Encyclopedia of Cognitive Science

#231, Categorial grammar and formal semantics

Michael Moortgat
Professor of Computational Linguistics
Utrecht Institute of Linguistics, OTS
Utrecht University
Trans 10, 3512 JK Utrecht
The Netherlands

tel: +31–30–2536043
fax: +31–30–2536000
e-mail: moortgat@let.uu.nl

Keywords

Contents

Categorial grammar and formal semantics, #231

1 Introduction

Categorial grammar is a linguistic framework with firm roots in type theory and constructive logic. Not surprisingly, this grammar formalism has been intensively studied from a logical and mathematical point of view. This article has a different emphasis, and focuses on the categorial modelling of the cognitive abilities that are at the basis of the acquisition, use and understanding of natural language. Two central questions are addressed in the sections that follow:

- What are the *constants* of grammatical reasoning? Can we provide an explanation for the uniformity of the form/meaning correspondence across languages in terms of these constants?
- How can we reconcile the idea of grammatical constants/invariants with the differences between languages, that is, with structural *variation* in the realization of the form/meaning correspondence?

The categorial perspective on these questions is well captured by the slogan 'parsing as deduction'. A grammar, essentially, is given by an assignment of types to the elementary units in the lexicon. The type-forming operations have the status of logical connectives: determining whether a complex configuration of grammatical material is well-formed amounts to presenting a derivation, or proof, in the logic for these connectives. Natural language expressions, or signs, have a form dimension and a meaning dimension. The categorial type language, consequently, is modeltheoretically interpreted with respect to these two dimensions, and a derivation encodes an effective procedure for building up the structural organization of an expression, and for associating this structure with a recipe for meaning assembly.

The article is organized as follows. In §2, we focus on the form dimension of expressions. We identify the logical constants of the computational system, and we study how the base logic for these constants can be extended with facilities for structural reasoning. In §3, we see how the logical rules of inference for the type-forming operations can be read as instructions for meaning assembly, and how the structural rules determine which components of an expression can enter into the assembly process. In §4, some important areas of current categorial research are surveyed. The final section provides some historical background and pointers to the literature.

2 Form: grammatical invariants and structural variation

2.1 The base logic

Natural language expressions are structured objects, with an articulation in dimensions such as linear order and hierarchical grouping. The type formulas of a categorial grammar are structured in a way that mirrors the composition of the expressions they categorize. The set Type of type formulas is obtained as the closure of a small set of *basic* (or *atomic*) types under a number of type-forming operations. Individual categorial grammars will differ with respect to the type-forming operations they employ. For the purposes of this article, the following clauses will be representative.

1. the set of *basic types* is a subset of Type;
2. if A is a formula in Type, then $\Diamond A$ and $\Box A$ are too;
3. if A and B are formulas in Type, then $A \bullet B$, A/B and $A\backslash B$ are too.

Basic types play a role similar to that of major constituents in phrase-structure grammar: they categorize expressions one can think of as 'complete'. Common examples would be the type s for sentences, n for common noun phrases, np for full noun phrases (also called determiner phrases). The unary and binary type-forming operations provide a vocabulary to categorize the constituent parts of complex expressions. Informally, a type formula $A \bullet B$ categorizes an expression that can be decomposed into a constituent of type A followed by a constituent of type B. An expression of type A/B is incomplete: it combines with an expression of type B on its right into an expression of type A. Similarly, an incomplete expression of type $B\backslash A$ combines with an expression of type B on its left to form an A type expression. The unary type-forming operations are more recent additions to the categorial vocabulary. They can be thought of as features: an expression of type $\Box A$ issues a request for a feature to be checked; such an expression can be used as a regular A as soon as the $\Box$ feature is eliminated. The operation $\Diamond$ provides the means to perform the required feature-checking.

Frame semantics. To make this informal description precise, one uses the *frame-based* models familiar from possible-world semantics. For the categorial type language, a *frame* is a tuple $\langle W, R_\diamond, R_\bullet \rangle$. W is a non-empty set, the set of expressions. $R_\diamond$ and $R_\bullet$ are binary and ternary relations over W, interpreting the unary and binary type-forming operations, respectively. One can think of $R_\bullet$ as the 'Merge' relation: $R_\bullet xyz$ holds in case x is the composition of the parts y and z. Similarly, $R_\diamond xy$ holds if the feature-checking relation connects y to x. One obtains a *model* by adding a *valuation* V to a frame F. The valuation assigns subsets of W to the atomic formulas. For complex types, the valuation respects the conditions below.

$x \in V(\Diamond A)$	iff there exists a y such that $R_\diamond xy$ and $y \in V(A)$
$x \in V(\Box A)$	iff for all y, $R_\diamond yx$ implies $y \in V(A)$
$x \in V(A \bullet B)$	iff there are y and z such that $y \in V(A)$, $z \in V(B)$ and $R_\bullet xyz$
$x \in V(C/B)$	iff for all y and z, if $y \in V(B)$ and $R_\bullet zxy$, then $z \in V(C)$
$x \in V(A\backslash C)$	iff for all y and z, if $y \in V(A)$ and $R_\bullet zyx$, then $z \in V(C)$

Type computations, soundness and completeness. Turning from the model-theoretic to the proof-theoretic point of view, we are interested in a deductive system to perform type computations of the form $A \to B$ ('type B is derivable from type A'). We want the deductive system to be faithful to the interpretation of the type-forming operations, in the following sense:

SOUNDNESS AND COMPLETENESS. $A \to B$ is provable if and only if for every frame F and every valuation V, $V(A) \subseteq V(B)$.

Soundness (the 'if' direction of the biconditional) makes sure that we cannot obtain a formal derivation for $A \to B$ if it is not the case that indeed, for all frames and valuations, the interpretation of type A is included in the interpretation of type B. More important is the requirement of completeness (the 'only if' direction): this guarantees that whenever the inclusion $V(A) \subseteq V(B)$ holds for arbitrary frames and valuations, there will be a proof of this fact.

An axiomatization satisfying the soundness and completeness requirements starts with an identity axiom $A \to A$, and an inference rule allowing one to conclude $A \to C$ from premises $A \to B$ and $B \to C$. Semantically, these express the reflexivity and transitivity of the derivability relation. In addition, one has the inference rules in (1) establishing the relationship between the interpretation of $\Diamond$ and $\Box$, and between $\bullet$ and left and right division $\backslash$ and $/$. The patterns in (1) turn $(\Diamond, \Box)$, $(\bullet, /)$ and $(\bullet, \backslash)$ into what are known as *residuated pairs* in algebra, or *adjoint functors* in category theory.

(1)
$$\begin{array}{llll} (R0) & \Diamond A \to B & \text{if and only if} & A \to \Box B \\ (R1) & A \bullet B \to C & \text{if and only if} & A \to C/B \\ (R2) & A \bullet B \to C & \text{if and only if} & B \to A\backslash C \end{array}$$

Sample theorems. Let us look at some elementary theorems of the grammatical base logic presented above. From the identity axiom, one obtains the Application schemata of (2b) in one step, using the residuation inferences in the 'if' direction. From Application, one derives the Lifting schemata of (2c), this time reasoning in the 'only if' direction.

(2)
$$\begin{array}{lllll} a. & A\backslash B \to A\backslash B & (Ax) & B/A \to B/A & (Ax) \\ b. & A \bullet (A\backslash B) \to B & (R2 \Leftarrow) & (B/A) \bullet A \to B & (R1 \Leftarrow) \\ c. & A \to B/(A\backslash B) & (R1 \Rightarrow) & A \to (B/A)\backslash B & (R2 \Rightarrow) \end{array}$$

The Application schemata are no doubt the most familiar laws of categorial combinatorics. The original categorial grammars of Ajdukiewicz and Bar-Hillel in fact were restricted to Application. Using the Application schemata, one can 'lexicalize' the rules of a context-free phrase structure grammar. Take the productions *S* $\longrightarrow$ *NP VP* and *VP* $\longrightarrow$ *TV NP* for the derivation of a Subject-Transitive Verb-Object (SVO) pattern. In categorial terms, the sentence is projected from the Transitive Verb, which is typed $(np\backslash s)/np$. The SVO pattern is obtained in two Application steps: rightward application consumes the Object, leftward application the Subject. The auxiliary label *VP* disappears; the complex type $np\backslash s$ expresses its combinatory role.

