed category $L(G,\alpha)$, for $\alpha \in s(\mathcal{T}(G))$, is the union of a

nonempty family of substitution classes of $L^F(G)$ which are contained in $s(L^F(G))$. Every substitution class of $L^F(G)$, different from non$(L^F(G))$, is the intersection of a nonempty family of typed categories.

We show that the perfect concordance can be reached for one-valued grammars. An AB-grammar is said to be *one-valued* (or: rigid), if $I_G(a)$ contains at most one type, for any $a \in \Sigma_G$. Less formally, a one-valued AB-grammar assigns at most one type to each word.

By $\mathcal{G}_1$ we denote the class of one-valued AB-grammars. If $G \in \mathcal{G}_1$, then every $X \in \Sigma_G^F$ is assigned at most one type. So $X \equiv_G^F Y$ if and only if X and Y are assigned either the same type, or no type in G. If X is assigned a type in G, then each substructure of X is assigned a unique type.

For $G \in \mathcal{G}_1$, the relations $\equiv_G^F$ and $\equiv_{L^F(G)}$ coincide on $s(L^F(G))$. Nonetheless, not every $G \in \mathcal{G}_1$ is well-constructed.

EXAMPLE 4. Consider G_3 with the alphabet $\{a, b, c\}$, $s_{G_3} = s$, which assigns: $a : s/(p/q)$, $b : p/q$, $c : q$. Then, $L^F(G_3) = \{(ab)_1\}$, (WC.2) holds, but (WC.1) fails, since $(bc)_1$ and c belong to non$(L^F(G_3))$, but $(bc)_1 : p$, $c : q$ in G_3.

$G \in \mathcal{G}_1$ is well-constructed if and only if every type from $s(\mathcal{T}(G))$ is assigned to some $X \in s(L^F(G))$ (a strengthening of (WC.2)).

Every $G \in \mathcal{G}_1$ can be (effectively) transformed into a well-constructed $G' \in \mathcal{G}_1$, F-equivalent to G. Furthermore, G' is unique up to renaming atomic types; see [11, 13]. This construction is different from the one for arbitrary AB-grammars. One defines a relation $<_G$ on Σ_G^F: $X <_G Y$ if and only if there exists $Z \in s(L^F(G))$ such that Y is the functor of Z and either X is the argument of Z or $X \equiv_{L^F(G)} Z$ (this relation is closely related to the following relation between types: $\alpha < \alpha\backslash\beta$, $\beta < \alpha\backslash\beta$, and similarly for β/α). This relation is invariant with respect to $\equiv_{L^F(G)}$, hence it yields the quotient relation $<_G^{\equiv}$ on $\Sigma_G^F / \equiv_{L^F(G)}$, and the latter relation is well-founded (i.e. every nonempty set of substitution classes has a minimal element). The type assignment of G' is defined by induction on $<_G^{\equiv}$. In particular, the atomic types of G' correspond to the minimal substitution classes (one of them is $L^F(G)$, and it corresponds to $s_{G'}$). Again, the construction can be done effectively, using $A(G)$.

For instance, G_3 can be transformed into G_4, assigning $a : s/p$, $b : p$, $c : \emptyset$, which is well-constructed.

For any well-constructed $G \in \mathcal{G}_1$, there holds:

(S-T$_1$) every substitution class of $L^F(G)$, different from the class of nonsense, equals some typed category $L^F(G, \alpha)$ for a unique $\alpha \in s(\mathcal{T}(G))$, and conversely, every typed category $L^F(G, \alpha)$ with $\alpha \in s(\mathcal{T}(G))$ equals some substitution class contained in $s(L^F(G))$.

Every $G \in \mathcal{G}_1$ satisfies the following conditions:

$$\text{if } Z[X] \in s(L^F(G)) \text{ and } Z[X] \equiv_G^F Z[Y] \text{ then } X \equiv_G^F Y\,, \tag{1}$$

$$\text{if } X \equiv_G^F Y \text{ then } Z[X] \equiv_G^F Z[Y]\,, \tag{2}$$

for all $X, Y \in \Sigma_G^F$ and $Z \in \Sigma_G^{F[]}$. (1) and (2) express essentially the same as "the fundamental theorems of the theory of syntactic categories" in Wybraniec-Skardowska [50]. If $G \in \mathcal{G}_1$ is well-constructed, then in (1), (2) $\equiv_G^F$ can be replaced by $\equiv_{L^F(G)}$.

Consequently, for any $G \in \mathcal{G}_1$, we obtain:

$$\forall_{X,Y \in s(L^F(G))}(X \equiv_{L^F(G)} Y \Leftrightarrow \exists_{Z \in \Sigma_G^{F[]}}(Z[X] \in L^F(G) \wedge Z[Y] \in L^F(G)))\,. \quad (3)$$

We prove ($\Rightarrow$). Assume that $X, Y \in s(L^F(G))$ and $X \equiv_{L^F(G)} Y$. Then, $Z[X] \in L^F(G)$, for some context Z. Consequently $Z[Y] \in L^F(G)$ for the same Z, which yields the right-hand side of (3). We prove ($\Leftarrow$). Let Z be a context such that $Z[X] \in L^F(G)$ and $Z[Y] \in L^F(G)$. Then, $Z[X] :_G s_G$ and $Z[Y] :_G s_G$, and consequently $Z[X] \equiv_G^F Z[Y]$. By (1), $X \equiv_G^F Y$, which yields $X \equiv_{L^F(G)} Y$.

(3) shows that the two versions of the principle of substitution are equivalent for one-valued AB-grammars (if restricted to substructures of sentences). (3) with $L \subseteq \Sigma^F$ in the place of $L^F(G)$ remains true for a wider class of fa-languages. For instance, the language of combinatory logic satisfies the equivalence, though it cannot be generated by any one-valued AB-grammar (in the combinatory term xx two types are needed for x). Tarski [47] regarded the property expressed by (3) as a characteristic feature of formal languages in mathematics and logic. Quite likely, in [2] Ajdukiewicz formulated the principle with 'there is a sentence' instead of 'every sentence' under the influence of Tarski's view.

The second version of the principle was criticized by some authors as inadequate for natural languages. Indeed, in English and probably all natural languages one can find many examples of expressions which are substitutable in some but not all contexts, even under the requirement that the substitution must preserve the sentence structure. For instance, in 'Mary calls John' one can replace 'John' by 'a friend', which is impossible in 'Mary calls the old John'; in 'the teacher examines a student' one can replace 'a student' by 'two students', which is impossible in 'a student calls the teacher'.

Ajdukiewicz was certainly aware of these problems, when he formulated the principle in [2] and at the same time claimed the universality of his approach. This can be explained quite simply. His method is applicable ("in principle") to arbitrary languages provided that these languages are reconstructed in the style of formal languages. Both formulations of the principle of substitution contain the qualification: A in meaning (sense) a. This makes it possible not only to treat homonyms as different expressions (like 'can$_1$', can$_2$' in Section 1) but also to remove all syntactic ambiguities: if a word appears in different contexts with different types, then one treats the differently typed words as different words. In this way, every AB-grammar can be transformed into a one-valued grammar: if $I_G(a) = \{\alpha_1, \ldots, \alpha_n\}$, then a is replaced by n copies $a(1), \ldots, a(n)$ typed $a(i) : \alpha_i$. The resulting one-valued AB-grammar satisfies (3). If the former grammar is well-constructed, then the latter grammar is well-constructed.

