Logic and **L**ogical **P**hilosophy
Volume 3 (1995), 47–86

Wojciech Buszkowski

GRAMMATICAL STRUCTURES AND LOGICAL DEDUCTIONS

CONTENTS

Received November 29, 1995

The three essays presented here concern natural connections between grammatical derivations and structures provided by certain standard grammar formalisms, on the one hand, and deductions in logical systems, on the other hand. In the first essay we analyse the adequacy of Polish notation for higher-order languages. The Ajdukiewicz algorithm (Ajdukiewicz 1935) is discussed in terms of generalized MP-deductions. We exhibit a failure in Ajdukiewicz's original version of the algorithm and give a correct one; we prove that generalized MP-deductions have the frontier property, which is essential for the plausibility of Polish notation. The second essay deals with logical systems corresponding to different grammar formalisms, as e.g. Finite State Acceptors, Context-Free Grammars, Categorial Grammars, and others. We show how can logical methods be used to establish certain linguistically significant properties of formal grammars. The third essay discusses the interplay between Natural Deduction proofs in grammar oriented logics and semantic structures expressible by typed lambda terms and combinators.

The results of section 1 have been published in Polish in [10] (here, proofs are simplified). In section 2, the logical proof of the equivalence of Recursive Transition Networks and Context-Free Grammars is new. Section 3 contains no new results. However, the main objective of this paper is to emphasize the unity of logical deductions and grammatical derivations, and that appears the first time in print. Of course, the very idea of linking logic and grammar is an old one. Let us explicitly mention Hiż [24] who qualifies type-theoretic approaches to the theory of grammar as the grammatical component of the doctrine of *logicism*.

1. MP-deductions and Polish notation

In a Hilbert system theorems are proven from a set of axioms/assumptions by few inference rules of the form:

$$\frac{A_1; \ldots ; A_n}{B},$$

where the premises $A_1, \ldots, A_n$ and the conclusion B are formulas of the system. For many systems, the only inference rule is *Modus Ponens*:

$$\text{(MP)}\ \frac{A \to B;\ A}{B}.$$

Logic interprets $A \to B$ as to mean something 'B can be inferred from A', hence (MP) is plausible. Semantics also refers to (MP) as *the function application principle*: if F is a function of type $A \to B$ and x is an object of type A, then $F(x)$ is an object of type B. The functional interpretation of (MP) underlies the functor-argument construction which is the basic construction of Fregean Grammar: if X is an expression of type $A \to B$ (the functor) and Y is an expression of type A (the argument), then XY is an expression of type B, and $\mu(XY) = \mu(X)(\mu(Y))$, where $\mu(X)$ stands for the semantic denotation of X.

The interplay between the logical and the functional meaning of (MP) and other inference rules (e.g. Conditional Syllogism versus Function Composition) witnesses general relations between logical deductions and grammatical constructions to be discussed in this chapter. In this section we focus on a simplest relation of that kind: MP-deductions versus Polish notation.

Each MP-deduction is graphically represented as a (rising up) tree in which the leaves are labelled by the assumptions and the internal nodes result from applying (MP) to their children; the outcome formula occupies the root of the tree. To save space, we represent trees in a linear form. We write $t : A$, if A is the outcome of tree t. Then, MP-deductions and their outcomes are recursively defined as follows:

(D1) $A : A$, for every formula A,

(D2) if $s : A \to B$ and $t : A$ then $[s, t] : B$.

In the pattern $s : A$, the bracketed string s uniquely determines the outcome A, since the conclusion of (MP) is uniquely determined by the premises. Accordingly, MP-deductions can also be represented by bracketed strings of formulas, the outcome formulas omitted.

A typically linguistic issue is to focus on the frontier of the tree: the plane string of assumptions appearing on the leaves. In Fregean Grammar, formulas of the conditional language are interpreted as types: $A \to B$ is the type of functors taking an argument of type A in order to form a complex expression of type B. The initial lexical description of the given language assigns a type to each symbol (i.e. atomic expression) of this language. Types of non-atomic expressions are calculated in the following way. Given a string $v_1 \dots v_n$ of symbols, one forms the string $A_1 \dots A_n$, of types corresponding to these symbols. One assigns to $v_1 \dots v_n$ precisely those types which are the outcomes of MP-deductions with the frontier $A_1 \dots A_n$. If t is an MP-deduction with outcome A and frontier $A_1 \dots A_n$, then the bracketing of t induces a grammatical structure on the string $v_1 \dots v_n$; further, if denotations $\mu(v_i)$

are fixed, then the denotation of the expression $v_1 \dots v_n$ supplied with this structure can be obtained by function application, going along the tree in the top-down direction.

From the linguistic point of view, a significant property of MP-deductions is the frontier property:

(FP) *each MP-deduction is uniquely determined by its frontier.*

Accordingly, in Fregean languages, well-formed expressions can uniquely be represented as strings of symbols (supplied with types) without parentheses and other structure markers. That is the leitmotive of Polish notation, going back to ŁUKASIEWICZ for the case of sentential logics and being extended to higher-order languages in AJDUKIEWICZ [2] whith origins in Logical Syntax of LEŚNIEWSKI; see HIŻ [24]. Precisely, Ajdukiewicz admits multiple-argument functors: the functor X is of type $A_1 \dots A_n \to B$, if together with arguments $Y_1, \dots, Y_n$ of type $A_1, \dots, A_n$, respectively, it forms the complex expression $XY_1 \dots Y_n$ of type B. For instance, if PN is the type of proper nouns and S is the type of sentential formulas, then the negation symbol $\neg$ is of type S→S, the implication symbol $\Rightarrow$ is of type S S→S, and the equality symbol $=$ is of type PN PN→S. The corresponding deductions are based on generalized MP-rules:

$$\text{(MP-n)}\ \frac{A_1 \dots A_n \to B;\ A_1; \dots;\ A_n}{B}.$$

Clearly, the formal notion of a generalized MP-deduction can be obtained by a straightforward modification of (D1), (D2). The plausibility of Polish notation for higher-order languages with multiple-argument functors relies upon the following strengthening of (FP):

(FP*) *each generalized MP-deduction is uniquely determined by its frontier.*

Actually, Ajdukiewicz gives no proof of (FP*). Instead, he proposes an algorithm which, for the given string $A_1 \dots A_n$, produces an outcome type A and a structure of this string. The Ajdukiewicz algorithm obeys *the left first routine*. In the string, one finds the left-most appearance of an interval of the form:

$$(B_1 \dots B_m \to A), B_1, \dots, B_m$$

and replaces this interval with B. Starting from the initial string, one repeats this reduction step as far, as possible. If this procedure terminates with a single type as the final string, this type is the outcome type; otherwise, the algorithm qualifies the initial string as ill-formed. Clearly, for any initial

string, there is precisely one left first reduction, and consequently, if this reduction is successful, the algorithm produces a unique outcome type as well as a unique structure.

The adequacy of the Ajdukiewicz algorithm depends on the question if all generalized MP-deductions can be determined from their frontiers according to the left first routine. Unfortunately, the answer is negative. Consider the string:

$$((\mathrm{PN}\to\mathrm{PN})\ \mathrm{PN}\ \mathrm{PN}\to\mathrm{S}),\ (\mathrm{PN}\to\mathrm{PN}),\ \mathrm{PN},\ (\mathrm{PN}\to\mathrm{PN}),\ \mathrm{PN}.$$

Following the left first routine, one first reduces the second and the third type to PN and then the last two types to PN, and the resulting string:

$$((\mathrm{PN}\to\mathrm{PN})\ \mathrm{PN}\ \mathrm{PN}\to\mathrm{S}),\ \mathrm{PN},\ \mathrm{PN}$$

is irreducible. But the initial string is the frontier of an MP-deduction with outcome S: first apply (MP) to the last two types, then (MP-3) to the whole. Passing to a grammar, consider the second-order predicate I defined as follows:

$$I(f,x,y)\ \text{ iff }\ f(x)=y.$$

Thus, I is of type (PN→PN) PN PN→S. The Polish expression $Ifxgy$ is well-formed; it means $f(x)=g(y)$. But this expression gives rise to the string of types written above, hence the Ajdukiewicz algorithm qualifies it as ill-formed. Of course, the MP-deduction mentioned above (resigning from the left-first routine) yields the proper outcome and the correct structure $(Ifx(gy))$. Concluding, not all well-formed expressions can correctly be analysed by the Ajdukiewicz algorithm based on the left first routine. Instead, one needs the *non-deterministic* version of the Ajdukiewicz procedure: intervals can be reduced independently of their position in the string. Non-determinism makes it possible to analyse the given string in different ways, and one cannot a priori exclude the situation in which different analyses lead to different outcome types or proof structures.

To justify Polish notation in its general scope, a proof of (FP*) must be given. Further, no a constraints upon the shape of admissible MP-deductions (like the left first routine) should be imposed, since they could eliminate some well-formed expressions (as in the example above). Below we sketch a proof of (FP*) which is based on certain fine properties of generalized MP-trees. The reader can see in this proof a development of proof theory for Hilbert systems in a direction non-orthodox from the logical perspective, but quite natural from the linguistic point of view. As a consequence, we also obtain a

proof of (FP), and we show that the left first routine happens to be adequate for MP-deductions (one-argument types).

Greek capitals denote finite strings of formulas (types). We write $\Gamma \vdash A$, if there is a generalized MP-deduction $t : A$ with frontier Γ (equivalently: Γ reduces to A according to the non-deterministic Ajdukiewicz procedure). The relation $\vdash$ fulfills the following conditions:

(C1) $A \vdash A$ (reflexivity),

(C2) if $\Gamma, A, \Gamma' \vdash B$ and $\Delta \vdash A$ then $\Gamma, \Delta, \Gamma' \vdash B$ (transitivity).

The relation $A < B$ 'type A is a right subtype of type B' is recursively defined by the clauses:

$$A < (B_1 \dots B_n \to A),$$

$$\text{if } A < B \text{ and } B < C \text{ then } A < C;$$

so, $<$ is the transitive closure of the relation defined by the first clause. Clearly, $<$ is irreflexive and almost-linear:

$$\text{if } A < C \text{ and } B < C \text{ then } A < B \text{ or } A = B \text{ or } B < A.$$

We also use the property:

$$(B_1 \dots B_n \to A) \not< (C_1 \dots C_n \to A);$$

otherwise, A would be a proper subtype of itself.

We formulate three crucial lemmas:

(L1) if $A_1 \dots A_n \vdash A$, $(n > 1)$, then $A < A_1$,

(L2) if $A < B$, then $B, \Delta \vdash A$, for some nonempty Δ,

(L3) if $\Gamma \vdash A$, then there is no nonempty string Δ such that $\Gamma, \Delta \vdash A$.

(L1) is an obvious property of generalized MP-deductions. For (L2) we consider two cases. (CASE 1) $B = (B_1 \dots B_n \to A)$. Then, put Δ equal to $B_1 \dots B_n$. (CASE 2) $A < C < B$. By induction, there are nonempty strings Δ', Δ'' such that $B, \Delta' \vdash C$ and $C, \Delta'' \vdash A$. Then, $B, \Delta', \Delta'' \vdash A$, by (C2).