For an illustration of the Lifting schemata, let us assume we assign the type np to simple proper names. Instances of Lifting would be the type transitions from np to $s/(np\backslash s)$ or $((np\backslash s)/np)/(np\backslash s)$. These lifted types would be appropriate for noun phrases with a distribution restricted to the subject position, in the case of $s/(np\backslash s)$, or the direct object position, in the case of $((np\backslash s)/np)/(np\backslash s)$. What the derivability arrow says here is that any expression that is assigned the type np will be able to occur in subject or object position, but that there can be expressions with a restricted subject or object distribution, expressed through the higher order types $s/(np\backslash s)$ or $((np\backslash s)/np)/(np\backslash s)$. One can think here of case-marked pronouns. Assuming that the lexical type assignment for *he/she* is $s/(np\backslash s)$, but that *him/her* are lexically typed as $((np\backslash s)/np)/(np\backslash s)$, our example grammar will correctly rule out 'him irritates she' while allowing 'he irritates her'.

Elementary theorems for the unary type-forming operations are established in (3).

(3)
$$\begin{array}{llll} \Box A \rightarrow \Box A & (Ax) & \Diamond A \rightarrow \Diamond A & (Ax) \\ \Diamond\Box A \rightarrow A & (R0 \Leftarrow) & A \rightarrow \Box\Diamond A & (R0 \Rightarrow) \end{array}$$

Polarity sensitive items (**??**) provide a simple illustration for the added expressivity of the unary type-forming operators. Consider the contrast between 'Nobody left yet' with the negative polarity item 'yet' and '*Somebody left yet'. In a type language with just the binary type-forming operations, one could assign both 'somebody' and 'nobody' the subject type $s/(np\backslash s)$, and 'yet' the modifier type $(np\backslash s)\backslash(np\backslash s)$. Such type assignment is too crude to block the ungrammatical '*Somebody left yet'. In the extended type language, the negative polarity trigger 'nobody' can be assigned the type $s/\Box\Diamond(np\backslash s)$, whereas 'somebody' keeps the undecorated type $s/(np\backslash s)$. By typing the negative polarity item 'yet' as $(np\backslash s)\backslash\Box\Diamond(np\backslash s)$ one can express the fact that it requires a trigger such as 'nobody' to check the $\Box\Diamond$ decoration in its numerator subtype. For the derivation of the simple sentence 'Nobody left' (where there is no polarity item to be checked), we rely on the fact that in the base logic, we have $s/\Box\Diamond(np\backslash s) \rightarrow s/(np\backslash s)$, i.e. the $\Box\Diamond$ decoration on argument subtypes can be simplified away, allowing the combination (in terms of the Application schema) of 'nobody' with a simple verb phrase 'left' of type $np\backslash s$.

Monotonicity properties. Apart from the theorems above, the base logic has the derived rules of inference below, expressing the monotonicity properties of the type-forming operations. With respect to the derivability relation, the operations $\Diamond$ and $\Box$ are order-preserving (isotone). The $\bullet$ operation is order-preserving in its two arguments; the division operations / and \ are order-preserving in their numerator, and order-reversing (antitone) in their denominator argument.

(4)
$$\begin{array}{lllll} A \rightarrow B & \text{implies} & \Diamond A \rightarrow \Diamond B & \text{and} & \Box A \rightarrow \Box B \\ & & A/C \rightarrow B/C & \text{and} & C\backslash A \rightarrow C\backslash B \\ & & C/B \rightarrow C/A & \text{and} & B\backslash C \rightarrow A\backslash C \\ & & A \bullet C \rightarrow B \bullet C & \text{and} & C \bullet A \rightarrow C \bullet B \end{array}$$

From a combinatorial point of view, one can see these rules as operations to produce an infinite number of type transformations from some small inventory of 'primitive' ones. Consider the Lifting schema. From it, one obtains the transformations known as Value Raising (for example, lifting a determiner type np/n to $(s/(np\backslash s))/n$) and Argument Lowering (for example, lowering a third-order verb phrase type $(s/(np\backslash s))\backslash s$ to first-order $np\backslash s$).

Alternative presentations, Natural Deduction. The categorial base logic allows many alternative axiomatizations, each serving its own function. The essential point is that the different presentations must find their justification in the modeltheoretic interpretation of the connectives, i.e. they are equivalent syntaxes for performing valid type computations. In the Gentzen sequent calculus, one replaces the arrows $A \to B$ by statements $\Gamma \Rightarrow B$ ('structure Γ is of type B). The antecedent Γ is built out of formulas by means of the structural connectives $\langle\cdot\rangle$ and $(\cdot \circ \cdot)$, counterparts of the logical connectives $\Diamond$ and $\bullet$. The purpose of this presentation is to show that the transitivity rule (the Cut rule) can be eliminated. Every logical rule of inference in the Gentzen calculus introduces a connective either in the antecedent or in the succedent, so that backward-chaining, cut-free proof search immediately yields a decision procedure for categorial derivability.

The derivational format of Combinatory Categorial Grammar (CCG) is a Hilbert-style presentation. Functional Application here is taken as the basic, primitive schema for type combination. To the Application schema are added extra schemata, such as Lifting, which is known as the combinator **T**. The CCG format of derivations is related to the Gentzen style as the the combinator presentation of intuitionistic logic is to its Gentzen presentation. The recursive generalization of the primitive type transformations under monotonicity is important for such 'combinatory' presentations of categorial derivability: without this generalization, one loses completeness.

In a third format, Natural Deduction (ND), every type-forming connective has an introduction and an elimination rule. As a result, ND doesn't have the pleasant proof search properties of the Gentzen calculus, but it is a perspicuous presentation of a derivation once it has been found. For this reason, ND is often used in linguistic discussion of categorial analyses. Also, ND is the most transparant format to associate meaning assembly with a derivation, as we will see in §3. We present the ND rules for the base logic below, using the Gentzen sequent style, which is explicit about the structural configuration of the antecedent assumptions.

$$[/\mathrm{I}]\frac{\Gamma \circ B \vdash A}{\Gamma \vdash A/B} \qquad \frac{\Gamma \vdash A/B \qquad \Delta \vdash B}{\Gamma \circ \Delta \vdash A}[/\mathrm{E}]$$

$$[\backslash\mathrm{I}]\frac{B \circ \Gamma \vdash A}{\Gamma \vdash B\backslash A} \qquad \frac{\Gamma \vdash B \qquad \Delta \vdash B\backslash A}{\Gamma \circ \Delta \vdash A}[\backslash\mathrm{E}]$$

$$[\bullet\mathrm{I}]\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma \circ \Delta \vdash A \bullet B} \qquad \frac{\Delta \vdash A \bullet B \qquad \Gamma[A \circ B] \vdash C}{\Gamma[\Delta] \vdash C}[\bullet\mathrm{E}]$$

Figure 1: Natural deduction: binary connectives. Notation: $\Gamma \vdash A$ for the deduction of a conclusion A from a configuration of assumptions Γ. Antecedent structures are built from formulas with $(- \circ -)$: the structural connective corresponding to the logical connective $(- \bullet -)$. Axioms: $A \vdash A$.

$$\frac{\Gamma \vdash \Box A}{\langle\Gamma\rangle \vdash A}\ (\Box E) \qquad \frac{\langle\Gamma\rangle \vdash A}{\Gamma \vdash \Box A}\ (\Box I)$$

$$\frac{\Gamma \vdash A}{\langle\Gamma\rangle \vdash \Diamond A}\ (\Diamond I) \qquad \frac{\Delta \vdash \Diamond A \quad \Gamma[\langle A\rangle] \vdash B}{\Gamma[\Delta] \vdash B}\ (\Diamond E)$$

Figure 2: Natural deduction: control features. $\langle\cdot\rangle$ as the structural counterpart of $\Diamond$.

Multimodal generalization. All that was said above can be straightforwardly generalized to a situation where one has not just one single merge and feature checking relation, but families of them. In modal logic terms, this means moving from a unimodal system to a multimodal one, with frames $\langle W, \{R_i^2\}_{i\in I}, \{R_j^3\}_{j\in J}\rangle$ where the different relations are kept apart by indexing them with a composition mode label. Similarly, in the formula language, we index the connectives for these composition modes. The concept of multiple modes of composition is not unfamiliar. For the binary operations, one can think of a distinction between the structure of words (morphology) and the structure of phrases (syntax): one can give a categorial analysis of morphological and syntactic composition in terms of $/, \bullet, \backslash$, but still one will want to keep these grammatical levels distinct, say as $\bullet_w$ versus $\bullet_\phi$. For the unary connectives $\diamondsuit, \Box$, multimodality makes it possible to distinguish a number of named features in the grammar, so that they can play different roles in controlling composition.

The multimodal perspective will turn out to be particularly useful once we move beyond the base logic and consider its structural extensions, where one then can have interaction between different binary composition modes (between morphology and syntax, in the case of complement inheritance, for example), and between specific unary control features and binary composition operations. Such interaction principles will be discussed in the next section.