As we have noted in Section 1, Ajdukiewicz wanted to extend the Łukasiewicz parenthesis-free notation (Polish notation) for richer languages. For propositional languages, considered by Łukasiewicz, there is one atomic type s, unary

connectives are typed s/s, binary connectives s/ss (or: $(s/s)/s$), and so on. Each well-formed string of symbols has a unique type and a unique structure, and they can be effectively computed from the string if one knows the number of arguments of each connective. For richer languages, the number of arguments is not sufficient; one must know the types of all symbols. We will explain the matter in more detail.

An AB-grammar G is said to be *unidirectional*, if all types in $\mathcal{T}(G)$ are $/-$types (resp. $\backslash-$types), this means, they do not contain $\backslash$ (resp. $/$). *The yield* of an fa-structure X is the string obtained from X by dropping all structure markers; *the ps-yield* of X is obtained by dropping all functor markers. Unidirectional one-valued AB-grammars are *categorially* and *structurally unambiguous*: for any string $u \in \Sigma_G^+$, there is at most one pair (X, α) such that u is the yield of X and $X :_G \alpha$; see [17].

This also holds for AB-grammars with Ajdukiewicz's many-argument types $\alpha/\beta_1 \ldots \beta_n$. It is noteworthy that Ajdukiewicz's approach in [2] needs some revision at this point. According to Ajdukiewicz, the reduction procedure can be performed in a fully deterministic way: at each step one finds the left-most occurrence of a sequence of types matching the left-hand side of (RED) and replaces this sequence with the type of the right-hand side of (RED). This deterministic algorithm always returns a unique fa-structure and a unique type of the entry or replies negatively, if it comes to an irreducible sequence. Unfortunately this procedure is incorrect for many-argument types; for one-argument types it works well. In [17], the following counterexample has been found:

$$s/(n/n)nn,\ n/n,\ n, n/n,\ n\,.$$

This sequence reduces to s, if one reduces the last two types to n, then the whole to s, but this reduction does not fulfil Ajdukiewicz's requirement. Following this requirement, one should first reduce the second and the third type (together) to n, then the last pair to n, and obtain the irreducible sequence $s/(n/n)nn,\ n,\ n$. One can consider a formal language, where $I : s/(n/n)nn$, $f : n/n$, $g : n/n$, $a : n$, $b : n$. Let the meaning of $Ifab$ be $f(a) = b$. The expression $Ifagb$ is well-formed (it means $f(a) = g(b)$), but Ajdukiewicz's procedure rejects it.

This inadequacy can be removed in two ways (see [17]): (i) the reduction procedure is executed in a non-deterministic way: at any stage one chooses a reducible pattern of types and rewrites it according to (RED) (this routine is standard for AB-grammars, but it makes the unambiguity properties nontrivial), (ii) the reduction procedure remains deterministic but admits partial reductions:

$$\alpha/\beta_1 \ldots \beta_n, \beta_1, \ldots, \beta_i \Rightarrow |\alpha/\beta_{i+1} \ldots \beta_n\,.$$

Types with $|$ can act as functors but not as arguments of other functors. Worth noticing, in [2] such types are used for languages with variable-binding operators. For the example mentioned above, one reduces the first three types, which yields $|s/n,\ n/n,\ n$, then the last pair to n, and finally $|s/n,\ n$ to s.

Bidirectional one-valued AB-grammars can be both structurally and cate-

gorially ambiguous. The sequence of types:

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

reduces to p with structure $(((p/(q\backslash q))/q, q)_1, q\backslash q)_1$ and to $p/(q\backslash q)$ with structure $(((p/(q\backslash q))/q), (q, q\backslash q)_2)_1$. The categorial and structural unambiguity, however, can be retained, if strings are replaced by phrase structures: for any phrase structure X there exists at most one pair (Y, α) such that Y is an fa-structure, α is a type, X is the ps-yield of Y, and the grammar assigns α to Y.

Therefore, for unidirectional one-valued AB-grammars one can represent fa-structures by their yields, and for bidirectional one-valued AB-grammars by their ps-yields. For any well-constructed unidirectional one-valued grammar G, the syntactically connected strings are precisely the yields of structures from $s(L^F(G))$, and the equivalence classes of $\equiv_{L(G)}$, restricted to syntactically connected strings, coincide with the typed categories $L(G, \alpha)$, for $\alpha \in s(\mathcal{T}(G))$. The same is true for well-constructed bidirectional one-valued AB-grammars if one replaces strings by phrase structures and $\equiv_{L(G)}$ by $\equiv_{L^P(G)}$.

Now we briefly discuss the problem of determining basic and functor categories of the given language, especially a natural language. In practice, many different aspects play a role, e.g. analogies with logical formalisms, tradition, semantics, particular features of the language under consideration (a discussion of various factors can be found in Marciszewski [36]). Semantics justifies the qualification of sentences and names (proper nouns) as two basic categories. Intransitive (resp. transitive) verbs are treated like unary (resp. binary) predicates in first-order logic. Complete noun phrases are treated like generalized quantifiers; they act as functors on unary predicates as arguments. These options, however, are not obligatory, and there are good reasons for alternative solutions. Keenan and Faltz [30] admit the basic category of complete noun phrases, whose subcategory consists of proper nouns; intransitive verbs act as functors on the complete noun phrases. The reasons are semantical: in [30] the basic ontological categories, corresponding to the basic syntactic categories, are boolean algebras with some natural (set-theoretic) interpretations of 'and', 'or', 'not'. The ontological category of individuals is not boolean, but that of generalized quantifiers (i.e. families of sets of individuals) is boolean, and proper nouns are interpreted as particular families of sets of individuals, namely the principal ultrafilters in the boolean algebra of all sets of individuals.

Some algorithms which extract a grammar from a finite set of sentences, represented as fa-structures, were proposed in [13] for one-valued grammars and [21] for arbitrary AB-grammars. These algorithms employ unification of types, an adaptation of the Curry algorithm for determining the principal type of a combinator. Kanazawa [27] further elaborated these methods toward a Gold-style paradigm of learning as identification in the limit (also for sentences represented as strings). In the last two decades this issue dominated the mathematical research in AB-grammars; see the textbook [39].

Finall, we will discuss semantic types, following Ajdukiewicz [3]. This paper outlines a semantical version of the approach from [2]. Semantic types correspond to ontological categories of denotations of expressions. Ajdukiewicz uses

w as the type of truth values and i as the type of individuals. $\frac{\beta}{\alpha}$ is the type of functions which send arguments of type α to values of type β. So $\frac{w}{w}$ corresponds to unary truth-value functions, $\frac{w}{ww}$ to binary truth-value functions, $\frac{w}{i}$ to unary predicates, $\frac{w}{ii}$ to binary predicates, and so on. The semantic category of type α can be defined as the set of expressions whose denotations are of type α. Of course, this definition assumes that the language is interpreted in a fixed model which determines the denotations of meaningful expressions. The denotation of a compound expression can be computed from the denotations of words by function application.

Today the ideas of [3] are standard in type-theoretic semantics. This paper, however, was published in 1960, several years before the first semantical work of R. Montague. Nonetheless Ajdukiewicz could be influenced by some earlier proposals of a similar character, e.g. [9, 45], and higher-order logics, well elaborated in this time.

