Categorial Grammars and Substructural Logics

Wojciech Buszkowski
Adam Mickiewicz University in Poznań
University of Warmia and Mazury in Olsztyn

1 Introduction

Substructural logics are formal logics whose Gentzen-style sequent systems abandon some/all structural rules (Weakening, Contraction, Exchange, Associativity). They have extensively been studied in current literature on nonclassical logics from different points of view: as sequent axiomatizations of relevant, multiple-valued and fuzzy logics [73, 44], as so-called resource-sensitive logics for computer science (here linear logics of Girard [42] are most representative), and as type logics for categorial grammars. In this paper we mainly focus on the latter perspective, but several basic ideas are also significant for other approaches. For example, applications of linear logics in linguistics have been proposed in the style of type logics (see Casadio [30], Lambek [61], de Groote and Lamarche [43]), and algebraic models for linear logics (phase-space models) have been constructed in a similar way as powerset models for type logics [12, 21].

In *type logics* formulae are interpreted as types. In semantic terms, $A \to B$ is a type of functions (procedures) which send inputs of type A to outputs of type B, and $A \otimes B$ is a type of pairs (f, g) such that f is of type A and g is of type B. In syntactic terms, $A \to B$ (resp. $B \leftarrow A$) is a type of functional expressions (functors) b which together with any argument a of type A form a complex expression ab (resp. ba) of type B.The scope of possible realizations is huge: from constructivism in mathematics to logics of computation, from combinators and lambda calculus to linear logics, from type theories of Russell and Church to theories of syntax, proposed by Leśniewski, Ajdukiewicz, Chomsky, Curry and Lambek.

Connections between logic and grammar is a leitmotive of philosophy of language and many threads of philosophical logic. Categorial grammar whose main components are: (i) an assignment of logical types to lexical atoms (words), (ii) a system of type logic which provides a syntactic description of complex expressions, is a kind of formal grammar especially close to symbolic logic (much more than e.g. context-free grammar and context-sensitive grammar which are purely combinatorial devices). The general possibility of language description by means of logical types and logical relations between them was regarded by Hiż as the doctrine of *grammar logicism*. It seems to be important that the impact is

double-side. Not only logic serves as a reservoir of ready tools for grammar, but also theories of grammar stimulate new developments of logic. In this paper we exhibit many logical formalisms created for purposes of grammatical analysis, and beyond doubt they influence certain current issues of symbolic logic.

Since Montague [67], semantics of natural language has extensively been studied within philosophical logic. Semantical problems seem to be more sophisticated and solutions less definitive. Type grammars are especially close to the description of syntax according to general semantic paradigms. Connections of type grammars with Montague semantics are based on the Curry-Howard isomorphism between proofs in Natural Deduction format and lambda terms. For Lambek logics, main results on the Curry-Howard line are due to van Benthem [89, 91], Moortgat [68, 69] and their collaborators. In this paper, we abandon semantical questions altogether.

In section 2 we present a variety of logical systems relevant to type logics. Starting from the basic logic AB (after Ajdukiewicz and Bar-Hillel), which underlies Classical Categorial Grammar, we continue with more flexible systems of the Lambek calculus, originated by Lambek [59, 60], to end with their extensions toward linear logics and action logics. Formal systems are compared with their algebraic models, especially those models which are significant for linguistics. In section 3 we consider categorial grammars, based on different type logics, and discuss characteristic topics like weak and strong generative capacity, complexity and learning algorithms.

This paper is a continuation of the author's [15, 20, 25]. We refer to those surveys and to other items in literature for a more detailed exposition of many technical matters. Here we mainly focus on basic concepts and methods and try to make the exposition readable for a wide community. Other general expositions can be found in [91, 69, 83].

Although this paper is written in the form of a survey, it quite often refers to earlier research of the author and his collaborators. It also announces some new, unpublished results: on the complexity of certain substructural logics (section 2), on rough set approach in formal learning theory and a PTIME construction of a CF-grammar equivalent to a given pregroup grammar (section 3).

2 Substructural logics as type logics

2.1 Basic type logics

Type logics are usually formalized as propositional logics. Variables $p, q, r, \ldots$ are atomic formulae. Complex formulae are formed out of atomic formulae by means of logical connectives. In the simplest case, one admits two connectives $\rightarrow$ (right conditional) and $\leftarrow$ (left conditional). Formulae will be denoted by $A, B, C, \ldots$. In linguistic literature, one often writes $A\backslash B$ for $A \rightarrow B$ and A/B for $A \leftarrow B$. It is convenient to present these logics as sequent systems, this means, systems operating on *sequents* $\Gamma \vdash A$ whose antecedent Γ is a finite sequence (string) of formulae. The intended meaning of the sequent

$A_1, \ldots, A_n \vdash A$ is the following: for any expressions a_i of type A_i, $i = 1, \ldots, n$, the complex expression $a_1 \ldots a_n$ is of type A. The latter complex is often understood as the concatenation of strings $a_1, \ldots, a_n$, but other interpretation are also in use; in general, $a_1 \ldots a_n$ is composed of $a_1, \ldots, a_n$ by some syntactic operation.

The logic AB can be presented as a rewriting system whose basic rules are:

$$(\text{AB1})\ A, (A \to B) \Rightarrow B,\ (\text{AB2})\ (B \leftarrow A), A \Rightarrow B,$$

for arbitrary formulae A, B. For instance, *John* of type NP together with *dreams* of type NP→S forms a sentence *John dreams* (of type S). In general, a string b of functor-type $A \to B$ (resp. $B \leftarrow A$) takes an argument a of type A on the left-hand (resp. right-hand) side, giving rise to a complex string ab (resp. ba) of type B. A string $A_1, \ldots, A_n$ of types *reduces* to type A in AB, if A is the result of a successive application of rules (AB1), (AB2) (a finite number of times, possibly zero). For instance, if *some* is of type Det and *poet* of type N, where Det=NP←N, then *some poet dreams* initiates the reduction:

$$(\text{NP}\leftarrow\text{N}),\text{N},(\text{NP}\to\text{S})\Rightarrow\text{NP},(\text{NP}\to\text{S})\Rightarrow\text{S}.$$

A reduction of Γ to A can be treated as an AB-proof of the sequent $\Gamma \vdash A$. We write $\Gamma \vdash_{\mathcal{L}} A$ if $\Gamma \vdash A$ is provable in logic $\mathcal{L}$. So, $\Gamma \vdash_{AB} A$ means that Γ reduces to A in the sense of AB.

Actually, Ajdukiewicz [2] introduced the $(\leftarrow)-$fragment of this logic; precisely, he regarded multiple-argument types $B \leftarrow A_1, \ldots, A_n$ with reduction rules:

$$(B \leftarrow A_1, \ldots, A_n), A_1, \ldots, A_n \Rightarrow B.$$

Bidirectional types are due to Bar-Hillel [4] and Lambek [59].

In Natural Deduction (ND) form AB can be axiomatized by axioms:

$$(\text{Id})\ A \vdash A$$

and two elimination rules for conditionals:

$$(\to\text{E})\ \frac{\Gamma \vdash A;\ \Delta \vdash A \to B}{\Gamma, \Delta \vdash B},\ (\leftarrow\text{E})\ \frac{\Gamma \vdash B \leftarrow A;\ \Delta \vdash A}{\Gamma, \Delta \vdash B}.$$

Proofs in the ND-format determine the so-called *functor-argument structures* (fa-structures) on antecedents of provable sequents. Types are atomic fa-structures. If X, Y are fa-structures, then $(XY)_1$ and $(XY)_2$ are fa-structures. X is the functor, and Y is the argument of $(XY)_1$, and the converse holds for $(XY)_2$. The logic ABs (i.e. AB in structure form) arises from AB (in ND-format) by replacing $\Gamma$with X, Δ with Y in premises of rules, and $\Gamma\Delta$ with $(XY)_2$ (resp. $(XY)_1$) in the conclusion of $(\to\text{E})$ (resp. $(\leftarrow\text{E})$). Every proof of $\Gamma \vdash A$ in AB determines a unique fa-structure X on Γ such that $X \vdash_{ABs} A$. The same could be done for the original version of AB. The notion of a functor-argument structure plays an important role in the theory of categorial grammars (see section 3).

Ajdukiewicz [2] described a simple parsing procedure for the $(\leftarrow)-$fragment of AB. Given a string Γ, find the left-most block of the form of $(B \leftarrow A), A$ and replace it with B. Then, $\Gamma \vdash_{AB} C$ iff Γ reduces to C by finitely many applications of left-most reductions. As shown in [19], this is not true for multiple-argument types (considered by Ajdukiewicz) but is true for one-argument types, considered here. The Ajdukiewicz procedure is fully deterministic and shows, in fact, that languages generated by rigid categorial grammars based on the $(\leftarrow)-$fragment of AB are deterministic context-free languages, i.e. they can be recognized by deterministic push-down automata (rigid grammars are defined below). This shows natural connections with the Polish notation (generalized to languages of higher-order type) [19].

The logic AB reflects the idea that types of linguistic expressions are completely determined by the declaration. A different approach was elaborated by Lambek [59]. Type-forming operations $\rightarrow, \leftarrow, \otimes$ are interpreted as operations in the algebra of languages. Let Σ be an alphabet. Σ^* denotes the set of all strings on Σ, and $\Sigma^+ = \Sigma^* - \{\epsilon\}$, wher ϵ is the empty string. An arbitrary subset of Σ^* (resp. Σ^+) is called a (resp. $\epsilon-$free) *language* on Σ.

Let L_1, L_2 be ϵ-free languages on Σ. One defines:

$$L_1 \otimes L_2 = \{ab : a \in L_1 \,\&\, b \in L_2\},$$

$$L_1 \rightarrow L_2 = \{c \in \Sigma^+ : L_1 \otimes \{c\} \subseteq L_2\},\ L_1 \leftarrow L_2 = \{c \in \Sigma^+ : \{c\} \otimes L_2 \subseteq L_1\}.$$

We write $L_1 L_2$ for $L_1 \otimes L_2$. If $L_1, L_2 \subseteq \Sigma^*$, then these operations are defined in a similar way with replacing Σ^+ by Σ^*. It makes a difference; for instance, if $L = \{a\}$, $a \neq \epsilon$, then $L \rightarrow L = \emptyset$ or $L \rightarrow L = \{\epsilon\}$, depending on the assumed universe of strings: Σ^+ or Σ^*. The operation $\otimes$ is called *product*.

The Lambek calculus (L) produces all sequents $A_1, \ldots, A_n \vdash A$ which are true in the algebra of $\epsilon-$free languages, this means, true under any assignment of arbitrary $\epsilon-$free languages for variables, if one interprets $\vdash$ as $\subseteq$ and $A_1, \ldots, A_n$ as $A_1 \otimes \cdots \otimes A_n$. An algebraic axiomatization of L is restricted to *simple* sequents $A \vdash B$, admits axioms (Id) and:

$$\text{(ASS1) } (A \otimes B) \otimes C \vdash A \otimes (B \otimes C),\ \text{(ASS2) } A \otimes (B \otimes C) \vdash (A \otimes B) \otimes C,$$

and the following inference rules:

$$\text{(RES1) } \frac{A \otimes B \vdash C}{B \vdash A \rightarrow C},\ \text{(RES2) } \frac{A \otimes B \vdash C}{A \vdash C \leftarrow B},$$

$$\text{(RES3) } \frac{B \vdash A \rightarrow C}{A \otimes B \vdash C},\ \text{(RES4) } \frac{A \vdash C \leftarrow B}{A \otimes B \vdash C}.$$

$$\text{(CUT1) } \frac{A \vdash B;\ B \vdash C}{A \vdash C}.$$

Clearly, rules (RES1) and (RES3) are mutually converse, and similarly (RES2) and (RES4). (CUT1) is the *cut*-rule, restricted to simple sequents. It is easy to show that all axioms are true, and all rules preserve the truth in

the algebra of $\epsilon-$free languages. Consequently, all sequents provable in L are true in this algebra (soundness). The converse statement (completeness: all sequents true in this algebra are provable in L) is a nontrivial result of Pentus [78]; one must assume that Σ contains at least two elements. The completeness theorem for the $(\rightarrow, \leftarrow)-$fragment of L is easier [12]. We sketch the proof later on.

The Lambek calculus is an extension of AB. From axioms $A \rightarrow B \vdash A \rightarrow B$ and $B \leftarrow A \vdash B \leftarrow A$ using (RES3) and (RES4) one infers $A \otimes (A \rightarrow B) \vdash B$ and $(B \leftarrow A) \otimes A \vdash B$, which correspond to (AB1), (AB2) (one can replace $\otimes$ by comma). Using (RES2) and (RES1) to the latter sequents, one derives:

$$\text{(L1) } A \vdash B \leftarrow (A \rightarrow B),\ A \vdash (B \leftarrow A) \rightarrow B,$$

(type-raising laws, not provable in AB). We provide a list of other L-provable sequents, going beyond AB:

(L2) $(A \rightarrow B) \otimes (B \rightarrow C) \vdash A \rightarrow C,\ (A \leftarrow B) \otimes (B \leftarrow C) \vdash A \leftarrow C$ (composition laws),

(L3) $A \rightarrow B \vdash (C \rightarrow A) \rightarrow (C \rightarrow B)$, $A \leftarrow B \vdash (A \leftarrow C) \leftarrow (B \leftarrow C)$ (Geach laws),

(L4) $(A \rightarrow B) \leftarrow C \vdash A \rightarrow (B \leftarrow C)$, $A \rightarrow (B \leftarrow C) \vdash (A \rightarrow B) \leftarrow C$ (associativity laws for conditionals),

(L5) $A \vdash B \rightarrow (B \otimes A)$, $A \vdash (A \otimes B) \leftarrow B$ (second type-raising laws),

(L6) $(A \rightarrow B) \otimes C \vdash A \rightarrow (B \otimes C)$, $A \otimes (B \leftarrow C) \vdash (A \otimes B) \leftarrow C$ (Grishin laws).

Let PN be the type of proper noun. Then, transitive verbs can be assigned type PN→S, and type NP, of noun phrase, can be defined as S←(PN→S). Since PN⊢NP is an instance of (L1), then every proper noun can also be regarded as a noun phrase. Raising proper nouns to the type of noun phrase was essential in Montague treatment of coordinate phrases like *John and some poet*. Let S←S be the type of *not*. Then, the sentence *not every poet dreams* admits a single analysis in AB, namely (not ((every poet) dreams)) (omit the functor indices in the corresponding functor-argument structure). In the Lambek calculus, it also admits structures: ((not (every poet)) dreams) and (((not every) poet) dreams). For the first one, S←S⊢NP←NP, by (L3) and the definition of NP. For the second one, S←S⊢Det←Det, where Det=NP←N is the type of determiner. A precise definition of structures determined by a categorial grammar will be given in section 3.