We prove (L3). Assume the opposite: $\Gamma \vdash A$ and $\Gamma, \Delta \vdash A$ with Δ nonempty. Let Γ be a shortest string which can be extended in this way. Γ is not a single type; otherwise, $\Gamma = A$, and (L1) yields $A < A$, which is

impossible. So, both Γ and Γ, Δ are frontiers of non-trivial deductions with outcome A. The down-most MP-rules applied in these deductions are:

$$\frac{B_1 \dots B_m \to A;\ B_1; \dots;\ B_m}{A},$$

$$\frac{C_1 \dots C_n \to A;\ C_1; \dots;\ C_n}{A},$$

respectively. Let B_0, C_0 abbreviate the first premises of these rules, and let C be the first type in Γ. By (L1), $B_0 < C$, $C_0 < C$, hence $B_0 < C_0$ or $B_0 = C_0$ or $C_0 < B_0$. By the last property of $<$ mentioned above, we get $B_0 = C_0$, and consequently, $m = n$ and $B_i = C_i$, for all $i = 1, \dots, n$. So, the frontier strings can be decomposed as:

$$\Gamma = \Gamma_0, \Gamma_1, \dots, \Gamma_n;\ \Gamma, \Delta = \Gamma'_0, \Gamma'_1, \dots, \Gamma'_n,$$

with $\Gamma_i \vdash B_i$, $\Gamma'_i \vdash C_i$, for all $i = 0, 1, \dots, n$. Since $\Gamma \neq \Gamma, \Delta$, there is a least i such that $\Gamma_i \neq \Gamma'_i$. Clearly, one of the two strings must be a proper initial interval of the other, and it is shorter than Γ. That contradicts the minimality of Γ.

To prove (FP*), we first show that the frontier uniquely determines the outcome type. Assume $\Gamma \vdash A$, $\Gamma \vdash B$ with $A \neq B$. Then, at least one of the two deductions must be non-trivial, hence Γ is not a single type. Using (L1) and almost-linearity, we get $A < B$ or $B < A$. Assume $A < B$ (the other case is dual). By (L2), $B, \Delta \vdash A$, for some nonempty Δ. Using (C2), we get $\Gamma, \Delta \vdash A$, which contradicts (L3).

We finish the proof of (FP*). Let Γ be the frontier of deductions t, t'. We show $t = t'$. We use induction on the length of Γ. By the preceding paragraph, there is a unique type A such that $\Gamma \vdash A$, hence $t : A$ and $t' : A$. If Γ is a single type, then $\Gamma = A$ and $t = t'$ are trivial deductions. Otherwise, both t and t' are non-trivial. We consider the down-most applications of MP-rules in t and t', as in the proof of (L3), and we get $m = n$, $B_i = C_i$, for all $i = 0, 1, \dots, n$, as above. Consequently:

$$\Gamma = \Gamma_0, \Gamma_1, \dots, \Gamma_n = \Gamma'_0, \Gamma'_1, \dots, \Gamma'_n,$$

and these intervals satisfy the conditions above. If $\Gamma_i \neq \Gamma'_i$, for some i, we would obtain a contradiction with (L3) (consider the least i fulfilling this inequality). So, $\Gamma_i = \Gamma'_i$, for all i. By induction, there are unique deductions $t_i : B_i$ with frontiers Γ_i, $i = 0, 1, \dots, n$, and we obtain:

$$t = [t_0, t_1, \dots, t_n] = t'.$$

For pure MP-deductions (equivalently: for one-argument types), the left first routine is adequate. By (FP) (which is a consequence of (FP*)), it is enough to prove the following: if $\Gamma \vdash A$ then Γ can be reduced to A according to the Ajdukiewicz algorithm. We use induction on the length of Γ. If Γ is a single type, then $\Gamma = A$, and the reduction is trivial. Otherwise, Γ is the frontier of a non-trivial deduction $t : A$. Let $B \to A$ and B be the premises of the down-most application of (MP) in t. Then, $\Gamma = \Gamma', \Gamma''$ with:

$$\Gamma' \vdash B \to A,\ \Gamma'' \vdash B.$$

By induction, there is an Ajdukiewicz reduction of Γ' to $B \to A$, which yields a reduction of Γ to $B \to A, \Gamma''$. There is also a reduction of Γ'' to B. If it is non-trivial, then, at each (non-final) step, $B <$ the first type in the string appearing at this step, and consequently, one cannot reduce the initial $B \to A$ with this type. So, we obtain an Ajdukiewicz reduction of $B \to A, \Gamma''$ to $B \to A, B$, which reduces to A in one step.

We have examined some special properties of MP-deductions which are significant for Logical Syntax: they justify the plausibility of Polish notation. By (FP*), this notation can be applied to functor-argument languages of arbitrary order; a standard example is Typed Combinatory Logic. Languages of Higher-Order Logic contain variable-binding operators, e.g. quantifiers $\forall x_A$, $\exists x_A$ (x_A is a variable of type A), and one may prohibit them to take the part of arguments in functor-argument constructions. Following Ajdukiewicz, we introduce the barrier | written before a type to block its argument role. For instance, quantifiers are of type |S→S; one can use this type as the first but not the second premise of (MP), and similarly for n-argument rules. All the results given above easily extend to types with barriers.

Quantifiers and similar operators (integral, sum, etc.) cause no problem due to the fact that their arguments are of a fixed type, and the operator scope (i.e. the argument) is a unique interval in the string ((L3) is crucial here). However, there is a trouble with polymorphic operators, e.g. the lambda abstractor, whose argument types may vary. To use Polish notation in Typed Lambda-Calculus, one must care for the definiteness of argument types: each occurrence of λx_A must be marked, say , $\lambda_B x_A$, where B is the type of the expected argument. Thus, $\lambda_B x_A$ is treated as a functor of type $B \to (A \to B)$. For Type-Free Lambda-Calculus, Polish notation cannot be used, unless one introduces some extra-devices, as e.g. infinite type-assignments for all symbols.

The Ajdukiewicz algorithm (obeying the left-first routine) can be executed by a deterministic push-down automaton. Consequently, given a finite

set T of one-argument types and a type A, the set of all strings Γ, of types from T, such that $\Gamma \vdash A$ is a deterministic context-free language. That may be of some interest for Logic: frontiers of deduction trees with a common outcome form a rather simple set in the hierarchy of languages. (Yet, Logic normally works with an infinite set T of axioms, being generated from finitely many axiom schemes by substitution.) It is noteworthy that the left first routine can be modified to fit generalized MP-deductions by employing Ajdukiewicz's barriers. Replace (MP-n), $n > 1$, with the following rules:

$$\frac{A_1 \dots A_n \to B}{\mid A_1 \dots A_n \to B},$$

$$\frac{\mid A_i \dots A_n \to B;\ A_i}{\mid A_{i+1} \dots A_n \to B},$$

$$\frac{\mid A_n \to B;\ A_n}{B}.$$

The role of $\mid$ is to block argument roles of the intermediate functor types, untill the last argument has been consumed. Using methods discussed above, one easily shows that generalized MP-deductions modified in this way can be executed according to the left-first routine, hence they can be simulated by deterministic push-down automata.

The logician may test other inference rules with regard to the frontier property. For Conditional Syllogism:

$$\frac{A \to B;\ B \to C}{A \to C},$$

only a half of (FP) holds. Due to the obvious associativity, all bracketings of the given string yield the same outcome (if any). Consequently, the frontier string uniquely determines an outcome but not a structure. If the latter rule acts together with (MP), determinacy totally disappears. The following deductions yield two different outcomes from the same frontier:

$$[(A \to A) \to ((A \to A) \to B),\ [A \to A,\ A \to A]] : (A \to A) \to B,$$

$$[[(A \to A) \to ((A \to A) \to B),\ A \to A],\ A \to A] : B,$$

where in the first deduction both rules are used, and in the second one (MP) is applied twice. The reader easily sees unary rules like Necessitation $A/\Box A$ and Generalization $A/\forall x A$ also destroy determinacy. The frontier property, then, seems to be characteristic of systems based on (MP) as the only inference rule. Does it suggest any structural explanation for the prominent role of this rule in both Logic and Linguistics?

2. Formal grammars as deductive systems

Connections between logical deductions and grammatical derivations, examined in the preceding section for Logical Grammar, can be seen in many other kinds of formal grammar inhabiting Mathematical Linguistics. In this section we survey basic forms of this correspondence.

First, we introduce some useful logical notions. A fundamental logical notion is *a consequence relation*, defined as a relation $\Gamma \vdash A$, where Γ is a set of formulas, and A is a formula. Both proof theory and linguistics need a refined notion of *a sequential consequence relation*: a relation $\Gamma \vdash A$, where Γ is a finite string of formulas, and A is a formula, which satisfies conditions (C1) and (C2) from section 1. The intended meaning of $\Gamma \vdash A$ is: there is a derivation tree of A from the string of assumptions Γ within the given deductive system.

For any deductive system $\mathcal{S}$, the corresponding relation $\Gamma \vdash_{\mathcal{S}} A$ can be axiomatized in the following way. Each inference rule:

$$\text{(R)}\ \frac{A_1, \ldots, A_n}{B}$$

gives rise to the consequence pattern:

$$\text{(R')}\ A_1, \ldots, A_n \vdash B.$$

Axioms of the system (if there are any) are represented as degenerate patterns $\vdash B$, that means, $\Lambda \vdash B$, where Λ stands for the empty string. The relation $\vdash_{\mathcal{S}}$ can be defined as the smallest sequential consequence relation which contains all consequence patterns determined by system $\mathcal{S}$. Equivalently, it can be presented in the form of a sequential deductive system whose axioms are:

$$\text{(AC1)}\ A \vdash A$$

and all consequence patterns corresponding to axioms and inference rules of $\mathcal{S}$, and the only deduction rule is *the cut rule*:

$$\text{(CUT)}\ \frac{\Gamma_1, A, \Gamma_2 \vdash B;\ \Delta \vdash A}{\Gamma_1, \Delta, \Gamma_2 \vdash B}.$$

Clearly, (AC1) and (CUT) simulate (C1) and (C2), respectively.

Formal grammars are usually defined by means of two components:

(I) a (finite) set of *lexical assumptions* $v : A$ such that v is a lexical atom (symbol, word, etc.), and A is a formula which denotes a *grammatical category*,

(II) a *derivational system*, consisting of production rules, transition rules, and so like, which is to generate complex expressions of respective categories.

In the framework of this paper, (II) is represented as a logical deductive system. The role of (I) is to translate logical deductions into grammatical judgements, roughly, according to the following correspondence: the expression $v_1 \dots v_n$ is assigned category A (write: $v_1 \dots v_n : A$) if, for some lexical assumptions $v_1 : A_1, \dots, v_n : A_n$, the pattern $A_1, \dots, A_n \vdash A$ is derivable in the deductive system. Thus, lexical assumptions actually take the part of hypothetical assumptions for logical deductions provided by (II).

Let us look at concrete examples. Following the linguistic tradition, we use two conditionals: $A \to B$ (*the right conditional*) and $B \leftarrow A$ (*the left conditional*). The former takes its argument on the left, and the latter on the right, which gives rise to inference patterns:

$$(\mathrm{MP}{\to})\ A,\ A \to B \vdash B,\ \ (\mathrm{MP}{\leftarrow})\ B \leftarrow A,\ A \vdash B.$$

In functional terms, $A \to B$ is the type of *left-looking* functors which together with an A-argument (on the left) form a complex expression of type B, while $B \leftarrow A$ is the type of *right-looking* functors which together with an A-argument (on the right) form a complex expression of type B. For instance, the type of Noun Phrase is S←VP (VP is the type of Verb Phrase), and VP equals PN→S (PN is the type of Proper Noun). CAUTION: According to this convention, all conditionals in section 1 should be reversed; also for multiple-argument functors, we should write $B \leftarrow A_1 \dots A_n$ instead of $A_1 \dots A_n \to B$.