Semantic types do not uniquely indicate the syntactic roles of expressions. For instance, Latin words 'Johannes' and 'Petrum' are assigned i, but 'Johannes' can only be the subject and 'Petrum' the direct object of a simple declarative sentence (so both 'Johannes amat Petrum' and 'Petrum amat Johannes' express the same statement, whose functor is 'amat', the first argument is 'Johannes', and the second argument is 'Petrum'). Ajdukiewicz supplies expressions with positional markers (indices) which indicate the positions of these expressions in a sentence, represented as a fa-structure with functors always preceding their arguments (so functor markers can be omitted). For 'Mary sings and Alice dances' (my example), represented as (and (sings Mary) (dances Alice)), the whole sentence is supplied with 1, the main functor 'and' with (1,0), its first argument (sings Mary) with (1,1), its second argument (dances Alice) with (1,2), 'sings' with (1,1,0), 'Mary' with (1,1,1), 'dances' with (1,2,0), 'Alice' with (1,2,1). Ajdukiewicz argues that these positional markers are similar to inflections, and the language in which all words are supplied with such markers is a "purely inflectional language".

In this way, Ajdukiewicz attributes to types an exclusively semantical role, while functor-argument relations are regulated by positional markers. Although the idea of a purely inflectional language seems attractive, this concrete realization is not satisfactory. Positional markers of words, proposed here, bring just another encoding of a single fa-structure; they lack sense, if not related to a particular structure (up to isomorphism), while inflections in inflectional languages are not restricted to any particular form of sentence. The markers, given above, are useless for 'dear Alice sings softly'.

AB-grammars and other type grammars, considered later on, are suitable for positional languages, as \ and / in directional types $\alpha\backslash\beta$, β/α encode the information on the positions of the functor and the argument and nothing else. Inflectional languages need more information encoded in types or some additional syntactic constraints.

To attain a better concordance of syntactic and semantic categories, [14] proposes a semantics sensitive to syntax. Syntactic types with \, / can be treated as semantic types. The ontological category of type $\alpha\backslash\beta$ (resp. β/α) consists of

pairs (r, f) (resp. (l, f)) such that f is a function from the ontological category of type α to that of type β, and r, l are position markers. If (r, f) (resp. (l, f)) is a denotation of X, then X can be interpreted as f, provided that X acts as the right (resp left) functor in the structure under consideration. In Section 3 we show that also stronger type logics can be related to models of this kind. Again, this approach works well for (fragments of) positional languages, but more syntactic information must be encoded in semantic types for inflectional languages. This way is quite opposite to that of [3].

3 Syntactic categories in Lambek grammars

AB-grammars implicitly assume the following rule: if $u : \alpha\backslash\beta$ (resp. $u : \beta/\alpha$), then for any string v such that $v : \alpha$, there holds $vu : \beta$ (resp. $uv : \beta$). Lambek [32] replaces 'if-then' by 'if and only if'. This leads to new reduction laws. Since $u : \alpha$ entails $uv : \beta$, for any v of type $\alpha\backslash\beta$, then u is of type $\beta/(\alpha\backslash\beta)$; this justifies the type-raising law $\alpha \Rightarrow \beta/(\alpha\backslash\beta)$ and its dual form $\alpha \Rightarrow (\beta/\alpha)\backslash\beta$ has a symmetric justification. In particular, $n \Rightarrow s/(n\backslash s)$ can be read: every proper noun is a noun phrase. This type-raising was implicitly applied by Montague [37] who lifted up proper nouns to the type of noun phrase in order to interpret the conjunction 'and' as a noun phrase conjunction (in models, as the intersection of families of sets of individuals) in contexts like 'John and a student'. Also, if $u : \alpha\backslash\beta$ and $v : \beta\backslash\gamma$, then $wuv : \gamma$, for any $w : \alpha$, and consequently $uv : \alpha\backslash\gamma$. This justifies the composition law $\alpha\backslash\beta, \beta\backslash\gamma \Rightarrow \alpha\backslash\gamma$, and a similar argument supports $\alpha/\beta, \beta/\gamma \Rightarrow \alpha/\gamma$. In particular, $s/s, s/(n\backslash s) \Rightarrow s/(n\backslash s)$ enables the classification of 'not every student' as a (negative) noun phrase.

Lambek's approach changes the *static* typing of expressions in AB-grammars into a *dynamic* one; each type can be transformed into infinitely many new types. On the level of syntax, these new types correspond to different syntactic roles of one expression; in semantics, they yield the types of possible denotations of the expression.

Two basic versions of the Lambek calculus are Associative Lambek Calculus (L), due to Lambek [32], and Nonassociative Lambek Calculus (NL), due to Lambek [33]. In both systems formulae are built from variables (atomic types) by means of three connectives $\cdot, \backslash, /$, called product, right residuation (right implication, right division) and left residuation (left implication, left division), respectively. These connectives are interpreted in algebras of languages $\mathcal{P}(\Sigma^+)$ as follows:

$$L_1 \cdot L_2 = \{uv : u \in L_1,\, v \in L_2\}\,,$$

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

for $L_1, L_2 \subseteq \Sigma^+$. Σ^+ can be replaced by Σ^*, if one regards languages with ϵ, and by Σ^P, if one deals with ps-languages (one replaces u by X, v by Y, and uv by (XY)). These algebras are called *language models*.

General algebraic models are *residuated groupoids*. A residuated groupoid is an ordered algebra $\mathcal{A} = (A, \cdot, \backslash, /, \leq)$ such that $(A, \leq)$ is a partially ordered set, and $\cdot, \backslash, /$ are binary operations on A, satisfying the residuation laws:

$$x \cdot y \leq z \text{ iff } y \leq x\backslash z \text{ iff } x \leq z/y\,,$$

for all $x, y, z \in A$. *A residuated semigroup* is a residuated groupoid such that $\cdot$ is associative, i.e. $(A, \cdot)$ is a semigroup. For any residuated groupoid, $\cdot$ is monotone in both arguments, hence $(A, \cdot, \leq)$ is a partially ordered groupoid. $\mathcal{P}(\Sigma^P)$ is a residuated groupoid ($\subseteq$ is the order), and $\mathcal{P}(\Sigma^+)$ is a residuated semigroup.

NL and L can be presented as (intuitionistic) sequent systems. An ND-system for the product-free L (i.e. L restricted to formulae without $\cdot$) adds to the ND-system for AB the introduction rules:

$$(\mathrm{I}\backslash)\ \frac{\alpha, \Gamma \Rightarrow \beta}{\Gamma \Rightarrow \alpha\backslash\beta}\,,\ \ (\mathrm{I}/)\ \frac{\Gamma, \alpha \Rightarrow \beta}{\Gamma \Rightarrow \beta/\alpha}\,,$$

with the constraint $\Gamma \neq \epsilon$. Dropping this constraint yields the product-free L^*; this system admits sequents with empty antecedents.

For NL, antecedents of sequents are formula structures (in the ps-format). The ND-system for AB is extended by $(\mathrm{I}\backslash)$ with the premise $(\alpha, \Gamma) \Rightarrow \beta$ and $(\mathrm{I}/)$ with the premise $(\Gamma, \alpha) \Rightarrow \beta$. Admitting the empty structure Λ (the unit of the operation $(\ ,\ )$), yields the product-free NL^*.One writes $\Rightarrow \alpha$ for $\Lambda \Rightarrow \alpha$.

