Categorial grammar and formal semantics

Michael Moortgat

11th July 2002

Abstract

This paper will appear, in a slightly shortened form, as an in-depth article (# 231) in the *Encyclopedia of Cognitive Science*, Nature Publishing Group, Macmillan Publishers Ltd. For alerts on the project's progress, visit www.cognitivescience.net.

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 Categories, types, processing, parsing, deduction.

Article definition Categorial grammar: a lexicalized grammar formalism based on logical type-theory. A categorial lexicon assigns one or more types to the atomic elements of a language; the assembly of form and meaning is accounted for in terms of the rules of inference for these types seen as formulas of a grammar logic. Cross-linguistic variation results from extending the invariant core of the grammar logic with facilities for structural reasoning.

Contents

1 Introduction

Categorial grammar, a linguistic framework with firm roots in type theory and constructive logic, is well represented in the logical and mathematical literature. This article puts the emphasis more on the categorial modelling of the cognitive abilities underlying the acquisition, use and understanding of natural language. The sections below address two central questions. First of all, what are the *invariants* of grammatical composition, and how do they capture the uniformities of the form/meaning correspondence across languages? Secondly, how can we reconcile the idea of grammatical invariants with structural *variation* in the realization of the form/meaning correspondence?

The slogan 'parsing as deduction' concisely expresses the categorial perspective on these questions. 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 an expression is well-formed amounts to presenting a derivation, or proof, in the logic for these connectives. Natural language expressions are *signs* with a form 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 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. The final section provides some background information and pointers to current areas of research.

2 Form: grammatical invariants and structural variation

2.1 The base logic

Natural language expressions are structured objects that come with a linear order and a hierarchical grouping. In categorial grammar, the traditional parts of speech assume the form of type formulas. The structure of these types mirrors the composition of the expressions they categorize. The set of type formulas Type is obtained as the closure of a small set Atom of *basic* types under a number of type-forming operations. Individual categorial grammars will differ with respect to the type-forming operations they employ. For the present purposes, the following clauses will be representative.

(1) (ATOMS) Atom is a subset of Type;
(UNARY) if A is a formula in Type, then $\Diamond A$ and $\Box A$ are too;
(BINARY) 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'. Examples could be the type np for proper names, s for sentences, n for common noun phrases. Languages can differ as to which basic type distinctions they make. The unary and binary operations provide a vocabulary to categorize expressions in terms of their constituent parts. Informally, a 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 with a fraction type A/B (or $B\backslash A$) is incomplete: it combines with an expression of type B on its right (or left, respectively) into an expression of type A. 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, Došen (1992) and Kurtonina (1995) make use of *frame-based* models familiar from possible-world semantics for modal logics. 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 assigning subsets of W to the atomic formulas. For complex types, the valuation respects the conditions below.

(2)

$x \in V(\diamondsuit A)$	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. On the proof-theoretic level, we are interested in a deductive system to perform type computations $A \to B$ ('type B is derivable from type A'). We want this system to be faithful to the interpretation of the type-forming operations, in the following sense:

(3) SOUNDNESS AND COMPLETENESS

$A \to B$ is provable iff $V(A) \subseteq V(B)$, for every frame F and valuation V.

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 (4) establishing the relationship between the interpretation of $\diamondsuit$ and $\Box$, and between $\bullet$ and left and right division $\backslash$ and $/$. The patterns in (4) turn $(\diamondsuit, \Box)$, $(\bullet, /)$ and $(\bullet, \backslash)$ into what are known as *residuated pairs* in algebra, or *adjoint functors* in category theory.

(4)

$(R0)$	$\diamondsuit A \to B$	if and only if	$A \to \Box B$
$(R1)$	$A \bullet B \to C$	if and only if	$A \to C/B$
$(R2)$	$A \bullet B \to C$	if and only if	$B \to A\backslash C$

Sample theorems. Let us look at some elementary theorems of the grammatical base logic. From the identity axiom, one obtains the Application schemata of (5b) in one step, using the residuation inferences in the 'if' direction. From Application, one derives the Lifting schemata of (5c), this time reasoning in the 'only if' direction.