A *Finite State Acceptor* (FSA) admits finitely many lexical assumptions $v : A \to B$ and axioms $\Lambda \vdash A$, where A, B are *states*. Axioms provide *initial states*, and lexical assumptions correspond to *transitions*. The only inference rule is (MP→). One distinguishes a finite set of *terminal states*. The string $X \in V^\star$ (V is the lexicon) is *accepted* by the FSA, if $X : A$, for some terminal state A. Let us recall that $X : A$ holds if, and only if, there is an (MP→)-deduction of A from some string Γ of formulas corresponding to atoms from X according to lexical assumptions. A crucial property is that axioms must appear on the left-most leaf of a deduction tree, since no nested conditionals are admitted.

In the above picture, grammar still remains external to the very logic; it is lexical assumptions which coordinate logical deductions with grammatical judgements. A more tight connection of grammar and logic can be reached in

the framework of Labelled Deductive Systems (LDS's), elaborated in GABBAY [18]. An LDS operates with *labelled formulas* $X : A$ such that A is a logical formula, and X is a label. Rules of the system perform at the same time a logical transformation of formulas and an algebraic operation on labels. For instance, the labelled (MP$\rightarrow$) is:

$$(\text{LMP}\rightarrow)\ X : A;\ Y : A \rightarrow B \vdash XY : B;$$

here we interpret X, Y as finite strings, and XY is the concatenation of strings X and Y. Clearly, an FSA can be represented by the LDS whose initial postulates are axioms $\Lambda : A$ and lexical assumptions $v : A \rightarrow B$, and (LMP$\rightarrow$) is the only deduction rule.

To see the real power of the logical interpretation of grammar, we discuss logical systems corresponding to other basic formalisms of Mathematical Linguistics.

Admitting axioms $\Lambda \vdash A \rightarrow B$, where A, B are states, yields FSA's with Λ-moves; they may change state A into state B without moving along the string. These new axioms can be eliminated: for any assumption $v : C \rightarrow D$, one introduces all assumptions $v : C \rightarrow D'$ such that there is a transition from D to D' by Λ-moves only, and for any axiom $\Lambda \vdash A$, one adds all axioms $\Lambda \vdash B$ such that B can be derived from A in a similar way. It is a known result that each FSA with Λ-moves can be simulated by a standard FSA. In logical terms, an n-step transition from D to D' amounts to applying (SYL) n times. Although (SYL) is not admissible in the deductive system of FSA ((MP$\rightarrow$) is the only rule), one easily checks the extended system with (SYL) and (MP$\rightarrow$) is conservative over the (MP$\rightarrow$)-system, as concerns patterns $\Gamma \vdash A$, where A is a state (not a transition). The conservativity of (SYL) over (MP$\rightarrow$) for unnested conditionals is the logical core of the equivalence of FSA's with Λ-moves and standard FSA's.

Admitting axioms with nested conditionals of the form:

$$\Lambda \vdash A_1 \rightarrow (A_2 \rightarrow \cdots (A_n \rightarrow B) \ldots),$$

where $A_1, \ldots, A_n, B$ are atomic, leads to the world of *Context-Free Grammar* (CFG). Now, atomic formulas represent *nonterminal symbols* which are to denote grammatical categories. The lexical component consists of assumptions $v : A$ assigning nonterminals A to lexical atoms v. The derivational system is based upon *production rules* which rewrite a string of nonterminals into a single nonterminal (the standard 'generative' presentation of CFG assumes the opposite direction of rewriting: from a single symbol to a

string, but our deductive approach prefers the 'analytic' routine). Clearly, the production rule:

$$A_n \dots A_2 A_1 \vdash B$$

can be replaced with the axiom written above. It is certainly a matter of taste whether production rules are represented by axioms or consequence patterns (inference rules). A logical hint to prefer axioms is to preserve the logical character of inferences: deduction rules should be logically valid, as e.g. (MP$\rightarrow$), (SYL), while axioms (and assumptions) may be nonlogical. Thus, each CFG gives rise to a deductive system with nonlogical axioms of the above form, assumptions $v : A$, and the only inference rule (MP$\rightarrow$).

As a less trivial application of the logical approach, we give a logical reconstruction of the known equivalence of CFG's and *Recursive Transition Networks* (RTN's) (see GAZDAR and MELLISH [19]). An RTN admits axioms $\Lambda \vdash A$, assumptions $v : A$ and special assumptions $A : B \rightarrow C$, where A, B, C are states. Like FSA, the RTN proceeds the string from the left to the right, but transitions are executed as follows. The subinterval Y of the string XYZ is treated as an atom which yields the transition $Y : B \rightarrow C$ provided the RTN assigns state A to Y and $A : B \rightarrow C$ is a special assumption. Less formally, after the RTN has passed the initial interval X and reached state B, it may alter the mode; starting from an initial state, it proceeds the string, till state A will be reached for the source string Y. Then, the RTN skips Y, changing state B to state C. RTN's are especially suitable to model recursive constructions in language. A simple example is:

- Axiom: $\Lambda \vdash I$ (I is the initial state),
- Assumptions: Mary, John, Jim: NP; sees, knows: V; that: WH,
- Special assumptions (transitions):
  NP: $I \rightarrow A$, VP: $A \rightarrow$S, V: $A \rightarrow B$,
  WH: $B \rightarrow C$, S: $C \rightarrow$S, NP: $B \rightarrow$S.

Here A, B, C are auxiliary states. One easily checks this RTN assigns state S (Sentence) to strings:

- Mary sees John,
- Mary sees that John knows that Mary knows Jim,
- Mary sees that John knows that Mary sees that John knows that Mary knows Jim,

ans so on. For the second string, starting from I, the RTN assigns C to *Mary sees that*, next it alters the mode to assign state S to the remaining interval *John knows that Mary knows Jim*, which is executed in a similar way. Incidentally, this RTN can still be simulated by an FSA. But, if one also regards recursion in NP to include expressions:

- the man who sees Jim,
- the man who knows that Mary knows Jim,

and so like, then the resulting RTN will already surpass the capacity of FSA's. We also note that initial states (axioms) need not appear; due to the form of lexical assumptions, the RTN can start the run from the state associated with the first symbol of the string.

Our main task is to explain the equivalence of RTN's and CFG's. The LDS corresponding to the given RTN is defined by:

- Axioms $\Lambda : A$, for initial states A (if there are any),
- Assumptions $v : A$ (reporting lexical assumptions),
- Assumptions $A : B \to C$ (reporting transitions),
- Rule (LMP$\to$).

This LDS is essentially an FSA-system in which states are included among lexical atoms. The recursive behavior will be attained, after one adds *the labelled cut rule*:

$$\text{(LCUT)}\ XAZ : B;\ Y : A \vdash XYZ : B,$$

where X, Y, Z are strings of lexical atoms and states. Notice that (LCUT) makes sense for those LDS's only which allow logical formulas to take the part of labels. Clearly, if X is a string of lexical atoms, and A is a state, then $X : A$ is derivable in the LDS (with (LCUT)) if, and only if, the RTN assigns state A to X. That also remains true, if the LDS will be enriched with identity assumptions $A : A$ which are related to (AC1) and mirror the double role of states: as formulas and labels. We denote this LDS by S(RTN).

Now, each CFG can be presented in the Chomsky Normal Form, with all production rules of the form $AB \vdash C$. Instead of replacing them with axioms:

$$\Lambda \vdash B \to (A \to C),$$

we can equivalently represent them in the form of unary inference rules:

$$B \vdash A \to C,$$

which eliminates nested conditionals. The corresponding LDS S(CFG) is given by:

- Axioms $\Lambda : A$, for Λ-rules $\Lambda \vdash A$ (if there are any),
- Assumptions $v : A$ (reporting lexical assumptions),
- Identity assumptions $A : A$ (here needed to include nonterminals into labels),
- U-rules $X : B \vdash X : A \to C$ (executing the unary rules),
- Rule (LMP$\to$).

The equivalence of RTN's and CFG's follows from the fact that systems S(RTN) and S(CFG) yield the same labelled formulas $X : A$ (we assume axioms and assumptions are the same in both systems, and U-rules of S(CFG) are naturally related to special assumptions of S(RTN)). This fact is not obvious, since systems S(RTN) and S(CFG) are apparently different. S(RTN) admits (LCUT) which is not a rule of S(CFG), and S(CFG) admits U-rules which are stronger than special assumptions of S(RTN) (notice that formulas A, B in (LCUT) must be atomic, hence U-rules cannot be derived in S(RTN)). To unify these systems, we add (LCUT) to S(CFG) and U-rules to S(RTN); the resulting systems are denoted by S(CFG)+CUT and S(RTN)+U, respectively. Clearly, the latter systems are equivalent. Now, the equivalence of RTN's and CFG's follows from two logical *elimination theorems*:

(T1) S(CFG) is closed under (LCUT) (equivalently: (LCUT) can be eliminated from any derivation in S(CFG)+CUT),

(T2) in S(RTN)+U, U-rules can be eliminated from any derivation of $X : A$ with A atomic.

(T1) is proven by a straightforward induction on derivations in S(CFG) (that is much simpler than cut-elimination proofs for Gentzen style systems, since LDS's admit no logical transformations of the antecedent formulas). To prove (T2) assume the U-rule $Y : C \vdash Y : B \to D$ be applied in a derivation of $X : A$. Since $B \to D$ is not atomic, the node $Y : B \to D$ must be a premise of a next rule; this next rule can be neither (LCUT), nor an U-rule, hence

it must be (LMP→) with premises $Z : B$ and $Y : B \to D$. We modify the derivation: first, we apply (LMP→) with premises $Z : B$ and $C : B \to D$ (notice the latter is a special assumption of S(RTN)), and second, we apply (LCUT) with premises $ZC : D$ and $Y : C$, which yields $ZY : D$ as in the initial derivation.

By (T1), if $X : A$ is derivable in S(RTN), then $X : A$ is derivable in S(CFG). Conversely, each formula $X : A$ derivable in S(CFG) is also derivable in S(CFG)+CUT equal to S(RTN)+U, hence it is derivable in S(RTN), by (T2). That yields the equivalence of RTN's and CFG's. Since *Push-Down Acceptors* (PDA's) are merely a reformulation of RTN's, also the equivalence of CFG's and PDA's may be regarded as a consequence of the above elimination theorems.

A variety of grammar formalisms arises from the ones discussed above by admitting more complex formulas, axioms, rules, etc.

Categorial Grammar admits assumptions $v : A$, where A may be an arbitrary formula built of $\to$ and $\leftarrow$. In the classical form, introduced by BAR-HILLEL [3], admissible deductions are pure MP-deductions, precisely, deductions based on (MP→) and (MP←). We refer to these grammars as *Basic Categorial Grammars* (BCG's). Clearly, BCG is a straightforward generalization of the Ajdukiewicz paradigm, discussed in section 1. Yet, the presence of two conditionals breaks down the frontier property. These are two deductions with the same frontier which yield different outcomes:

$$[[(B \leftarrow (A \to A)) \leftarrow A,\ A],\ A \to A] : B,$$

$$[(B \leftarrow (A \to A)) \leftarrow A,\ [A,\ A \to A]] : B \leftarrow (A \to A).$$

In linguistic terms, one says that BCG's are both *categorially* and *structurally ambiguous*. Categorial ambiguity means that one string can be assigned more than one category. Structural ambiguity means that one string can be assigned to a certain category by means of at least two different deduction trees (which generate different structures on this string). Interestingly, MP-deductions with two conditionals are still categorially unambigous for atomic outcomes: $\Gamma \vdash A$ and $\Gamma \vdash B$ entail $A = B$, if A, B are atomic. That follows from the obvious fact:

(OC) if $\Gamma \vdash A$ by means of (MP→) and (MP←) only, then each atomic formula has a even number of occurrences in $\Gamma \vdash A$ (as a subformula).