Proof-theoretic investigations of the Lambek calculus often employ another axiomatization, namely a Gentzen-style sequent system. It operates with arbitrary sequents $\Gamma \vdash A$ and admits axioms (Id) and the following inference rules, naturally divided in left- and right-introduction rules for connectives:

$$(\otimes\text{L})\ \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, A \otimes B, \Delta \vdash C},\ (\otimes\text{R})\ \frac{\Gamma \vdash A;\ \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B},$$

$$(\to\mathrm{L})\ \frac{\Gamma, B, \Delta \vdash C;\ \Phi \vdash A}{\Gamma, \Phi, A \to B, \Delta \vdash C},\ (\to\mathrm{R})\ \frac{A, \Gamma \vdash B}{\Gamma \vdash A \to B},$$
$$(\leftarrow\mathrm{L})\ \frac{\Gamma, B, \Delta \vdash C;\ \Phi \vdash A}{\Gamma, B \leftarrow A, \Phi, \Delta \vdash C},\ (\to\mathrm{R})\ \frac{\Gamma, A \vdash B}{\Gamma \vdash B \leftarrow A},$$
$$(\mathrm{CUT})\ \frac{\Gamma, A, \Delta \vdash B;\ \Phi \vdash A}{\Gamma, \Phi, \Delta \vdash B}.$$

For L, one assumes $\Gamma \neq \epsilon$ in rules ($\to$R), ($\leftarrow$R), and consequently, all provable sequents $\Gamma \vdash A$ have nonempty antecedents Γ. The latter system is equivalent to the former one in the following sense: (1) all simple sequents provable in the former system are provable in the latter, (2) if $A_1, \ldots, A_n \vdash A$ is provable in the latter system, then $A_1 \otimes \cdots \otimes A_n \vdash A$ is provable in the former.

Lambek [59] proves the cut-elimination theorem: any sequent provable in L can be proved in L without (CUT). Equivalently, the set of provable sequents of L without (CUT) is closed under (CUT) (this means: (CUT) is admissible in the cut-free fragment of L). One must prove: if both premises of (CUT) are provable in L without (CUT), then the conclusion of (CUT) is provable in L without (CUT). The proof can be arranged as follows. We apply induction I: on the complexity of the cut-formula A, i.e. the total number of occurrences of symbols in A. For each case, we apply induction II: on the length of the proof of the left premise of (CUT), i.e. the total number of sequents appearing in this proof (represented as a tree). For some cases, we apply induction III: on the length of the proof of the right premise of (CUT). Let us consider one case: $A = A_1 \to A_2$. We switch on induction II. If $\Gamma, A, \Delta \vdash B$ is an axiom (Id), then Γ and Δ are empty, and $\Gamma, \Phi, \Delta \vdash B$ equals $\Phi \vdash A$. If $\Gamma, A, \Delta \vdash B$ arises by any rule except for ($\to$L), introducing the formula A, then the thesis immediately follows from the hypothesis of induction II. Let $\Gamma, A, \Delta \vdash B$ arise by the case, excluded in the preceding sentence. We switch on induction III. If $\Phi \vdash A$ is an axiom (Id), then the conclusion of (CUT) equals the left premise. If $\Phi \vdash A$ arises by any rule except for ($\to$R), then the thesis immediately follows from the hypothesis of induction III. Let $\Phi \vdash A$ arises by ($\to$R), so the premise is $A_1, \Phi \vdash A_2$. But, $\Gamma, A, \Delta \vdash B$ arises by ($\to$L) with premises $\Gamma', A_2, \Delta \vdash B$, $\Gamma'' \vdash A_1$, where $\Gamma = \Gamma', \Gamma''$. By the hypothesis of induction I, we get $\Gamma'', \Phi \vdash A_2$, and by the hypothesis of induction I again, $\Gamma', \Gamma'', \Phi, \Delta \vdash B$ is provable in L without (CUT).

Since all introduction rules increase the complexity of sequents, this means: the complexity of the conclusion is greater than the complexity of each premise, and the premises are formed out of subformulae of formulae appearing in the conclusion, then L admits a standard proof-search decision procedure. For example, we show that $(A \otimes B) \leftarrow C \vdash A \otimes (B \leftarrow C)$ is not provable in L. The sequent is not an axiom (Id). It could be the conclusion of only two rules: (1) ($\leftarrow$L) whose right premise is $\vdash C$, (2) ($\otimes$R) whose right premise is $\vdash B \leftarrow C$. Neither of these possibilities holds, since no sequent with the empty antecedent is provable in L.

If we omit the restriction $\Gamma \neq \epsilon$ in rules ($\to$R), ($\leftarrow$R), then we get a stronger system L* which is complete with respect to the algebra of all languages. From

$p \vdash p$ one infers $\vdash p \to p$, by ($\to$R), which yields $(p \to p) \to p \vdash p$, by ($\to$L). This proof is correct in L*, but not in L. We have shown that L* is stronger than L also in the scope of sequents with nonempty antecedents. The cut-elimination theorem holds for L*. The sequent considered in the preceding paragraph is not provable in L*, if A, B, C are different variables: for case (1), we notice that $\vdash C$ is unprovable, and for case (2), $\vdash B \leftarrow C$ can arise by ($\leftarrow$R) whose premise is $C \vdash B$, but the latter sequent is unprovable.

We say that formulae A, B are *equivalent* in logic $\mathcal{L}$, if $A \vdash_{\mathcal{L}} B$ and $B \vdash_{\mathcal{L}} A$. For example, $A \to B$ and $(B \leftarrow (A \to B)) \to B$ are equivalent in L, and consequently, in L*. Formulae $A \leftarrow A$ and $(A \leftarrow A) \leftarrow (A \leftarrow A)$ are equivalent in L* (not in L). This phenomenon causes certain troubles in natural language processing. If adjectives are assigned type Adj=N$\leftarrow$N and adjective modifiers (e.g. *very*) type Adj$\leftarrow$Adj, then L* treats both types as equivalent. So, *a famous poet* and *a very famous poet* are well-formed noun phrases, whence also *a very poet* must be accepted. For this reason, linguists prefer the weaker system L. On the other hand, L* is a more standard logical system: it produces not only provable sequents but also provable formulae (theorems).

Let us sketch the proof that the $(\to, \leftarrow)-$fragment of L* is complete with respect to the algebra of languages. Soundness is easy. For completeness, we construct a canonical model [12, 17]. We fix a sequent $\Gamma_0 \vdash A_0$. Let S denote the set of all subformulae of formulae appearing in this sequent. Clearly, S is a finite set. We consider languages on S, i.e. subsets of S^*. An assignment f of languages to variables is defined as follows:

$$f(p) = \{\Gamma \in S^* : \Gamma \vdash_{L^*} p\}.$$

We prove:

$$f(A) = \{\Gamma \in S^* : \Gamma \vdash_{L^*} A\},$$

by induction on the complexity of formula $A \in S$. If $A = p$, then the latter holds, by the definition of f. Let $A = B \to C$. Assume $\Gamma \in f(B \to C)$. Since $B \vdash B$ is provable, then $B \in f(B)$, by the induction hypothesis. Consequently, string B, Γ belongs to $f(C)$, which yields $B, \Gamma \vdash_{L^*} C$, by the induction hypothesis. We get $\Gamma \vdash_{L^*} B \to C$, by ($\to$R). Assume $\Gamma \vdash_{L^*} B \to C$. Let $\Delta \in f(B)$. Then, $\Delta \vdash_{L^*} B$, by the induction hypothesis. Since $B, (B \to C) \vdash C$ is provable, by (Id) and ($\to$L), then $\Delta, \Gamma \vdash_{L^*} C$, by (CUT). Using the induction hypothesis again, we conclude that string Δ, Γ belongs to $f(C)$. Consequently, $\Gamma \in f(B \to C)$. For $A = C \leftarrow B$, the reasoning is dual.

Assume that $\Gamma_0 \vdash A_0$ is not provable in L*. By the above equality, $\Gamma_0 \in f(\Gamma_0)$, but $\Gamma_0 \notin f(A_0)$ (remind that comma is interpreted as product). Thus, $\Gamma_0 \vdash A_0$ is not true under the assignment f.

The completeness of the product-free fragment of L with respect to the algebra of $\epsilon-$free languages can be shown in a similar way (replace S^* by S^+). Using a standard encoding of strings on S by means of strings on $\{0, 1\}$, one proves the completeness of both systems with respect to the algebras of languages on the binary alphabet.

Lambek [60] introduced the nonassociative version of L, naturally related to tree structures of linguistic expressions. *Formula structures* are defined as follows:

(FS1) any formula is a formula structure,

(FS2) if X, Y are formula structures, then $(X \circ Y)$ is a formula structure.

By $X[Y]$ we denote a formula structure X with a designated occurrence of a substructure Y; in this context, $X[Z]$ denotes the result of substituting Z for Y in X.

The nonassociative Lambek calculus (NL) operates with sequents $X \vdash A$ such that X is a formula structure and A is a formula. Its axioms are (Id) and inference rules are the following:

$$(\otimes\mathrm{L})\ \frac{X[A \circ B] \vdash C}{X[A \otimes B] \vdash C},\ (\otimes\mathrm{R})\ \frac{X \vdash A;\ Y \vdash B}{X \circ Y \vdash A \otimes B},$$

$$(\rightarrow\mathrm{L})\ \frac{X[B] \vdash C;\ Y \vdash A}{X[Y \circ (A \rightarrow B)] \vdash C},\ (\rightarrow\mathrm{R})\ \frac{A \circ X \vdash B}{X \vdash A \rightarrow B},$$

$$(\leftarrow\mathrm{L})\ \frac{X[B] \vdash C;\ Y \vdash A}{X[(B \leftarrow A) \circ Y] \vdash C},\ (\leftarrow\mathrm{R})\ \frac{X \circ A \vdash B}{X \vdash B \leftarrow A},$$

$$(\mathrm{CUT})\ \frac{X[A] \vdash B;\ Y \vdash A}{X[Y] \vdash B}.$$

The cut-elimination theorem can be proved for NL as above. Then, NL admits a proof-search decision procedure. Actually, NL is computationally easier than L. The decision problem for L and L* is NP-complete [79], while it is P for NL [43]. The difference is more strike for consequence relations determined by these logics. For a set of sequents T, by $\mathcal{L}(T)$ we denote the logic $\mathcal{L}$ enriched with all sequents from T as additional axioms. Since the cut-elimination theorem does not hold, in general, for such systems, rule (CUT) must be included in the list of inference rules. For any finite set T, the decision problem for $\mathrm{NL}(T)$ is P, and similarly for the global problem (T is a datum) [26], while it is undecidable (Σ^0_1-complete) for the case of $\mathrm{L}(T)$ and $\mathrm{L}^*(T)$, even for some fixed finite set T of sequents of the form $p, q \vdash r$ or $p \rightarrow q \vdash r$ [10].

An algebraic axiomatization of NL arises from that of L by dropping axioms (ASS1), (ASS2). NL is essentially weaker than L. (L1) and (L5) are provable in NL, but (L2), (L3), (L4) and (L6) are not provable. Lambek [60] argues that associativity can lead to overgeneration. For example, on the basis of sentences like *John sees him* one assigns type (S←PN)→S to *him*, which is a natural type of noun phrase on the object position. Taking PN←PN as the type of adjective *poor* and (PN→S)←PN as the type of transitive verb *sees*, L produces type S of both *John sees poor Bob* and *John sees poor him*, while NL does not compute type S of the latter string. Lambek's argument needs completion. Since L is sound with respect to the algebra of languages, then it cannot produce incorrect types of complex strings, if types of lexical atoms are

stipulated correctly. This example shows that *him* cannot be correctly assigned type (S←PN)→S, if concatenation is treated as an associative operation. After one has avoided associativity, the type assignment is correct.

2.2 Algebraic models and extended type logics

We discuss more general models for type logics. This perspective gives rise to several interesting extensions of basic type logics.

Algebras of languages are instances of residuated semigroups. *A residuated semigroup* is a structure $\mathcal{M} = (M, \leq, \cdot, \rightarrow, \leftarrow)$ such that $(M, \leq)$ is a poset, $(M, \cdot)$ is a semigroup (this means: $\cdot$ is an associative, binary operation on M), and $\rightarrow, \leftarrow$ are binary operations on M, satisfying the equivalences:

$$\text{(RES) } ab \leq c \text{ iff } b \leq a \rightarrow c \text{ iff } a \leq c \leftarrow b,$$

for all $a, b, c \in M$. As a consequence, one derives monotonicity conditions:

(MON1) if $a \leq b$ then $ca \leq cb$ and $ac \leq bc$,

(MON2) if $a \leq b$ then $c \rightarrow a \leq c \rightarrow b$ and $a \leftarrow c \leq b \leftarrow c$,

(MON3) if $a \leq b$ then $b \rightarrow c \leq a \rightarrow c$ and $c \leftarrow b \leq c \leftarrow a$,

for all $a, b, c \in M$. Operations $\rightarrow, \leftarrow$ are called *residuals* with respect to product. Dropping associativity of product, one defines the notion of *a residuated groupoid* (monotonicity conditions hold also in this case). A semigroup with identity 1, satisfying $a \cdot 1 = a = 1 \cdot a$, is called a monoid. In a natural way, one defines the notions of a residuated monoid and a residuated groupoid with identity. If $(M, \leq)$ is a lattice with the meet operation $\wedge$ and the join operation $\vee$, then the corresponding residuated monoid is called *a residuated lattice*. It can contain the least element 0 and/or the greatest element $\top$ (sometimes, we write $\bot$ for 0).

The algebra of languages can be expanded to a residuated lattice by setting: $L_1 \wedge L_2 = L_1 \cap L_2$, $L_1 \vee L_2 = L_1 \cup L_2$, $1 = \{\epsilon\}$, $0 = \emptyset$, $\top = \Sigma^*$. This algebra is a special case of a powerset algebra. Let $(M, \cdot, 1)$ be a monoid. On the powerset of M one defines operations $\cdot$, $\wedge$, $\vee$, $\rightarrow$ and $\leftarrow$ as for languages ($\cdot$ equals $\otimes$), replacing $c \in \Sigma^+$ by $c \in M$, and $\leq$ as the set inclusion. With $\mathbf{1} = \{1\}$, the resulting structure is a residuated lattice, called *the powerset algebra* over monoid $(M, \cdot, 1)$. In a similar way, one defines the powerset algebra over semigroup $(M, \cdot)$ (now, it is a residuated lattice without identity).

Other natural examples of residuated lattices are relation algabras. On the set of binary relations on a universe U, one defines operations:

$$R \circ S = \{(x, y) \in U^2 : \exists z((x, z) \in R \text{ and } (z, y) \in S)\},$$

$$R \rightarrow S = \{(x, y) \in U^2 : R \circ \{(x, y)\} \subseteq S\},$$

$$S \leftarrow R = \{(x, y) \in U^2 : \{(x, y)\} \circ R \subseteq S\},$$

and $\wedge$, $\vee$ as above. Setting $1 = \{(x, x) : x \in U\}$, $0 = \emptyset$, one obtains a residuated lattice.