(5)

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.	$A \to B/(A\backslash B)$	$(R1 \Rightarrow)$	$A \to (B/A)\backslash B$	$(R2 \Rightarrow)$

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, one types the Transitive Verb as $(np\backslash s)/np$, thus projecting the SVO pattern 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.

Instances of Lifting would be type transitions from np (the type assigned to simple proper names) to $s/(np\backslash s)$ or $((np\backslash s)/np)\backslash(np\backslash s)$. These lifted types are 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 $((np\backslash s)/np)\backslash(np\backslash s)$. 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. One can think of case-marked pronouns, as Lambek (1958) already pointed out. With $s/(np\backslash s)$ as the lexical type assignment for 'he'/'she', but $((np\backslash s)/np)/(np\backslash s)$ for 'him'/'her', we correctly rule out 'him irritates she' while allowing 'he irritates her'.

Elementary theorems for the unary type-forming operations are established in (6).

(6)
$$\begin{array}{llll} \Box A \to \Box A & (Ax) & \Diamond A \to \Diamond A & (Ax) \\ \Diamond\Box A \to A & (R0 \Leftarrow) & A \to \Box\Diamond A & (R0 \Rightarrow) \end{array}$$

An illustration of the added expressivity of the unary operators can be found in Bernardi (2002), where they are used to control the distribution of polarity sensitive items. 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, both 'somebody' and 'nobody' would receive 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 expresses 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' (with no polarity item to be checked), we rely on the fact that in the base logic, we have $s/\Box\Diamond(np\backslash s) \to s/(np\backslash s)$, 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 these theorems, the base logic has (7) as derived rules of inference. 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 $\backslash$ are order-preserving in their numerator, and order-reversing (antitone) in their denominator argument.

(7)
$$\begin{array}{lllll} A \to B & \text{implies} & \Diamond A \to \Diamond B & \text{and} & \Box A \to \Box B \\ & & A/C \to B/C & \text{and} & C\backslash A \to C\backslash B \\ & & C/B \to C/A & \text{and} & B\backslash C \to A\backslash C \\ & & A \bullet C \to B \bullet C & \text{and} & C \bullet A \to C \bullet B \end{array}$$

From a combinatorial point of view, these rules 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. one has to prove 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 structure-building operations $\langle\cdot\rangle$ 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, as shown in (Lambek 1958) for the binary and (Moortgat 1996) for the unary connectives.

The derivational format of Combinatory Categorial Grammar (CCG, (Steedman 2000b) and references cited there) 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, 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.

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

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

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

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

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

Figure 1: Natural deduction. Notation: $\Gamma \vdash A$ for the deduction of a conclusion A from a configuration of assumptions Γ. Axioms: $A \vdash A$. Antecedent structures are built from formulas with the structure-building operations $\langle \cdot \rangle$ and $(\cdot \circ \cdot)$. These are the structural counterparts of $\Diamond$ and $\bullet$, respectively, as the $\Diamond, \bullet$ Introduction rules show.

Multimodal generalization. One can straightforwardly generalize the base logic to a system 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 to a multimodal system, with frames $\langle W, \{R^2_i\}_{i \in I}, \{R^3_j\}_{j \in J} \rangle$ 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 composition modes 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 morphology and syntax 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 $\Diamond, \Box$, 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 turns 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 are discussed below.

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 (3) does not impose any restrictions on the interpretation of $R_\bullet$ and $R_\Diamond$. In this sense, the base logic can be said to capture the *invariants* of grammatical composition. Although the base logic already has a rich deductive structure, 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. To overcome the problem of structural rigidity, one extends the base logic 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. The postulates in (8) create a hierarchy of categorial systems: adding structural options, flexibility of type combination increases, but structural discrimination deteriorates.