Accordingly, if the BCG is *deterministic*, that means, each lexical atom is assigned at most one type, then each string is assigned at most one atomic type (basic grammatical category).

The key idea of Categorial Grammar is to store all the linguistic information in lexical assumptions, whereas the derivational machinery totally relies on some logically universal principles. In the BCG format, lexical assumptions provide syntactic types of lexical atoms, and types of complexes are to be deduced, using bidirectional Modus Ponens. This kind of grammatical description is naturally related to a type-theoretic semantics for the language under consideration. Both $A \to B$ and $B \leftarrow A$ are semantically interpreted as types of functions from A-objects to B-objects. Thus, (MP$\to$) and (MP$\leftarrow$) mirror the universal *function application principle*: function F from A to B applied to an A-object yields a B-object. If semantic denotations of all lexical atoms are fixed, then denotations of complex expressions are to be computed by function application according to admissible MP-deductions.

There are no a priori reasons to restrict Categorial Grammar to pure MP-deductions. In literature, there were proposed many extended systems, admitting other inference rules, as e.g.:

$$\text{(SYL}\to\text{)}\ \frac{A \to B;\ B \to C}{A \to C},\ \text{(SYL}\leftarrow\text{)}\ \frac{C \leftarrow B;\ B \leftarrow A}{C \leftarrow A},$$

$$\text{(TR}\to\text{)}\ \frac{A}{(B \leftarrow A) \to B},\ \text{(TR}\leftarrow\text{)}\ \frac{A}{B \leftarrow (A \to B)}.$$

Clearly, (SYL$\to$) and (SYL$\leftarrow$) are directional versions of Conditional Syllogism, and they are semantically executed by function composition. (TR$\to$) and (TR$\leftarrow$) are *Type Raising Rules*; semantically, they correspond to the tranformation F which to each A-object x assigns the mapping F_x that maps each function f from A to B to the B-object $f(x)$. For instance, each PN-denotation, i.e. an individual, is assigned an NP-denotation, where:

$$\text{NP}{=}\text{S}{\leftarrow}(\text{PN}{\to}\text{S}).$$

Here PN$\to$S is the type of Verb Phrase (Unary Predicate), as e.g. *cries* in *Susan cries*, hence NP is the type of Noun Phrase, as e.g. *some girl* in *some girl cries*. The transformation F shifts each individual x to the NP-denotation F_x that assigns the truth value $f(x)$ to each VP-denotation f. If sets are identified with characteristic functions, then an individual x is transformed into the family of all sets of individuals which contain x (the principal ultrafilter generated by x). That reminds the Leibniz idea of identifying individuals by their properties, and HIŻ [25] provides a logical discussion of this transformation in relation to Russell's antinomy.

Thus, a categorial grammar admitting (TR$\leftarrow$) can 'lift up' each PN expression to type NP, which enables one, for instance, to interpret complexes

like *John and some girl*, *John but not every student* as Boolean operations on NP-denotations. A standard illustration of (SYL←) is the following. Consider the NP-expression *a tall girl*. It consists of a Determiner, an Adjective, and a Noun (type N), and the first two types are:

$$\text{Det} = \text{NP}{\leftarrow}\text{N},\ \text{Adj} = \text{N}{\leftarrow}\text{N}.$$

Now, the pure MP-analysis produces the deduction:

$$[\text{NP}{\leftarrow}\text{N},\ [\text{N}{\leftarrow}\text{N},\ \text{N}]]\text{: NP},$$

which recognizes *a tall girl* as an NP with the structure:

$$[\text{a},\ [\text{tall},\ \text{girl}]].$$

The interval *a tall* is not a constituent. If (SYL←) is allowed, then the alternative deduction:

$$[[\text{NP}{\leftarrow}\text{N},\ \text{N}{\leftarrow}\text{N}],\ \text{N}]\text{: NP}$$

is also possible. Here (SYL←) is responsible for recognizing *a tall* as a constituent of type NP←N = Det, which agrees with the linguistic practice of qualifying such expressions to *restricted determiners* (see KEENAN and FALTZ [27]).

We focus on the logically most significant aspects of the above framework. As it has been the case for BCG, the deductive machinery remains logical: new inference rules do not depend on the particular language, but they rely upon some universal properties of logical constants (here: conditionals) and can be executed by means of general semantic transformations. In the next section we show that these transformations can be defined in an appropriate version of Typed Lambda-Calculus, a basic formalism for logical semantics. Another interpretation of new rules appeals to algebras of residuation (residuated algebras); see [7, 11].

Besides logical inference rules, exemplified by (SYL→), (SYL←), etc., one may assume *logical axioms*, as e.g.

$$(\text{ASYL}{\rightarrow})\ (B \rightarrow C) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)),$$

which is a version of Conditional Syllogism. Of course, axiom (ASYL→) can simulate the action of rule (SYL→), but it produces more consequences. If D abbreviates the formula above, then the axiom $\Lambda \vdash D$ together with (MP→) $D,\ D \rightarrow E \vdash E$ yields $D \rightarrow E \vdash E$ by (CUT), which is not derivable

with (SYL←) only. In semantics, logical axioms can be interpreted by typed combinators (e.g. the combinator B represents (ASYL→)) which provide certain standard semantic transformations. This style of language description is exploited in *Combinatory Categorial Grammar* of STEEDMAN [46], and we shall write more on this approach in the next section.

In a historical development, new logical axioms and rules were affixed to Categorial Grammar in order to improve the semantic adequacy of grammatical structures. Conditional Syllogism was assumed in GEACH [20], while Type Raising Rules can be credited to MONTAGUE [35]. As early as in LAMBEK [28], they were justified within a deduction theory, naturally extending MP-systems. The key idea is to consider sequential consequence relations with *introduction rules* for conditionals:

$$(\mathrm{I}{\rightarrow})\ \frac{A, \Gamma \vdash B}{\Gamma \vdash A \rightarrow B},$$

$$(\mathrm{I}{\leftarrow})\ \frac{\Gamma, A \vdash B}{\Gamma \vdash B \leftarrow A}.$$

Observe that with the presence of (CUT) rules (MP→) and (MP←) are equivalent to the following *elimination rules*:

$$(\mathrm{E}{\rightarrow})\ \frac{\Gamma \vdash A;\ \Delta \vdash A \rightarrow B}{\Gamma, \Delta \vdash B},$$

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

The interplay of introduction-elimination rules is characteristic of logical Natural Deduction Systems. Accordingly, sequential deductive systems admitting I-rules and E-rules for conditionals are certain Natural Deduction extensions of pure MP-systems, considered before. VAN BENTHEM [4] was the first who exploited the Natural Deduction approach to grammar formalisms in a fully conscious way, an some main lines of his theory will be sketched in the next section.

Using (I→), (I←), one easily derives the afore mentioned SYL-rules and TR-rules. For (SYL→), from:

$$A,\ A \rightarrow B,\ B \rightarrow C \vdash C,$$

which holds by (MP→) and (CUT), we infer:

$$A \rightarrow B,\ B \rightarrow C \vdash A \rightarrow C,$$

by (I→). The remaining cases are left to the reader.

In the above discussion, we have considered purely conditional formulas. With new logical constants, deductive systems become more flexible also for linguistic purposes. Already LAMBEK [28] uses *product* $\circ$, corresponding to juxtaposition of strings, which is regulated by rules:

$$(\mathrm{I}\circ)\ \frac{\Gamma \vdash A;\ \Delta \vdash B}{\Gamma, \Delta \vdash A \circ B},$$

$$(\mathrm{I}\circ\mathrm{L})\ \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, A \circ B, \Delta \vdash C}.$$

The second rule is a Gentzen-style left-introduction rule, not in the Natural Deduction format. Gentzen-style systems for the Lambek calculus are discussed in [8, 6, 36]. One can also admit *conjunction* $\wedge$ with rules:

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

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

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

Continuing this way leads to still richer sequential logics, including Linear Logics of GIRARD [21], Bilinear Logics of LAMBEK [31] and YETTER [51], Arrow Logics of VAN BENTHEM [6], Modal Lambek Systems of MOORTGAT [37], Labelled Deductive Systems of GABBAY [18], and others. Here we point out some simple connections between these enriched logics and formal grammars.

Generative Grammars admit production rules of the form:

$$A_1 \dots A_m \vdash B_1 \dots B_n,$$

which can be simulated in sequential deductive systems by patterns:

$$A_1 \dots A_m \vdash B_1 \circ \dots \circ B_n$$

or, in the MP-format, by axioms:

$$A_m \to (A_{m-1} \to \dots (A_1 \to B_1 \circ \dots \circ B_n) \dots).$$

Without product, there is no natural translation of general rewriting rules into logical schemes (but, see [9] for an 'artificial' translation into purely conditional schemes).

Context-Sensitive Grammars use context-dependent production rules, like:

$$C, A_1, \ldots, A_n \vdash C, B;$$

that means, the CF-rule $A_1, \ldots, A_n \vdash B$ may be applied only if A_1 is preceded by symbol C. Again, using product, the above rule can be replaced with the axiom:

$$A_n \to (A_{n-1} \to \cdots (C \to C \circ B) \ldots),$$

but without product no direct translation is possible.

Conjunction can be used to collapse a finite number of category symbols into one symbol. Thus, a nondeterministic categorial grammar which assigns, say, $A_1, \ldots, A_n$ to v can be transformed into a deterministic one which assigns $A_1 \wedge \cdots \wedge A_n$ to v. Some work on categorial grammars with conjunction and other booleans has been done by KANAZAWA [26]. In the theory of FSA's, conjunction formulas can be applied to give a logical interpretation of the classical construction of a deterministic FSA equivalent to the given nondeterministic FSA. Recall that a deterministic FSA associates to each lexical atom v a mapping T_v from states to states such that $v : A \to B$ is a transition if, and only if, $B = T_v(A)$ (intuitively, the next state is uniquely determined by the current state and the scanned symbol); one also assumes there is exactly one initial state. Scott's powerset construction modifies the given (nondeterministic) FSA into a deterministic one by taking sets of states of the former FSA as states of the latter FSA. For each atom v, the mapping T_v sends each state-set W into the set of all states B such that, for some $A \in W$, $v : A \to B$ is admissible by the former FSA. The set of all initial states of the former FSA is the only initial state of the latter FSA, and final states of the latter FSA are those state-sets which contain at least one final state of the former FSA.