Both algebras of languages and relation algebras are *complete* lattices, this means: they are closed under infinite joins and meets. Relation algebras provide an extensional semantics for programs in the style of dynamic logics. Interesting interaction between dynamic logics and natural language semantics have been explored by many authors; see e.g. van Benthem [91, 92] and the references there.

One can interpret formulae and sequents in these general algebras in a straightforward way; $\vdash$ is interpreted as $\leq$. L is complete with respect to residuated semigroups, L* with respect to residuated monoids, NL with respect to residuated groupoids, and analogous completeness theorems hold for systems enriched with connectives $\wedge$, $\vee$ and constants $1, 0, \top$, admitting the corresponding, additional axioms and inference rules (in the sequent form):

$$(1\mathrm{L})\ \frac{\Gamma, \Delta \vdash A}{\Gamma, 1, \Delta \vdash A},\ (1\mathrm{R})\ \vdash 1,$$

$$(0\mathrm{L})\ \Gamma, 0, \Delta \vdash A,$$

$$(\wedge\mathrm{L1})\ \frac{\Gamma, A, \Delta \vdash C}{\Gamma, A \wedge B, \Delta \vdash C},\ (\wedge\mathrm{L2})\ \frac{\Gamma, B, \Delta \vdash C}{\Gamma, A \wedge B, \Delta \vdash C},$$

$$(\wedge\mathrm{R})\ \frac{\Gamma \vdash A;\ \Gamma \vdash B}{\Gamma \vdash A \wedge B},$$

$$(\vee\mathrm{L})\ \frac{\Gamma, A, \Delta \vdash C;\ \Gamma, B, \Delta \vdash C}{\Gamma, A \vee B, \Delta \vdash C},$$

$$(\vee\mathrm{R1})\ \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B},\ (\vee\mathrm{R2})\ \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B},$$

$$(\top)\ \Gamma \vdash \top$$

The system L* extended to the language with $\wedge, \vee, 0, 1$ and supplied with the above axioms and inference rules for these connectives and constants is called *Full Lambek Calculus* (FL). The cut-elimination theorem holds for FL [73]. FL is complete with respect to the class of all residuated lattices. Since the distribution of $\wedge$ over $\vee$ is not provable in FL, this system is sound, but not complete with respect to relation algebras and algebras of languages. Completeness theorems hold for fragments of FL. The $(\vee, 0, 1)-$free fragment of FL is complete with respect to relation algebras [3, 28]. Also L with $\wedge$ is complete with respect to restricted relation algebras, consisting of subrelations of a transitive relation T (we write $(x, y) \in T$ in definitions of $\circ, \rightarrow, \leftarrow$). Similar completeness theorems hold for L* (resp. L) with $\wedge$ with respect to the class of powerset algebras over monoids (resp. semigroups). Since every residuated monoid (resp. semigroup) with meet is embeddable into the powerset algebra over some monoid (resp. semigroup), then these systems are even strongly complete with respect to powerset algebras [12, 21](the embedding h does not preserve 1; one only

gets: $1 \leq a$ iff $1 \in h(a)$). The product-free L and L* with $\wedge$ are strongly complete with respect to algebras of $\epsilon-$free languages and algebras of languages, respectively [12, 17].

An important operation on languages is *Kleene star* *, defined as follows: $L^* = \bigcup_{n\in\omega} L^n$, where $L^0 = \{\epsilon\}$, $L^{n+1} = L^n L$. In a simlar way, one defines * in powerset algebras (now $L^0 = \{1\}$, for $L \subseteq M$) and relation algebras (now R^0 is the identity relation). Let Σ be a finite alphabet. If we treat symbols from Σ as individual constants of the language of FL with *, then we can define a mapping $L(a) = \{a\}$, for $a \in \Sigma$. The mapping is naturally extended to a homomorphism from the algebra of variable-free formulae to the algebra of languages on Σ. *Regular expressions* can be identified with variable-free formulae of the $(\vee, \otimes, *, 0, 1)-$fragment of the language. Languages of the form $L(A)$ (A is a regular expression) are called *regular languages* on Σ. An equation $A = B$ is said to be *true for regular expressions* if $L(A) = L(B)$.

It is known that the equations true for regular expressions cannot be axiomatized by any finite set of equations [81]. Kozen [47] proved the following completeness theorem: $L(A) = L(B)$ iff $A = B$ is true in all Kleene algebras, i.e. algebras $\mathcal{M} = (M, \vee, \cdot, ^*, 0, 1)$ such that $(M, \vee, 0)$ is a join semilattice with the least element 0, $(M, \cdot, 1)$ is a monoid, $\otimes$ distributes over $\vee$, 0 is an annihilator for $\otimes$, and the * fulfills the axioms:

$$\text{(K1) } 1 \vee aa^* \leq a^*;\ 1 \vee a^*a \leq a^*,$$

$$\text{(K2) if } ab \leq b \text{ then } a^*b \leq b; \text{ if } ba \leq b \text{ then } ba^* \leq b$$

(here: $a \leq b$ iff $a \vee b = b$). Clearly, algebras of languages, powerset algebras and relation algebras are Kleene algebras. Kleene algebras do not form a variety (an equational class).

Pratt [80] introduced action algebras as Kleene algebras with residuals $\rightarrow, \leftarrow$. Precisely, *an action algebra* is an algebra $\mathcal{M} = (M, \vee, \cdot, ^*, \rightarrow, \leftarrow, 0, 1)$ such that $(M, \vee, \cdot, \rightarrow, \leftarrow, 0, 1)$ is a residuated join semilattice and * fulfills (K1), (K2). Kleene (resp. action) algebras supplied with meet are called Kleene (resp. action) lattices. As shown in [80], action algebras (lattices) form a finitely based variety. Furthermore, for regular expressions A, B, $L(A) = L(B)$ iff $A = B$ is true in all action algebras. Accordingly, in the language with residuals, one obtains a finite, purely equational axiomatization of the algebra of regular expressions (not containing residuals).

Type logics with * is an interesting research area. It is not known if the equational theory of all action algebras (lattices) is decidable [46]. The equational theory of all Kleene algebras is decidable (PSPACE-complete), since $L(A) = L(B)$ can be verified by means of finite-state automata.

A Kleene algebra is said to be **-continuous* if $xa^*y = \sup\{xa^ny : n \in \omega\}$. Since every action algebra is a Kleene algebra, the same definition can be applied to action algebras. All natural Kleene (action) algebras are *-continuous: algebras of languages, powerset algebras over monoids, relation algebras. It follows from the Kozen completeness theorem that the equational theory of all Kleene algebras equals that of all *-continuous Kleene algebras. It is not the

case for action algebras (lattices). The equational theory of *-continuous action lattices can be axiomatized in the form of a sequent system which extends FL by the following inference rules:

$$(*\mathrm{L})\ \frac{(\Gamma, A^n, \Delta \vdash B)_{n\in\omega}}{\Gamma, A^*, \Delta \vdash B},$$

$$(*\mathrm{R})\ \frac{\Gamma_1 \vdash A; \ldots; \Gamma_n \vdash A}{\Gamma_1, \ldots, \Gamma_n \vdash A^*},$$

for any $n \in \omega$. Thus, (*R) represents an infinite family of finitary rules, while (*L) is an infinitary rule (a kind of $\omega-$rule). This system provides all sequents true in all *-continuous action lattices [75]. Actually, it provides order formulae $a \leq b$, but every order formula is an equation $a \vee b = b$, and every equation $a = b$ can be represented by means of two order formulae $a \leq b$, $b \leq a$. We denote this system by ACTω.

ACTω admits cut-elimination, and the provability problem for ACTω is Π^0_1 (this means, the relation $\Gamma \vdash_{\mathcal{L}} A$, for $\mathcal{L}$=ACTω, is a complement of a recursively enumerable relation) [75]. Then, ACTω is a conservative extension of FL. In [27], it has been shown that ACTω is Π^0_1-hard (the total language problem for context-free grammars is reducible to ACTω). Consequently, ACTω is Π^0_1-complete. Hence, the equational theory of *-continuous action algebras (lattices) admits no finitary, recursive axiomatization, so it is essentially stronger than the equational theory of all action algebras (lattices). It can be shown that the equational theory of algebras of languages (with residuals and *) is even more complex.

Noncommutative Linear Logic (NcLL) extends FL by new connectives and constants: $\oplus$ (par) is a dual for $\otimes$, ${}^l,{}^r$ are two linear negations, and 0 is a dual for 1 (now, the least element of a lattice will be denoted by $\bot$). Our notation is not standard in this domain; following Troelstra [87], we denote lattice operations by $\wedge, \vee$ and the correesponding constants by $\bot, \top$, which disagrees with the notation of Girard [42] and Abrusci [1]. Symbols r, l for negations are used by Lambek [62] in pregroups. We also write NcLL instead of the more standard NLL to avoid collision with our symbol NL for the nonassociative Lambek calculus.

A system of NcLL, given by Abrusci [1], is a classical sequent system, which means that one regards sequents of the form $\Gamma \vdash \Delta$, where Γ, Δ are finite sequences of formulae. Conditionals can be defined: $A \to B = A^r \oplus B$, $B \leftarrow A = B \oplus A^l$, so it suffices to write rules for ${}^l,{}^r, \otimes, \oplus, \wedge, \vee, 0, 1, \bot, \top$. The axioms are (Id), (1R), and ($\top$), ($\bot$L), (0L) in the form:

$$(\top)\ \Gamma \vdash \Delta_1, \top, \Delta_2,$$

$$(\bot\mathrm{L})\ \Gamma_1, \bot, \Gamma_2 \vdash \Delta,$$

$$(0\mathrm{L})\ 0 \vdash .$$

The inference rules are:

$$(\mathrm{rL})\ \frac{\Gamma \vdash \Delta, A}{\Gamma, A^r \vdash \Delta},\ (\mathrm{rR})\ \frac{A, \Gamma \vdash \Delta}{\Gamma \vdash A^r, \Delta},$$

$$(\mathrm{lL})\ \frac{\Gamma \vdash A, \Delta}{A^l, \Gamma \vdash \Delta},\ (\mathrm{lR})\ \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \Delta, A^l},$$

$$(\otimes\mathrm{L})\ \frac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, A \otimes B, \Gamma_2 \vdash \Delta},\ (\otimes\mathrm{R})\ \frac{\Gamma_1 \vdash \Delta_1, A;\ \Gamma_2 \vdash B, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, A \otimes B, \Delta_2},$$

$$(\oplus\mathrm{L})\ \frac{\Gamma_1, A \vdash \Delta_1;\ B, \Gamma_2 \vdash \Delta_2}{\Gamma_1, A \oplus B, \Gamma_2 \vdash \Delta_1, \Delta_2},\ (\oplus\mathrm{R})\ \frac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, A \oplus B, \Delta_2},$$

$$(1\mathrm{L})\ \frac{\Gamma_1, \Gamma_2 \vdash \Delta}{\Gamma_1, 1, \Gamma_2 \vdash \Delta},\ (0\mathrm{R})\ \frac{\Gamma \vdash \Delta_1, \Delta_2}{\Gamma \vdash \Delta_1, 0, \Delta_2}.$$

Rules for $\wedge, \vee$ are similar to those in FL, adapted to classical sequents. For instance, ($\vee$L) admits the form:

$$(\vee\mathrm{L})\ \frac{\Gamma_1, A, \Gamma_2 \vdash \Delta;\ \Gamma_1, B, \Gamma_2 \vdash \Delta}{\Gamma_1, A \vee B, \Gamma_2 \vdash \Delta}.$$

We prove $A \otimes (A \to B) \vdash B$ in NcLL. $A \vdash A$ and $B \vdash B$ are axioms (Id). Then, $A, A^r \vdash$, by (rL), whence $A, A^r \oplus B \vdash B$, by ($\oplus$L). Finally, $A \otimes (A^r \oplus B) \vdash B$, by ($\otimes$L).

A sequent $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$ is deductively equivalent to the simple sequent:

$$A_1 \otimes \cdots \otimes A_m \vdash B_1 \oplus \cdots \oplus B_n.$$

So, comma in the antecedent is interpreted as product, while comma in the succedent is interpreted as par. Dualities are expressed by several deductive equivalences of formulae, e.g. $(A \otimes B)^r$ with $B^r \oplus A^r$, $(A \otimes B)^l$ with $B^l \oplus A^l$, A^{rl} and A^{lr} with A. One refers to $\otimes$ (resp. $\oplus$) as the *multiplicative* conjunction (resp. disjunction), while to $\wedge$ (resp. $\vee$) as the *additive* conjuction (resp. disjunction).

NcLL admits cut-elimination [1]. Actually, for proof-theoretic results it is more expedient to use a one-side sequent system whose sequents are of the form $\vdash \Delta$ (classical systems with negations can be presented in this form). Then, the rule (CUT) splits in two rules:

$$(\mathrm{CUTl})\ \frac{\Delta_1, A^l;\ A, \Delta_2}{\Delta_1, \Delta_2},\ (\mathrm{CUTr})\ \frac{\Delta_1, A;\ A^r, \Delta_2}{\Delta_1, \Delta_2}.$$

Using (CUTr), (CUTl), one easily shows that rules ($\to$L), ($\to$R), ($\leftarrow$L), ($\leftarrow$R) are admissible in NcLL, and consequently, NcLL is a decidable extension of FL. NcLL is a conservative extension of L*. Systems L*, FL can be treated as intuitionistic fragments of NcLL. Yetter [95] proposed *Cyclic Noncommutative Linear Logic* (CLL) in which two negations collapse to one, and $\vdash \Gamma, \Delta$ is provable iff $\vdash \Delta, \Gamma$ is so. CLL admits cut-elimination, is decidable and conservatively extends L*.

Linear Logic (LL) of Girard [42] can be defined as NcLL with one additional (structural) rule of Exchange:

$$\text{(C-EXC)}\ \frac{\Delta_1, A, B, \Delta_2}{\Delta_1, B, A, \Delta_2}.$$

This is Exchange for classical one-side sequent systems. For double-side sequent systems one should add the exchange of A and B in the antecedent. For intuitionistic systems, the rule Exchange admits the form:

$$\text{(EXC)}\ \frac{\Gamma_1, A, B, \Gamma_2 \vdash C}{\Gamma_1, B, A, \Gamma_2 \vdash C}.$$

For systems with Exchange, Γ, Δ in sequents can be treated as multisets, i.e. sets with possible repetitions of elements (their order is not essential). Inference rules admit simplified forms: all designated formulae can be put on the right (or on the left). In LL A^r and A^l are deductively equivalent, so there is, in fact, one (linear) negation, usually denoted by $^\perp$. Similarly, two conditionals collapse to one (linear) implication. The latter also holds for intuitionistic systems, e.g. L, L*, FL with (EXC); following Ono [73], we denote these systems by L_e, L^*_e, FL^*_e.