2.2 The structural module

The laws of the base logic do not depend on specific structural properties of the 'Merge' and feature-checking relations: the completeness theorem for **B** does not impose any restrictions on the interpretation of $R_\bullet$ and $R_\diamondsuit$. In this sense, the base logic can be said to capture the *invariants* of grammatical composition. Although **B** already has a rich deductive structure as we have seen above, the system also has its limitations. If an expression can occur in different structural configurations, one would like to relate these configurations. In the base logic, this cannot be done: type assignment is structurally rigid, in the sense that different structural environments will lead to different type assignments. An example would be the lexical entry for the relative pronoun in (**??**). The type $(n\backslash n)/(np\backslash s)$ is the solution to the type-assignment equation for the configuration where the relative pronoun binds the subject of the relative clause body. Direct object relativization (as in 'the book that Alice read') would require another type assignment.

To overcome the problem of structural rigidity, one extends **B** with facilities for structural reasoning. Technically, such facilities have the status of *non-logical* axioms, or postulates. They can be introduced in a global, or in a controlled fashion. We discuss these in turn.

Global structural rules. Consider first the postulates in (5). The associativity postulate A_l says that in order to decide whether a left-branching configuration is well-formed, one may have to consider a right-branching alternative. Similarly for A_r, where one makes a judgment about the well-formedness of a right-branching configuration on the basis of a left-branching alternative. The commutativity postulate C says that changing the linear order of the grammatical resources does not affect derivability. If this postulate holds, the distinction between left-incompleteness and right-incompleteness collapses.

(5)
$$\begin{array}{ll} (A \bullet B) \bullet C \to A \bullet (B \bullet C) & \mathsf{A}_l \\ A \bullet (B \bullet C) \to (A \bullet B) \bullet C & \mathsf{A}_r \\ A \bullet B \to B \bullet A & \mathsf{C} \end{array}$$

The structural options of (5) create a hierarchy of categorial systems. We introduce some terminology. Adding A_l and A_r together to the $/, \bullet, \backslash$ fragment of the base logic, one obtains the system known as **L**, after the associative calculus of Lambek 1958. The $/, \bullet, \backslash$ fragment of the base logic is also know as **NL**: originally this systems was presented as the type calculus obtained by dropping the associativity postulates from **L**. Adding the commutativity postulate to **L** produces the system known as **LP** (the Lambek calculus with permutation). This system coincides with the multiplicative fragment of Linear Logic, which has a commutative product operation matched by a single linear implication.

One can view the categorial hierarchy from two angles: as one ascends the hierarchy, flexibility of type combination increases, but structural discrimination deteriorates. We illustrate this with the facilities for rebracketing in **L**. Below in (6) the type transitions known as the Geach laws. With the aid of A_r (A_l) and residuation reasoning in the base logic, one derives these as theorems. Alternatively, one could add G_r (G_l) as postulates to the base logic, and derive the product forms of associative rebracketing as theorems. From the Geach laws, one immediately derives the schemata of functional Composition (combinator **B** in CCG): B_r and B_l are the simplest forms.

(6) $$\begin{array}{llll} \mathsf{G}_r & A/B \to (A/C)/(B/C) & B\backslash A \to (C\backslash B)\backslash(C\backslash A) & \mathsf{G}_l \\ \mathsf{B}_r & (A/B)\bullet(B/C) \to A/C & (C\backslash B)\bullet(B\backslash A) \to C\backslash A & \mathsf{B}_l \end{array}$$

There are many places in grammar where facilities for rebracketing, and the non-standard constituent structures they engender, are helpful. Cases of non-constituent coordination, as in (7), can serve as an example. On the assumption that the coordination particles *and/but* make a conjunction phrase X out of conjuncts of that same type, the type s/np is a solution for X in the right-node raising construction of (7a). To obtain that solution, one needs a rebracketing postulate (A_r, or G_r) in addition to the residuation reasoning of the base logic. Similarly, in the case of (7b), the solution $X = ((iv/np)/np)\backslash iv$ turns the composition of the noun phrases into a function consuming the ditransitive *offered* and returning an intransitive verb phrase (iv as an abbreviation for $np\backslash s$). For this solution, one needs the postulates (A_l, or G_l) in addition to the residuation reasoning of the base logic. But where an appeal to rebracketing seems to make sense in the context of coordination particles, the postulates A_l and A_r together have the effect that hierarchical constituent structure could not play any role in determining grammaticality — obviously a claim that few linguists would seriously uphold. The ensuing loss of discrimination is illustrated in (7c,d). A typing $tv\backslash iv$ is appropriate for the reflexive pronoun. But in the presence of global associativity, there is no way of distinguishing the well-formed (7c) from the ill-formed (7d).

(7) *a* $\underbrace{\text{the Lobster loves}}_{s/np}$ but $\underbrace{\text{the Gryphon hates}}_{s/np}$ Turtle Soup

b Alice offered $\underbrace{\text{Tweedledum a biscuit}}_{((iv/np)/np)\backslash iv}$ and $\underbrace{\text{Tweedledee a cup of tea}}_{((iv/np)/np)\backslash iv}$

c the Mad Hatter $\underbrace{\text{loves}}_{(np\backslash s)/np}$ himself (himself : $tv\backslash iv = ((np\backslash s)/np)\backslash(np\backslash s))$

d *the Mad Hatter $\underbrace{\text{thinks Alice loves}}_{(np\backslash s)/np}$ himself

Controlled structural reasoning. To constrain the structural extensions of the base logic, various strategies have been pursued. In the rule-based approach of Combinatory Categorial Grammar, one augments the Application/Lifting basis with structural combinators which, in an unconstrained form, would be overgenerating. An example is the following form of the combinator **B**, known as forward crossed composition, used to derive the crossing dependencies of verbal clusters in the Dutch subordinate clause.

(8)
a. (omdat) Alice de taartjes wil$_{vp/inf}$ stelen$_{np\backslash inf}$
b. (because) Alice the tarts wants steal
c. (because) Alice wants to steal the tarts

(9) $$\mathsf{B}_\times \quad (A/B)\bullet(C\backslash B) \to C\backslash A$$

In the base logic (and in **L**) this combinator is unsound: its derivation involves a form of commutativity. To avoid collapse to the stronger system **LP**, where the combinator would be sound indeed, CCG imposes type constraints on such rules (restricting the middle term B in this case to certain verbal categories). In addition, the set of rule schemata (combinators) is kept finite, so that one can avoid the consequences of the recursive generalization of rules under monotonicity.

The alternative is to exploit the intrinsic *logical* instruments for structural control offered by richer type systems with multimodal interaction principles. Suppose one distinguishes regular phrasal composition ($\bullet_1$) from the composition operation that forms Dutch verbal clusters ($\bullet_0$), and types the cluster-forming 'wil' as $vp/_0 inf$ and the transitive infinitive 'stelen' as $np\backslash_1 inf$. One can then impose an interaction postulate/combinator $\mathsf{B}'_\times$ (or the equivalent product form) that allows restructuring when these two modes are in construction with each other, without attributing commutativity/associativity behaviour to the individual modes.

(10)
$$\begin{array}{ll} \mathsf{B}'_\times & (A/_0 B) \bullet_0 (C\backslash_1 B) \to C\backslash_1 A \\ & A \bullet_1 (B \bullet_0 C) \to B \bullet_0 (A \bullet_1 C) \end{array}$$

In a similar vein, one can introduce interaction postulates relating the unary operators to binary composition. Consider the postulates below, implementing controlled forms of reordering and restructuring restricted to expressions carrying the licensing $\Diamond$ feature. With a lexical type assignment $(n\backslash n)/(s/\Diamond\Box np)$ to the relative pronouns 'that'/'which'/'whom', these postulates allow medial and peripheral extraction from right branches. We give an example in Figure 2.2.