If $S = \{A_1, \ldots, A_n\}$ is a nonempty set of states, then $\bigwedge W$ denotes the conjunction $A_1 \wedge \ldots \wedge A_n$. Each transition $v : W \to W'$ of the resulting deterministic FSA can be modelled as the labelled formula $v : \bigwedge W \to \bigwedge W'$, and the only initial state of this FSA can be represented as $\bigwedge I$, where I is the set of initial states of the nondeterministic FSA. Now, the equivalence of the nondeterministic FSA and the deterministic one is based on the following property of derivability with rules (MP$\to$), (I$\wedge$), (I$\wedge$L1), (I$\wedge$L2):

$$\bigwedge I, \bigwedge I \to \bigwedge W_1, \bigwedge W_1 \to \bigwedge W_2, \ldots, \bigwedge W_n \to \bigwedge W \vdash \bigwedge W,$$

holds true, for transitions $v_1 : \bigwedge I \to \bigwedge W_1$, ..., $v_n : \bigwedge W_n \to \bigwedge W$ of the deterministic FSA if and only if, for every $B \in W$, there are $A_0 \in I$,

$A_1 \in W_1, \ldots, A_n \in W_n$ such that $v_1 : A_0 \to A_1$, ..., $v_n : A_n \to B$ are transitions of the nondeterministic FSA and:

$$A_0, A_0 \to A_1, A_1 \to A_2, \ldots, A_n \to B \vdash B$$

holds true (the latter is a pure MP-deduction).

Throughout this section we have departed rather far from the simple picture of section 1 where the focus is MP-trees, directly modelling grammatical structures. However, sequential deductive systems can be refined to make derivation structures explicit. One considers sequents $\Gamma \vdash A$ in which Γ is a bracketed string of formulas. Then, (MP$\to$) and (MP$\leftarrow$) take the form:

$$\text{(SMP}\to\text{)}\ [A, A \to B] \vdash B,\ \text{(SMP}\leftarrow\text{)}\ [B \leftarrow A, A] \vdash B,$$

and the rule (CUT) is changed into:

$$\text{(SCUT)}\ \frac{\Gamma[A] \vdash B;\ \Delta \vdash A}{\Gamma[\Delta] \vdash B},$$

where $\Gamma[A]$ denotes a bracketed string Γ with a designated occurrence of formula A, and $\Gamma[\Delta]$ denotes the substitution of Δ for the designated occurrence of A in Γ. For the case $\Delta = \Lambda$, that means, A is an axiom, the substitution removes the node A from the tree. Now, if $\Gamma \vdash A$ is derivable, then grammatical structures of expressions are produced by replacing each leave A by a lexical atom v such that $v : A$ is a lexical assumption. Again, this correspondence can be inserted in the very logic by applying Labelled Deductive Systems. The structured rule (LMP$\to$) is:

$$\text{(SLMP}\to\text{)}\ \frac{X : A,\ Y : A \to B}{[X, Y] : B},$$

where X, Y denote bracketed strings of lexical atoms, and similarly for $\leftarrow$.

Structures may contain more information; in the presence of two conditionals, it is natural to make the position of the functor explicit in the structure. We write $[X, Y]_1$, if the functor is X, and $[X, Y]_2$, if the functor is Y. Structures sensitive to the position of functors are called *functor-argument structures*. Now, SMP-rules should be written:

$$\text{(SMP}\to\text{)}\ [A, A \to B]_2 \vdash B,\ \text{(SMP}\leftarrow\text{)}\ [B \leftarrow A, A]_1 \vdash B,$$

and similarly for labelled patterns.

Structured sequential and labelled systems make it possible to directly account for the correspondence between logical deductions and grammatical

structures. As shown in [9], Lambek style grammars are *structurally complete* in the sense that they provide all possible functor-argument structures on the admissible strings. On the contrary, BCG's are structurally more restrictive than CFG's: the latter generate all phrase language of finite index (with respect to the basic intersubstitutability congruence), while the former only those phrase languages of finite index which satisfy the additional requirement that the least distance from each node to the leaves is bounded for the language.

At the end, we note that more advanced grammar formalisms cannot be modelled by purely sentential logics, discussed in this section. Computational Linguistics uses First-Order Logic to execute Unification Systems, as e.g. Definite Clause Grammars (Pereira and Shieber [41]) and Unification Categorial Grammars (Uszkoreit [49]). Actually, many linguistic goals can be accomplished with the aid of systems essentially weaker than the Classical First-Order Logic, namely first-order expansions of the sequential systems discussed above.

3. Natural Deduction, lambdas and semantics

In section 2 we have mentioned rules (I$\rightarrow$), (I$\leftarrow$), (E$\rightarrow$) and (E$\leftarrow$) as introduction and elimination rules for directed conditionals. Introduction and elimination rules for logical constants are characteristic of Natural Deduction systems. *The Curry-Howard isomorphism* establishes a strict correspondence between Natural Deduction proofs and typed lambda terms [22]. Typed lambda terms are a canonical encoding for semantic denotations of linguistic expressions. Thus, grammatical derivations represented as Natural Deduction proofs allow one to directly compute semantic denotations of the derived expressions, which is a leitmotive of Computational Semantics.

The logico-linguistic side of the Curry-Howard isomorphism was thoroughly elaborated by van Benthem in a series of papers and books, the most representative ones being [4, 5, 6], and his collaborators and students, as e.g. Zwarts [53], Moortgat [36], Wansing [50], Sanchez Valenzia [42] and Hendriks [23]. Some origins can be traced back to Curry [14] and Montague [35]. Actually, most people interested in natural language semantics used typed lambda terms for the semantic representation of language, and certain ideas from their enterprises are closely related to this subject. A typical example is Cresswell [13] whose theory of lambda-categorial languages studies lambda terms as semantic structures of expressions, but the deductive aspects are hidden, and the same

might be said of the American semantic school (D. Davidson, D. Lewis, B. Hall-Partee, D. Dowty and others). However, the parallel problem of computing semantic denotations along the derivation tree of a Phrase Structure Grammar has been investigated in e.g. LEWIS [34], PARTEE [40], SUPPES [47]. An essentially equivalent approach with Hilbert style proofs instead of Natural Deduction proofs and typed combinators instead of typed lambda terms has been developed by STEEDMAN [1, 45, 46] with origins in CURRY [14] and SHAUMYAN [44]. More advanced theories which confront unification systems with the second-order lambda calculus have been proposed in Unification Categorial Grammar [49]; also see LEISS and EMMS [33]. Also, deeper connections between grammatical derivations, logical deductions and category-theoretic formalisms, the latter being algebraically oriented extensions of the lambda calculus, can be found in recent papers of LAMBEK [29, 30, 31].

Following the approach of VAN BENTHEM [4], we consider *semantic types* which are formed out of atomic types **e** (entity), **t** (truth value), and possibly others, by means of the rule: if A and B are types, then $A \to B$ is a type (VAN BENTHEM writes (A, B)). To each type A one assigns a nonempty set D_A, called *the ontological category of type A*. D_e and D_t are the set of entities (i.e. individuals) and the set of (classical) truth values, respectively; $D_{A \to B}$ is the set of all functions from D_A to D_B. Linguistic expressions are assigned semantic types according to the ontological types of their (intended) denotations. Below we list semantic types corresponding to some basic grammatical categories.

PN	$\mathbf{e}$
S	$\mathbf{t}$
VP	$\mathbf{e} \to \mathbf{t}$
TV	$\mathbf{e} \to (\mathbf{e} \to \mathbf{t})$
N	$\mathbf{e} \to \mathbf{t}$
NP	$(\mathbf{e} \to \mathbf{t}) \to \mathbf{t}$
Det	$(\mathbf{e} \to \mathbf{t}) \to ((\mathbf{e} \to \mathbf{t}) \to \mathbf{t})$
Prep	$(\mathbf{e} \to \mathbf{t}) \to (\mathbf{e} \to \mathbf{t})$
Adv	$(\mathbf{e} \to \mathbf{t}) \to (\mathbf{e} \to \mathbf{t})$

Here, as usual, PN stands for Proper Noun, S for Sentence, VP for Verb Phrase, TV for Transitive Verb (Phrase), N for Common Noun, NP for Noun Phrase, Det for Determiner, Prep for Prepositional Phrase, Adv for Adverb. Observe that different syntactic categories may fall into a common semantic type, witness VP and N, Prep and Adv. Clearly, the above correspondence is merely a first approximation. Intensional treatments use

the special type **s** for *possible worlds*, and each *extensional* type A is accompanied with type $\mathbf{s} \to A$ (the type of *senses* of objects of type A); see MONTAGUE [35]. KEENAN and FALTZ [27] prefer to use **t** and **NP** as the basic semantic types, where **NP** is the family of NP-denotations, i.e. families of sets of individuals, while VP-expressions are assigned type $\mathbf{NP} \to \mathbf{t}$, being understood as the type of complete homomorphisms from the Boolean algebra **NP** to the Boolean algebra of truth values. For the sake of brevity, we stick at the simplified, purely extensional hierarchy of types, based on **e** and **t**.

Typed lambda-terms (shortly: terms) are defined as follows. For each type A, there are infinitely many variables of type A: $x_A, y_A, z_A, \ldots$, and, possibly, some constants of type A. One defines *terms of type* A by the following recursion:

(TER.1) variables and constants of type A are terms of type A,

(TER.2) if s is a term of type $A \to B$, and t is a term of type A, then (st) is a term of type B,

(TER.3) if s is a term of type B, and x is a variable of type A, then (λxs) is a term of type $A \to B$.

(TER.2) and (TER.3) are the formation rules of *application* and *lambda abstraction*, respectively. Given a valuation μ which to each variable and constant assigns an object of the corresponding type, one naturally extends it to a function which to each term t of type A assigns an object $\mu(t) \in D_A$ (see e.g. [6, 22]).

In natural language semantics, terms take the part of semantic structures of expressions. A key idea, going back to MONTAGUE and CRESSWELL, is to regard lexical units as constants; complex terms are semantic structures of expressions which are obtained from these terms by dropping all formal symbols (lambdas, variables, parentheses). A baby example is the sentence *John works* with the structure:

$$((\lambda x_{\mathbf{e}\to\mathbf{t}}(x_{\mathbf{e}\to\mathbf{t}}\mathbf{John}))\,\mathbf{works});$$

here, **John** and **works** are constants of type **e** and $\mathbf{e} \to \mathbf{t}$, respectively. Clearly, the above structure is a term of type **t**. Analogously, the sentence *John hits Paul* is given the structure:

$$((\lambda x_{\mathbf{e}\to\mathbf{t}}(x_{\mathbf{e}\to\mathbf{t}}\mathbf{John}))(\mathbf{hits\ Paul}));$$

here, **hits** is a constant of type $\mathbf{e} \to (\mathbf{e} \to \mathbf{t})$.

Both the examples above illustrate interchanging functors and arguments by lambda transformations. For any term t of type A, by t^B we denote the term:

$$(\lambda x_{A\to B}(x_{A\to B}t))$$

of type $(A \to B) \to B$; here, $x_{A\to B}$ is a variable not free in t. If s is an arbitrary term of type $A \to B$, then (st) and $(t^B s)$ have the same denotation, that means, $\mu((st)) = \mu((t^B s))$, for any valuation μ. Consequently, t^B takes the part of a functor which with argument s of type $A \to B$ yields the same value as functor s with argument t. Using this notation, the above structures can be written:

$$(\mathbf{John^t\ works});\ (\mathbf{John^t}\ (\mathbf{hits\ Paul})).$$

Another linguistically significant action of lambda-terms is to provide the composition of two functions. If s, t are terms of type $B \to C$ and $A \to B$, respectively, then the term:

$$u = (\lambda x_A(s(tx_A))),$$

of type $A \to C$, fulfils $\mu(u) = \mu(s) \circ \mu(t)$. For instance, consider constants **not**, **every** and **girl**, of type $\mathbf{t} \to \mathbf{t}$, $(\mathbf{e} \to \mathbf{t}) \to \mathbf{NP}$ and $\mathbf{e} \to \mathbf{t}$, respectively, where **NP** stands for the semantic type corresponding to NP. Then, the negative NP *not every girl* is given the structure:

$$(\lambda x_{\mathbf{e}\to\mathbf{t}}(\mathbf{not}((\mathbf{every\ girl})x_{\mathbf{e}\to\mathbf{t}}))),$$

of type **NP**. Notice that pure application cannot join the three constants into a meaningful whole. Also, complexes like *John and Mary*, *works and rests* etc. can correctly be interpreted with **and** of the basic type $\mathbf{t} \to (\mathbf{t} \to \mathbf{t})$. For the first case, we form the auxiliary structure:

$$(\lambda x_{\mathbf{e}\to\mathbf{t}}((\mathbf{and}(x_{\mathbf{e}\to\mathbf{t}}\mathbf{John}))(x_{\mathbf{e}\to\mathbf{t}}\mathbf{Mary}))),$$

of type **NP**, which can be modified to an equivalent structure by interchanging functor **and** and its first argument, and the second case is treated in a similar way. We refer the reader to [36, 6, 11] for further semantic roles of lambda-terms.