(8)
$$\begin{array}{cc} (A \bullet B) \bullet C \to A \bullet (B \bullet C) & \mathsf{A}_l \\ A \bullet (B \bullet C) \to (A \bullet B) \bullet C & \mathsf{A}_r \\ A \bullet B \to B \bullet A & \mathsf{C} \end{array}$$

The rebracketing postulates A_l and A_r added to the $/, \bullet, \backslash$ fragment of the base logic, produce the system known as **L**, the associative calculus of (Lambek 1958). The $/, \bullet, \backslash$ fragment of the base logic itself is know as **NL**: in (Lambek 1961) this systems was obtained by dropping the associativity postulates from **L**. Characteristic theorems of **L** are the type transitions in (9): the Geach laws $\mathsf{G}_r, \mathsf{G}_l$, and the functional composition schemata (known as combinator **B** in CCG) of which $\mathsf{B}_r, \mathsf{B}_l$ are the simplest forms.

(9)
$$\begin{array}{cccc} \mathsf{G}_r & A/B \to (A/C)/(B/C) & B\backslash A \to (C\backslash B)\backslash(C\backslash A) & \mathsf{G}_l \\ \mathsf{B}_r & (A/B) \bullet (B/C) \to A/C & (C\backslash B) \bullet (B\backslash A) \to C\backslash A & \mathsf{B}_l \end{array}$$

Adding the commutativity postulate to **L** produces **LP** (Lambek calculus with permutation), a system coinciding with the multiplicative fragment of linear logic, which has a commutative product operation matched by a single linear implication. The distinction between left-incompleteness and right-incompleteness collapses in the presence of C.

Extending the base logic with facilities for structural reasoning has consequences for the interpretation of the type-forming operations, discussed in (Došen 1992; Kurtonina 1995). An interpretation with respect to *arbitrary* frames, obviously, is not available any more. Instead, each postulate introduces a corresponding frame constraint restricting the interpretation of the Merge relation $R_\bullet$, 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 concatenation, i.e. $x = y \cdot z$. Pentus (1994) proves that **L** indeed is complete with respect to this concatenation interpretation.

Controlled structural reasoning There are many natural language phenomena that seem to require some of the flexibility offered by the postulates (8). Cases of non-constituent coordination can be naturally handled with the possibilities for type-combination that follow from the rebracketing postulates. Displacement phenomena are ubiquitous in natural language, and seem to require some form of commutativity. At the same time, it is clear that in a global form, these structural options overgenerate. Commutativity would entail that well-formedness is preserved under arbitrary changes in word order; free rebracketing makes constituent structure irrelevant for determining grammaticality.

To obtain *controlled* 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. One then imposes type-restrictions on these extra combinators. 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 resource management offered by richer type systems with unary control features and multimodal interaction principles. To compare these two strategies, consider the following cases of extraction.

(10) *a.* what Alice found
b. what Alice found there

$$\dfrac{\dfrac{\text{what}}{wh/(s/np)} \quad \dfrac{\dfrac{\dfrac{\text{Alice}}{np}}{s/(np\backslash s)}\,\mathsf{T} \quad \dfrac{\dfrac{\text{found}}{(np\backslash s)/np} \quad \dfrac{\text{there}}{(np\backslash s)\backslash(np\backslash s)}}{(np\backslash s)/np}\,\mathsf{B}_r}{s/np}}{wh}\,\mathsf{B}_{l\times}$$

Figure 2: *Wh*-extraction: combinator-style derivation. The clause body 'Alice found there' is assigned type s/np by means of the backwards crossed composition combinator $\mathsf{B}_{l\times}$. The rule can apply because the cancelled $(np\backslash s)$ satisfies the type-restriction on $\mathsf{B}_{l\times}$.

In CCG, the peripheral case of extraction (10a) are derived from an assignment $wh/(s/np)$ to the *wh*-pronoun by lifting the type for 'Alice' to $s/(np\backslash s)$ which is then composed with the transitive verb type $(np\backslash s)/np$ for 'found' by means of B_r. To obtain the non-peripheral case of extraction in (10b), one needs the combinator $\mathsf{B}_{l\times}$, a form of composition which depends on the commutativity postulate. To avoid collapse into **LP**, one imposes a side-condition on the rule, restricting the middle term B to certain verbal categories, in this case $(np\backslash s)$.