In L one adds rules for product:

$$(\mathrm{E}\,\cdot)\ \frac{\Gamma, \alpha, \beta, \Gamma' \Rightarrow \gamma\,;\ \Delta \Rightarrow \alpha \cdot \beta}{\Gamma, \Delta, \Gamma' \Rightarrow \gamma}\,,\ \ (\mathrm{I}\,\cdot)\ \frac{\Gamma \Rightarrow \alpha\,;\ \Delta \Rightarrow \beta}{\Gamma, \Delta \Rightarrow \alpha \cdot \beta}\,.$$

In NL, the first premise of $(\mathrm{E}\,\cdot)$ is $\Gamma[(\alpha, \beta)] \Rightarrow \gamma$ and the conclusion is $\Gamma[\Delta] \Rightarrow \gamma$; the conclusion of $(\mathrm{I}\,\cdot)$ is $(\Gamma, \Delta) \Rightarrow \alpha \cdot \beta$.

In algebraic models, sequents are interpreted as follows. The nonempty antecedent Γ is interpreted as the formula $f(\Gamma)$, obtained by replacing each comma by $\cdot$, and $\Rightarrow$ is interpreted as $\leq$. The empty antecedent is interpreted as 1, i.e. the unit element, satisfying $1 \cdot x = x = x \cdot 1$, for any element x. The language $\{\epsilon\}$ is the unit in language models $\mathcal{P}(\Sigma^*)$. Σ^{P*} denotes $\Sigma^P \cup \{\Lambda\}$ (the empty structure is the unit in Σ^{P*}). Then, $\{\Lambda\}$ is the unit in $\mathcal{P}(\Sigma^{P*})$. A sequent $\Gamma \Rightarrow \alpha$ is *true* in $\mathcal{A}$ for a valuation μ (i.e. a homomorphism from the free algebra of formulae to $\mathcal{A}$), if $\mu(f(\Gamma)) \leq \mu(\alpha)$; it is *valid* in a class of algebras, if it is true in every algebra from this class for all valuations.

L is complete with respect to residuated semigroups, this means: the sequents provable in L are precisely the sequents valid in this class. L^* is complete with respect to residuated monoids (i.e. residuated semigroups with 1), NL with respect to residuated groupoids, and NL^* with respect to residuated unital groupoids (i.e. residuated groupoids with 1). In each system, $\Gamma \Rightarrow \alpha$ is provable if and only if $f(\Gamma) \Rightarrow \alpha$ is provable.

These logics are often presented as sequent systems in which the elimination rules are replaced by the left introduction rules (they introduce a connective in

the antecedent). In NL the left introduction rule for product is: from $\Gamma[(\alpha,\beta)] \Rightarrow \gamma$ infer $\Gamma[\alpha\cdot\beta] \Rightarrow \gamma$ (see [16] for the full list). The cut-rule:

$$\text{(CUT) in NL } \frac{\Gamma[\alpha] \Rightarrow \beta\,;\ \Delta \Rightarrow \alpha}{\Gamma[\Delta] \Rightarrow \beta} \quad \text{(CUT) in L } \frac{\Gamma, \alpha, \Gamma' \Rightarrow \beta\,;\ \Delta \Rightarrow \alpha}{\Gamma, \Delta, \Gamma' \Rightarrow \beta}$$

plays a special role. It is a structural rule (introducing no new formula). It is necessary in theories (logics augmented with assumptions, i.e. nonlogical axioms) for their completeness, but not in pure logics: every provable sequent can be proved without (CUT). This *cut-elimination theorem* was proved by Lambek [32] for L and [33] for NL. It also holds for other type logics considered here. The most important consequences are the decidability of these logics and *the subformula property*: every provable sequent $\Gamma \Rightarrow \alpha$ has a (cut-free) proof in which all sequents consist of subformulae of the formulae occurring in $\Gamma \Rightarrow \alpha$.

Other standard structural rules are exchange (e), integrality (or: left weakening) (i), contraction (c), and associativity (a):

$$\text{(e) } \frac{\Gamma[(\Delta_1, \Delta_2)] \Rightarrow \alpha}{\Gamma[(\Delta_2, \Delta_1)] \Rightarrow \alpha}, \quad \text{(i) } \frac{\Gamma[\Delta_i] \Rightarrow \alpha}{\Gamma[(\Delta_1, \Delta_2)] \Rightarrow \alpha},$$

$$\text{(c) } \frac{\Gamma[(\Delta, \Delta)] \Rightarrow \alpha}{\Gamma[\Delta] \Rightarrow \alpha} \quad \text{(a) } \frac{\Gamma[((\Delta_1, \Delta_2), \Delta_3)] \Rightarrow \alpha}{\overline{\Gamma[(\Delta_1, (\Delta_2, \Delta_3))] \Rightarrow \alpha}}.$$

NL with (a) is equivalent to L. Logics with (e) are interpreted in commutative algebras ($x \cdot y = y \cdot x$ for all elements x, y); then $x\backslash y = y/x$, and one writes $x \to y$ for both. So the connectives of L with (e) are $\cdot, \to$, and similarly for NL with (e).

Substructural logics are often defined as axiomatic and rule extensions of Full Lambek Calculus (FL): L* enriched with constants 1, 0 and lattice connectives $\wedge, \vee$ (optionally also constants $\bot, \top$, interpreted as the least element and the greatest element). 1 is interpreted as the unit element for product and 0 as an arbitrary element (one defines two negations $\sim x = x\backslash 0$, $-x = 0/x$; in commutative algebras they collapse to one negation $\neg$). The algebraic models of FL are residuated lattices, i.e. lattice-ordered residuated monoids; see [23]. The nonassociative version FNL corresponds to lattice-ordered residuated unital groupoids; see [23, 19]. According to the terminology of linear logics, $\cdot, \backslash, /, 1, 0$ are called *multiplicative* connectives and constants, while $\wedge, \vee, \bot, \top$ are called *additive* connectives and constants. FL is a conservative extension of L* and FNL of NL*. L* is the multiplicative fragment of FL (without 1, 0).

Many important nonclassical logics belong to this family. For instance, intuitionistic logic (IL) amounts to FL with (e), (i), (c) and the definition $0 = \bot$, multiplicative-additive linear logic (MALL) to FL with (e) and the axiom $\neg\neg\alpha \Rightarrow \alpha$, and Łukasiewicz logic Ł$_\infty$ to FL with (e), (i), the definition $0 = \bot$ and the axiom $(\alpha \to \beta) \to \beta \Rightarrow \alpha \vee \beta$. Noncommutative and nonassociative versions of these logics are also studied.

Type grammars usually apply multiplicative substructural logics, which can be interpreted in language models: NL, NL*, L, L*. Logics of semantic types

are formalized with (e) and, possibly, other structural rules. Logics with $\wedge, \vee$ are less popular, although there are good reasons for them; see below. One also considers logics with new residuated operations, e.g. unary modalities $\Diamond_i$ and their residuals $\Box_i^{\downarrow}$. In algebras, $\Diamond$ and $\Box^{\downarrow}$ are connected by the unary residuation law: $\Diamond x \leq y$ iff $x \leq \Box^{\downarrow} y$. (In general, $\Box^{\downarrow}$ is different from $\Box$, i.e. the De Morgan dual of $\Diamond$.) L and NL with unary modalities were applied by e.g. Moortgat [38] and Morrill [40] to allow a controlled usage of some structural rules (in a similar role that exponentials are used in linear logics).