The relation of noncommutative and commutative Linear Logics to language structures is not direct. As a matter of fact, there are no natural operations on languages (or: on binary relations) which interpret $\oplus, ^r, ^l$, and even $\otimes$, preserving laws provable in NcLL or LL.

Algebraic models for NcLL are bilinear algebras; the term is due to Lambek [61]. *A bilinear algebra* is a structure $\mathcal{M} = (M, \leq, \cdot, \rightarrow, \leftarrow, 1, 0)$ such that $(M, \leq, \cdot, \rightarrow, \leftarrow, 1)$ is a residuated monoid, and $0 \in M$ satisfies the following equations:

$$\text{(BL0)}\ a = (0 \leftarrow a) \rightarrow 0;\ a = 0 \leftarrow (a \rightarrow 0),$$

for any $a \in M$. Notice that $a \leq (0 \leftarrow a) \rightarrow 0$, $a \leq 0 \leftarrow (a \rightarrow 0)$ hold in all residuated semigroups (remind type-raising laws (L1)). 0 is called *a dualizing element*. One defines $a^r = a \rightarrow 0$, $a^l = 0 \leftarrow a$. One shows $(b^l a^l)^r = (b^r a^r)^l$, for all $a, b \in M$, and this element is denoted by $a \oplus b$.

The multiplicative fragment of NcLL is complete with respect to bilinear algebras, and NcLL is complete with respect to *bilinear lattices*, i.e. bilinear algebras which are lattices with operations $\wedge, \vee$ [1]. CLL is complete with respect to *cyclic bilinear lattices*, in which 0 satisifes $a \rightarrow 0 = 0 \leftarrow a$ (then, $a^r = a^l$). LL is complete with respect to *commutative bilinear lattices*, in which $ab = ba$, for all elements a, b.

Phase-space models of Girard [42] are special bilinear lattices, similar to powerset algebras over monoids. *A phase-space model* (for LL) is a structure $\mathcal{M} = (M, \cdot, 1, 0)$ such that $(M, \cdot, 1)$ is a commutative monoid, and $0 \subseteq M$. A set $X \subseteq M$ is called *a fact*, if $X = Y \rightarrow 0$, for some $Y \subseteq M$ (the operation $\rightarrow$ on subsets of M is defined as for powerset algebras). Let $F(\mathcal{M})$ denote the family of all facts (in $\mathcal{M}$). $F(\mathcal{M})$ is a commutative bilinear lattice with operations and

designated elements defined as follows:

$$X^{\perp} = X \to 0;\ X \otimes Y = (X \cdot Y)^{\perp\perp};\ X \oplus Y = (X^{\perp} \cdot Y^{\perp})^{\perp};$$

$$X \wedge Y = X \cap Y;\ X \vee Y = (X^{\perp} \cap Y^{\perp})^{\perp};$$

$$\mathbf{0} = 0;\ \mathbf{1} = 0^{\perp};\ \top = M;\ \perp = \top^{\perp}.$$

One shows that $F(\mathcal{M})$ is a closure system, this means, it is closed under arbitrary meets. LL is complete with respect to phase-space models. Phase-space models for CLL are similar to those for LL except that the monoid need not be commutative, and 0 satisfies: $ab \in 0$ iff $ba \in 0$. Again, the completeness theorem holds [95]. Phase-space models for NcLL involve a more sophisticated condition for 0 [1].

Accordingly, operations in phase-space models are defined, using a kind of double negation, and none of them can naturally be applied in algebras of languages (except $\wedge$, but it is restricted to facts, and facts are very special languages). On the other hand, NcLL and CLL are conservative extensions of L*, so the former can be used as provers of theorems of the latter. As shown in [23], the operations in phase-space models can be defined by means of $\to$, $\wedge$, and 0 only, whence they can be expressed in the product-free L* with a designated constant 0 and either (EXC), for the case of LL-models, or a special rule:

$$\text{(C0)}\ \frac{\Gamma, \Delta \vdash 0}{\Delta, \Gamma \vdash 0},$$

for the case of cyclic phase-space models. This yields several interesting consequences concerning both model-theoretic and computational aspects of type logics. The system L^{*}_{e} is identical with the logic BCI, studied in the world of substructural logics, connected with the lambda calculus (B,C,I are symbols for certain combinators, i.e. closed lambda-terms). Since BCI possesses finite model property (FMP) [17], then LL possesses FMP [23]; FMP for LL has independently been proved by Lafont [58]. Similarly, since L* with (C0) possesses FMP, then CLL possesses FMP. Since the multiplicative LL is NP-complete [65], then BCI is NP-complete, and this also holds for its single-variable fragment. Since the multiplicative CLL is NP-complete [79], then the product-free L* with (C0) is NP-complete (this also holds for its $\to$ −fragment). Detailed proofs will be given in a forthcoming paper.

Lambek [62] introduces *Compact Bilinear Logic* (CBL) which strengthens and, at the same time, simplifies the multiplicative NcLL. One identifies $\otimes$ with $\oplus$, and 1 with 0. Algebraic models of CBL are pregroups. *A pregroup* is a structure $\mathcal{M} = (M, \leq, \cdot, ^{r}, ^{l}, 1)$ such that $(M, \leq, \cdot, 1)$ is a partially ordered monoid (i.e. a monoid with a partial ordering $\leq$, satisfying (MON1)), and ^{r},^{l} are unary operations on M, fulfilling the conditions:

$$\text{(PRE)}\ aa^{r} \leq 1 \leq a^{r}a;\ a^{l}a \leq 1 \leq aa^{l},$$

for all $a \in M$. The element a^{r} (resp. a^{l}) is called the *right* (resp. *left*) *adjoint* of a. The class of pregroups extends the class of partially ordered groups;

in the latter $a^r = a^l = a^{-1}$. Every commutative pregroup is a group; then, linguistically interesting pregroups must be noncommutative. In a pregroup, one defines $a \to b = a^r b$, $b \leftarrow a = ba^l$ and obtains a residuated monoid. Consequently, all sequents provable in L* are provable in CBL (modulo this translation), but not conversely (the converses of Grishin laws (L6) are provable in CBL, but not in L* [62]).

In pregroups, $(ab)^r = b^r a^r$, $(ab)^l = b^l a^l$, $a^{rl} = a = a^{lr}$ and $1^r = 1^l = 1$. Consequently, any formula is equivalent to a product $A_1 \cdot \cdots \cdot A_k$ such that $k \geq 0$ and each A_i is a variable with some number of r's or l's. It is expedient to use the following notation: $A^{(0)}$ equals A, $A^{(n)}$ equals A with n r's, if $n > 0$, and $A^{(n)}$ is A with n l's, if $n < 0$. CBL, restricted to formulae in which adjoints apply to variables only, can be presented as a rewriting system with the following rules of Contraction (CON) and Expansion (EXP):

$$\text{(CON) } \Gamma, p^{(n)}, p^{(n+1)}, \Delta \Rightarrow \Gamma, \Delta,$$

$$\text{(EXP) }) \ \Gamma, \Delta \Rightarrow \Gamma, p^{(n+1)}, p^{(n)}, \Delta.$$

Lambek [62] admits additional assumptions of the form $p \Rightarrow q$. Precisely, for a fixed finite poset $(P, \leq)$, he treats members of P as individual constants and considers an extension of CBL which assumes all arrows true in this poset. Then, he adds a new rewriting rule, called Inductive Step (IND):

$$\text{(IND) } \Gamma, p^{(n)}, \Delta \Rightarrow \Gamma, q^{(n)}, \Delta,$$

where $p \leq q$ if n is even, and $q \leq p$ if n is odd.

Lambek [62] proves an important normalization theorem ('switching lemma') for CBL: if $\Gamma \Rightarrow \Delta$ in CBL, then there exists Φ such that $\Gamma \Rightarrow \Phi$, by (CON) and (IND) only, and $\Phi \Rightarrow \Delta$, by (EXP) and (IND) only. Consequently, if a sequent $\Gamma \Rightarrow A$ is provable, and A does not contain product, then Γ reduces to A by (CON) and (IND) only. As shown in [24], this theorem implies PTIME-decidability of CBL and is equivalent to the cut-elimination theorem for some sequent systems for CBL.

3 Categorial grammars

3.1 Classical categorial grammars

A classical categorial grammar (CCG) can be defined as a quadruple $G = (\Sigma_G, I_G, S_G, \text{AB})$; Σ_G is a nonempty finite lexicon (alphabet), I_G is a finite relation between elements of Σ_G and types, and S_G is a designated variable. AB denotes the basic type logic of Ajdukiewicz and Bar-Hillel; so, types are formulae of this logic. Σ_G, I_G, S_G are called *the lexicon*, *the initial type assignment* and *the principal type*, respectively, of G. We write $G : v \mapsto A$ for $(v, A) \in I_G$. We say that G *assigns* type A to string $a = v_1 \dots v_n$, $v_i \in \Sigma_G$, if there are types A_i such that $G : v_i \mapsto A_i$, for $i = 1, \dots, n$, satisfying $A_1 \dots A_n \vdash_{AB} A$; we

write $a \mapsto_G A$. *The language* of G ($L(G)$) is the set of all strings a such that $a \mapsto_G S_G$. Other type grammars, considered later on, are defined in a similar way; only AB has to be replaced by another type logic.

The fa-structures on Σ_G are defined as fa-structures of types except that atomic fa-structures are elements of Σ_G. By $F(\Sigma_G)$ we denote the set of fa-structures on Σ_G. We say that G assigns type A to structure $X \in F(\Sigma_G)$ if there is an fa-structure X' of types which arises from X by replacing each atom v by a type B such that $G : v \mapsto B$, satisfying $X' \vdash_{ABs} A$; we again write $X \mapsto_G A$. *The f-language* of G ($L^f(G)$) is the set of all $X \in F(\Sigma_G)$ such that $X \mapsto_G S_G$.

For instance, let G admit the following assignment:

$$\text{Kazimierz} \mapsto A, \text{ teaches} \mapsto B, \text{ excellently} \mapsto C,$$

where $A =$PN (proper noun), $B = A \rightarrow$S, $C = B \rightarrow B$, and $S_G =$S (sentence). Then, $L^f(G)$ contains infinitely many structures, e.g.:

1. (Kazimierz teaches)$_2$,
2. (Kazimierz (teaches excellently)$_2$)$_2$,
3. (Kazimierz ((teaches excellently)$_2$ excellently)$_2$)$_2$,

and so on. $L(G)$ contains the strings of words resulting from the above structures.

The above grammar is a 1-valued (rigid) CCG, i.e. it assigns at most one type to any lexical atom. This property is characteristic of grammars designed for formal languages of logic and mathematics, while natural language usually requires several types to be assigned to one lexical atom. The negation 'not' is assigned type S←S, as in the sentence 'not every man works' with structure (not ((every man) works)), but also type $C \leftarrow C$ (C defined as above), as in 'John works not hardly'.

Phrase structures are fa-structures without functor indices. For $X \in F(\Sigma_G)$, by $p(X)$ we denote the phrase structure resulting from X after one has dropped all functor indices. *The phrase language* of G ($L^p(G)$) consists of all $p(X)$, for $X \in L^f(G)$.

CCG's are closely related to Chomsky's phrase-structure grammars. *A context-free grammar* (CF-grammar) is a quadruple $G = (\Sigma_G, N_G, R_G, S_G)$ such that Σ_G, N_G are disjoint finite alphabets, $S_G \in N_G$, and R_G is a finite subset of $N_G \times (\Sigma_G \cup N_G)^\star$. Elements of Σ_G, N_G, R_G are called *terminal symbols*, *nonterminal symbols* and *production rules*, respectively, of G, and S_G is called *the initial symbol* of G. Production rules are written $A \mapsto a$ instead of (A, a). We say that string b is *directly derivable* from string a in G (write $a \Rightarrow_G b$) if there exist strings c, d, e and rule $A \mapsto e$ in R_G such that $a = cAd$, $b = ced$. We say that string b is *derivable* from string a in G (write $a \Rightarrow^\star_G b$) if there exists a sequence $(a_0, \ldots, a_n)$ such that $n \geq 0$, $a_0 = a$, $a_n = b$ and $a_{i-1} \Rightarrow_G a_i$, for all $i = 1, \ldots, n$. *The language* of G ($L(G)$) is the set of all $a \in \Sigma^\star_G$ such that $S_G \Rightarrow^\star_G a$.

Grammars G, G' are said to be (weakly) *equivalent* if $L(G) = L(G')$. It is well-known that every CF-grammar G such that $\epsilon \notin L(G)$ can be transformed into an equivalent CF-grammar G' in *the Chomsky normal form*: all production rules of G' are of the form $A \mapsto v$ or $A \mapsto BC$, where $A, B, C \in N_{G'}$, $v \in V_{G'}$.

The CCG from the above example can be replaced with a CF-grammar (in the Chomsky normal form) whose production rules are as follows:

$$\mathrm{S} \mapsto AB,\ B \mapsto BC,$$

$$A \mapsto \text{Kazimierz},\ B \mapsto \text{teaches},\ C \mapsto \text{excellently},$$

with $S_G = \mathrm{S}$, $N_G = \{S, A, B, C\}$, and Σ_G consisting of lexical atoms 'Kazimierz', 'teaches' and 'excellently'. A derivation of 'Kazimierz teaches excellently' is:

$$\mathrm{S} \Rightarrow AB \Rightarrow ABC \Rightarrow^* \text{Kazimierz teaches excellently}.$$

Every CF-derivation determines a unique phrase structure on the derived string; the above derivation leads to structure (Kazimierz (teaches excellently)). *The phrase language* of CF-grammar G ($L^p(G)$) consists of all phrase structures on strings from $L(G)$ which are determined by possible derivations of these strings. One could also save nonterminals appearing in the derivation; the resulting labelled phrase structure (Kazimierz ${}_A$(teaches ${}_B$excellently$_C$)$_B$)$_S$ is a linear representation of a derivation tree.

A principal difference between CCG and CF-grammar is that the former is *lexical*, that means: all particular linguistic information is put in the initial assignment of types to lexical atoms, and the derivation procedure is based on universal rules, common for all languages, whereas the latter puts the linguistic information in the production rules which underly the derivation procedure. Lexicality is characteristic of all basic kinds of type grammar: the universal rules for derivation procedures are provable sequents of some logics, being independent of the particular language.