(11) $\mathsf{B}_{l\times} \quad (B/C) \bullet (B\backslash A) \to A/C \quad$ where B is a predicate category

The $\diamondsuit, \square$ connectives make it possible to avoid extra-logical type-restrictions. The postulates $P1/P2$ below implement a controlled form of rebracketing and reordering for formulas carrying the $\diamondsuit$ control feature, as shown in (Moortgat 1999). With a lexical type assignment $wh/(s/\diamondsuit\square np)$ to the *wh*-pronoun, one obtains peripheral and medial extraction from right branches. Under this analysis, one does not attribute any associativity/commutativity to the $\bullet$ operation itself; displacement effects arise through the *interaction* of the Merge operation with a gap hypothesis carrying the licensing $\diamondsuit$ feature. A derivation is given in Figure 3.

(12) $P1 \quad (A \bullet B) \bullet \diamondsuit C \to (A \bullet \diamondsuit C) \bullet B$
$P2 \quad (A \bullet B) \bullet \diamondsuit C \to A \bullet (B \bullet \diamondsuit C)$

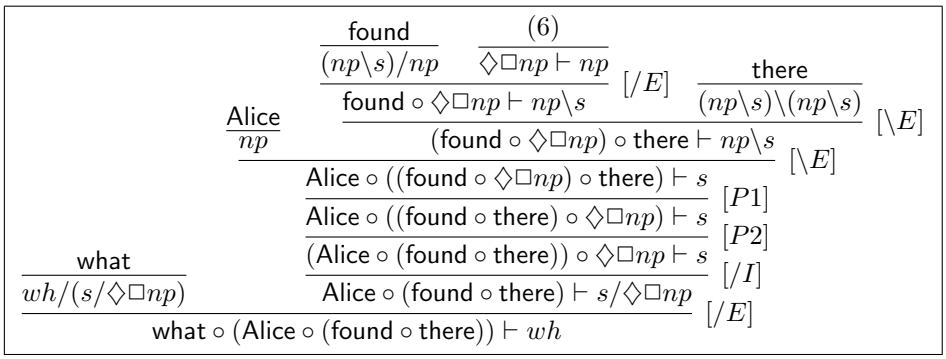


Figure 3: *Wh*-extraction: $\Diamond$ control. The type-assignment to the relativizer 'what' expresses the fact that the relative clause body is a sentence built with the help of a 'gap' hypothesis of type $\Diamond\Box np$. The feature-marked hypothesis has to be withdrawn at the right periphery, but 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 $\Diamond\Box np \rightarrow np$. The 'gap' hypothesis is then used as a regular direct object with respect to the selecting verb 'found'.

2.3 Generative capacity and computational complexity

The modular view on grammatical invariants and structural variation invites a comparison between the categorial landscape and the Chomsky hierarchy. For a recent survey, see (Buszkowski 1997). The discovery in the Eighties of dependency patterns that cannot be adequately captured by context-free grammars has led to an interest in 'mildly context-sensitive' formalisms, i.e. systems with an expressivity beyond context-free, but sufficiently restricted to have polynomial parsing algorithms. The classical Ajdukiewicz/Bar-Hillel 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 same is true for the base logic described in §2.1 (**?**). The correctness of Chomsky's conjecture that context-free equivalence extends to the Lambek calculus **L** was finally established in (Pentus 1993). This result does not have a direct corollary for polynomial parsability, because the construction of a context-free grammar from an **L** grammar is of exponential complexity.