Systems not allowing empty antecedents of sequents, like NL, L, are not popular among logicians; no single formula α (i.e. no sequent $\Rightarrow \alpha$) is provable, hence these logics cannot be presented as Hilbert-style systems, nor easily compared with other nonclassical logics. In type grammars, however, they are extensively used, starting from Lambek [32] (also AB is a logic of this kind). Logics with empty antecedents are too strong, in a sense. Adjectives can be typed N/N, where N is the type of common noun (so adjectives are treated as common noun modifiers) and adverbs $(N/N)/(N/N)$ (adjective modifiers). In semantics, adjectives act as functions which send a set of individuals to its subset and adverbs act as higher-order functions which modify adjectival functions. In L* $\alpha/\alpha \Leftrightarrow (\alpha/\alpha)/(\alpha/\alpha)$ is provable (this means: sequents in both directions are provable), hence adjectives are indistinguishable from adverbs. This is unacceptable for linguistics ('a beautiful student' is correct, but 'a very student' is not). One can solve this problem by introducing new types, e.g. a new atomic type for adjectives, but it complicates the grammar and is less satisfactory from the semantic viewpoint.

On the other hand, both kinds of logics (i.e. with and without empty antecedents) are closely related. A faithful interpretation of FL in its version without empty antecedents has been shown in [20]; this interpretation also works for several stronger logics, allowing cut elimination.

On the basis of L and its variants, one can analyze compound expressions more easily and deeply than in AB-grammars. With 'John': n, 'likes': $(n\backslash s)/n$, 'some': $((s/n)\backslash s)/N$, 'teacher': N, we have 'John likes some teacher': s, since the sequent:

$$n, (n\backslash s)/n, ((s/n)\backslash s)/N, N \Rightarrow s$$

is provable in L. This sequent is not provable in AB; an AB-grammar must also assign $n\backslash(s/n)$ to 'likes'. (The associative law $\alpha\backslash(\beta/\gamma) \Leftrightarrow (\alpha\backslash\beta)/\gamma$ is provable in L.) Another example is 'he likes her' with 'he': $s/(n\backslash s)$ and 'her': $(s/n)\backslash s$. The sequent:

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

is provable in L but not in AB, and an AB-grammar needs additional types, e.g. 'he': $(s/n)/((n\backslash s)/n)$. (The Geach law $\alpha/\beta \Rightarrow (\alpha/\gamma)/(\beta/\gamma)$ is provable in L, and similarly for its dual with $\backslash$.) In general, AB-grammars must assign many types to one word to account for different syntactic roles of this word, while L-grammars can derive the other types from some main types and explain logical relations between types.

Associative logics are essentially stronger than their nonassociative versions. Type-raising laws and expansion laws are provable in NL, but associative laws, composition laws and Geach laws are not. It may be argued that associative logics are too strong for type grammars. Lambek's example uses 'Mary': n, 'poor': n/n (as 'poor John': n), 'him': $(s/n)\backslash s$, and 'likes', 'John' typed as above. On the basis of L 'Mary likes poor John': s, but also 'Mary likes poor him': s, though the latter is incorrect in English. The sequent:

$$n, (n\backslash s)/n, n/n, (s/n)\backslash s \Rightarrow s \tag{4}$$

is provable in L but not in NL (this means: no sequent $\Gamma \Rightarrow s$ such that the antecedent of (4) is the yield of Γ is provable in NL). This was a reason for replacing L by NL in [33]. Moortgat [38] treats NL as a basic type logic and increases its power by adding modalities $\Diamond, \Box^{\downarrow}$.

Now we briefly discuss some relations of Lambek grammars to AB-grammars.

Every L-grammar G can be transformed into an infinite AB-grammar $\overline{G}$: $I_{\overline{G}}(a)$ contains all types β such that $\alpha \Rightarrow \beta$ is provable in L, for some $\alpha \in I_G(a)$. One shows $L(G, \alpha) = L(\overline{G}, \alpha)$, for any type α, and consequently, $L(G) = L(\overline{G})$. An analogous transformation works for NL-grammars.

Since the antecedents of sequents in NL are formula structures of the form of phrase structures, then NL-grammars naturally assign types to phrase structures. Similarly as in Section 2, I_G is extended for all structures from Σ_G^P by setting: $I_G((XY)) = \{(\Gamma, \Delta) : \Gamma \in I_G(X), \Delta \in I_G(Y)\}$. For NL-grammars and L-grammars, the types of fa-structures can be defined as those assigned by the infinite AB-grammar $\overline{G}$; by dropping structure markers we get the types of ps-structures and strings. For NL-grammars, the second typing of phrase structures is equivalent to the first one. In NL-grammars and L-grammars, different fa-structures with the same ps-yield are indistinguishable (this means: they are assigned the same types) and in L-grammars this also holds for different phrase structures with the same yield. The possibility of interchanging functors and arguments is a result of type-raising laws.

Consequently, for an NL-grammar G, $L^F(G)$ consists of all fa-structures whose ps-yield belongs to $L^P(G)$; for an L-grammar G, both $L^F(G)$ and $L^P(G)$ consist of all structures (of the appropriate kind) whose yield belongs to $L(G)$. So NL-grammars are grammars of phrase structures, and L-grammars are grammars of strings.

Kandulski [28, 29] proves the P-equivalence of type grammars based on NL and AB; hence NL-grammars are equivalent to $\epsilon-$free CFGs. This proof shows that for any NL-grammar G, one can construct an AB-grammar G' such that $L^P(G) = L^P(G')$ and $I_G(a) \subseteq I_{G'}(a) \subseteq I_{\overline{G}}(a)$, for any $a \in \Sigma_G$. So G' is a finite fragment of the infinite AB-grammar $\overline{G}$. Analogous results for product-free NL-grammars were earlier proved in [12].

The equivalence of L-grammars and $\epsilon-$free CFGs was proved by Pentus [43]. Hence L-grammars are equivalent to AB-grammars. The P-equivalence does not hold: L-grammars allow all possible phrase structures of the generated strings, hence they can generate ps-languages of infinite degree.

[15] shows that every product-free L-grammar G is equivalent to some AB-grammar G', extending G and being a finite fragment of $\overline{G}$, as above. G' can be treated as a natural AB-grammar equivalent to G.

We turn to the main topic of this paper: syntactic categories.

By the equivalence results discussed above, the theory of AB-grammars, presented in Section 2, can be applied, essentially, to NL-grammars and L-grammars if we replace the latter with (natural) AB-grammars equivalent to them. In this sense Ajdukiewicz's approach to syntactic categories preserves its merits for grammars based on Lambek logics. As a rule, a direct application of this approach to Lambek grammars is impossible or, at least, problematic.

The strict concordance of typed categories and substitution classes is impossible even for one-valued Lambek grammars. In opposition to AB, Lambek logics provide laws of the form $\alpha \Rightarrow \beta$ such that $\alpha \neq \beta$, e.g. type-raising laws (in NL and L) and Geach laws (in L). If $\alpha \Rightarrow \beta$ is provable, then $L(G, \alpha) \subseteq L(G, \beta)$; if, additionally, $\beta \Rightarrow \alpha$ is not provable and $\beta \in I_G(a)$, then $a \in L(G, \beta)$, $a \notin L(G, \alpha)$, hence $L(G, \alpha) \subset L(G, \beta)$. The same holds for typed categories consisting of structures. Accordingly, typed categories do not partition the universe of (well-formed) expressions, hence they cannot coincide with substitution classes.