(11)
$$\begin{array}{ll} P1 & (A \bullet B) \bullet \Diamond C \to (A \bullet \Diamond C) \bullet B \\ P2 & (A \bullet B) \bullet \Diamond C \to A \bullet (B \bullet \Diamond C) \end{array}$$

The general theory of $\Diamond, \Box$ as control operators has been investigated in [21]. These authors establish a number of embedding theorems showing that the full logical space between the base logic and **LP**/Linear Logic can be navigated in terms of the control connectives, both in the 'licensing' direction illustrated above (allowing structural inferences that would be unavailable without the control features) and in the 'constraining' sense (blocking structural options that would be licit in the absence of the control features). These results show that $\Diamond, \Box$ are indeed the right logical constants for capturing the traffic laws of structural communication. On the other hand, we know that natural languages inhabit only a small region of the structural landscape that is opened up by the $\Diamond, \Box$ connectives. An important theme of current research has to do with charting the boundaries of structural variation: Can one narrow down the class of allowable postulates for structural reasoning in terms of substantive linguistic constraints? We return to this question below.

$$\begin{prooftree}
\AxiomC{what}
\UnaryInfC{$wh/(s/\Diamond\Box np)$}
\AxiomC{Alice}
\UnaryInfC{np}
\AxiomC{found}
\UnaryInfC{$(np\backslash s)/np$}
\AxiomC{(3)}
\UnaryInfC{$\Diamond\Box np \vdash np$}
\RightLabel{$[/E]$}
\BinaryInfC{found $\circ \Diamond\Box np \vdash np\backslash s$}
\AxiomC{there $\vdash (np\backslash s)\backslash(np\backslash s)$}
\RightLabel{$[\backslash E]$}
\BinaryInfC{(found $\circ \Diamond\Box np) \circ$ there $\vdash np\backslash s$}
\RightLabel{$[\backslash E]$}
\BinaryInfC{Alice $\circ$ ((found $\circ \Diamond\Box np) \circ$ there) $\vdash s$}
\RightLabel{$[P1]$}
\UnaryInfC{Alice $\circ$ ((found $\circ$ there) $\circ \Diamond\Box np) \vdash s$}
\RightLabel{$[P2]$}
\UnaryInfC{(Alice $\circ$ (found $\circ$ there)) $\circ \Diamond\Box np \vdash s$}
\RightLabel{$[/I]$}
\UnaryInfC{Alice $\circ$ (found $\circ$ there) $\vdash s/\Diamond\Box np$}
\RightLabel{$[/E]$}
\BinaryInfC{what $\circ$ (Alice $\circ$ (found $\circ$ there)) $\vdash wh$}
\end{prooftree}$$

Figure 3: Non-peripheral extraction. The type-assignment to the relativizer 'what' expresses the fact that the relative clause body is a sentence built with the help of a 'gap' hypothesis of type $\Diamond\Box np$. The feature-marked hypothesis has to be withdrawn at the right periphery, but it is not selected in that position. It is related to the non-peripheral direct object position within the relative clause body by virtue of the postulates $P1$ and $P2$. Once it has found the direct object position, the licensing feature $\Diamond$ has done its work and can be cleaned up by the law $\Diamond\Box np \to np$. The 'gap' hypothesis is then used as a regular direct object with respect to the selecting verb 'found'.

Frame constraints and specialised models. To conclude this section, some remarks on the modeltheoretic consequences of adding facilities for structural reasoning have to be made. For a type logical system with structural postulates, obviously, an interpretation with respect to *arbitrary* frames is not available any more. Instead, each postulate introduces a corresponding frame constraint restricting the interpretation of the $R_\bullet$ and/or $R_\diamond$ relations involved, and completeness is stated with respect to frames respecting the relevant constraints. A Commutativity postulate, for example, would impose the semantic constraint that for all $x, y, z \in W$, $R_\bullet xyz$ implies $R_\bullet xzy$. Similarly for the other postulates discussed. In the presence of such semantic constraints, it will often be the case that one can specialize the abstract relational interpretation to more concrete models. A good example is the system **L** with its associative composition relation $R_\bullet$. In this case, one can read $R_\bullet xyz$ as $x = y \cdot z$, where '$\cdot$' is the concatenation operation. It has been shown by Pentus [31] that indeed **L** is complete with respect to this concatenation (or free semigroup) interpretation.

3 Meaning assembly: the Curry-Howard correspondence

In the previous section, we have given an interpretation of the categorial type language in the dimension of grammatical form. We now turn to the meaning dimension of linguistic signs. The execution of the parsing-as-deduction programme in this dimension can be exactly parallel to what we have seen for syntax. For modeltheoretic interpretation of the type language, a family of semantic domains is set up. Proofs (derivations) in the type logic are interpretated as programs, proof search as computation — computation of meaning assembly in this case.

3.1 Type-theoretic semantics and the lambda calculus

For semantic interpretation, we associate every type A with a semantic domain D_A. Expressions of type A find their denotations in D_A. Semantic domains can be set up in two ways: directly on the basis of the types as discussed in the previous section, or indirectly, via a mapping from syntactic to semantic types. The indirect option can be attractive for a number of reasons. On the level of atomic types, one may want to make different basic distinctions depending on whether one uses syntactic or semantic criteria. For complex types, a map from syntactic to semantic types makes it possible to forget information that is relevant only for the way expressions are to be configured in the form dimension.

Semantic and syntactic types. For a simple extensional interpretation, the set of atomic semantic types $\mathsf{SemAtom}$ could consist of types e and t, with D_e the domain of discourse (a non-empty set of entities, objects), and $D_t = \{0, 1\}$, the set of truth values. The full set of semantic types $\mathsf{SemType}$ is then obtained by closing $\mathsf{SemAtom}$ under the rule that if A and B are in $\mathsf{SemType}$, then $A \to B$ is also. $D_{A \to B}$, the semantic domain for a functional type $A \to B$, is the set of functions from D_A to D_B. The mapping from syntactic to semantic types $(\cdot)^*$ could now stipulate for basic syntactic types that $np^* = e$, $s^* = t$, and $n^* = (e \to t)$. Sentences, in this way, denote truth values; (proper) noun phrases individuals; common nouns functions from individuals to truth values. For complex syntactic types, we set $(A/B)^* = (B\backslash A)^* = B^* \to A^*$. On the level of semantic types, the directionality of the slash connective is no longer taken into account. The distinction between numerator and denominator — domain and range of the interpreting functions — is kept. Notice that both verb phrases with syntactic type $np\backslash s$ and common nouns are mapped to the semantic type $e \to t$.

The language of the simply typed lambda calculus. We build up the set of meaningful expressions (terms) of semantic type A, starting from a denumerably infinite set of variables for each type. For each expression t of type A, we specify its interpretation $[\![t]\!]^g$ relative to an assignment function g which assigns to each variable of type A a member of D_A.

Variables Let x be a variable of type A. Then x is a term of type A. Interpretation: $[\![x]\!]^g = g(x)$.

Application Let t and u be terms of type $A \to B$ and A respectively. Then $(t\ u)$ is a term of type B. Interpretation: $[\![(t\ u)]\!]^g = [\![t]\!]^g\ ([\![u]\!]^g)$, i.e. the value one obtains when applying the function $[\![t]\!]^g$ to $[\![u]\!]^g$.

Abstraction Let x be a variable of type A and t a term of type B. Then $\lambda x.t$ is a term of type $A \to B$. Interpretation: $[\![\lambda x.t]\!]^g$ is that function h from D_A into D_B such that for all objects $k \in D_A$,

$h(k) = [\![t]\!]^{g[x:=k]}$, where $g[x := k]$ is the assignment that is exactly like g except for the possible difference that it assigns the object k to the variable x.

Given this interpretation, certain equalities hold between terms. One can see them as syntactic simplifications, replacing a more complex term (the *redex*) by a simpler one with the same interpretation (the *contractum*).

(12)
$$\begin{array}{lcll} (\lambda x.t)\ u & \rightsquigarrow_\beta & t[u/x] & \text{provided } u \text{ is free for } x \text{ in } t \\ \lambda x.(t\ x) & \rightsquigarrow_\eta & t & \text{provided } x \text{ is not free in } t \end{array}$$

3.2 Formulas-as-types, proofs as programs

Curry's basic insight was that one can see the functional types of type theory as logical implications, giving rise to a one-to-one correspondence between typed lambda terms and natural deduction proofs in positive intuitionistic logic. A natural deduction presentation for $\rightarrow$ starts from identity axioms $A \vdash A$ and has the introduction and elimination rules below, where Γ, Δ represent finite lists of formulas, and where $\Gamma - A$ is the result of dropping no, some or all occurrences of A from Γ.

(13)
$$\frac{\Gamma \vdash A \rightarrow B \quad \Delta \vdash B}{\Gamma, \Delta \vdash B} \rightarrow \text{Elim} \qquad \frac{\Gamma \vdash B}{\Gamma - A \vdash A \rightarrow B} \rightarrow \text{Intro}$$

Let us write $\Gamma(t)$ for the string of types of free occurrences of variables in a term t. Each term t of type A now encodes a natural deduction proof of the sequent $\Gamma(t) \vdash A$. The Variable clause in the definition of well-formed terms corresponds to the axiom sequent, the Application clause to $\rightarrow$ Elimination, and the Abstraction clause to $\rightarrow$ Introduction, where the dropped A assumption corresponds to the variable bound by the lambda abstractor. In the opposite direction, every natural deduction proof is encoded by a lambda term. The *normalization* of natural deduction proofs corresponds to the β/η reductions of terms.