For the structural extensions of the base logic discussed in §2.2, the challenge is to identify appropriate constraints: it is clear that arbitrary combinator extensions, or structural rule packages, lead to excessive expressivity. But Vijay-Shanker and Weir (1994) show that an appropriately restricted version of CCG is weakly equivalent to the linear indexed grammars, hence polynomially parsable. In a similar spirit, Moot (2002) 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. The general theory of $\Diamond, \Box$ as control operators has been investigated in (Kurtonina and Moortgat 1997). These authors establish a number of embedding theorems showing that the full logical space between the base logic and **LP** 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).

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, discussed in §3) that produce a certain string. In this area, Tiede (2001) has obtained interesting results, showing that while the Lambek systems **(N)L** are weakly CF, their expressivity in terms of strong capacity goes beyond that of CF grammars.

2.4 Language learning

Kanazawa (1998) has studied formal learning theory for categorial grammar within Gold's paradigm of identification in the limit on the basis of positive data. The focus 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). It is a matter of dispute whether Gold's very abstract formulation of the learning problem is directly relevant for first language acquisition. An alternative purely inductive approach, learning a subclass of the shallow context-free languages, is presented in (Adriaans 2002).

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 Foret (2001) is promising in this respect: she mixes unification/substitution with Lambek-style deduction, suggesting modulation of learnability questions 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.

3 Meaning assembly: the Curry-Howard correspondence

Categorial grammar adheres to the *truth-conditional* theory of semantics: the interpretation process establishes a systematic relationship between linguistic expressions and states of affairs in the world in such a way that specifying the meaning of a sentence comes down to giving its truth conditions. As in the previous section, model theory provides the tools to carry out this program. For semantic interpretation this involves the construction of a set-theoretic model of 'the world' in terms of objects and configurations of such objects; these set-theoretic constructs then serve as the semantic values of natural language expressions.

The integrated treatment of syntax and semantics, which is now seen as the most attractive aspect of categorial grammar, is of relatively recent origin. The original Lambek systems (Lambek 1958; Lambek 1961) were presented as *syntactic* type calculi. About the same time, Curry (1961) 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. The cornerstone of his Universal Grammar programme is a precise implementation of Frege's *Compositionality Principle*. Informally, this fundamental principle in natural language semantics 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 (1987) 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 which we describe below. For expository purposes, the discussion below is restricted to functional types; the full Curry-Howard interpretation involves extension to the other type-forming operations.

3.1 Modeltheoretic semantics, type theory 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 is 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. Finally, the semantic type system naturally fits the language of the typed lambda calculus, which we can then use, together with its standard interpretation, to specify the instructions for meaning assembly.

Semantic and syntactic types. For a simple extensional interpretation, the set of atomic semantic types 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 SemType is then obtained by closing SemAtom under the rule that if A and B are in 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. In §3.2, we will present a procedure to associate a derivation $A_1, \ldots, A_n \vdash B$ with a term t of type B representing a recipe for meaning assembly with parameters $x_1, \ldots, x_n$ for the lexical assumptions $A_1, \ldots, A_n$. To prepare the ground, 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 $k \in D_A$, $h(k) = [\![t]\!]^{g'}$, where g' 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*).

(13)

$(\lambda x.t)\ u$	$\rightsquigarrow_\beta$	$t[u/x]$	provided u is free for x in t
$\lambda x.(t\ x)$	$\rightsquigarrow_\eta$	t	provided x is not free in t

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 $\Gamma - A$ results from dropping, some or all occurrences of A from Γ.

$$\frac{\Gamma \vdash A \rightarrow B \quad \Delta \vdash B}{\Gamma, \Delta \vdash B} \rightarrow \text{Elim} \qquad \frac{\Gamma \vdash B}{\Gamma - A \vdash A \rightarrow B} \rightarrow \text{Intro} \tag{14}$$

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.

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

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

Figure 4: Natural Deduction rules: term labeling.

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.