The Gaifman theorem [5] establishes the (weak) equivalence of CCG's and CF-grammars (for $\epsilon-$free languages). It is easy to show that every CCG is equivalent to some CF-grammar. Let G be a CCG, and let T_G denote the set of all types appearing in I_G. By $T(G)$ we denote the set of all subtypes of types from T_G. Clearly, $T(G)$ is finite and contains all types assigned by G to any strings. A CF-grammar G' is defined by: $V_{G'} = \Sigma_G$, $N_{G'} = T(G)$, $S_{G'} = S_G$, and $R_{G'}$ consists of all rules:

$$B \mapsto A(A \to B),\ B \mapsto (B \leftarrow A)A,$$

for $(A \to B), (B \leftarrow A) \in T(G)$, and all lexical rules $A \mapsto v$, for $(v, A) \in I_G$. One easily proves:

$$A_1 \dots A_n \vdash_{AB} A \text{ iff } A \Rightarrow^\star_{G'} A_1 \dots A_n,$$

for all $A_i, A \in T(G)$, and consequently, $L(G) = L(G')$.

The converse direction is more sophisticated. It is easier if one assumes that the CF-grammar G is in *the Greibach normal form*, that means: all production

rules are of the form $A \mapsto v$, $A \mapsto vB$ or $A \mapsto vBC$, for $A, B, C \in N_G$, $v \in \Sigma_G$ (every CF-grammar G such that $\epsilon \notin L(G)$ is equivalent to a CF-grammar in the Greibach normal form). We identify nonterminal symbols of G with primitive types. A CCG G' is defined by: $V_{G'} = \Sigma_G$, $S_{G'} = S_G$, and $I_{G'}$ consists of:

1. all pairs (v, A) such that $(A \mapsto v) \in R_G$,
2. all pairs $(v, A \leftarrow B)$ such that $(A \mapsto vB) \in R_G$,
3. all pairs $(v, (A \leftarrow C) \leftarrow B)$ such that $(A \mapsto vBC) \in R_G$.

By induction on n, one proves:

$$A \Rightarrow_G^\star v_1 \dots v_n \text{ iff } v_1 \dots v_n \mapsto_{G'} A,$$

for all $A \in N_G$, $v_i \in \Sigma_G$, which yields $L(G) = L(G')$.

CCG's are not equivalent to CF-grammars on the level of phrase structures. Let $P(\Sigma)$ denote the set of all phrase structures on alphabet Σ. Let $L \subseteq P(\Sigma)$. We say that $X \in P(\Sigma)$ is equivalent to $Y \in P(\Sigma)$ with respect to L (write $X \sim_L Y$) if, for all $Z \in P(\Sigma)$, $Z[X] \in L$ iff $Z[Y] \in L$ (as usual in logic, $Z[X]$ denotes Z with a distinguished occurrence of substructure X, and $Z[Y]$ denotes the result of replacing X with Y in Z). By the classical theorem of Thatcher, $L = L^p(G)$, for some CF-grammar G, iff the relation $\sim_L$ is of finite index. By *the external degree* of $X \in P(\Sigma)$ (think of X as a tree) we mean the length of the shortest branch of X, and *the degree* of X is the maximal external degree of substructures of X. For instance, v is of degree 0, (vw) is of degree 1, and $((vw)(v'w'))$ is of degree 2. It has been shown in [13] that $L = L^p(G)$, for some CCG G, iff both $\sim_L$ is of finite index and all structures in L are of bounded degree. Accordingly, phrase languages of CCG's are a narrower class than those of CF-grammars. For instance, the CF-grammar given by rules S$\mapsto$SS, S$\mapsto v$ produces all possible phrase structures on $\{v\}$, hence its phrase language is of unbounded degree, and consequently, it cannot be generated by any CCG.

Phrase languages of CF-grammars and CCG's are regular tree languages in the sense of tree automata (see [40]); this follows from the above characterization of them as languages of finite index. Tiede [86] obtains analogous results for ND proof trees associated with CCG's and other kinds of categorial grammars. For grammars based on the nonassociative Lambek calculus, some related results have been proven by Kandulski [54].

A similar characterization can be given for functor languages generated by CCG's: for $L \subseteq F(\Sigma)$, there is a CCG G such that $L = L^f(G)$ iff both the relation $\sim_L$ is of finite index and all structures in L are of bounded functor degree (one counts the length of functor branches only; see [15, 20]). Consequently, functor languages of CCGs are regular tree languages. Standard techniques of tree automata [40] yield the decidability of the emptiness problem, the inclusion problem and the equality problem for these languages; for a discussion, see [15]. In particular, the inclusion problem $L^f(G) \subseteq L^f(G')$ is decidable, which is crucial for some learnability results in the next subsection.

3.2 Learning grammars

Formal learning theory is understood here in the sense of symbolic learning, following the Gold paradigm of learning from positive data and identification of language in the limit [41, 74]. In a broader perspective, it is a branch of inductive logic, since one searches for a general knowledge (a grammar for the whole language) on the basis of some finite data (some finite set of expressions from the language).

In [14, 29], the method of unification, known in computational logic, e.g. logic programming, was applied to learning procedures for CCG's. Kanazawa [49, 50] used these procedures to construct convergent learning functions for several classes of CCG's. Negative postulates were considered in [29, 66, 34, 35]. Unification in categorial grammar was also studied in e.g. [90, 88]. Several authors examined learnability of other kinds of categorial grammars; see [37, 7, 70] with both negative and positive results (see below). We show here that the very method can be extended to other grammar formalisms, e.g. context-free grammars and context-sensitive grammars.

We recall some basic notions of formal learning theory.

A grammar system is a triple (Ω, E, L) such that Ω and E are countably infinite sets, and L is a function from Ω into the powerset of E. Elements of Ω are called *grammars*, elements of E are called *expressions*, and $L(G)$, for $G \in \Omega$, is called *the language* of G.

A learning function for the grammar system is a partial function from E^+ into Ω. Intuitively, the learning function assigns grammars to finite samples of a language. Let $(s_i)_{i\in\omega}$ be an infinite sequence of expressions. One says that a learning function φ *converges* on this sequence to $G \in \Omega$ if $\varphi((s_i)_{i<n})$ is defined and equals G, for all but finitely many $n \in \omega$.

Let $\mathcal{G} \subseteq \Omega$. We denote $L(\mathcal{G}) = \{L(G) : G \in \mathcal{G}\}$. A sequence $(s_i)_{i\in\omega}$ is called *a text* for a language $L \subseteq E$ if $L = \{s_i : i \in \omega\}$. One says that a learning function φ *learns* $\mathcal{G}$ if, for every $L \in L(\mathcal{G})$ and every text for L, there is $G \in \mathcal{G}$ such that $L = L(G)$ and φ converges to G on this text. A class $\mathcal{G}$ is said to be (effectively) *learnable* if there exists a computable learning function φ that learns $\mathcal{G}$.

Let $\mathcal{L}$ be a class of subsets of E (languages on E). One says that $\mathcal{L}$ *admits a limit point* if there exists a stricly ascending chain $(L_n)_{n\in\omega}$ of languages from $\mathcal{L}$ such that the join of this chain belongs to $\mathcal{L}$. It is known that if $L(\mathcal{G})$ admits a limit point then $\mathcal{G}$ is not (even uneffectively) learnable. As a consequence, if $\mathcal{G}$ generates all finite languages and at least one infinite language, then $\mathcal{G}$ is not learnable. This holds for all standard classes of formal grammars, e.g. regular grammars, CF-grammars, CCG's and so on.

Accordingly, learnability can be gained for some restricted classes only, as e.g. context-sensitive grammars with at most k production rules [85]. Kanazawa [49, 50] shows that rigid CCG's and $k-$valued CCG's are learnable.

Wright [94] has defined the following property of a class $\mathcal{L}$ of languages: $\mathcal{L}$ has *finite elasticity* if there exists no pair $((s_i)_{i\in\omega}, (L_i)_{i\in\omega})$, $s_i \in E$, $L_i \in \mathcal{L}$, such that $s_i \notin L_i$ but $s_0, \ldots, s_i \in L_{i+1}$, for all $i \in \omega$. Kapur [55] has proven

that finite elasticity entails the following condition:

(D) for every $L \in \mathcal{L}$, there exists a finite set $D_L \subseteq L$ such that L is the smallest language $L' \in \mathcal{L}$, satisfying $D_L \subseteq L'$.

In [25], it is shown that finite elasticity is equivalent to the following:

(D') for every $L \subseteq E$, there exists a finite set $D_L \subseteq L$ such that, for every $L' \in \mathcal{L}$, if $D_L \subseteq L'$ then $L \subseteq L'$.

This yields Kapur's theorem, since (D) follows from (D'). If $\mathcal{L}$ is a closure system, that means: $\mathcal{L}$ is closed under arbitrary meets, then $\mathcal{L}$ satisfies (D') iff it admits no infinite ascending chains, that means: there is no infinite sequence (L_i), of languages from $\mathcal{L}$, such that $L_i \subset L_{i+1}$, for all $i \in \omega$ [25].

Let $\mathcal{G} \subseteq \Omega$. We always assume that $\mathcal{G}$ is recursively enumerable and the universal membership problem $s \in L(G)$, for $s \in E$, $G \in \mathcal{G}$, is decidable. If $\mathcal{G}$ is recursively enumerable, then there exists a computable sequence $(G_n)_{n\in\omega}$, of all grammars from $\mathcal{G}$.

If $\mathcal{G}$ satisfies the above assumption and $L(\mathcal{G})$ satisfies (D), then $\mathcal{G}$ is learnable [55]. We sketch the proof. We consider two cases.

Case 1. The inclusion problem $L(G_1) \subseteq L(G_2)$, for $G_1, G_2 \in \mathcal{G}$, is decidable (this holds for structure languages of CCG's and CF-grammars). Let $(G_n)_{n\in\omega}$ be a computable sequence of all grammars from $\mathcal{G}$. The learning function φ is defined as follows. Let $(s_i)_{i\leq n}$ be a finite sequence of expressions from E. One considers grammars $G_0, \ldots, G_n$. One marks those grammars G_j, $j = 0, \ldots, n$, which satisfy $s_0, \ldots, s_n \in L(G_j)$ (if no G_j fulfilling this condition exists, then φ is undefined). Then, one takes the first marked grammar G_j whose language is minimal (with respect to inclusion) among languages of marked grammars (this means, there exists no marked grammar G_k such that $L(G_k) \subset L(G_j)$), and this grammar is the value of φ for the given sequence. Now, let (s_n) be a text for some language $L \in L(\mathcal{G})$. By (D), there exists a finite set $D_L \subseteq L$ such that L is the smallest language from $L(\mathcal{G})$ which contains D_L. There is $n' \in \omega$ such that $D_L \subseteq \{s_0, \ldots, s_m\}$, for all $m \geq n'$. Let G_i be the first grammar in the sequence (G_n) such that $L(G_n) = L$. Clearly, for each $m \geq \max(i, n')$, G_i will be taken as the value of φ for $(s_i)_{i\leq m}$.

Case 2. The inclusion problem is undecidable (this holds for string languages of CCG's and CF-grammars). The construction is similar to the one for Case 1 except that we additionally employ a fixed sequence $(e_n)_{n\in\omega}$ of all expressions from E and define $E_n = \{e_0, \ldots, e_n\}$. The value of φ for $(s_i)_{i\leq n}$ is defined as the first marked grammar G_j such that $L(G_j) \cap E_n$ is minimal among languages $L(G_k) \cap E_n$, for marked grammars G_k. Now, G_i, defined in Case 1, need not be the value of φ on $(s_i)_{i\leq m}$, for all $m \geq n'$; it may happen that $\varphi((s_i)_{i\leq m}) = G_l$, for some $l < i$ such that $L(G_l) \cap E_m = L(G_i) \cap E_m$. This may, however, happen, for finitely many m only, since L is the smallest language from $L(\mathcal{G})$ containing D_L. Accordingly, φ converges to G_i.

We have only regarded positive data. Many learning procedures take into account positive and negative data. *A pn-text* for a language L can be defined

as a pair $((e_n, d_n)_{n\in\omega}, d)$ such that $(e_n)_{n\in\omega}$ is a sequence of all expressions from E, and $d_n = 1$ if $e_n \in L$, $d_n = 0$ if $e_n \notin L$.

It was observed by Gold [41] that any class of grammars (satisfying the general assumptions above) is learnable from positive and negative data. Let $(G_n)_{n\in\omega}$ be a computable sequence of all grammars from $\mathcal{G}$. φ is defined on $(s_i, d_i)_{i\leq n}$ as the first grammar G_j in the sequence which is compatible with the data. One easily shows that, for any pn-text for $L \in L(\mathcal{G})$, φ converges to a grammar for L.

Many results in formal learning theory were obtained by analogous methods; the main technical issue was to prove finite elasticity of a class of languages. Algorithms based on the search for the first grammar in the sequence of all grammars which is compatible with the data have some drawbacks. To explain the drawbacks, we use some notions of knowledge representation theory, and precisely, rough set theory of Pawlak [76].

In rough set theory, a set X of objects is represented by a pair of two sets: the lower approximation of X ($\mathrm{LA}(X)$) and the upper approximation of X ($\mathrm{UA}(X)$). The lower approximation of X consists of those objects which necessarily belong to X on the basis of our knowledge, and the upper approximation of X consists of those objects which possibly belong to X on the basis of our knowledge. These notions were usually applied to information systems (relational databases). We show that they are useful in formal learning theory.

Let $\mathcal{G}$ be a class of grammars. For a language $L \in L(\mathcal{G})$ and a set $D \subseteq L$, *the lower approximation* and *the upper approximation* of L *determined by* D (with respect to $\mathcal{G}$) are defined as follows:

$$\mathrm{LA}^{\mathcal{G}}(L, D) = \bigcap\{L' \in L(\mathcal{G}) : D \subseteq L'\};\ \mathrm{UA}^{\mathcal{G}}(L, D) = \bigcup\{L' \in L(\mathcal{G}) : D \subseteq L'\}.$$

These definition are very natural. If we only know that expressions from D belong to L, and $L \in L(\mathcal{G})$, then precisely the expressions from $\mathrm{LA}^{\mathcal{G}}(L, D)$ necessarily belong to L on the basis of our knowledge, and precisely the expressions from $\mathrm{UA}^{\mathcal{G}}(L, D)$ possibly belong to L on the basis of our knowledge.

The above algorithms compute neither $\mathrm{LA}^{\mathcal{G}}(L, D)$, nor $\mathrm{UA}^{\mathcal{G}}(L, D)$, if D is the set of input data. If G_i is the first grammar compatible with D, then $\mathrm{LA}^{\mathcal{G}}(L, D) \subseteq L \subseteq \mathrm{UA}^{\mathcal{G}}(L, D)$, but these inclusions do not justify any new positive or negative inference about the extent of L which is not transparent in the input data. Of course, when the algorithm finds the proper grammar for L, the grammar produces the lower approximation of L determined by the input data. The problem is that the learner usually does not know that the proper grammar has just been found; she cannot exclude that larger data will enforce the change of the hypothesis. Only for very special cases, e.g. if the determined language L has been proved to be maximal in $L(\mathcal{G})$, one can conclude that the searching procedure is ultimately finished.