Translating Curry's 'formulas-as-types' idea to the categorial type logics we are discussing, we have to take the differences between intuitionistic logic and the grammatical resource logic into account. Below we repeat the natural deduction presentation of the base logic, now taking term-decorated formulas as basic declarative units. Judgements take the form of sequents $\Gamma \vdash t : A$. The antecedent Γ is a structure with leafs $x_1 : A_1, \ldots, x_n : A_n$. The x_i are unique variables of type A_i^*, where $(\cdot)^*$ is the mapping from syntactic to semantic types. The succedent is a term t of type A^* with exactly the free variables $x_1, \ldots, x_n$, representing a program which given inputs $k_1, \ldots, k_n$ produces $[\![t]\!]$ under the assignment that maps the variables x_i to the objects k_i. The x_i in other words are the parameters of the meaning assembly procedure. A derivation starts from axioms $x : A \vdash x : A$. The Elimination and Introduction rules have a version for the right and the left implication. On the meaning assembly level, this syntactic difference is ironed out, as we already saw that $(A/B)^* = (B\backslash A)^*$. As a consequence, we don't have the *isomorphic* (one-to-one) correspondence between terms and proofs of Curry's original program. But we do read off meaning assembly from the categorial derivation.

$$[/I]\frac{\Gamma \circ x : B \vdash t : A}{\Gamma \vdash \lambda x.t : A/B} \qquad \frac{\Gamma \vdash t : A/B \quad \Delta \vdash u : B}{\Gamma \circ \Delta \vdash (t\ u) : A}[/E]$$

$$[\backslash I]\frac{x : B \circ \Gamma \vdash t : A}{\Gamma \vdash \lambda x.t : B\backslash A} \qquad \frac{\Gamma \vdash u : B \quad \Delta \vdash t : B\backslash A}{\Gamma \circ \Delta \vdash (t\ u) : A}[\backslash E]$$

A second difference between the programs/computations that can be obtained in intuitionistic implicational logic, and the recipes for meaning assembly associated with categorial derivations has to do with the resource management of assumptions in a derivation. The formulation of the $\rightarrow$ introduction rule makes it clear that in intuitionistic logic, the number of occurrences of assumptions (the 'multiplicity' of the logical resources) is not critical. One can make this style of resource management explicit in the form of structural rules of Contraction and Weakening, allowing for the duplication and waste of resources.

(14)
$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\ C \qquad\qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ W$$

In contrast, the categorial type logics are resource sensitive systems where each assumption has to be used exactly once. At the level of **LP**, we have the following correspondence between resource constraints and restrictions on the lambda terms coding derivations:

1. no empty antecedents: each subterm contains a free variable;
2. no Weakening: each λ operator binds a variable free in its scope;
3. no Contraction: each λ operator binds at most one occurrence of a variable in its scope.

Moving from **LP** to the grammatical base logic imposes even tighter restrictions on binding: in the absense of Associativity and Commutativity, the slash introduction rules responsible for the λ operator can only reach the immediate daughters of a structural domain.

3.3 The syntax/semantics interface

Applied to the composition of natural language meaning, the 'proofs-as-programs' approach has some interesting consequences for the syntax/semantics interface.

A first point to notice is the strictly modular treatment of derivational versus lexical semantics. The proof term that is read off a derivation is a *uniform instruction* for meaning assembly that fully abstracts from the contribution of the particular lexical items on which it is built. As a result, no assumptions about lexical semantics can be built into the meaning assembly process as represented by a derivation. We illustrate the interplay between lexical and derivational semantics in Figures 3.3 and 3.3. Whereas the proof term in Fig 3.3 is a faithful encoding of the derivation (modulo directionality and structural operations), the term one obtains in Fig 3.3 after substitution of lexical meaning programs and β simplification has lost the transparency with respect to the derivation.

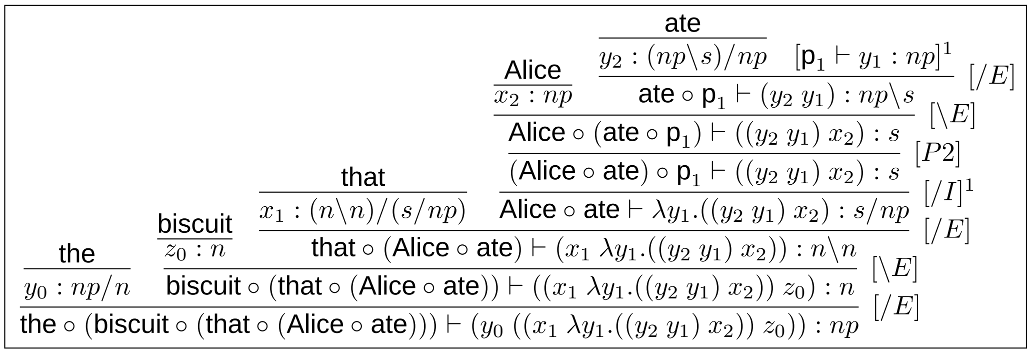
ate
$y_2 : (np\backslash s)/np$
$[\mathsf{p}_1 \vdash y_1 : np]^1$
$[/E]$
Alice
$x_2 : np$
ate $\circ$ $\mathsf{p}_1 \vdash (y_2\ y_1) : np\backslash s$
$[\backslash E]$
Alice $\circ$ (ate $\circ$ p_1) $\vdash ((y_2\ y_1)\ x_2) : s$
$[P2]$
(Alice $\circ$ ate) $\circ$ $\mathsf{p}_1 \vdash ((y_2\ y_1)\ x_2) : s$
$[/I]^1$
that
$x_1 : (n\backslash n)/(s/np)$
Alice $\circ$ ate $\vdash \lambda y_1.((y_2\ y_1)\ x_2) : s/np$
$[/E]$
biscuit
$z_0 : n$
that $\circ$ (Alice $\circ$ ate) $\vdash (x_1\ \lambda y_1.((y_2\ y_1)\ x_2)) : n\backslash n$
$[\backslash E]$
the
$y_0 : np/n$
biscuit $\circ$ (that $\circ$ (Alice $\circ$ ate)) $\vdash ((x_1\ \lambda y_1.((y_2\ y_1)\ x_2))\ z_0) : n$
$[/E]$
the $\circ$ (biscuit $\circ$ (that $\circ$ (Alice $\circ$ ate))) $\vdash (y_0\ ((x_1\ \lambda y_1.((y_2\ y_1)\ x_2))\ z_0)) : np$

Figure 4: Computation of the *proof term* for the np 'the biscuits that Alice ate'. Leafs are labeled with variables. The derivation produces a meaning recipe with parameters for the lexical meaning programs.

The second feature is the limited semantic expressivity of a structure-sensitive type logic: many forms of meaning assembly that can be straightforwardly expressed in the language of the lambda calculus cannot be obtained as Curry-Howard images of the Introduction/Elimination inferences of the categorial base logic.

To resolve the tension between structure-sensitivity and semantic expressivity, categorial grammars can exploit a combination of two strategies. Structural reasoning (in terms of combinators or structural postulates) makes it possible to explicitly determine which positions are accessible for semantic manipulation (binding). The example of controlled wh-extraction in 2.2 is an illustration. Secondly, lexical

1.	the : $np/n - \iota$	Lex
2.	biscuit : $n -$ **biscuit**	Lex
3.	that : $(n\backslash n)/(s/np) - \lambda z_{15}.\lambda x_{16}.\lambda y_{16}.((z_{15}\ y_{16}) \wedge (x_{16}\ y_{16}))$	Lex
4.	alice : $np -$ **a**	Lex
5.	ate : $(np\backslash s)/np -$ **eat**	Lex
6.	$\vert$ $\mathrm{p}_1 : np - y_1$	Hyp
7.	$\vert$ ate $\circ\ \mathrm{p}_1 : np\backslash s - (\mathbf{eat}\ y_1)$	$/E\ (5,6)$
8.	$\vert$ alice $\circ$ (ate $\circ\ \mathrm{p}_1) : s - ((\mathbf{eat}\ y_1)\ \mathbf{a})$	$\backslash E\ (4,7)$
9.	$\vert$ (alice $\circ$ ate) $\circ\ \mathrm{p}_1 : s - ((\mathbf{eat}\ y_1)\ \mathbf{a})$	$P2\ (8)$
10.	alice $\circ$ ate : $s/np - \lambda y_1.((\mathbf{eat}\ y_1)\ \mathbf{a})$	$/I\ (6,9)$
11.	that $\circ$ (alice $\circ$ ate) : $n\backslash n - \lambda x_{16}.\lambda y_{16}.(((\mathbf{eat}\ y_{16})\ \mathbf{a}) \wedge (x_{16}\ y_{16}))$	$/E\ (3,10)$
12.	biscuit $\circ$ (that $\circ$ (alice $\circ$ ate)) : $n - \lambda y_{16}.(((\mathbf{eat}\ y_{16})\ \mathbf{a}) \wedge (\mathbf{biscuit}\ y_{16}))$	$\backslash E\ (2,11)$
13.	the $\circ$ (biscuit $\circ$ (that $\circ$ (alice $\circ$ ate))) : $np - (\iota\ \lambda y_{16}.(((\mathbf{eat}\ y_{16})\ \mathbf{a}) \wedge (\mathbf{biscuit}\ y_{16})))$	$/E\ (1,12)$