In language models of L, all typed categories are generated from basic categories by the operations $\cdot, \backslash, /$. Let P be a set of atomic types. The basic categories are the values of a map $c : P \mapsto \mathcal{P}(\Sigma^+)$. The map c is uniquely extended to all types on P, by the homomorphism equations:

$$c(\alpha \cdot \beta) = c(\alpha) \cdot c(\beta),\ c(\alpha \backslash \beta) = c(\alpha) \backslash c(\beta),\ c(\alpha / \beta) = c(\alpha) / c(\beta). \tag{5}$$

For an L-grammar G, by P_G we denote the set of atomic types involved in G. For any $p \in P_G$, we define $c_G(p) = L(G, p)$. The map c_G can be extended as above. We are faced with the natural problem of whether typed categories in the grammar are compatible with typed categories in the model. [10] shows that the strong condition: $c_G(\alpha) = L(G, \alpha)$, for all product-free types α on P_G, cannot be attained for L-grammars, nor AB-grammars.

A type grammar G is said to be *weakly complete*, if this condition holds for all $\alpha \in s(\mathcal{T}(G))$, and *correct*, if $a \in c_G(\alpha)$ whenever $\alpha \in I_G(a)$, for all types α and words a.

Every weakly complete grammar is correct. Not all grammars are correct. For instance, the AB-grammar G with $a : s/(s/s)$, $b : s/s$ is incorrect; $bb \in c_G(s/s)$, but $bb \notin L(G, s/s)$, hence $abb \notin c_G(s)$, and consequently, $a \notin c_G(s/(s/s))$. The analogous grammar based on L is weakly complete, hence correct. Nonetheless there exist incorrect L-grammars. By extending Σ_G, every product-free L-grammar G can be extended to some weakly complete L-grammar which is equivalent to G for strings on Σ_G. If an AB-grammar is correct, then it is equivalent to the L-grammar having the same lexicon, the same initial type assignment and the same designated type. If G is correct, then c_G is the least map c satisfying: $a \in c(\alpha)$ whenever $\alpha \in I_G(a)$ (with respect to the partial ordering: $c \leq c'$ if and only if $c(p) \subseteq c'(p)$ for all $p \in P_G$).

This resembles the characterization of context-free languages as the minimal solutions of finite systems of linear equations in languages. Analogous results can be obtained for NL-grammars.

Notice that the incorrect AB-grammar, presented above, is one-valued and well-constructed. So even such AB-grammars, though fully compatible with Ajdukiewicz's postulates for syntactic categories, need not be compatible with Lambek's approach. This is not surprising, since the two approaches essentially differ in the interpretation of functor types. For AB-grammars, language models should be replaced by more general structures; instead of (5) one only assumes $c(\alpha\backslash\beta) \subseteq c(\alpha)\backslash c(\beta)$, and similarly for β/α.

The mathematical research in type logics focused on the completeness of type logics with respect to language models, the equivalence of categorial grammars and production grammars (especially CFGs), the computational complexity of type logics and categorial grammars, and others. Here we do not discuss these matters in detail; the reader is referred to [16, 18, 38, 39].

Pentus [44] proves the completeness of L with respect to language models $\mathcal{P}(\Sigma^+)$; this also holds for L* and the corresponding language models $\mathcal{P}(\Sigma^*)$. The strong completeness does not hold, but it holds for the product-free fragments of these logics, even with $\wedge, \top$. In language models we interpret $\wedge, \vee$ as the set-theoretic intersection ($\cap$) and union ($\cup$) of languages, and $\top$ as the total language.

Types with $\wedge, \vee$ are not often used in type grammars, but there are good reasons to work with them. Lambek [33] applied $\wedge$ to replace a multi-valued type assignment $a : \alpha_1, \ldots, \alpha_n$ with the one-valued assignment $a : \alpha_1 \wedge \cdots \wedge \alpha_n$. Kanazawa [26] considered types sensitive to features, e.g. np$\wedge$sing (singular noun phrase), np$\wedge$pl (plural noun phrase); he also proved that FL-grammars generate some languages which are not context-free. The two types of noun phrase, $s/(n\backslash s)$ (subject) and $(s/n)\backslash s$ (object), yield the type $((s/(n\backslash s)) \wedge ((s/n)\backslash s))/N$ of determiners. Lambek [34] links the subtypes with the main type by nonlogical axioms (assumptions), e.g. $\pi_k \Rightarrow \pi$, $s_k \Rightarrow s$; see Section 1. A similar effect can be reached by defining $\pi = \pi_1 \vee \pi_2 \vee \pi_3$, $s = s_1 \vee s_2$; then, the type change formalism remains a pure logic (it is a theory in [34], a violation of lexicality).

FL is not complete with respect to language models, e.g. the distributive laws for $\wedge, \vee$ are valid in language models but not provable in FL. One can add to FL the axiom:

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

The resulting logic is called Distributive Full Lambek Calculus (DFL), and DFNL is an analogous extension of FNL. The remaining distributive laws are provable. The present axiomatization does not allow cut elimination, but cut-free systems for these logics exist; see [31].

An interesting linguistic interpretation of FL and related logics, not assuming the distributive laws for $\wedge, \vee$, uses Syntactic Concept Lattices (SCLs), applied by Clark [22] in some learning procedures for formal grammars. By a context

on Σ one means a pair $(u, w) \in (\Sigma^*)^2$. Let $L \subseteq \Sigma^*$ be a fixed language. For any $U \subseteq \Sigma^*$, one defines $U^{\triangleright}$ as the set of all contexts (u, w) such that $uvw \in L$, for all $v \in U$. For any $S \subseteq (\Sigma^*)^2$, one defines $S^{\triangleleft}$ as the set of all $v \in \Sigma^*$ such that $uvw \in L$, for all $(u, w) \in S$. The operations ${}^{\triangleright}, {}^{\triangleleft}$ form a Galois connection ($U \subseteq S^{\triangleleft}$ if and only if $S \subseteq U^{\triangleright}$), and consequently, the operation $C(U) = U^{\triangleright\triangleleft}$ is a closure operation on $\mathcal{P}(\Sigma^*)$. Furthermore, C is a nucleus (it satisfies $C(U) \cdot C(V) \subseteq C(U \cdot V)$). The closed sets (i.e. satisfying $C(U) = U$) are called the syntactic concepts determined by L. Let C_L denote the family of syntactic concepts determined by L. One shows that C_L is closed under $\backslash, /$ (defined in $\mathcal{P}(\Sigma^*)$) and $\cap$. One also defines $U \cdot_C V = C(U \cdot V)$, $U \cup_C V = C(U \cup V)$, $1_C = C(\{\epsilon\})$. C_L with these operations is a residuated lattice (not necessarily distributive), called the SCL determined by L. FL is strongly complete with respect to SCLs [49]. Analogous results can be obtained for FNL and logics not allowing empty antecedents if one modifies SCLs appropriately.

Syntactic concepts in the sense of [22] can be interpreted as syntactic categories determined by the language L. This is a generalization of Ajdukiewicz's idea of a syntactic category as a substitution class. Although syntactic concepts are not equivalence classes of the relation of mutual substitutability, they are determined by sets of contexts. For instance, in the nonassociative format, the category of singular noun phrases is determined by the single context (_ exists), since (X exists) is a correct sentence if and only if X is a singular noun phrase. This remarkable generalization of Ajdukiewicz's approach has not yet been developed in the theory of type grammars. A similar idea of syntactic categories was elaborated in the *contextual grammars* of S. Marcus.