We show below that algorithms based on unification, proposed in [14, 29] and further elaborated by Kanazawa [49, 50], Marciniec [66] and others, do compute the lower and the upper approximation.

First, we demonstrate these algorithms for CCG (learning from structures).

Let us consider the grammar system (Ω, E, L) such that Ω is the class of CCG's (with a fixed lexicon Σ), $E = F(\Sigma)$, and $L(G) = L^f(G)$, for $G \in \Omega$. We assume that all grammars in Ω have the same principal type S. Since Σ and S are fixed, a grammar G can be identified with I_G.

First, we recall a unification-based learning procedure from [29]. Let $D \subseteq F(\Sigma)$ be a nonempty, finite set. We define a CCG $\mathrm{GF}(D)$, called *the general form of a grammar* for D. S is treated as a constant, whereas other atomic types are treated as variables. We assign S to all structures from D and distinct variables to all occurrences of argument substructures of these structures. Then, we assign types to functor substructures of these structures according to the rules:

(fr) if $(XY)_2$ is assigned B and X is assigned A then Y is assigned $A \to B$,

(fl) if $(XY)_1$ is assigned B and Y is assigned A then X is assigned $B \leftarrow A$.

$\mathrm{GF}(D)$ contains all assignments $v \mapsto A$ obtained in this way, for $v \in \Sigma$; the principal type of $\mathrm{GF}(D)$ is S. For instance, if D consists of structures (Joan works)$_2$ and (Joan (works hardly)$_2$)$_2$, then $\mathrm{GF}(D)$ contains the assignments:

$$\text{Joan} \mapsto x, y,\ \text{works} \mapsto z,\ x \to \mathrm{S},\ \text{hardly} \mapsto z \to (y \to \mathrm{S}).$$

A substitution is a mapping from variables to types; it is naturally extended to a mapping from types to types. If G is a CCG, and σ is a substitution, then $G\sigma$ is a (unique) CCG which assigns $v \mapsto A\sigma$, whenever $G : v \mapsto A$. We have $L^f(G) \subseteq L^f(G\sigma)$, for all G, σ. For $G, G' \in \Omega$, we write $G \subseteq G'$ if $I_G \subseteq I_{G'}$. The following lemma is crucial [29].

(T1) Let $D \subseteq F(V)$ be nonempty and finite. Then, for every $G \in \Omega$, $D \subseteq L^f(G)$ iff there exists σ such that $GF(D)\sigma \subseteq G$.

The following notions come from the logical theory of unification; we modify definitions from [29]. A substitution σ is called *a unifier* of $G \in \Omega$ if, for all $v \in \Sigma$ and types A, B such that $G : v \mapsto A$ and $G : v \mapsto B$, there holds $A\sigma = B\sigma$. It is called *a most general unifier* (m.g.u.) of G if it is a unifier of G and, for every unifier η of G, there exists a substitution γ such that $\eta = \sigma\gamma$. G is said to be *unifiable* if there exists a unifier of G. The standard algorithm of unification can be used to decide whether a given CCG G is unifiable and, if so, to produce an m.g.u. of G (it is unique up to variants). For rigid CCG's, and any nonempty, finite $D \subseteq F(V)$, there holds the following theorem [29].

(T2) There exists a rigid $G \in \Omega$ such that $D \subseteq L^f(G)$ iff $\mathrm{GF}(D)$ is unifiable. If σ is an m.g.u. of $\mathrm{GF}(D)$, then $\mathrm{GF}(D)\sigma$ is a rigid CCG from Ω whose functor language is the smallest language $L(G)$ such that $G \in \Omega$ is rigid and $D \subseteq L(G)$.

$\mathrm{GF}(D)$ from the above example is unifiable; an m.g.u. σ is the substitution: y/x, $z/(x \to \mathrm{S})$; this means: $\sigma(y) = x$, $\sigma(z) = x \to$S. Hence $\mathrm{GF}(D)\sigma$ is given by:

$$\text{Joan} \mapsto x,\ \text{works} \mapsto x \to \mathrm{S},\ \text{hardly} \mapsto (x \to \mathrm{S}) \to (x \to \mathrm{S}).$$

This agrees with a standard analysis of these sentences: x is proper noun, $x \mapsto$S is the type of verb phrase, and the last type is a type of adverb.

Let us define RG(D) =GF(D)σ, if GF(D) is unifiable and σ is an m.g.u. of GF(D). It follows from (T2) that RG(D) is a 1-valued CCG such that L(RG(D)) is the smallest 1-valued language containing D. Let $\mathcal{G}_1$ denote the class of 1-valued (rigid) CCG's. Kanazawa [49] proves that $L(\mathcal{G}_1)$ has finite elasticity. First, he proves that $L(\mathcal{G}_1)$ admits no infinite ascending chains. Then, it can be shown that $L(\mathcal{G}_1)$ enriched with the total language $F(\Sigma)$ is a closure system (use (T2)); now, apply our earlier results on closure systems [25]. The function $\varphi((s_i)_{i\leq n})$ =RG($\{s_i\}_{i\leq n}$) learns the class $\mathcal{G}_1$ (up to isomorphism of grammars).

This learning algorithm is a special case of a general method, which can be described as follows. Denote $\mathcal{T}_D = \{I_G(v) : v \in \Sigma\}$, where G =GF(D). $\mathcal{T}_D$ is a finite family of finite sets of types. For any family $\mathcal{T} = \{T_1, \ldots, T_n\}$ of that kind, a substitution σ is called a unifier of $\mathcal{T}$, if, for any $i = 1, \ldots, n$ and for all $A, B \in T_i$, $\sigma(A) = \sigma(B)$. So, σ is a unifier of GF(D) in the sense defined above iff σ is a unifier of $\mathcal{T}_D$. The notion of an m.g.u. of $\mathcal{T}$ is defined in the standard way.

A substitution α induces an equivalence relation $\equiv_\alpha$ on types: $A \equiv_\alpha B$ iff $\alpha(A) = \alpha(B)$. Each equivalence relation $\equiv$ on types partitions any set T, of types, in equivalence classes (restricted to T). We define $T/\equiv = \{[A]_\equiv \cap T : A \in T\}$. We also define:

$$\mathcal{T}/\equiv = T_1/\equiv \cup \cdots \cup T_n/\equiv, \text{ for } \mathcal{T} = \{T_1, \ldots, T_n\}.$$

This family is called *the partition* of $\mathcal{T}$ *determined* by $\equiv$.

We assume that $\mathcal{G} \subseteq \Omega$ is a recursive class of CCG's, with the universal membership problem being decidable, which satisfies the following conditions (a generalization of conditions formulated in [66]):

(G1) if $G \subseteq G'$ and $G' \in \mathcal{G}$ then $G \in \mathcal{G}$,

(G2) for any finite set $D \subseteq E$ and any substitution α, if α(GF(D)) $\in \mathcal{G}$ then σ(GF(D)) $\in \mathcal{G}$, for any substutution σ which is an m.g.u. of $\mathcal{T}_D/\equiv_\alpha$.

(G1) and (G2) are satisfied for many natural classes of CCG's. A CCG is said to be $k-$valued, if it assigns at most k types to each atom from Σ. By $\mathcal{G}_k$ we denote the class of $k-$valued CCG's. Obviously, $\mathcal{G}_k$ satisfies (G1). We show that it satisfies (G2). Assume α(GF(D)) $\in \mathcal{G}_k$. α is a unifier of $\mathcal{T}_D/\equiv_\alpha$, whence there exists an m.g.u. σ of $\mathcal{T}_D/\equiv_\alpha$. Clearly, $\mathcal{T}_D/\equiv_\alpha = \mathcal{T}_D/\equiv_\sigma$, for any m.g.u. σ, so σ(GF(D)) $\in \mathcal{G}_k$.

The order of type A ($o(A)$) is defined as follows [15]:

$$o(p) = 0, \text{ for variables } p;\ o(A \to B) = o(B \leftarrow A) = \max(o(B), o(A) + 1).$$

The order of G ($o(G)$) is the maximal order of types in I_G. The class of all CCG's G such that $o(G) \leq n$, for a fixed $n \geq 0$, satisfies (G1),(G2). Classes satisfying (G1),(G2) are closed under meets, so the class of $k-$valued CCG's of a bounded order satisfies (G1),(G2), and similarly for classes of $k-$valued

grammars, fulfilling negative constraints of Marciniec [66], e.g. not generating expressions from a fixed, finite set D'.

Let $\mathcal{G}$ be as above. For any finite set $D \subseteq F(\Sigma)$, we define the set $\mathrm{OUT}^{\mathcal{G}}(D)$ as the set of all CCG's $\sigma(\mathrm{GF}(D))$ such that σ is an m.g.u. of a partition of $\mathcal{T}_D$, and $\sigma(\mathrm{GF}(D)) \in \mathcal{G}$ (since $\mathcal{G}$ is recursive, the latter condition can effectively be verified). Using (T2) and (G1),(G2), we prove:

$$\text{(T3) } \mathrm{LA}^{\mathcal{G}}(L, D) = \bigcap\{L(G) : G \in \mathrm{OUT}^{\mathcal{G}}(D)\},$$

$$\text{(T4) } \mathrm{UA}^{\mathcal{G}}(L, D) = \{X \in F(\Sigma) : \mathrm{OUT}^{\mathcal{G}}(D \cup \{X\}) \neq \emptyset\},$$

for all $L \in L(\mathcal{G})$ and all nonempty, finite $D \subseteq L$.

Consequently, both the lower and the upper approximation of the language $L \in L(\mathcal{G})$ determined by $D = \{s_i\}_{i \leq n}$ can effectively be computed on each step of the learning procedure (notice that $\mathrm{OUT}^{\mathcal{G}}(D)$ is a finite set of grammars up to grammar isomorphism). Structure languages of CCG's are closed under finite meets [15], whence the lower approximation can be presented by means of a CCG, not necessarily belonging to the class $\mathcal{G}$. For instance, the meet of two 2-valued languages need not be a 2-valued language. The upper approximation is a recursive language; it can be presented by means of a Turing machine.

We come to an idea of approximate learning, which seems to be more attractive than the simple paradigm discussed above. The learning function φ computes finite sets of grammars, compatible with the data, and containing grammars generating all minimal languages (from the class) compatible with the data. Then, the meet of all languages generated by the grammars computed on the given step equals the lower approximation of the searched language, determined by the data.

If $L(\mathcal{G})$ satisfies (D), then, from some step n, one of the grammars computed by φ must be a proper grammar for the searched language; this grammar generates the smallest language among languages generated by other computed grammars. So, φ learns $\mathcal{G}$ in the previous sense. It is the case for $\mathcal{G}_k$, $k \geq 1$. Using finite elasticity of $\mathcal{G}_1$, Kanazawa [49, 50] proves that each class $L(\mathcal{G}_k)$ has finite elasticity, and consequently, $\mathcal{G}_k$ is learnable. In a similar way, one can show learnability of different classes restricted by additional constraints [35].

Quite analogous procedures can be designed for string languages. It, however, requires a modification of the construction of $\mathrm{OUT}^{\mathcal{G}}(D)$. If D is a finite set of strings, then we consider all sets $D' \subseteq F(\Sigma)$ arising from D after one has defined some fa-structure on each string from D (this is a finite family of sets of structures). We define $\mathrm{OUT}^{\mathcal{G}}(D)$ as the join of all sets $\mathrm{OUT}^{\mathcal{G}}(D')$, the latter being defined as above. Theorems (T3), (T4) remain true.

Categorial grammars based on NL and L behave differently. For L, all possible structures on a given string can be generated, whence learnability from strings amounts to learnability from structures. In both cases, the corresponding class of languages (even for 1-valued grammars) admits a limit point; for NL, learnability from structures is possible [7]. Fulop [38] studies learnabilty from semantic structures (represented by means of lambda terms).

Unification-based learning is not restricted to CCG's. For context-free grammars, we can use similar constructions. Now, D is a finite set of phrase structures. We assign S to all structures from D and different variables to substructures of these structures. Finally, D gives rise to a finite set of productions, generating all structures from D. For example, if D contains (Joan (works hardly)), then GF(D) will contain productions:

$$\mathrm{S} \mapsto xy,\ x \mapsto \text{Joan},\ y \mapsto zu,\ z \mapsto \text{works},\ u \mapsto \text{hardly}.$$

Substitutions are restricted to variable-for-variable substitutions. Analogues of (T2), (T3), (T4) can be proved. Using this methods, one can show that the class of CF-grammars, admitting at most k productions, is learnable both from structures and from strings. The generalization for context-sensitive grammars is also possible, though less elegant.

3.3 Grammars based on substructural logics

AB is a very poor logic: it is a logic of a purely hypothetical reasoning, based on modus ponens in the form (AB1), (AB2). In this subsection we consider categorial grammars, employing richer substructural logics. Categorial grammars based on a logic $\mathcal{L}$ are defined like CCG's except that AB is replaced by $\mathcal{L}$. In particular, Lambek categorial grammars are based on L or L*, nonassociative Lambek categorial grammars on NL, commutative Lambek categorial grammars on L_e or L_e^* and so on.

Proof-theoretic properties of substructural logics are useful for explaining many basic aspects of categorial grammars. The first of them is *generative capacity*: what are the classes of languages generated by different kinds of categorial grammars? The cut-elimination theorem which holds for all substructural logics, discussed in section 2, can be used to show that every $\epsilon-$free context-free language is generated by some categorial grammar based on $\mathcal{L}$, where $\mathcal{L}$ is any fixed (noncommutative) substructural logic between L and ACTω. By the Gaifman theorem, every $\epsilon-$free CF-grammar is weakly equivalent to a CCG of order not greater than 1. Using cut-elimination, one easily proves that any sequent $\Gamma \vdash p$ such that Γ is a string of types of order not greater than 1, and p is a variable (constant), is provable in $\mathcal{L}$ iff it is provable in AB. Consequently, every $\epsilon-$free CF-grammar is weakly equivalent to a categorial grammar based on $\mathcal{L}$. This remains true for CLL and CBL [79, 22].

Interestingly, the above fact plays an essential role in the proof of the Π_1^0-hardness of ACTω. The total language problem $L(G) = \Sigma^+$ for $\epsilon-$free CF-grammars is reducible to the same problem for CCG's of order not greater than 1. The latter can be replaced by categorial grammars based on FL, which is a conservative subsystem of ACTω. The condition $L(G) = \Sigma^+$ can be expressed by a sequent of ACTω [27].

For several substructural logics $\mathcal{L}$, it has been proved that categorial grammars based on $\mathcal{L}$ generate precisely the $\epsilon-$free context-free languages. One must prove that every categorial grammar based on $\mathcal{L}$ is equivalent to some CF-grammar. For different logics, different proofs have been found.