Figure 5: Substitution of lexical semantics. Boldface for non-logical constants. In steps 11 and 12, β conversion is applied on the fly to the application terms obtained from the slash elimination rules. The proof tree is presented in the linear or Fitch style Natural Deduction format.

meaning programs do not have to obey the resource constraints of the derivational semantics. Specifically, we do not impose the single-bind condition on lexical meanings (although the ban on vacuous abstraction does make sense, also in the lexicon.) An example of multiple binding is the lexical lambda term for the relative pronoun 'that' in 3.3, a program which computes property intersection. Another example would be a reflexive pronoun like 'himself', which we typed $tv\backslash iv$ above. It consumes its tv argument in a resource-sensitive way, realizing the identification of subject and object arguments of the verb through its lexical lambda term $\lambda x \lambda y.((x\ y)\ y)$.

Quantifier scope, anaphora. The interplay between these two strategies in current research is nicely illustrated by the construal of quantifier scope ambiguities and antecedent-anaphor dependencies. Generalized quantifier expressions like 'everyone', 'someone', 'nobody', require an interpretation as sets of properties, i.e. they find a denotation in $D_{(e\to t)\to t}$. A syntactic type compatible with such denotations would be $s/(np\backslash s)$. But there are two problems with such a type. First of all, it is restricted to subject position, and one wouldn't like to resort to multiple type assignments for non-subject occurrences. Secondly, it doesn't allow non-local scope readings, as in (c) below, where the embedded quantifier takes scope at the main clause level.

(15)
a. Alice thinks someone left.
b. $((\mathbf{think}\ (\exists\ \lambda x.(\mathbf{leave}\ x)))\ \mathbf{a})$
c. $(\exists\ \lambda x.((\mathbf{think}\ (\mathbf{leave}\ x))\ \mathbf{a}))$

What is needed here is the syntactic/semantic behaviour expressed by the following natural deduction rule: a GQ expression occupies a syntactic noun phrase position within a sentential domain, binding the corresponding e type variable when it takes scope.

(16)
$$\frac{\Delta \vdash t : GQ \qquad \Gamma[x : np] \vdash u : s}{\Gamma[\Delta] \vdash (t\ \lambda x.u) : s}\ GQ\ \text{Elim}$$

There have been various solutions to this problem. In the multimodal style, one defines GQ as $s/_i(np\backslash_j s)$ and formulates structural postulates for the i and j modes to allow non-local scoping. The semantic action $(t\ \lambda x.u)$ is produced automatically as the Curry-Howard image of the slash introduction and elimination steps. An alternative approach in the combinatory style postulates a set of type-changing rule schemata which, in combination with the basic function application schema, produce the required scope flexibility. These approaches are discussed in depth in Carpenter [9]. A more radical alternative in [13] uses the $\lambda\mu$ calculus, an extension of the lambda calculus inspired by the continuation semantics of programming language theory. A scopally ambiguous sentence in this system has a unique syntactic derivation/proof. The scope ambiguity results from different possibilies of reducing the proof term to normal form.

A common property of these approaches is the prediction that in sentences with multiple quantifiers, the full set of combinatorially possible scope relations would be available. This prediction is not correct. Within the broad class of generalized quantifier expressions, one can identify various subclasses with distinct scopal possibilities. One can accommodate such differential scopal behaviour by refining type assignment from $s/_i(np\backslash_j s)$ to $s'/_i(np\backslash_j s'')$, where s' and s'' are feature-decorated s types. The derivability patterns $\Diamond\Box s \to s \to \Box\Diamond s$ then provide a way of proof-theoretically forcing certain scope construals while ruling out others. Alternatively, one can impose extra-logical constraints on derivations.

The construal of antecedent-anaphora relations, like that of quantifier scope, involves non-local dependencies beyond the reach of the grammatical base logic, as in the example below, where the anaphor in the subordinate clause has to pick up its antecedent in the main clause.

(17) $a.$ *Alice* thinks *she* dreams
$b.$ ((**think** (**dream a**)) **a**)

In addition, meaning composition for anaphora resolution involves a duplication of resources, in the sense that one would like to make the pronoun 'she' in the example above responsible for the copying of the antecedent meaning. Jaeger [17] offers an in-depth discussion of the options. [Identity semantics $\lambda x.x$ for the pronoun, in combination with a restricted copying rule in syntax, in the form of a controlled structural rule of Contraction. Jacobson's combinator based alternative. Combination: 'she': $\lambda x.\langle x, x\rangle$ (pair formation is the Curry-Howard image of $\bullet$ Introduction); $np\backslash_i(np \bullet_j np)$ with structural rules for the modes.]

4 Themes and variations

We close with a brief discussion of some active areas of current research.

4.1 Extending the type language

The connectives we have discussed above offer a type-driven perspective on the structure-building operations of language, and their correlates for meaning assembly. This is the core part of the categorial vocabulary. To deal with other aspects of grammatical organization, the type language can be extended with appropriate type-forming operations. By introducing Boolean connectives, one provides the means for conjunctive or disjunctive reasoning about type assignments. Various sorts of type polymorphism can be analysed by moving from a propositional type language to a language with explicit quantification. In the first order case, the atomic formulae become predicates, their argument positions filled by terms built up from feature constants and variables. A third person singular feminine noun phrase, for example, could be typed as $np(3, \mathit{sing}, \mathit{fem})$, and from this type, one can derive the generalized form $\exists G.np(3, \mathit{sing}, G)$. Notions of underspecification, unification or generalization in this way can be expressed in terms of explicit quantification over feature variables. In the second order case, the type language is extended with quantification over *type* variables. Chameleon words, such as the coordination particles, can then be assigned a type schema $\forall X.(X\backslash X)/X$ which adapts to the context it finds itself in via the logical rules for the quantifier. Morrill's book [28] provides many illustrations of the use of these extended type systems in linguistic analysis, and a comparison with the implicit way of handling quantification in unification-based formalisms, or in hybrid categorial-unificational systems.

In general, there is a trade-off here between the expressivity of the multiplicative component and the need for extra first or second order devices for polymorphism. With a more expressive multiplicative system, one can establish a derivational relationship between types in terms of the logical and structural rules for the composition operations and the control features. As a result, it will often be possible to reduce 'variable' polymorphism of the type discussed here, to derivational polymorphism.

Complexity is one of the reasons to optimally exploit the multiplicative component: the resource-conscious multiplicative systems we have considered all enjoy *decidability*, whereas the second order Lambek calculus for example is undecidable.

4.2 Generative capacity and computational complexity

The relations between the categorial landscape and the Chomsky hierarchy have been well investigated. The reader is refered to [5] for a recent survey. The discovery of dependency patterns in natural languages that cannot be adequately captured by context-free grammars has led to an interest in what are known as 'mildly context-sensitive' formalisms, i.e. systems with an expressivity beyond context-free, but sufficiently restricted to have polynomial parsing algorithms. The classical **AB** grammars have long been known to be weakly equivalent to context-free grammars, hence to be too poor to serve as models of Universal Grammar. The correctness of Chomsky's conjecture that this equivalence extends to the Lambek calculus **L** was finally established by Pentus in [30]. This result does not have a direct corollary for polynomial parsability, unfortunately, because the construction of a context-free grammar from an **L** grammar is of exponential complexity. The base logic, as we have described it above, is of polynomial complexity. The question is, then, how to extend this system with constrained facilities for structural reasoning. It is clear that arbitrary combinator extensions, or structural rule packages, lead to Turing expressivity. But it can be shown that an appropriately restricted version of CCG is weakly equivalent to the linear indexed grammars, hence polynomially parsable. In a similar spirit, Moot [26] shows how with appropriate restrictions on lexical assignments and structural postulates, one can carve out a class of multimodal categorial grammars equivalent with Lexicalized Tree Adjoining Grammars and inheriting the polynomial parsability of these systems. More important than weak generative capacity are issues of strong capacity, which in the categorial tradition would mean the proof structures (or their lambda terms) that produce a certain string. In this area, [36] has obtained some interesting results, showing that while systems like **NL** or **L** are weakly CF, their expressivity in terms of strong capacity goes beyond that of CF grammars.