Typed lambda-terms can equivalently be represented as Natural Deduction proofs in the positive logic of implication, based upon axioms (Id) $A \vdash A$ and rules:

$$(\mathrm{E}^{\star}\ \to)\ \frac{\Gamma \vdash A \to B;\ \Delta \vdash A}{\Gamma, \Delta \vdash B},$$

$$(\mathrm{I}^{\star} \rightarrow) \ \frac{\Gamma \vdash B}{\Gamma' \vdash A \rightarrow B},$$

where Γ' results from dropping a finite number (possibly equal to zero) of occurrences of A in string Γ. The correspondence is quite clear, if formation rules for terms are given by the labelled deductive system:

(TER.1') $x_A : A$, $c_A : A$ (c_A is a constant of type A),

(TER.2') if $s : A \rightarrow B$ and $t : A$, then $(st) : B$,

(TER.3') if $s : B$ and $x : A$, then $(\lambda x s) : A \rightarrow B$.

Clearly, each Natural Deduction proof corresponds to a proof in this LDS, and conversely, with (Id) parallel to (TER.1'), ($\mathrm{E}^{\star} \rightarrow$) to (TER.2'), and ($\mathrm{I}^{\star} \rightarrow$) to (TER.3'). A Natural Deduction proof must be labelled in the following way: each assumption A, being removed by an application of ($\mathrm{I}^{\star} \rightarrow$), is replaced by a variable x_A, and these and only these occurrences of A which are removed by the same instance of ($\mathrm{I}^{\star} \rightarrow$) are replaced by the same variable (in terms of GIRARD et al. [22], assumptions are distributed in different boxes according to their cooccurrence in introduction rules). If Γ' equals Γ (that means, no occurrence of A is removed), then (TER.3') uses a variable x not free in s, and otherwise, x is precisely the variable which replaces the removed occurrences of A.

In this way, the lambda-term corresponding to *John hits Paul* represents the following proof.

$$\dfrac{\dfrac{\dfrac{(\mathbf{e} \rightarrow \mathbf{t})_x \quad \mathbf{e}}{\mathbf{t}}}{(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}} \quad \dfrac{\mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \quad \mathbf{e}}{\mathbf{e} \rightarrow \mathbf{t}}}{\mathbf{t}}$$

Here $(\mathbf{e} \rightarrow \mathbf{t})_x$ symbolizes assumption $\mathbf{e} \rightarrow \mathbf{t}$ being removed by the only application of ($\mathrm{I}^{\star} \rightarrow$). Equivalently, this proof can be written in a form making 'active' assumptions explicit.

$$\dfrac{\dfrac{\dfrac{\mathbf{e} \rightarrow \mathbf{t} \vdash \mathbf{e} \rightarrow \mathbf{t} \quad \mathbf{e} \vdash \mathbf{e}}{\mathbf{e} \rightarrow \mathbf{t}, \mathbf{e} \vdash \mathbf{t}}}{\mathbf{e} \vdash (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}} \quad \dfrac{\mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \vdash \mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \quad \mathbf{e} \vdash \mathbf{e}}{\mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}), \mathbf{e} \vdash \mathbf{e} \rightarrow \mathbf{t}}}{\mathbf{e}, \mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}), \mathbf{e} \vdash \mathbf{t}}$$

The latter is precisely a proof tree in the system based on (Id), ($E^\star \rightarrow$) and ($I^\star \rightarrow$), and it can be rewritten as the following formation tree for the lambda-term given above for *John hits Paul*.

$$\dfrac{\dfrac{\dfrac{x : \mathbf{e} \rightarrow \mathbf{t} \qquad \mathbf{John} : \mathbf{e}}{(x\,\mathbf{John}) : \mathbf{t}}}{(\lambda x(x\,\mathbf{John})) : (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}} \qquad \dfrac{\mathbf{hits} : \mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \quad \mathbf{Paul} : \mathbf{e}}{(\mathbf{hits}\,\mathbf{Paul}) : \mathbf{e} \rightarrow \mathbf{t}}}{((\lambda x(x\,\mathbf{John}))(\mathbf{hits}\,\mathbf{Paul})) : \mathbf{t}}$$

Accordingly, each Natural Deduction proof of sequent $A_1 \ldots A_n \vdash A$ can be represented by a term t, of type A, containing precisely n free variables of type $A_1, \ldots, A_n$, respectively; these variables appear in t in the horizontal order determined by the order of assumptions $A_1, \ldots, A_n$. Clearly, free variables can be replaced by constants of the corresponding type, as in the example above. Now, if $\mathbf{v_1}, \ldots, \mathbf{v_n}$ are constants such that:

$$\mu(\mathbf{v_i}) = a_i \in D_{A_i}, \text{ for } i = 1, \ldots, n,$$

then, for each term t representing a Natural Deduction proof of the above sequent, the object $\mu(t) \in D_A$ may be regarded as the denotation of the expression $v_1 \ldots v_n$ determined by the given proof. In this sense, semantic denotations can be computed according to possible Natural Deductions proofs from the string of assumptions corresponding to lexical units of the expression in question.

The full lambda language equivalent to Natural Deduction proofs of the full positive logic of implication is too strong to be treated as an adequate generator of admissible semantic structures. For instance, ($I^\star \rightarrow$) with $\Gamma' = \Gamma$ justifies the inference:

$$\dfrac{B \vdash B}{B \vdash A \rightarrow B},$$

being represented by the semantic transformation:

$$(\lambda y_A x_B) : A \rightarrow B,$$

which lifts up each object $b \in D_B$ to the constant function $F_b \in D_{A \rightarrow B}$ such that $F_b(a) = b$, for all $a \in D_A$. No linguistically sound interpretation of this move is possible, however. For the PN *John* cannot function as a functor of type, say, $\mathbf{t} \rightarrow \mathbf{e}$. This special case of ($I^\star \rightarrow$) yields the logical Thinning which corresponds to the monotonicity of consequence: everything which follows from the given assumptions also follows from any larger array

of assumptions. Clearly, Thinning is not plausible for semantic transformations, and the same may be said of Contraction: each repetition of A in the assumption string can be removed. Contraction is equivalent to $(\mathrm{I}^\star \rightarrow)$ with Γ' having two or more occurrences of A removed. A sample argument is:

$$\dfrac{\dfrac{\dfrac{\mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \vdash \mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \quad \mathbf{e} \vdash \mathbf{e}}{\mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}), \mathbf{e} \vdash \mathbf{e} \rightarrow \mathbf{t}} \quad \mathbf{e} \vdash \mathbf{e}}{\mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}), \mathbf{e}, \mathbf{e} \vdash \mathbf{t}}}{\mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \vdash \mathbf{e} \rightarrow \mathbf{t}}$$

where $(\mathrm{I}^\star \rightarrow)$ is used at the root and removes two occurences of $\mathbf{e}$ at once. Accepting this argument would allow each TV-expression to act as a VP-expression, which is linguistically disputable. But, the semantic structure for *John and Mary*, given above, uses λ binding two occurrences of $x_{\mathbf{e}\rightarrow\mathbf{t}}$, and no Contraction-free analysis of this case is possible.

It would be difficult to fix one definite system which could provide precisely the linguistically sound deductions, but attempts in this direction lead to different interesting subsystems of positive logic. VAN BENTHEM [4] focuses on a Contraction- and Thinning-free system which is given by (Id), $(\mathrm{E}^\star \rightarrow)$ and $(\mathrm{I}^\star \rightarrow)$ with Γ' having exactly one occurrence of A less than Γ and $\Gamma' \neq \Lambda$. Proofs in this system are represented the terms fulfilling the following constraints:

(CON.1) each subterm contains a free variable (or a constant),

(CON.2) each occurrence of λ binds at least one variable free in its scope,

(CON.3) each occurrence of λ binds at most one variable free in its scope.

Clearly, (CON.1) enforces the nonemptiness of Γ' in $(\mathrm{I}^\star \rightarrow)$, (CON.2) blocks Thinning, and (CON.3) blocks Contraction. Removing (CON.1) yields the BCI-logic, known from investigations in Combinatory Logic and Relevant Logic (see ONO and KOMORI [39], DOŠEN and SCHROEDER-HEISTER [16]). Also, dropping (CON.1) and (CON.2) corresponds to the BCK-logic, while dropping (CON.1) and (CON.3) to the BCIW-logic (the marks B, C, I, W, K come from Combinatory Logic). A systematic account of systems determined by each of the eight subsets of this set of constraints is given in [8].

Even VAN BENTHEM's system is still too strong in some respect. It overgenerates syntactic structures, since it is closed under *the permutation rule*:

$$(\mathrm{PER})\ \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}.$$

Consequently, if X is qualified an expression of type C, then every permutation of X is so; thus, not only *every girl whistles* but also *whistles girl every* is assigned type $\mathbf{t}$. Further, it overgenerates semantic structures. For instance, the transformation:

$$(\lambda y_{\mathbf{e}}(\lambda x_{\mathbf{e}}((\mathbf{loves}\, x_e)y_e)))$$

allows one to interpret *loves* as to mean *is loved by*, and similarly, each binary function $F \in D_{A\to(B\to C)}$ is transformed into its reversal $F' \in D_{B\to(A\to C)}$ such that $F'(b)(a) = F(a)(b)$ (or, according to the Schönfinkel convention, $F'(b,a) = F(a,b)$).

Essentially weaker systems can be obtained from the directional calculi, considered in section 2. Recall that types are formed out of atomic types by means of two conditionals $\to$ and $\leftarrow$. The (product-free) Lambek Calculus is given the Natural Deduction form by axioms (Id) and rules (E$\to$), (E$\leftarrow$), (I$\to$) and (I$\leftarrow$) (with $\Gamma \neq \Lambda$ in I-rules). Notice that now $\leftarrow$ takes the part of $\to$ from the nondirectional systems, discussed above (compare (E$\leftarrow$) and (E* $\to$), but (I$\leftarrow$) is remarkably weaker than (I* $\to$) (only the left-most assumption is to be removed).

As suggested in [8, 9], the van Benthem correspondence between proofs and (special) terms can be extended to directional systems with applying a directional version of the lambda calculus. With directional types and two lambdas λ^r, λ^l, new formation rules for terms are (TER.1) and:

(TER.2^r) if s is a term of type A, and t is a term of type $A \to B$, then (st) is a term of type B,

(TER.2^l) if s is a term of type $B \leftarrow A$, and t is a term of type A, then (st) is a term of type B,

(TER.3^r) if s is a term of type B, then $(\lambda^r x_A B)$ is a term of type $A \to B$,

(TER.3^l) if s is a term of type B, then $(\lambda^l x_A B)$ is a term of type $B \leftarrow A$.