Let T denote a finite set of types. For a logic $\mathcal{L}$ and a type A, $L^{\mathcal{L}}(T, A)$ denotes the set of all $\Gamma \in T^+$ such that $\Gamma \vdash_{\mathcal{L}} A$. It suffices to prove that $L^{\mathcal{L}}(T, A)$ is a context-free language, for any finite set T and any (atomic) type A.

The first kind of proofs is based on different forms of *normalization* of proofs in $\mathcal{L}$. In [13, 49, 50] a normalization theorem for NL was established. According to the theorem, if $A_1, \ldots, A_n \vdash_{NL} B$, then there exist $C_1, \ldots, C_n, C$ such that A_i reduces to C_i, $i = 1, \ldots, n$, by some NL-derivable rewriting rules, decreasing the complexity of types, $C_1 \ldots C_n$ reduces to C by AB-rules, and C expands to B by some NL-derivable rewriting rules, increasing the complexity of types. Such derivations can easily be simulated by a CF-grammar. This method has been applied in [52, 53] to grammars based on NL with (EXC) and NL with arbitrary modalities. A similar idea works for grammars based on CBL (see subsection 2.2), where normalization is given by the switching lemma.

The second kind of proofs uses *interpolation*. Actually, two forms of interpolation are possible.

Associative and multiplicative systems admit Roorda interpolation. For p atomic, let $|A|_p$ denote the number of occurrences of p in A. $|\Gamma|_p$ is defined in a similar way. $|\Gamma|$ denotes the total number of occurrences of atomic types in Γ, and similarly for $|A|$. The following lemma was proved by Roorda [84].

(INT) Let $\Gamma\Delta\Gamma' \vdash B$ be provable in L with $\Delta \neq \epsilon$. Then, there exists type A (an interpolant of Δ), satisfying: (i) $\Delta \vdash A$ is provable in L, (ii) $\Gamma A \Gamma' \vdash B$ is provable in L, (iii) for any atomic p, $|A|_p$ is not greater than the minimum of $|\Delta|_p$, $|\Gamma\Gamma' B|_p$.

A similar lemma holds for L* and (multiplicative) CLL. Using this lemma, Pentus [77] proves that type grammars based on L generate context-free languages. Let $T(P, m)$ denote the set of all types of complexity not greater than m, formed out of atomic types from P. Pentus proves a binary reduction property: if $A_1, \ldots, A_n, A$ belong to $T(P, m)$ and $A_1, \ldots, A_n \vdash_L A$, where $n \geq 2$, then there exist $i \in \{1, \ldots, n-1\}$ and B in $T(P, m)$ such that B is an interpolant of $A_i A_{i+1}$ in this sequent (conditions (i), (ii) are satisfied). So, a CF-grammar for the language $L^L(T(P, m), p)$ can be constructed; $T(P, m)$ is the set of nonterminals and production rules are (reversed) binary sequents provable in L (restricted to nonterminals). An analogous construction works for L* and CLL. In [9], this method has been applied to L with assumptions of the form $p \vdash q$, and in [6] to CBL.

Nonassociative systems admit another form of interpolation, not admissible for associative ones. Let T be a finite set of types, closed under subtypes. $T-$sequents are sequents using types from T only.

(NINT) Let $X[Y] \vdash B$ be a provable $T-$sequent. Then, there exists $A \in T$ such that $X[A] \vdash B$ and $Y \vdash A$ are provable.

(NINT) for NL (also with (EXC)) was proved by J¨ager [45]. It follows that, for $A \in T$, the language $L^{NL}(T, A)$ can be generated by a CF-grammar

with production rules $A \mapsto B, C$, for provable sequents $B, C \vdash A$. This yields a new proof of the fact that categorial grammars based on NL are equivalent to CF-grammars. The proof of (NINT) proceeds by induction on proofs in NL. Actually, (CUT) does not destroy the proof, so (NINT) also holds for NL enriched with assumptions. In [26], this method was refined to show PTIME-decidability of NL (also with assumptions).

Some form of (NINT) remains true for NL with $\wedge$ and $\vee$ (for $\vee$, the distribution of $\wedge$ over $\vee$ must be added). The interpolant A belongs to a larger set T' which is a closure of T under lattice operations. Consequently, categorial grammars based on NL with $\wedge$ or $\wedge$ and $\vee$ with distribution do not surpass context-free languages [36]. The problem is open for NL with $\wedge$ and $\vee$ without distribution. The situation is different for L and L* with $\wedge$: they can generate meets of two context-free languages, which are not context-free, in general [48].

An interesting problem is to extend type logics to be able to express negative facts. We briefly discuss some attempts from [18].

A De Morgan negation $\neg$ satisfies Double Negation $\neg\neg a = a$ and Transposition: if $a \leq b$ then $\neg b \leq \neg a$ [32]. L enriched with $\neg$ with additional axioms: $\neg\neg A \vdash A$, $A \vdash \neg\neg A$ and the rule:

$$\text{(TRA)}\ \frac{A \vdash B}{\neg B \vdash \neg A}$$

does not admit cut elimination. With (CUT), it is complete with respect to residuated semigroups with De Morgan negation. Its product free fragment is strongly complete with respect to powerset residuated semigroups with a quasi-Boolean complement in the sense of [8]: for a fixed involutive mapping $f : M \mapsto M$ and $P \subseteq M$, one defines $-P = M - f[P]$. The full system is strongly complete with respect to cone models over semigroups (the complement is quasi-Boolean). These facts follow from representation theorems: (i) for every residuated semigroup with De Morgan negation, its product free reduct is embeddable into some powerset residuated semigroup with quasi-Boolean complement, (ii) every residuated semigroup with De Morgan negation is embeddable into some residuated semigroup of cones (over some semigroup) with quasi-Boolen complement.

The decidability of L with De Morgan negation seems to be an open problem, and similarly for other type logics, mentioned above. Strong FMP for residuated groupoids, recently proved in [36], implies the decidability of the universal theory of residuated groupoids. Consequently, NL can be effectively used to derive positive and negative information from positive and negative assumptions.

We have discussed frames in which logical constants of type logics are interpreted by means of some operations on sets which are naturally determined by the underlying semigroup (groupoid) structure. A more general approach uses Kripke-style frames for modal logics [91, 93]. A frame is a pair (M, R) such that M is a set of states, and R is a ternary relation on M. For sets $P_1, P_2 \subseteq M$, one defines:

(K1) $P_1 \cdot P_2 = \{z \in M : (\exists x \in P_1, y \in P_2)\, R(x, y, z)\}$,

(K2) $P_1 \backslash P_2 = \{y \in M : (\forall x \in P_1, z \in M)(R(x,y,z) \Rightarrow z \in P_2)\}$,

(K3) $P_1 / P_2 = \{x \in M : (\forall y \in P_2, z \in M)(R(x,y,z) \Rightarrow z \in P_1)\}$.

The structure $(P(M), \subseteq, \cdot, \backslash, /)$ is a residuated groupoid. Every powerset residuated groupoid is a structure of this form, since one can define: $R(x,y,z)$ iff $z = xy$. Consequently, NL is strongly complete with respect to Kripke models. A similar fact holds for L and Kripke frames, satisfying associativity: (i) if $R(x,y,z)$ and $R(z,u,v)$ then there exists w such that $R(y,u,w)$ and $R(x,w,v)$, (ii) if $R(y,u,w)$ and $R(x,w,v)$ then there exists z such that $R(x,y,z)$ and $R(z,u,v)$. Kripke frames are largely discussed by Moortgat [69] in connection with modal extensions of NL.

NL with modalities can be presented as Generalized Lambek Calculus (GLC) [16, 20]; Dunn [33] studies a closely related system with more general models. One starts from abstract algebras (M, F) such that M is a nonempty set and $F = (f_i)_{i \in I}$ is an indexed family of operations on M. On the powerset $P(M)$, one defines powerset operations:

$$f_i(P_1, \ldots, P_n) = \{f_i(x_1, \ldots, x_n) : x_1 \in P_1, \ldots, x_n \in P_n\},$$

and for each $1 \leq j \leq n$ (n is the arity of f_i), the residuation operation f_i^j is defined as follows: $f_i^j(P_1, \ldots, P_n)$ equals the set of all $x_j \in M$ such that $f_i(x_1, \ldots, x_n) \in P_j$, for all $x_k \in P_k$, $k \neq j$. The resulting structures are obvious generalizations of powerset residuated groupoids. Types of GLC are formed out of primitive types by means of logical constants $\otimes_i, \rightarrow_i^j$ and the structure operation $\circ_i$, for $i \in I$ (their arity is that of f_i). The axioms are (Id), and the inference rules are:

$$(\otimes_i \mathrm{L})\ \frac{X[\circ_i(A_1, \ldots, A_n)] \vdash B}{X[\otimes_i(A_1, \ldots, A_n] \vdash B},$$

$$(\otimes_i \mathrm{R})\ \frac{X_1 \vdash A_1; \ldots; X_n \vdash A_n}{\circ_i(X_1, \ldots, X_n) \vdash \otimes_i(A_1, \ldots, A_n)},$$

$$(\rightarrow_i^j \mathrm{L})\ \frac{X[A_j] \vdash B;\ Y_1 \vdash A_1; \ldots; Y_n \vdash A_n}{x[\circ_i(Y_1, \ldots, \rightarrow_i^j (A_1, \ldots, A_n), \ldots, Y_n)] \vdash B},$$

$$(\rightarrow_i^j \mathrm{R})\ \frac{\circ_i(A_1, \ldots, X, \ldots, A_n) \vdash A_j}{X \vdash \rightarrow_i^j (A_1, \ldots, A_n)},$$

and (NCUT). In rule $(\rightarrow_i^j \mathrm{L})$, the premise $Y_j \vdash A_j$ is dropped and the type $\rightarrow_i^j (\ldots)$ is the $j-$th argument of $\circ_i$ in the conclusion. In rule $(\rightarrow_i^j \mathrm{R})$, X is the $j-$th argument of $\circ_i$ in the premise. GLC admits cut elimination. As shown in [56], GLC is strongly complete with respect to powerset frames over abstract algebras. Clearly, $\otimes_i$ is interpreted by f_i and $\rightarrow_i^j$ by f_i^j. By cut elimination, GLC is decidable; actually, all finitely axiomatizable theories over GLC are PTIME decidable [26]. This also holds, obviously, for multi-modal systems without additional structural postulates (rules). Grammars based on GLC generate context-free languages [53, 26].

Systems with many products can be used in linguistics to formalize different modes of composition of expressions. Instead of concatenation, one may use other operations which compose a complex expression from simpler ones. A good candidate is substitution. We enrich the lexicon Σ with variables $x_1, x_2, \ldots$ and define $f_i(a,b)$ to be equal to the result of substitution of b for x_i in a. In this way, we can analyse discontinuous phrases. For instance, 'if ... then ...' can be identified with 'if x_1 then x_2' and be assigned type $\rightarrow_1^1 (\rightarrow_2^1 (\mathrm{S},\mathrm{S}),\mathrm{S})$. A similar approach to discontinuity has been proposed by Morrill [71, 72].

At the end, we return to a recent proposal of Lambek [62] to use free pregroups for natural language grammar; see the end of subsection 2.2. *A pregroup grammar* assigns to words strings $A_1 \ldots A_k$, each A_i being of the form $p^{(n)}$. By the normalization theorem ('switching lemma'), in reduction of Γ to p only rules (CON) and (IND) can be employed. In [22], the weak equivalence of pregroup grammars and CF-grammars has been proved. One part of the theorem easily follows from the fact that, for any finite set P of atomic types, and any $n \geq 0$, the set $S(P,n)$ of all types $p^{(m)}$ such that $p \in P$ and the absolute value of m is not greater than n is finite. The language $L^{CBL}(S(P,n),p)$ is generated by a CF-grammar whose nonterminals are types from $S(P,n)$ and production rules correspond to (reversed) (CON) and (IND) (without the context). The language on Σ, generated by the pregroup grammar, can be obtained from the latter by inverse homomorphism and homomorphism; so, it is context-free.

We show here that an effective construction of an inverse homomorphism image of $L(S(P,n),p)$ yields, in fact, a PTIME construction of a CF-grammar equivalent to the given pregroup grammar. This construction is due to K. Moroz and the author.(In [6] an EXPTIME construction is presented, which uses a kind of Pentus-style binary reduction lemma.)

Let I_G be the initial type assignment of the pregroup grammar G. For $a \in \Sigma$, $I_G(a)$ is a finite set of strings of simple types of the form $p^{(m)}$. Let N be the maximum of absolute values of m in types $p^{(m)}$ appearing in I_G. For finite P, the set $S(P,N)$ is finite. A CF-grammar G' such that $L(G')$ equals $L(S(P,N),S)$ has the following productions:

$$X \mapsto EX;\ X \mapsto XE;\ E \mapsto p^{(n)}, q^{(n+1)},$$

for any $X \in S(P,N)$, $p^{(n)}, q^{(n+1)} \in S(P,N)$. such that $p \leq q$, if n is even, and $q \leq p$, if n is odd. E is a new nonterminal, representing the empty string.

Let M be a finite-state automaton, which accepts all strings of the form $a_1 x_1 \ldots a_n x_n$ such that $a_i \in \Sigma$ and $x_i \in I_G(a_i)$. The states of M are q_0 and states $[y]$ such that y is a terminal segment of some string x appearing in I_G. q_0 is the initial state, and $[\epsilon]$ is the final state. The (nondeterministic) transition function is defined as follows: (i) $\delta(q_0, a)$ equals the set of all $[x]$ such that $x \in I_G(a)$, and similarly for $\delta([\epsilon], a)$, (ii) $\delta([Xy], X)$ equals $\{[y]\}$.

We modify G' to admit symbols $a \in \Sigma$ on the right hand of nonterminals from $S(P,N)$. It suffices to add productions $X \mapsto a, X$. Let G'' denote the modified CF-grammar. A routine construction yields a CF-grammar for the meet of $L(G'')$ and $L(M)$. Nonterminals are triples (q, X, q') such that X is a

nonterminal of G'' (it may be a symbol from Σ) and q, q' are states of M. The initial symbol is $(q_0, S, [\epsilon])$, and productions are:

$$(q_1, X, q_2) \mapsto (q_1, Y, q_3), (q_3, Z, q_2),$$

for any production $X \mapsto Y, Z$ of G'' and all states q_1, q_2, q_3. One adds lexical and deleting rules:

$$([\epsilon], a, [x]) \mapsto a;\ (q_0, a, [x]) \mapsto a,\ \text{for } x \in I_G(a),$$

$$([Xy], X, [y]) \mapsto \epsilon.$$

The resulting CF-grammar generates the language of the initial pregroup grammar. It is easy to see that the size of this construction is polynomial in the size of G. Further, we can eliminate nullary rules and unary rules to obtain a CF-grammar in Chomsky Normal Form; the latter construction is also PTIME.