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\ C \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ W \tag{15}$$

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 5 and 6. Whereas the proof term in Fig 5 is a faithful encoding of the derivation (modulo directionality and structural operations), the term one obtains in Fig 6 after substitution of lexical meaning programs and β simplification has lost the transparency with respect to the derivation.

$$\dfrac{\dfrac{\mathsf{Noun}}{z_0 : n} \quad \dfrac{\dfrac{\mathsf{that}}{x_1 : (n\backslash n)/(s/np)} \quad \dfrac{\dfrac{\dfrac{\mathsf{Subj}}{x_2 : np} \quad \dfrac{\dfrac{\mathsf{TV}}{y_2 : (np\backslash s)/np} \quad [np \vdash y_1 : np]^1}{\mathsf{TV} \circ np \vdash (y_2\ y_1) : np\backslash s}\,[/E]}{\mathsf{Subj} \circ (\mathsf{TV} \circ np) \vdash ((y_2\ y_1)\ x_2) : s}\,[\backslash E]}{(\mathsf{Subj} \circ \mathsf{TV}) \circ np \vdash ((y_2\ y_1)\ x_2) : s}\,[P2]}{\mathsf{Subj} \circ \mathsf{TV} \vdash \lambda y_1.((y_2\ y_1)\ x_2) : s/np}\,[/I]^1}{\mathsf{that} \circ (\mathsf{Subj} \circ \mathsf{TV}) \vdash (x_1\ \lambda y_1.((y_2\ y_1)\ x_2)) : n\backslash n}\,[/E]}{\mathsf{Noun} \circ (\mathsf{that} \circ (\mathsf{Subj} \circ \mathsf{TV})) \vdash ((x_1\ \lambda y_1.((y_2\ y_1)\ x_2))\ z_0) : n}\,[\backslash E]$$

Figure 5: Computation of the *proof term* for the pattern 'Noun that Subj Transitive-Verb'. Leafs are labeled with variables. The derivation produces a meaning recipe with parameters for the lexical meaning programs. The recipe can be applied to any particular choice of lexical items fitting the type requirements: 'biscuit that Alice ate', 'book that Carroll wrote', etc.

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 Figure 3 is an illustration. Secondly, lexical 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 Figure 6, a program which computes property intersection. Another example would be a reflexive pronoun like 'himself'. With a type $((np\backslash s)/np)\backslash(np\backslash s)$, it consumes its transitive verb argument in a

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

Figure 6: Substitution of lexical semantics in the pattern 'Noun that Subj Transitive-Verb'. Boldface for non-logical constants. In steps 10 and 11, β conversion is applied on the fly to the application terms obtained from the slash elimination rules. The derivation is presented in the linear or Fitch style Natural Deduction format.

resource-sensitive way. The identification of subject and object arguments of the verb is realized through its lexical lambda term $\lambda x \lambda y.((x\ y)\ y)$.

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 (16c) below, where the embedded quantifier takes scope at the main clause level.

(16)
a. Alice thinks someone left.
b. $((\mathbf{think}\ (\exists\ \lambda x.(\mathbf{leave}\ x)))\ \mathbf{a})$
c. $(\exists\ \lambda x.((\mathbf{think}\ (\mathbf{leave}\ x))\ \mathbf{a}))$
d. *Alice* thinks *she* dreams
e. $((\mathbf{think}\ (\mathbf{dream}\ \mathbf{a}))\ \mathbf{a})$

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 (16d), where the anaphor in the subordinate clause can pick up its antecedent in the main clause. 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.

Proposals for dealing with these problems rely either on combinator-style type-shifting rule schemata or on structural extensions of the Lambek calculus. For quantifier scope construal, these options are discussed in depth in Carpenter (1998). For anaphora resolution, Jäger (2001) offers a comparison of the CCG approach of (Jacobson 1999) with a type-logical treatment based on identity semantics for anaphora, in combination with a restricted copying rule in syntax, in the form of a controlled structural rule of Contraction. An alternative perspective on scope and anaphora, more in the spirit of Curry's tectogrammatic programme, simplifies the categorial type theory to a non-directional **LP** system, and enforces structural control by introducing lambda term labeling also for the *form* dimension of grammatical signs. Oehrle (1994) is an early formulation of this approach, which has recently found new advocates.