This directional lambda calculus has thoroughly been discussed in WANSING [50]. It can be shown that proofs in the (product-free) Lambek Calculus are represented by directional terms fulfilling the constraints (CON.1) and

(DCON^r) each occurrence of λ^r binds precisely the right-most occurrence of a free variable in its scope,

(DCON^l) each occurrence of λ^l binds precisely the left-most occurrence of a free variable in its scope.

Variations of these constraints can be considered, as well.

In semantics, $A \rightarrow B$ and $B \leftarrow A$ can be interpreted in the same way: both stand for the type of functions from A-objects to B-objects. In (TER.2^r), functor t follows argument s, while in (TER.2^l), functor s precedes argument t, but the meaning remains the same: the application of the functor to the argument. In (TER.3^r) and (TER.3^l), both $(\lambda^r x_A s)$ and $(\lambda^l x_A s)$ denote the same function from A-objects to B-objects. So, denotations of complex expressions can be computed along Natural Deduction proofs, as it has been the case for nondirectional systems. Now, (PER) is not admissible, and arguments of binary functions cannot be commuted. Since (SYL$\rightarrow$) and (SYL$\leftarrow$) are derivable, with proofs being represented by the terms:

$$(\lambda^l x_A((x_A y_{A\rightarrow B})z_{B\rightarrow C})),$$

$$(\lambda^r z_A(x_{C\leftarrow B}(y_{B\leftarrow A} z_A))),$$

then function composition is admissible. The nondirectional transformation of Type Raising:

$$(\mathrm{TR})A \vdash (A \rightarrow B) \rightarrow B$$

is directionally imitated by (TR$\rightarrow$) and (TR$\leftarrow$) (replace the line by $\vdash$). In directional systems, sentence *John works* can be analysed in a direct way, as the structure:

$$(\textbf{John works}),$$

since **works** is of (directional) type $\mathbf{e} \rightarrow \mathbf{t}$, hence it takes the argument on the left, and similarly for *John hits Paul*.

There are, however, semantic structures which require the nondirectional approach. One of them is non-sentential conjunction, as in *John and Mary, works and rests*, which cannot be accounted by means of λ^r and λ^l; the basic type $(\mathbf{t} \rightarrow \mathbf{t}) \leftarrow \mathbf{t}$ cannot be transformed into a type $(A \rightarrow A) \leftarrow A$ in the directional framework. Also, the double quantifier sentence *every student reads some book* admits two correct nondirectional deductions, the first with the wide scope of *every* and the second with the wide scope of *some*, in which the NP's *every student* and *some book* are assigned type $(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$, and the TV *reads* is assigned type $\mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t})$. On the other hand, the directional approach requires two nonequivalent types of NP's: $\mathbf{t} \leftarrow (\mathbf{e} \rightarrow \mathbf{t})$ for the subject position and $(\mathbf{t} \leftarrow \mathbf{e}) \rightarrow \mathbf{t}$ for the object position, with type $(\mathbf{e} \rightarrow \mathbf{t}) \leftarrow \mathbf{e}$ for the TV *reads*. Accordingly, directionality may increase the number of types needed in lexical assumptions.

Directionality is, apparently, a concession to syntax, while nondirectional systems are strictly semantical in spirit. Some authors propose mixed systems in which directional deductions are supplied with nondirectional

semantic structures in the form of standard lambda-terms. This approach seems to be predominant in linguistic writings; an especially advanced study of mixed systems can be found in MOORTGAT [36] and HENDRIKS [23]. Hendriks considers sequents of the form:

$$A_1 : x_1, \ldots, A_n : x_n \vdash B : t$$

such that $A_1, \ldots, A_n, B$ are directional types, $x_1, \ldots, x_n$ are distinct variables whose types are the semantic counterparts of $A_1, \ldots, A_n$, respectively, and t is a standard term whose free variables are precisely $x_1, \ldots, x_n$. Axioms and inference rules operate on sequents of this form. For instance, the counterparts of (I→) and (I←) are:

$$\frac{A : x, A_1 : x_1, \ldots, A_n : x_n \vdash B : t}{A_1 : x_1, \ldots, A_n : x_n \vdash A \to B : (\lambda x t)},$$

$$\frac{A_1 : x_1, \ldots, A_n : x_n, A : x \vdash B : t}{A_1 : x_1, \ldots, A_n : x_n \vdash B \leftarrow A : (\lambda x t)}.$$

Mixed systems preserve the standard form of semantic structures and block overgeneration by a kind of syntactic control for semantic description. Actually, both Moortgat and Hendriks focus on Gentzen style formalisms and prove several cut elimination and normalization theorems for them.

An alternative (but essentially equivalent) approach to semantic structures is Combinatory Categorial Grammar, developed by STEEDMAN and his collaborators [1, 45, 46]. Lambda-terms without free variables are called *combinators*. Some most important combinators are:

$\mathbf{I} = \lambda x_A x_A$, type: $A \to A$,

$\mathbf{K} = \lambda x_A (\lambda y_B x_A)$, type: $A \to (B \to A)$,

$\mathbf{W} = \lambda x (\lambda y ((xy)y))$, type: $(A \to (A \to B)) \to (A \to B)$, where $y : A$ and $x : A \to (A \to B)$,

$\mathbf{C} = \lambda x (\lambda y (\lambda z ((xz)y)))$, $x : A \to (B \to C)$, $y : B$, $z : A$, type of $\mathbf{C}$: $(A \to (B \to C)) \to (B \to (A \to C))$,

$\mathbf{B} = \lambda x (\lambda y (\lambda z (x(yz))))$, $x : B \to C$, $y : A \to B$, $z : A$, type of $\mathbf{B}$: $(B \to C) \to ((A \to B) \to (A \to C))$.

Clearly, $\mathbf{I}$ yields the identity mapping, $\mathbf{K}$ produces constant functions, $\mathbf{W}$ diagonalizes binary functions, $\mathbf{C}$ commutes arguments of binary functions, and $\mathbf{B}$ yields the composition of two functions. *Combinatory terms* are formed

out of combinators and variables by application (rule (TER.2)). It is well known [15] that each lambda-term is equivalent to some combinatory term built from variables and two combinators: **K** and

$$\mathbf{S} = \lambda x(\lambda y(\lambda z((xz)(yz)))),$$

of type $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$, with $x : A \to (B \to C)$, $y : A \to B$, $z : A$. Consequently, all semantic structures expressible in the lambda language can also be expressed by means of combinatory terms.

Steedman explains the role of particular combinators in the semantic description of different constructions in natural language. A simple example is *Harry cooked and might eat the beans* with directional types:

$$\mathbf{cooked} : (\mathbf{NP} \to \mathbf{t}) \leftarrow \mathbf{NP},\ \mathbf{might} : (\mathbf{NP} \to \mathbf{t}) \leftarrow \mathbf{VP},$$

$$\mathbf{eat} : \mathbf{VP} \leftarrow \mathbf{NP},\ \mathbf{Harry},\ \mathbf{the\ beans} : \mathbf{NP},$$

where $\mathbf{VP} = \mathbf{e} \to \mathbf{t}$. As above, $\to$ and $\leftarrow$ are semantically synonymous. Here *and* should act as the Boolean conjuction on the semantic denotations of VP's *cooked* and *might eat*, which would require computing the denotation of the latter. Clearly, if F, G are denotations of *might, eat*, respectively, then $(\mathbf{B}F)G$ provides the denotation of *might eat*, i.e. the function $F \circ G$. Using combinators, Steedman provides adequate semantic structures for various 'difficult' constituents of sentences, including *wh*-constructions, non-adjacent connections and others.

Semantic structures obeying constraints (CON.1)-(CON.3) can be produced by means of combinators **B**, **C** and **I**. Szabolcsi [49] finds out that constructions like *file without reading* in e.g. *Harry will copy and file without reading some articles* require combinator **S** which is not expressible by means of **B**, **C** and **I**. For with type $(\mathbf{VP} \to \mathbf{VP}) \leftarrow \mathbf{NP}$ of *without reading* and type $\mathbf{VP} \leftarrow \mathbf{NP}$ of *file* the type $\mathbf{VP} \leftarrow \mathbf{NP}$ of *file without reading* can be derived by so-called Backward Crossed Substitution:

$$B \leftarrow C,\ (B \to A) \leftarrow C \vdash A \leftarrow C;$$

if F, G represent the assumptions, then $(SG)F$ is a function from C to A which correctly describes the meaning of this construction. That is another evidence for the need of a larger lambda language than the fragment delimited by van Benthem.

Returning to our leitmotive: logical proofs versus grammatical structures, we notice that Combinatory Categorial Grammar is naturally related

to Hilbert style systems rather than Natural Deduction systems. Since formation rules for combinatory terms are (TER.1) and (TER.2), and lambda abstraction is abandoned, then the corresponding deductions do not employ I-rules, and the remaining E-rules reduce to MP-rules (plus (CUT)). Fixed combinators, as e.g. **I**, **B** etc., appearing in combinatory terms provide logical axioms for the corresponding deductions, and variables take the part of assumptions.

All possible combinatory terms are equivalent to all possible proofs (from assumptions) in the full system of positive implication, with axioms:

(K) $A \to (B \to A)$,

(S) $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$,

and the only rule (MP). Notice that these axioms amount to types of combinators **K**, **S** which generate all combinators. Restricted systems of combinators correspond to subsystems of positive logic. For instance, combinatory terms formed out of **B**, **C** and **I** are equivalent to proofs in the BCI-logic whose axioms are types of these three combinators, and similarly one obtains the BCK-logic, the BCIW-logic and so on.

Above, we have merely outlined basic ideas of generating semantic structures by means of lambda-terms and combinators according to Natural Deduction proofs and Hilbert style proofs, respectively. Now, we briefly discuss some more fine properties of these formalisms which are of both logical and linguistic significance.

Different deductions of sequent $\Gamma \vdash A$ are represented by different terms which yield different semantic structures of the underlying expression (with the same outcome type). Structures s, t are said to be *equivalent*, if $\mu(s) = \mu(t)$, for every valuation μ. As shown by VAN BENTHEM [4], for any expression, there are only finitely many equivalence classes of structures (fulfilling (CON.1)–(CON.3)) with the given outcome type. In other words, each expression admits only finitely many readings in the given semantic category. The proof is based on normalization properties of the systems in question. A *redex* is a term $(\lambda x s)t$, and the term $s[x := t]$ (i.e. the substitution of t for every free occurrence of x in s) is *the contractum* of this redex. Another form of a redex is $\lambda x(sx)$ (x not free in s) with the contractum s. A *reduction* of a term consists in successive replacing redexes appearing in the term by their contracta. A term is in the normal form, if it contains no redex. In the typed lambda calculus, each term t can be reduced to exactly one term in the normal form (up to renaming bound variables) which is called *the normal form* of t (see e.g. [22]). Clearly, reduction always produces terms equivalent to

the initial term, hence each term is equivalent to its normal form. Applied to Natural Deduction proofs, reduction eliminates circularities like inferring $\Gamma \vdash A \to B$ from $\Gamma, A \vdash B$, by ($\mathrm{I}^\star \to$), and next inferring $\Gamma, A \vdash B$, by (Id) and ($\mathrm{E}^\star \to$). For any sequent $\Gamma \vdash A$, there are only finitely many normal deductions in the van Benthem calculus; equivalently, there are only finitely many terms in the normal form with the outcome type A and the fixed string of free variables and constants. That proves van Benthem's theorem. As stated here, the proof is sound for the contraction-free fragments of the lambda language.