Finally, we will briefly discuss logics of semantic types. Following van Benthem [7], we admit two atomic types: e (entity) and t (truth value); they correspond to i and w of Ajdukiewicz [3]. Other atomic types can also be considered, e.g. the type of quantifiers (interpreting noun phrases) was taken as atomic in [30]. Functional types are of the form $\alpha \to \beta$; one also writes $(\alpha\beta)$, for brevity.

Some simple semantic types are: (tt) (unary truth-value function), $(t(tt))$ (binary truth-value function), (et) (unary predicate, i.e. set of individuals), $(e(et))$ (binary predicate), $((et)t)$ (quantifier, i.e. family of sets of individuals), $((et)(et))$ (operation on unary predicates, e.g. complement), $((et)((et)(et)))$ (binary operation on unary predicates, e.g. union, intersection).

No serious presentation of type-theoretic semantics applied to language is possible in this short essay; the reader may consult any book on Montague Grammar and similar approaches, e.g. [37, 30, 7, 42, 39].

The product-free L with (e) was proposed by van Benthem [7] as a logic of semantic types (sometimes referred to as the Lambek-van Benthem calculus). In this logic one proves semantic counterparts of the laws provable in L. The type-raising law takes the form $\alpha \Rightarrow ((\alpha\beta)\beta)$. Its instance $e \Rightarrow ((et)t)$ changes the type of individuals to the type of quantifiers, a move anticipated by R. Montague; see the beginning of this section. The Geach law $(\alpha\beta) \Rightarrow ((\gamma\alpha)(\gamma\beta))$ yields $(tt) \Rightarrow ((et)(et))$; this shifts the initial type (tt) of 'not' to the type of boolean complement on sets of individuals, as in (John (is (not happy))). Also $(tt) \Rightarrow (((et)t)((et)t))$ shifts (tt) to the type of boolean complement on families

of sets of individuals, as in ((not (every student)) came).

The Curry-Howard isomorphism is a correspondence between proofs in ND-systems and typed lambda-terms. The ND-system for the product-free L with (e) admits the axioms (Id) and the rules:

$$(\mathrm{E}\!\to)\ \frac{\Gamma \Rightarrow \alpha \to \beta\,;\ \Delta \Rightarrow \alpha}{\Gamma, \Delta \Rightarrow \beta}\ ,\ \ (\mathrm{I}\!\to)\ \frac{\Gamma, \alpha \Rightarrow \beta}{\Gamma \Rightarrow \alpha \to \beta}\,.$$

Here the antecedents of sequents are (nonempty) finite multisets of formulae; the comma stands for the union of multisets. Proofs can be encoded by typed lambda-terms. The axiom $\alpha \Rightarrow \alpha$ is encoded by $x : \alpha$. The elimination rule (E→) corresponds to application: if $M : \alpha \to \beta$ and $N : \alpha$ then $MN : \beta$, and the introduction rule (I→) to abstraction: if $M : \beta$ and $x : \alpha$ then $\lambda x.M : \alpha \to \beta$.

Let us consider the proof of $e \Rightarrow ((et)t)$ and its encoding. From $(et) \Rightarrow (et)$ $(x : (et))$ and $e \Rightarrow e$ $(y : e)$, we get $(et), e \Rightarrow t$ $(xy : t)$, by (E→). Then (I→) yields $e \Rightarrow ((et)t)$ $(\lambda x.xy : ((et)t))$.

One interprets this formalism in a fixed model, i.e. a hierarchy of semantic domains (ontological categories) D_α, for semantic types α. If one valuates the free variable y with an individual $d \in D_e$, then $\lambda x.xy$ denotes the family of all $U \subseteq D_e$ such that $d \in U$ (sets, families of sets, etc. are identified with their characteristic functions). Thus, the proof of $e \Rightarrow ((et)t)$ is encoded by a lambda-term which determines a semantic transformation: the initial denotation d (an individual) is sent to the higher-order denotation $\lambda x^{(et)}.xy$ (a quantifier, here: the principal ultrafilter determined by d). This is a general paradigm: by the Curry-Howard isomorphism, proofs in this system determine semantic transformations, which modify the initial denotations of expressions.

A proof of $(tt) \Rightarrow ((et)(et))$ is encoded by $\lambda x^{(et)}y^e.z^{(tt)}(xy) : ((et)(et))$. If z is valuated by the truth-value function of negation, then this term denotes the boolean complement on sets of individuals, i.e. the denotation of 'not' in e.g. (John (is (not happy))).

On the other hand, $(t(tt)) \Rightarrow ((et)((et)(et)))$, needed to shift the initial type of 'and', 'or' to the type of predicate conjunction, as in (Mary (sings and dances)), cannot be proved in L with (e). The contraction rule (c) is needed to infer this law from $(t(tt)), (et), e, (et), e \Rightarrow t$ (provable in L with (e)). Thus, it is reasonable to admit more structural rules in logics of semantic types.

Proofs in the ND-system for L with (e) precisely correspond to the lambda-terms fulfilling the following constraints: (c1) each subterm has a free variable, (c2) no subterm has more than one occurrence of the same free variable, (c3) for any subterm $\lambda x.M$, x is free in M. Dropping (c1) amounts to admitting empty antecedents, dropping (c2) adds contraction, and dropping (c3) adds integrality.

It is well known that the normalization procedure for typed lambda-terms corresponds to proof normalization in ND-systems. Every sequent provable in the product-free L with (e) possesses only finitely many normal proofs, so the corresponding lambda-terms define only finitely many semantic transformations. In other words, *every provable sequent admits only finitely many semantic readings*; this result is due to van Benthem [7]. This does not hold for logics with contraction.

Syntactic types can be mapped to semantic types. The map m can be defined on atomic types by $m(s) = t$, $m(n) = e$, $m(N) = (et)$ etc.; then, it is extended to all types by $m(\alpha\backslash\beta) = m(\beta/\alpha) = (m(\alpha)m(\beta))$. Every proof in the product-free L can be interpreted as a proof in L with (e), if one replaces each type α by $m(\alpha)$. Consequently, syntactic derivations in L-grammars determine semantic transformations, described above.

The map m is not one-one: different syntactic types collapse to one semantic type. Semantic types ignore the directionality. On the basic level, N and $n\backslash s$ are interpreted as (et), although the syntactic roles of common nouns and verb phrases are completely different.

As we have noted at the end of Section 2, semantics can be made sensitive to syntactic roles. Directional types $\alpha\backslash\beta$ and β/α can be used as semantic types. $D_{\alpha\backslash\beta}$ (resp. $D_{\beta/\alpha}$) is defined as the set of pairs (r, f) (resp. (l, f)) such that $f : D_\alpha \mapsto D_\beta$.

In this approach, noncommutative type logics L, L^* etc. can directly be applied as logics of semantic types. The appropriate version of Curry-Howard isomorphism relates proofs in these logics (presented as ND-systems) with directional lambda-terms, employing directional types, two lambdas λ^r and λ^l, and appropriately modified applications. The term construction rules are: (1) $(MN)_2 : \beta$, if $M : \alpha$, $N : \alpha\backslash\beta$, (2) $(MN)_1 : \beta$, if $M : \beta/\alpha$, $N : \alpha$, (3) $\lambda^r x.M : \alpha\backslash\beta$, if $x : \alpha$, $M : \beta$, (4) $\lambda^l x.M : \beta/\alpha$, if $x : \alpha$, $M : \beta$. The directional lambda calculus behaves like the standard one with regard to fundamental logical and computational properties (strong normalization, the Curry-Howard isomorphism). These ideas were announced in e.g. [14, 16] and further worked out by Wansing [48].