3.4 Processing issues

The interpretation procedure discussed above is essentially dynamic: interpretations are assembled 'on line' in the course of the derivation process, rather than being computed *post hoc* from a given static structure. This has led to a distinctly 'categorial' view on processing issues.

Incrementality, information structure. The flexible notion of derivational constituency engendered by type-changing principles makes left-to-right parsing directly compatible with incremental interpretation. The resulting categorial modeling of natural language processing has been worked out in (Steedman 2000a). This work shows that derivational constituency is guided by *prosodic* articulation (intonation contour). To do justice to this dimension of grammatical organization, one needs a richer notion of semantic interpretation, accommodating notions of focus and information structure. Steedman's proposals are formulated in the CCG style; Hendriks (1999) analyses information packaging and intonation contour in multimodal type-logical terms.

Proof nets. A novel computational view on natural language processing derives from the *proof net* approach. Proof nets were originally developed in the context of Linear Logic, where they elegantly capture the essence of resource-sensitive derivations in graph-theoretical terms. Moot and Puite (2002) refine the proof net techniques for use with the grammatical type logics discussed in this article, where apart from resource multiplicity also structural patterns have to be taken into account.

Johnson (1998) and Morrill (2000) have pointed out that proof nets offer an attractive perspective on performance phenomena. 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 7.

Insert Fig 7 here

4 Exploration

4.1 Variants and alternatives

Pregroup grammars. An interesting variation on the categorial theme has been developed by Jim Lambek in a number of recent papers (Lambek 1999; Lambek 2001). 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 $a^l a \to 1 \to a a^l$ and $a a^r \to 1 \to a^r a$, 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 (1999) 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 $1 \to a a^l$ and $1 \to a^r a$ 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.

(17)

	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	$np^r\, s\, np^l$	$\to n$

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 (2001) 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 the algebraic formulation of (Stabler 1997; Stabler 1999). 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.

4.2 Further reading

The Supplementary References provide material for further exploration. We present brief guidelines below.

The history of categorial grammar is generally traced back to the work of Ajdukiewicz in the Thirties (Ajdukiewicz 1935), which was later taken up by Bar-Hillel in the Fifties (Bar-Hillel 1953). Jim Lambek's early papers (Lambek 1958; Lambek 1961), virtually unnoticed at the time, have proved to be 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 seminal 1958 paper is available electronically through *JSTOR*, and reprinted in (Buszkowski et al. 1998), a collection which contains more of the early papers.

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 (Oehrle et al. 1988) give a good picture of the types of categorial research in this period, both within the rule-based and within the logical traditions. Van Benthem's contribution to this volume has been instrumental in introducing Lambek's logical approach to the linguistic community.

The advent 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. *Language in*

Action (van Benthem 1995) 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. *Substructural Logics* (Restall 2000) is an accessible textbook on this subject, doing justice both to Linear Logic and to its many predecessors in modal logic. The connections between linear logic, categorial grammar, and computational formulations of minimalist grammars are explored in a special issue of *Language and Computation* (Retoré and Stabler 2002). *Proofs and types* (Girard, Lafont, and Taylor 1988) is a good source for the Curry-Howard interpretation.

Apart from the chapter on categorial type logics (Moortgat 1997), which is the primary source for this article, the *Handbook of Logic and Language* (van Benthem and ter Meulen 1997) contains a number of further in-depth chapters that can be consulted for the connections between categorial type systems and mathematical linguistics and proof theory, formal learning theory, type theory, and Montague Grammar.

There is a choice of monographs and collections illustrating the different styles of current categorial research. Steedman's recent books *Surface Structure and Interpretation* and *The Syntactic Process* (Steedman 1996; Steedman 2000b) well represent the agenda of Combinatory Categorial Grammar. For the deductive approach, the reader can turn to *Type Logical Grammar* (Morrill 1994), which offers a rich fragment of syntactic and semantic phenomena in the grammar of English, using a variety of type-forming operations (Boolean, quantificational) in addition to the composition operators discussed here. *Type Logical Semantics* (Carpenter 1998) is a general introduction to natural language semantics studied from the type-logical perspective; this book includes a detailed discussion of quantifier scope ambiguities as a case study. The collection (Kruijff and Oehrle 2002) reflects current categorial views on anaphora and binding.