References

[1] V.M. Abrusci, Phase Semantics and Sequent Calculus for Pure Noncommutative Linear Logic, *Journal of Symbolic Logic* 56 (1991), 1403-1451.

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

[3] H. Andréka and S. Mikulás, Lambek Calculus and Its Relational Semantics. Completeness and Incompleteness, *Journal of Logic, Language and Information* 3 (1994), 1-37.

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

[5] Y. Bar-Hillel, C. Gaifman and E. Shamir, On categorial and phrase structure grammars, *Bull. Res Council Israel* F 9 (1960), 155-166.

[6] D. Bechet, Parsing pregroup grammars and Lambek calculus using partial composition, to appear in *Studia Logica*.

[7] D. Bechet and A. Foret, $k-$valued non-associative Lambek grammars are learnable from generalized functor-argument structures, *Theoretical Computer Science* 355(2) (2006), 139-152.

[8] A. Białynicki-Birula and H. Rasiowa, On the Representation of Quasi-Boolean Algebras, *Bull. Acad. Polonaise Scie.* 5 (1957), 259-261.

[9] M. Bulińska, The Pentus Theorem for Lambek Calculus with Simple Nonlogical Axioms, *Studia Logica* 81 (2005), 43-59.

[10] W. Buszkowski, Some Decision Problems in the Theory of Syntactic Categories, *Zeitschrift f. math. Logik u. Grundlagen der Mathematik* 28 (1982), 539-548.

[11] W. Buszkowski, The equivalence of unidirectional Lambek categorial grammars and context-free grammars, *Zeitschrift f. math. Logik u. Grundlagen der Mathematik* 31 (1985), 369-384.

[12] W. Buszkowski, Completeness Results for Lambek Syntactic Calculus, *Zeitschrift f. math. Logik u. Grundlagen der Mathematik* 32 (1986), 13-28.

[13] W. Buszkowski, Generative Capacity of Nonassociative Lambek Calculus, *Bull. Polish Academy Scie. Math.* 34 (1986), 507-516.

[14] W. Buszkowski, Discovery Procedures for Categorial Grammars, in: E. Klein and J. van Benthem, eds., *Categories, Polymorphism and Unification*, University of Amsterdam, 1987.

[15] W. Buszkowski, Generative Power of Categorial Grammars, in: R.T. Oehrle, E. Bach and D. Wheeler, eds., *Categorial Grammars and Natural Language Structures*, D. Reidel, Dordrecht, 1988, 69-94.

[16] W. Buszkowski, *Logical Foundations of Ajdukiewicz-Lambek Categorial Grammars*, Polish Scientific Publishers, Warsaw, 1989. In Polish.

[17] W. Buszkowski, The finite model property for BCI and related systems, *Studia Logica* 57 (1996), 303-323.

[18] W. Buszkowski, Categorial Grammars with Negative Information, in: H. Wansing (ed.), *Negation. A notion in focus*, de Gruyter, Berlin, 1996, 107-126.

[19] W. Buszkowski, The Ajdukiewicz Calculus, Polish Notation and Hilbert Style Proofs, in: K. Kijania-Placek and J. Woleński (eds.), *The Lvov-Warsaw School and Contemporary Philosophy*, Kluwer, Dordrecht, 1998, 241-252.

[20] W. Buszkowski, Mathematical Linguistics and Proof Theory, in: J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, Elsevier, Amsterdam, 1997, 683-736.

[21] W. Buszkowski, Algebraic structures for categorial grammars, *Theoretical Computer Science* 199 (1998), 5-24.

[22] W. Buszkowski, Lambek Grammars Based on Pregroups, in: P. de Groote, G. Morrill and C. Retoré (eds.), *Logical Aspects of Computational Linguistics*, LNAI 2099, Springer, Berlin, 2001, 95-109.

[23] W. Buszkowski, Finite Models of Some Substructural Logics, *Mathematical Logic Quarterly* 48 (2002), 63-72.

[24] W. Buszkowski, Sequent systems for Compact Bilinear Logic, *Mathematical Logic Quarterly*49 (2003), 467-474.

[25] W. Buszkowski, Type Logics in Grammar, in: V.F. Hendricks and J. Malinowski, eds., *Trends in Logic*, Kluwer, 2003, 337-382.

[26] W. Buszkowski, Lambek Calculus with Nonlogical Axioms, in: C. Casadio, P.J. Scott and R. Seely, eds., *Language and Grammar. Studies in Mathematical Linguistics and Natural Language*, CSLI Lecture Notes 168, Stanford, 2005, 77-93.

[27] W. Buszkowski, On Action Logic, to appear in *Journal of Logic and Computation*.

[28] W. Buszkowski and M. Kołowska-Gawiejnowicz, Representation of Residuated Semigroups in Some Algebras of Relations. The Method of Canonical Models, *Fundamenta Informaticae* 31 (1997), 1-12.

[29] W. Buszkowski and G. Penn, Categorial Grammars Determined from Linguistic Data by Unification, *Studia Logica* 49 (1990), 431-454.

[30] C. Casadio, Non-Commutative Linear Logic in Linguistics, *Grammars* 3/4 (2001), 1-19.

[31] C. Casadio and J. Lambek, An Algebraic Analysis of Clitic Pronouns in Italian, in: P. de Groote, G. Morrill and C. Retoré (eds.), *Logical Aspects of Computational Linguistics*, LNAI 2099, Springer, Berlin, 2001, 110-124.

[32] J.M. Dunn, Perp and Star: Two Treatments of Negation, in: J. Tomberlin (ed.), *Philosophical Perspectives (Philosophy of Language and Logic)* 7 (1993), 331-357.

[33] J.M. Dunn, Partial Gaggles Applied to Logics with Restricted Structural Rules, in: P. Schroeder-Heister and K. Došen (eds.), *Substructural Logics*, Clarendon Press, Oxford, 1993, 63-108.

[34] B. Dziemidowicz, Optimal Unification and Learning Algorithms for Categorial Grammars, *Fundamenta Informaticae* 49 (2002), 297-308.

[35] B. Dziemidowicz, On Learnability of Restricted Classes of Categorial Grammars, to appear in *Studia Logica*.

[36] M. Farulewski, Finite Model Property for Some Substructural Logics, PhD Thesis, in preparation.

[37] A. Foret, On Mixing Deduction and Substitution in Lambek Categorial Grammars, in: P. de Groote, G. Morrill and C. Retoré, eds., *Logical Aspects of Computational Linguistics*, LNAI 2099, Springer, 2001, 158-174.

[38] S.A. Fulop, Semantic Bootstrapping of Type-Logical Grammar, *Journal of Logic, Language and Information* 14 (2005), 49-86.

[39] D.M. Gabbay, *Labelled Deductive Systems*, Oxford University Press, Oxford, 1996.

[40] M. Gécség and M. Steinby, *Tree Automata*, Akademiai Kiadó, Budapest, 1984.

[41] E.M. Gold, Language Identification in the Limit, *Information and Control* 10 (1867), 447-474.

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

[43] P. de Groote and F. Lamarche, Classical Non-Associative Lambek Calculus, *Studia Logica* 71.3 (2002), 355-388.

[44] P. Hajek, *Metamathematics of Fuzzy Logic*, Trends in Logic 4, Kluwer, 1998.

[45] G. Jäger, On the Generative Capacity of Multi-modal Categorial Grammars, *Research on Language and Computation* 1 (2003), 105-125.

[46] P. Jipsen, From Semirings to Residuated Kleene Algebras, *Studia Logica* 76 (2004), 291-303.

[47] D. Kozen, A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events, *Information and Computation* 110.2 (1994), 366-390.

[48] M. Kanazawa, The Lambek Calculus Enriched with Additional Connectives, *Journal of Logic, Language and Information* 1.2 (1992), 141-171.

[49] M. Kanazawa, Identification in the Limit of Categorial Grammars, *Journal of Logic, Language and Information* 5 (1996), 115-155.

[50] M. Kanazawa, *Learnable Classes of Categorial Grammars*, CSLI Publications, Stanford, 1988.

[51] M. Kandulski, The Equivalence of Nonassociative Lambek Categorial Grammars and Context-Free Grammars, *Zeitschrift f. math. Logik u. Grundlagen der Mathematik* 34 (1988), 41-52.

[52] M. Kandulski, Normal Form of Derivations in the Nonassociative and Commutative Lambek Calculus with Product, *Mathematical Logic Quarterly* 39 (1993), 103-114.

[53] M. Kandulski, On Generalized Ajdukiewicz and Lambek Calculi and Grammars, *Fundamenta Informaticae* 30.2 (1997), 169-181.

[54] M. Kandulski, Derived Tree Languages of Nonassociative Lambek Categorial Grammars with Product, *Fundamenta Informaticae* 55(3,4) (2003), 349-362.

[55] S. Kapur, *Computational Learning of Languages*, Ph.D. Thesis, Cornell University, 1991.

[56] M. Kołowska-Gawiejnowicz, Powerset Residuated Algebras and Generalized Lambek Calculus, *Mathematical Logic Quarterly* 43 (1997), 60-72.

[57] N. Kurtonina, *Frames and Labels. A modal analysis of categorial inference*, Ph.D. Thesis, University of Utrecht, 1995.

[58] Y. Lafont, The finite model property for various fragments of linear logic, *Journal of Symbolic Logic* 62 (1997), 1202-1208.

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

[60] J. Lambek, On the Calculus of Syntactic Types, in: R. Jakobson (ed.), *Structure of Language and Its Mathematical Aspects*, Proc. Symp. Appl. Math., AMS, Providence, 1961, 166-178.

[61] J. Lambek, Bilinear Logic in Algebra and Linguistics, in: J.Y. Girard, Y. Lafont and L. Regnier (eds.), *Advances in Linear Logic*, Cambridge University Press, Cambridge, 1995, 43-59.

[62] J. Lambek, Type Grammars Revisited, in: A. Lecomte, F. Lamarche and G. Perrier (eds.), *Logical Aspects of Computational Linguistics*, LNAI 1582, Springer, Berlin, 1999, 1-27.

[63] J. Lambek, Type Grammars as Pregroups, *Grammars* 4 (2001), 21-39.

[64] S. Leśniewski, Grundzüge einer neuen System der Grundlagen der Mathematik, *Fundamenta Mathematicae* 14 (1929), 1-81.

[65] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar, Decision problems for propositional linear logic, *Annals of Pure and Applied Logic* 56 (1992), 239-311.

[66] J. Marciniec, Learning Categorial Grammars by Unification with Negative Constraints, *Journal of Applied Non-Classical Logics* 4 (1994), 181-200.

[67] R. Montague, *Formal Philosophy*, Selected papers of R. Montague edited by R. Thomason, Yale University Press, New Haven, 1974.

[68] M. Moortgat, *Categorial Investigations. Logical and Linguistic Aspects of the Lambek Calculus*, Foris, Dordrecht, 1988.

[69] M. Moortgat, Categorial Type Logics, in: J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, Elsevier, Amsterdam, 1997, 93-177.

[70] M. Moortgat, Structural Equations in Language Learning, in: P. de Groote, G. Morrill and C. Retoré (eds.), *Logical Aspects of Computational Linguistics*, LNAI 2099, Springer, Berlin, 2001, 1-16.

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

[72] G. Morrill, A Generalised Discontinuity, manuscript, LICS, Ottawa, 2003.

[73] H. Ono, Semantics of Substructural Logics, in: P. Schroeder-Heister and K. Dosen, eds., *Substructural Logics*, Clarendon Press, Oxford, 1993, 259-291.

[74] D. Osherson, D. de Jongh, E. Martin and S. Weinstein, Formal learning theory, in: J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, Elsevier, Amsterdam, 1997, 737-775.

[75] E. Palka, An infinitary sequent system for the equational theory of *-continuous action lattices, to appear in *Fundamenta Informaticae*.

[76] Z. Pawlak, *Rough Sets. Theoretical Aspects of Reasoning about Data*, Kluwer, Dordrecht, 1991.

[77] M. Pentus, Lambek Grammars are Context-Free, *Proc. of 8th IEEE Symposium on Logic in Computer Science*, 1993, 429-433.

[78] M. Pentus, Models for the Lambek Calculus, *Annals of Pure and Applied Logic* 75 (1995), 179-213.

[79] M. Pentus, Lambek calculus is NP-complete, manuscript, Moscow State University, 2003.

[80] V. Pratt, Action Logic and Pure Induction, in: J. van Eijck (ed.), *Logics in AI*, LNAI 478, Springer, Berlin, 1991, 97-120.

[81] V. Redko, On defining relations for the algebra of regular events, *Ukrain. Mat. Z.* 16 (1964), 120-126. In Russian.

[82] G. Restall, *An Introduction to Substructural Logics*, Routledge, London, 2001.

[83] C. Retoré, *The Logic of Categorial Grammars. Lecture Notes.*, INRIA, 2005.

[84] D. Roorda, *Resource Logics. Proof-Theoretical Investigations*, Ph.D. Thesis, University of Amsterdam, 1991.

[85] T. Shinohara, Inductive Inference of Monotonic Formal Systems from Positive Data, in: S. Arikawa et al. (eds.), *Algorithmic Learning Theory*, Springer, Tokyo, 1990, 339-351.

[86] H-J. Tiede, *Deductive Systems and Grammars*, Ph.D. Thesis, Indiana University, 1999.

[87] A.S. Troelstra, *Lectures on Linear Logic*, CSLI Lecture Notes 29, Stanford, 1992.

[88] H. Uszkoreit, Categorial Unification Grammars, *Proc. 11th International Conference on Computational Linguistics*, Bonn, 1986, 187-194.

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

[90] J. van Benthem, Categorial Equations, in: E. Klein and J. van Benthem (eds.), *Categories, Polymorphism and Unification*, University of Amsterdam, 1987, 1-17.

[91] J. van Benthem, *Language in Action. Categories, Lambdas and Dynamic Logic*, North Holland, Amsterdam, 1991.

[92] J. van Benthem, *Exploring Logical Dynamics*, CSLI, Stanford, 1996.

[93] J. van Benthem, The Categorial Fine-Structure of Natural Language, in: C. Casadio, P.J. Scott and R. Seely, eds., *Language and Grammar. Studies in Mathematical Linguistics and Natural Language*, CSLI Lecture Notes 168, Stanford, 2005, 3-29.

[94] K. Wright, Identification of unions of languages drawn from an identifiable class, in: *The 1989 Workshop on Computational Learning Theory*, Morgan Kaufmann, San Mateo, 1989, 328-333.

[95] D.N. Yetter, Quantales and (Non-Commutative) Linear Logic, *Journal of Symbolic Logic* 55 (1996), 41-64.