4.3 Variants and alternatives

Pregroup grammars. An interesting variation on the categorial theme has been developed by Jim Lambek in a number of recent papers ([24, 1, 10]). The approach makes use of *pregroups*, algebraic structures closely related to the residuation-based models for the categorial type systems discussed here. A pregroup is a partially ordered monoid in which each element a has a left and a right adjoint, a^l, a^r, satisfying $a^l a \rightarrow 1 \rightarrow a a^l$ and $a a^r \rightarrow 1 \rightarrow a^r a$, respectively. Type assignment takes the form of associating a word with one or more elements from the free pregroup generated by a partially ordered set of basic types. For the connection with categorial type formulas, one can use the translations $a/b = ab^l$ and $b\backslash a = b^r a$. Parsing, in the pregroup setting, is extremely straightforward. Lambek [24] proves that one only has to perform the contractions replacing $a^l a$ and $a^l a$ by the multiplicative unit 1. This is essentially a check for well-bracketing — an operation that can be entrusted to a pushdown automaton. The expansions $1 \rightarrow a a^l$ and $1 \rightarrow a^r a$ are needed to prove equations like $(ab)^l = b^l a^l$. We have used the latter to obtain the pregroup version of the higher-order relative pronoun type $(n\backslash n)/(s/np)$ in the example below.

(18)

	book	that	Carroll	wrote	
CATEGORIAL TYPES :	n	$(n\backslash n)/(s/np)$	np	$(np\backslash s)/np$	
PREGROUP ASSIGNMENT :	n	$n^r\, n\, np^{ll}\, s^l$	np	$np^r\, s\, np^l$	$\rightarrow n$

Comparing the pregroup approach with the original categorial type system, one notices that the pregroup notation has associativity built in. This has pleasant consequences. In the standard Lambek calculus, the choice between $(np\backslash s)/np$ and $np\backslash(s/np)$ as the lexical type assignment for a transitive verb is in a certain sense arbitrary, given the fact that the associativity postulates make these types interderivable. The pregroup category format removes this notational overspecification: the two types translate to $np^r\, s\, np^l$. In general, every sequent derivable in the Lambek calculus will be derivable in the corresponding pregroup. The converse is not true: the pregroup image of the types $(a \bullet b)/c$ and $a \bullet (b/c)$, for example, is $a\,b\,c^l$, but these two types are not interderivable in **L**.

With respect to generative capacity, Buszkowski ([6]) shows that the pregroup grammars are equivalent to context-free grammars. They share, in other words, the expressive limitations of the original categorial grammars. To overcome these limitations in the analyses of German word order and Romance clitics referred to above, the authors rely on a combination of metarules and derivational constraints.

Minimalist grammars. Whereas the Chomskyan tradition of generative grammar and the categorial tradition have been moving in separate orbits for a long time, there are surprising convergences between resource-sensitive logics and Chomksy's recent 'Minimalist Program' when this is made mathematically precise, as in Stabler's [33] algebraic formulation. A minimalist grammar, in this format, consists of a lexicon of type assignments, closed under the structure-building operations Merge and Move. Type declarations are built up out of two sets of features with matching input/output polarities: category features and control features. The former govern the Merge operation, in which one easily recognizes the Modus Ponens/Application rule of categorial deduction. The control features explicitly license structural reasoning (Move), much like the unary multiplicatives $\Diamond, \Box$. The Stabler grammars have been shown to be weakly equivalent to Multiple Context Free Grammars, hence to fall within the class of mildy context-sensitive formalisms.

Comparing them with categorial logics, one notices that the minimalist category concept is essentially first-order: no use is made of hypothetical reasoning with respect to Merge. The restriction to Modus Ponens doesn't seem to be an essential limitation of the minimalist design, however. It would be interesting to extend minimalist grammars with facilities for hypothetical reasoning, which, as we have seen above, plays such a central role in the meaning assembly process.

Retoré and Stabler's theme issue of *Journal of Language and Computation* on the connections between categorial grammar and minimalism gives a good idea of this line of research.

4.4 Learning

Formal learnability theory for categorial grammar has been studied within the Gold paradigm of identification in the limit on the basis of positive data. A learner, in Gold's model, is presented with an infinite sequence of sentences from the language to be learned. For each sentence in the sequence, the learner formulates a conjecture (a grammar) as to the identity of the target language. Learning is considered to be successful if from a certain point on, the conjecture of the learner doesn't change anymore on the exposure to new data, and if at that point the conjecture is in fact a correct grammar for the target language. A *class* of languages is learnable in Gold's model if there is a learner that acquires a correct grammar from any infinite sequence of sentences from a language in that class.

Kanazawa [19] has obtained a number of important results within the Gold framework, building on the unification-based 'discovery procedures' for categorial grammars proposed by Buszkowski and Penn [8]. The focus of Kanazawa's work is on classical categorial grammars, using only the Application rules, and on combinatory extensions with extra rule schemata. On the input side, Kanazawa considers both learning from strings, and from function-argument structures. On the output side, the class of *rigid* grammars (where the grammar assigns a unique type to each word) is compared with the class of k-valued grammars (where at most k types are assigned to a lexical item).

The discussion in the previous section suggests some directions for further research in this area. First of all, one would like to obtain learnability results for classes of Lambek-style categorial grammars, where the learner has access to both the Elimination rules and the Introduction rules for the type-forming operators. Secondly, one would like to go beyond systems with a hard-wired structural component, in order to investigate the learnability effects of different choices of structural packages, in combination with an invariant base logic. The work of [11] is promising in this respect. Her approach to type conjoinability mixes unification/substitution with Lambek-style deduction, suggesting modulation of the conjoinability question in terms of different structural postulates. Finally, the role of *semantic* information in learning needs further investigation. The challenge here is to find a level of informativity that would be realistic in the setting of first language acquisition. Presenting the learner with a full λ recipe for the input data gives away too much of the problem to qualify as realistic in this sense. Function argument structures do provide important information about semantic dependencies, but only for languages where the dependency articulation coincides with surface constituent structure. To overcome this limitation, [**?**, **?**] study learning on the basis of 'non-projective' dependency structures, allowing discontinuous dependencies with respect to surface word order, and multiple dependencies in the case of hypothetical resources. Learning, in this setting, can be naturally broken up in two subtasks: finding the solutions to the type equations presented by the input dependency structures, and solving the structural equations that relate these dependency structures to the surface constituent articulation.

4.5 Processing issues

Incrementality, information structure. The flexible notion of derivational constituency engendered by type-changing principles has been exploited in the work of Steedman and others to model the incrementality of natural language processing. Given the categorial view on the syntax-semantics interface, such left-to-right parsing is directly compatible with incremental interpretation. In recent work, Steedman has shown that derivational constituency is guided by the *prosodic* articulation (intonation contour), which in turn gives rise to a richer notion of semantic interpretation in terms of focus and information structure. Steedman's proposals are formulated in the CCG style; Hendriks [14] analyses information packaging and intonation contour in multimodal type-logical terms.

Proof nets. A novel view on processing derives from the *proof net* approach, a graph-theoretic modelling of proofs originally developed in the context of Linear Logic. Proof nets offer a particularly attractive perspective on performance phenomena, as has been remarked by Johnson and Morrill [18, 29]. A net can be built in a left-to-right incremental fashion by establishing possible linkings between the input/output connectors of lexical items as they are presented in real time. This suggests a simple complexity measure on a traversal, given by the number of unresolved dependencies between literals. This complexity measure on incremental proof net construction makes the right predictions about a number of well-known processing issues, such as the difficulty of center embedding, garden path effects, attachment preferences, and preferred scope construals in ambiguous constructions. An illustration is presented in Figure XXX. In Linear Logic, as we have seen, resource occurrence is taken into account, but resource structure is ignored. Moot and Puite [27] introduce an adequate notion of proof nets for the grammatical type logics discussed in this article. These more structured nets can be the basis for further exploration of processing issues.

5 Historical remarks, reading suggestions

The history of categorial grammar is generally traced back to the work of Ajdukiewicz in the Thirties, which was later taken up by Bar-Hillel in the Fifties. The 1958 and 1961 papers of Jim Lambek are of central importance for the development of the field. In these papers, the type-forming operations are for the first time treated as logical connectives; logical proof theory takes the place of the stipulated rule schemata of the earlier systems. The highly readable seminal paper The Mathematics of Sentence Structure is available electronically.