Within this framework the collapse of different syntactic categories into one semantic category can easily be removed. Although common nouns may be interpreted as functions from D_e to D_t, they are not treated as functors: f is neither (r, f), nor (l, f). Thus, D_N is a basic ontological category, consisting of such functions, but $D_N \neq D_{e\backslash t}$, the latter consisting of pairs (r, f), for $f \in D_N$. As a consequence, the semantic category of type N does not collapse with the semantic category of type $e\backslash t$. Going this way, one attains a better compatibility of syntactic and semantic categories in Lambek grammars.

References

[1] K. Ajdukiewicz, W sprawie 'uniwersaliów'. (On the Problem of 'Universals'.) *Przegląd Filozoficzny* 37, (1934), 219–234.

[2] K. Ajdukiewicz, Die syntaktische Konnexität. (Syntactic Connexion.) *Studia Philosophica* 1, (1935), 1–27.

[3] K. Ajdukiewicz, Związki składniowe między członami zdań oznajmujących. (Syntactical Connections between Constituents of Declarative Sentences.) *Studia Filozoficzne* 6.21, (1960), 73–86.

[4] K. Ajdukiewicz, *The Scientific World-Perspective and Other Essays, 1931-1963.*, (J. Giedymin, Ed.), D. Reidel, Dordrecht, 1978.

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

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

[7] J. van Benthem, *Essays in Logical Semantics.* D. Reidel, Dordrecht, 1986.

[8] J. van Benthem and A. ter Meulen (Eds.), *Handbook of Logic and Language*, Elsevier, The MIT Press, Amsterdam, 1997.

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

[10] W. Buszkowski, Compatibility of a categorial grammar with an associated category system. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 28.3, (1982), 229–238.

[11] W. Buszkowski, Typed functorial languages. *Bull. Pol. Acad. Sci. Math.* 34.7-8, (1986), 495–505.

[12] W. Buszkowski, Generative capacity of Nonassociative Lambek Calculus. *Bull. Pol. Acad. Sci. Math.* 34.7-8, (1986), 507–516.

[13] W. Buszkowski, Solvable problems for classical categorial grammars. *Bull. Pol. Acad. Sci. Math.* 35.5-6, (1987), 373–382.

[14] W. Buszkowski, *Logiczne podstawy gramatyk kategorialnych Ajdukiewicza-Lambeka.* (Logical Foundations of Ajdukiewicz-Lambek Categorial Grammars.) Państwowe Wydawnictwo Naukowe, Warszawa, 1989.

[15] W. Buszkowski, Extending Lambek grammars to basic categorial grammars. *Journal of Logic, Language and Information* 5.3-4, (1996), 279–295.

[16] W. Buszkowski, Mathematical Linguistics and Proof Theory. In: [8], 683–736.

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

[18] W. Buszkowski, Lambek calculus and substructural logics. *Linguistic Analysis* 36.1-4, (2010), 15–48.

[19] W. Buszkowski, Interpolation and FEP for logics of residuated algebras. *Logic Journal of the IGPL* 19.3, (2011), 437–454.

[20] W. Buszkowski, An interpretation of Full Lambek Calculus in its variant without empty antecedents of sequents. In: *Logical Aspects of Computational Linguistics 2014.* (N. Asher and S. Soloviev, Eds.), LNCS 8535, Springer, 2014, 30–43.

[21] W. Buszkowski and G. Penn, Categorial grammars determined from linguistic data by unification. *Studia Logica* 49.4, (1990), 431–454.

[22] A. Clark, A learnable representation for syntax using residuated lattices. *Lecture Notes in Artificial Intelligence* 5591, 2011, 183–198.

[23] N. Galatos, P. Jipsen, T. Kowalski and H. Ono, *Residuated Lattices: An Algebraic Glimpse at Substructural Logics.* Elsevier, Amsterdam, 2007.

[24] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, 1979.

[25] E. Husserl, *Logische Untersuchungen.* Halle, 1900-1901.

[26] M. Kanazawa, The Lambek calculus enriched with additional connectives. *Journal of Logic, Language and Information* 1.2, (1992), 141–171.

[27] M. Kanazawa, Identification in the limit of categorial grammars. *Journal of Logic, Language and Information* 5.2, (1996), 115–155.

[28] M. Kandulski, The equivalence of nonassociative Lambek categorial grammars and context-free grammars. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 34.1, (1988), 41–52.

[29] M. Kandulski, Phrase structure languages generated by categorial grammars with product. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 34.3, (1988), 373–383.

[30] E.L. Keenan and L.M. Faltz, *Boolean Semantics for Natural Language.* D. Reidel, Dordrecht, 1985.

[31] M. Kozak, Distributive Full Lambek Calculus has the finite model property. *Studia Logica* 91.2, (2009), 201–216.

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

[33] J. Lambek, On the calculus of syntactic types. In: *Structure of Language and Its Mathematical Aspects.* (R. Jakobson, Ed.), AMS, Providence, 1961, 166–178.

[34] J. Lambek, *From Word to Sentence: a computational algebraic approach to grammar.* Polimetrica, 2008.

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

[36] W. Marciszewski, How freely can categories be assigned to expressions of natural language? A case study. In: *Categorial Grammar*, (W. Buszkowski, W. Marciszewski and J. van Benthem, Eds.), J. Benjamins, Amsterdam, 1988, 197–220.

[37] R. Montague, *Formal Philosophy.* (R.H. Thomason, Ed.), Yale University Press, New Haven, 1974.

[38] M. Moortgat, Categorial Type Logics. In: [8], 93–177.

[39] R. Moot and C. Retoré, *The Logic of Categorial Grammars. A Deductive Account of Natural Language Syntax and Semantics.* Lecture Notes in Computer Science 6850, Springer, 2012.

[40] G. Morrill, *Type-Logical Grammar. Categorial Logic of Signs.* Kluwer, Dordrecht, 1994.

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

[42] B. H. Partee, A. ter Meulen and R.E. Wall, *Mathematical Methods in Linguistics.* Kluwer, Dordrecht, 1990.

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

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

[45] R. Suszko, Syntactic structure and semantical reference. I. *Studia Logica* 8, (1958), 213–244. II. *Studia Logica* 9, (1960), 63–91.

[46] M. Tałasiewicz, *Philosophy of Syntax. Foundational Topics.* Trends in Logic 29, Springer, 2010.

[47] A. Tarski, *Pojęcie prawdy w językach nauk dedukcyjnych.* Warszawa, 1933. English version: The concept of truth in formalized languages. In: A. Tarski, *Logic, Semantics, Metamathematics.* Clarendon, Oxford, 1956.

[48] H. Wansing, *The logic of information structures.* PhD Thesis, University of Amsterdam, 1992.

[49] C. Wurm, Completeness of Full Lambek Calculus for Syntactic Concept Lattices. *Lecture Notes in Computer Science* 8036, 2013, 126–141.

[50] U. Wybraniec-Skardowska, *Theory of Language Syntax. Categorial Approach.* Kluwer, Dordrecht, 1991.