Another interesting question concerns the equivalence of Natural Deduction systems and Hilbert style systems. Given a Natural Deduction format, one seeks for a Hilbert style system which produces the same sequents, and conversely (actually, one wants a kind of strong equivalence: both systems should provide the same semantic structures of derivable sequents, at least up to the semantic equivalence of terms, defined above). Some negative results have been obtained by Zielonka [52] for directional systems and myself [8] for nondirectional systems. Namely, the (product-free) Lambek calculus admits no finite Hilbert style axiomatization, and the same is true for the van Benthem system. Since no sequent of the form $\vdash A$ is provable in these systems, Hilbert style presentations have no logical axioms; instead, they use unary rules of the form $A \vdash B$. Systems of Lambek and van Benthem can easily be axiomatized by infinitely many unary rules plus (MP) (the first axiomatization of that style is due to Cohen [12]), but the infinite collection of rules cannot be reduced to a finite one. These results are relevant to some earlier research in semantics, when people have improved the flexibility of purely applicative systems by affixing some logically sound inference patterns, as e.g. (SYL), (TR); it follows that none of those calculi can simulate all possible Lambek style deductions and produce a combinatorily complete system of semantic structures. On the contrary, the logic BCI, being a 'slight' extension of the van Benthem calculus (one admits $\Gamma' = \Lambda$ in ($\mathrm{I}^\star \to$)), possesses a finite Hilbert style axiomatization (given by **B**, **C** and **I**), and so do its strengthenings BCIW, BCK etc. (For directional versions of these systems, the question remains open.) Thus, semantic structures fulfilling (CON.2) and (CON.3) can be generated from the finite base **B**, **C**, **I**, enriched by variables and constants, by application only, whereas no finite base can generate all structures obeying (CON.1)–(CON.3) (here, elements of a base cannot be combinators, but they must be some terms fulfilling the constraints; in particular, they must contain free variables).

An (infinite) Hilbert style axiomatization of the van Benthem system is given by the rules:

(TR) $A \vdash (A \to B) \to B$,

(C') $A \to (B \to C) \vdash B \to (A \to C)$,

together with all rules produced from them by means of the higher level rules:

$$\text{(M1)}\ \frac{A \vdash B}{C \to B \vdash C \to A},$$

$$\text{(M2)}\ \frac{A \vdash B}{A \to C \vdash B \to C},$$

which express the antitonicity of $\to$ in the first argument and the monotonicity in the second one. One also needs MP (in the sequential format, (CUT) and (Id) are affixed). That the resulting system is equivalent to the van Benthem system is proven by showing ($\mathrm{I}^\star$ $\to$) ($\Gamma' \neq \Lambda$ has one occurrence of A less than Γ) be derivable in the former system, which is, actually, a Deduction Theorem for the Hilbert style system. The nonexistence of a finite axiomatization follows from the fact that (M1), (M2) cannot be replaced by any finite collection of patterns derivable with the aid of them [8]. Yet, even this infinite axiomatization is of linguistic significance. Consider the expression $v_1 \dots v_n$ with the lexical assignment $v_i : A_i$, $i = 1, \dots, n$. To derive $v_1 \dots v_n : B$ by means of the latter system one, first, transforms each A_i into a type B_i, by a finite application of unary rules, and second, executes an MP-deduction for $B_1 \dots B_n \vdash B$ (this arrangement of proofs is always possible, since each application of a unary rule after MP can be changed into an application of a unary rule before MP). Semantically, all denotations b of $v_1 \dots v_n$ can be produced in the following way: first, the initial denotations $a_i \in D_{A_i}$ are transformed into some denotations $b_i \in D_{B_i}$ by the axiomatically given semantic mappings, and second, the object b is computed from $b_1, \dots, b_n$ by function application only. Consequently, the non-applicative component of structure generation can be reduced to the lexical level, while keeping function application as the only construction joining constituents into an whole. Semantic lexicalization in this sense is possible for all combinatorily complete families of semantic structures.

At the end, we merely mention some major topics not discussed here. The above considerations are restricted to the pure lambda calculus and the corresponding purely propositional (implicative) systems. This framework may be extended in different ways. New logical constants (product, booleans, etc.) lead to extended versions of the lambda calculus, and we refer the reader to VAN BENTHEM [6] and LAMBEK/SCOTT [32] for a detailed discussion. Another extension is Higher-Order Logic in which one has the logical constant $=$ in each type; then, the semantic equivalence of structures

admit a direct treatment, and the Montague theory of intension can be incorporated (here the references are Gallin [17] and again van Benthem [6]). A natural problem is to characterize the objects which are denoted by lambda terms of a given form; as shown in van Benthem [6], in finite models (that means, all sets D_A are finite), all permutation invariant objects are definable by (pure) lambda terms, which evidently fails for infinite models (van Benthem has some interesting partial results). Recently, Sanchez Valenzia [42, 43]] has developed a theory of monotonic inference, based on Natural Deduction supplied with monotonicity marking. In his approach, types may contain an information of the positive (negative) character of expressions. For instance, *every* is typed $\mathbf{NP}^- \to (\mathbf{VP}^+ \to \mathbf{t})$, since it denotes a function antitone in the NP-argument and monotone in the VP-argument, and *some* is typed $\mathbf{NP}^+ \to (\mathbf{VP}^+ \to \mathbf{t})$, since it denotes a function monotone in both arguments (the order here is the natural Boolean order in ontological categories; see [27]). Natural Deduction rules are sensitive to monotonicity marking, which propagates the markers along the tree. The resulting marked structures provide information of the role of expressions in natural inferences like:

$$\frac{\text{every } X \text{ is } Y;\ X' \subset X}{\text{every } X' \text{ is } Y}.$$

References

[1] A. E. Ades and Mark J. Steedman, On the Order of Words, *Linguistics and Philosophy* 4 (1982), 517–558.

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

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

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

[5] J. van Benthem, The Lambek Calculus, in: [38].

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

[7] W. Buszkowski, Completeness Results for Lambek Syntactic Calculus, *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 32, (1986), 13–28.

[8] W. Buszkowski, The logic of types, in: J. T. Srzednicki (ed.), *Initiatives in Logic*, Nijhoff, Dordrecht, 1987.

[9] W. Buszkowski, Generative Power of Categorial Grammars, in: [38].

[10] W. Buszkowski, The Ajdukiewicz Calculus and Polish Notation, in Polish, in: J. Pogonowski (ed.), *Eufonia i Logos*, The Book devoted to Professors Maria Steffen-Batogowa and Tadeusz Batóg, Adam Mickiewicz University Press, Poznań, 1995.

[11] W. Buszkowski, W. Marciszewski and J. van Benthem (eds.), *Categorial Grammar*, J. Benjamins, Amsterdam, 1988.

[12] J. M. Cohen, The equivalence of two concepts of categorial grammar, *Information and Control* 10 (1967), 475–484.

[13] M. J. Cresswell, *Logics and Languages*, Methuen, London, 1974.

[14] H. B. Curry, Some Logical Aspects of Grammatical Structure, in: R. Jakobson (ed.), *Structure of Language and Its Mathematical Aspects*, AMS, Providence, 1961.

[15] H. B. Curry and R. Feys, *Combinatory Logic*, I, North Holland, Amsterdam, 1958.

[16] K. Došen and P. Schroeder-Heister (eds.), *Substructural Logics*, Oxford University Press, Oxford, 1993.

[17] D. Gallin, *Intensional and Higher-order Modal Logic*, North Holland, Amsterdam, 1975.

[18] D. Gabbay, *Labelled Deductive Systems I*, CIS-München, Munich, 1991.

[19] G. Gazdar and C. Mellish, *Natural Language Processing in Prolog. An Introduction to Computational Linguistics*, Addison-Wesley, Workingham, 1989.

[20] P. T. Geach, A program for syntax, *Synthese* 22 (1968), 3–17; reprinted in [11].

[21] J. Y. Girard, Linear Logic, *Theoretical Computer Science* 50 (1987), 1–102.

[22] J. Y. Girard with P. Taylor and Y. Lafont, *Proofs and Types*, Cambridge University Press, Cambridge, 1989.

[23] H. Hendriks, *Studied Flexibility*, Ph. D. Thesis, University of Utrecht, 1993.

[24] H. Hiż, Grammar Logicism, *The Monist* 51, (1967), 110–127; reprinted in [11].

[25] H. Hiż, On the Abstractness of Individuals, in: M. Munity (ed.), *Identity and Individuals*, New York University Press, 1971.

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

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

[28] J. Lambek, The mathematics of sentence structure, *American Mathematical Monthly* 65 (1958), 155–170.

[29] J. Lambek, On categorial and categorical grammars, in: [38].

[30] J. Lambek, Logics without structural rules, in: [16].

[31] J. Lambek, From Categorial Grammar to Bilinear Logic, in [16].

[32] J. Lambek and P. J. Scott, *Introduction to higher order categorical logic*, Cambridge University Press, Cambridge, 1986.

[33] H. Leiss and M. Emms, *Second-order Lambek Calculus and Cut Elimination*, DYANA Reports, 1993.

[34] D. Lewis, General Semantics, in: D. Davidson and G. Harman (eds.), *Semantics of Natural Language*, D. Reidel, Dordrecht, 1972.

[35] R. Montague, *Formal Philosophy*, (ed. by R. Thomason), Yale University Press, New Haven, 1974.

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

[37] M. Moortgat, Residuation in Mixed Lambek Systems, manuscript, University of Utrecht, 1994.

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

[39] H. Ono and Y. Komori, Logics without the contraction rule, *Journal of Symbolic Logic* 50 (1985), 169–201.

[40] B. Partee, Montague Grammar and Transformational Grammar, *Linguistic Inquiry* 6 (1975).

[41] F. C. Pereira and S. M. Shieber, *Prolog and Natural Language Analysis*, CSLI Lecture Notes, 10, Chicago University Press, Stanford, 1987.

[42] V. Sanchez Valenzia, *Studies on Natural Logic and Categorial Grammar*, Ph. D. Thesis, University of Amsterdam, 1991.

[43] V. Sanchez Valenzia, Natural Logic: Parsing Driven Inference, *to appear.*

[44] S. K. Shaumyan, *Applicational Grammar as a Semantic Theory of Natural Language*, Edinburgh University Press, Edinburgh, 1977.

[45] M. Steedman, Combinators and grammars, in: [38].

[46] M. Steedman, Categorial grammar. Tutorial overview, *Lingua* 90 (1993), 221-258.

[47] P. Suppes, Logical inference in English, *Studia Logica* 38 (1979), 375–391.

[48] A. Szabolcsi, *ECP in Categorial Grammar*, Max Planck Institute, Nijmegen, 1983.

[49] H. Uszkoreit, Categorial Unification Grammar, *Proc. 11th International Conference on Computational Linguistics*, Bonn, 1986.

[50] H. Wansing, *The Logic of Information Structures*, Ph. D. Thesis, University of Amsterdam, 1992.

[51] D. N. Yetter, Quantales and (Non-Commutative) Linear Logic, *The Journal of Symbolic Logic* 55 (1990), 41–64.

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

[53] F. Zwarts, *Categoriale grammatica en algebraische semantiek*, Ph. D. Thesis, University of Groningen, 1986.

WOJCIECH BUSZKOWSKI
Faculty of Mathematics and Computer Science
Adam Mickiewicz University
ul. Matejki 48/49
60-769 Poznań, POLAND