The integrated treatment of syntax and semantics is now seen as the most attractive aspect of categorial grammar. Historically, the categorial view on the syntax-semantics interface has oscillated a great deal. The original Lambek systems were presented as *syntactic* type calculi. About the same time, Haskell Curry was advocating the use of purely *semantic* types in natural language analysis. Curry in fact criticized Lambek for the admixture of syntactic considerations in his category concept, coining the famous distinction between *tectogrammatic* and *phenogrammatic* organization. The tectogrammatic level, in Curry's view, provides the appropriate information for meaning composition; the phenogrammatic pertains to the way this abstract grammatical structure is represented in terms of surface expressions. About the actual mapping between the two levels, Curry provides no specific information.

The design of the syntax-semantics interface becomes of central importance in Richard Montague's work, who proposes a precise implementation of Frege's *Compositionality Principle*. This fundamental principle in natural language semantics informally requires that the meaning of a complex expression be given as a function of the meaning of its constituent parts, and the way they are put together. In Montague's algebraic setup, compositionality takes the form of a *homomorphism*, that is, a structure-preserving mapping, between a syntactic and a semantic algebra. Ironically, when Van Benthem reintroduced semantic interpretation in the discussion of Lambek's syntactic calculi, it was by establishing the connection between categorial derivations and Curry's own 'formulas-as-types' program.

In the Eighties, the shift towards 'lexicalized' grammar formalisms brings a revival of interest in categorial grammar, which is recognized as the lexicalized framework *par excellence*. The proceedings of the 1985 Tucson conference give a good picture of the types of categorial research in this period, both within the rule-based and within the logical traditions.

The introduction of Linear Logic (Girard 1987), and the wave of research on 'substructural' styles of inference with controlled options for resource management rather than hard-wired global choices, have been important factors for the recent development of categorial grammar. Van Benthem's book *Language in Action* is a detailed study of the relations between categorial derivations, type theory and lambda calculus, and of the place of categorial grammars within the general landscape of resource-sensitive logics. Restall's *Substructural Logics* is an accessible textbook on this subject, doing justice both to Linear Logic and to its many predecessors in modal logic. Apart from the chapter on categorial type logics [25], which is the primary source for this article, the *Handbook of Logic and Language* contains a number of further in-depth chapters charting the connections between categorial type systems and mathematical linguistics and proof theory, type theory, and Montague Grammar.

There is a choice of monographs illustrating the different styles of categorial research. Steedman's recent books *Surface Structure and Interpretation* and *The Syntactic Process* well represent the agenda of Combinatory Categorial Grammar. For the deductive approach, the reader can turn to Morrill's *Type Logical Grammar*, which offers a rich fragment of syntactic and semantic phenomena in the grammar of English. Carpenter's *Type Logical Semantics* is a general introduction to natural language semantics studied from the type-logical perspective, with a detailed treatment of quantifier scope ambiguities as a case study. Jaeger's book *Anaphora and Categorial Grammar* compares Jacobson's CCG approach with the type-logical analysis.

For the presentation of new results in categorial research, the conference *Logical Aspects of Computational Linguistics* has become one of the major platforms. Proceedings are easily available in the Springer Lecture Notes in Computer Science series.

The most versatile computational tool for categorial exploration is Richard Moot's grammar development environment GRAIL. The kernel of this system is a general type-logical theorem prover based on proof nets and structural graph rewriting. The user interacts with the kernel via a graphical user interface, which provides control over the lexicon and the structural module, and which gives access to a full-fledged proofnet based debugger. The system is publicly available under the GNU Public License. A number of sample fragments can be accessed online.

References

[1] Bargelli, D. and J. Lambek, ‘An Algebraic Approach to French Sentence Structure’, in P.de Groote, G.Morrill, C.Retoré (Eds.), *Logical Aspects of Computational Linguistics*. LNAI 2099, Springer, Berlin, 2001, 62–78.

[2] Benthem, J. van, *Language in Action. Categories, Lambdas, and Dynamic Logic*. Studies in Logic, North-Holland, Amsterdam, 1991 (Student edition: MIT Press, Cambridge, MA, 1995)

[3] Benthem, J. van, ‘The semantics of variety in categorial grammar’. In [7], 37–55.

[4] Bernardi, R. (2002), *Title?*. PhD Dissertation, OTS Utrecht University.

[5] Buszkowski, W., ‘Mathematical linguistics and proof theory’. Chapter 12 in Van Benthem and ter Meulen (eds.) *Handbook of Logic and Language*. Elsevier, 1997, pp. 683–736.

[6] Buszkowski, W., ‘Lambek Grammars Based on Pregroups’, in P.de Groote, G.Morrill, C.Retoré (Eds.), *Logical Aspects of Computational Linguistics*. LNAI 2099, Springer, Berlin, 2001, 95–109.

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

[8] Buszkowski, W. and G. Penn (1990)

[9] Carpenter, B. (1996), *Type-Logical Semantics*. MIT Press, Cambridge, MA.

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

[11] Foret, A. (2001), ‘On mixing deduction and substitution in Lambek categorial grammars’, in P.de Groote, G.Morrill, C.Retoré (Eds.), *Logical Aspects of Computational Linguistics*. LNAI 2099, Springer, Berlin, 2001, 158–174.

[12] Groote, Ph. de (1999), ‘The non-associative Lambek calculus with product in polynomial time’. *Proceedings TAB99*. Springer LNAI 1617. 128–139.

[13] Groote, Ph. de (2001), ‘Type raising, continuations, and classical logic’. In Van Rooij and Stokhof (eds.) *Proceedings 13th Amsterdam Colloquium*. 97—101.

[14] Hendriks, H. (1999) ‘The logic of tune. A proof-theoretic account of intonation’. In Lecomte, ed., Proceedings LACL. Springer.

[15] Heylen, D. (1999) *Types and Sorts.* PhD Thesis, Utrecht Institute of Linguistics OTS.

[16] Jacobson, P. (1999) ‘Towards a variable-free semantics’. *Linguistics and Philosophy*, **22**(2):117–184.

[17] Jaeger, G. (2002) *Anaphora and type-logical grammar*. Habilitation.

[18] Johnson, M. (1998) ‘Proof nets and the complexity of processing center-embedded constructions’. *Journal of Logic, Language and Information*, **7**(4):433–447.

[19] Kanazawa, M. (1998) *Learnable classes of categorial grammars*. CSLI Publications, Stanford.

[20] Kruijff, G.-J. and D. Oehrle (eds) (2002), *Resource Sensitivity, Binding, and Anaphora*. Kluwer.

[21] Kurtonina, N. and M. Moortgat (1997) ‘Structural control’. In P. Blackburn and M. de Rijke (eds.) *Specifying Syntactic Structures*. CSLI, Stanford, 75–113.

[22] Kruijff, G.-J. and Richard T. Oehrle, eds., *Resource Sensitivity in Binding and Anaphora*. Studies in Linguistics & Philosophy, Kluwer Academic Publishers, in preparation.

[23] Lambek, J., 'The mathematics of sentence structure'. American Mathematical Monthly **65**, 154–170. Reprinted in [7], pp 153–172.

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

[25] Moortgat, M., 'Categorial type logics'. Chapter 2 in Van Benthem and ter Meulen (eds.) *Handbook of Logic and Language*. Elsevier, 1997, pp. 93–177.

[26] Moot, R. (2001), *Proof Nets for Linguistic Analysis*. PhD Dissertation. OTS, Utrecht University.

[27] Moot, R. and Q. Puite (2002) 'Proof nets for the multimodal Lambek calculus'. *Studia Logica*.

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

[29] Morrill, G. (2000) 'Incremental Processing and Acceptability'. *Computational Linguistics* **26**(3), 319–338.

[30] Pentus, M. (1993) 'Lambek grammars are context-free'. In Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science, pages 429-433, Montreal, Canada, 19-23 June 1993. IEEE Computer Society Press.

[31] Pentus, M. (1994) 'Language completeness of the Lambek calculus'. In Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science, pages 487-496, Paris, France, 4-7 July 1994. IEEE Computer Society Press.

[32] Restall, G. (2000) *An Introduction to Substructural Logics*. Routledge. London.

[33] Stabler, E. (1997), 'Derivational minimalism'. In Retoré (ed.) *Logical Aspects of Computational Linguistics*. Springer LNCS 1328, 68–95.

[34] Steedman, M., *Surface Structure and Interpretation*. Linguistic Inquiry Monograph, MIT Press, Cambridge MA, 1996.

[35] Steedman, M., *The Syntactic Process*, MIT Press/Bradford Books, Cambridge MA, 2000.

[36] Tiede, H.-J. (1999), *Deductive Systems and Grammars: Proofs as Grammatical Structures*. PhD Thesis, Indiana University.