A 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 at `http://www.let.uu.nl/~Richard.Moot/personal/grail.html`. A number of sample fragments can be accessed online at `http://www.grail.let.uu.nl/tour.pdf`.

Text References

Adriaans, P. (2002). Learning shallow context-free languages under simple distributions. In K. Vermeulen and A. Copestake (Eds.), *Algebras, Diagrams and Decisions in Language, Logic and Computation*, CSLI Lecture Notes. Stanford: CSLI.

Bernardi, R. (2002). *Reasoning with polarities in categorial type logic*. Ph. D. thesis, Utrecht Institute of Linguistics OTS, Utrecht University.

Buszkowski, W. (1997). Mathematical linguistics and proof theory. In J. van Benthem and A. ter Meulen (Eds.), *Handbook of Logic and Language*, Chapter 12, pp. 683–736. Elsevier/MIT Press.

Buszkowski, W. (2001). Lambek grammars based on pregroups. In P. de Groote, G. Morrill, and C. Retoré (Eds.), *Logical Aspects of Computational Linguistics*, Volume 2099 of *Lecture Notes in Artificial Intelligence*, Berlin, pp. 95–109. Springer.

Carpenter, B. (1998). *Type-logical Semantics*. Cambridge, Massachusetts: MIT Press.

Curry, H. B. (1961). Some logical aspects of grammatical structure. In R. Jacobson (Ed.), *Structure of Language and its Mathematical Aspects*, Volume XII of *Proceedings of the Symposia in Applied Mathematics*, pp. 56–68. American Mathematical Society.

Došen, K. (1992). A brief survey of frames for the Lambek calculus. *Zeitschrift für mathematischen Logik und Grundlagen der Mathematik 38*, 179–187.

Foret, A. (2001). On mixing deduction and substitution in Lambek categorial grammars. In P. de Groote, G. Morrill, and C. Retoré (Eds.), *Logical Aspects of Computational Linguistics*, Volume 2099 of *Lecture Notes in Artificial Intelligence*, pp. 158–174. Berlin: Springer.

Hendriks, H. (1999). The logic of tune. a proof-theoretic analysis of intonation. In A. Lecomte, F. Lamarche, and G. Perrier (Eds.), *Logical Aspects of Computational Linguistics*, Volume 1582 of *Lecture Notes in Artificial Intelligence*, pp. 132–159. Springer.

Jacobson, P. (1999). Towards a variable-free semantics. *Linguistics and Philosophy 22*(2), 117–184.

Jäger, G. (2001). Anaphora and quantification in categorial grammar. In M. Moortgat (Ed.), *Logical Aspects of Computational Linguistics*, Volume 2014 of *Lecture Notes in Artificial Intelligence*, pp. 70–90. Springer.

Johnson, M. (1998). Proof nets and the complexity of processing center-embedded constructions. *Journal of Logic, Language and Information 7*(4), 443–447.

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

Kurtonina, N. (1995). *Frames and Labels. A Modal Analysis of Categorial Inference*. Ph. D. thesis, OTS Utrecht, ILLC Amsterdam.

Kurtonina, N. and M. Moortgat (1997). Structural control. In P. Blackburn and M. de Rijke (Eds.), *Specifying Syntactic Structures*, pp. 75–113. Stanford: CSLI Publications.

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

Lambek, J. (1961). On the calculus of syntactic types. In R. Jacobson (Ed.), *Structure of Language and its Mathematical Aspects*, Volume XII of *Proceedings of the Symposia in Applied Mathematics*, pp. 166–178. American Mathematical Society.

Moortgat, M. (1996). Multimodal linguistic inference. *Journal of Logic, Language and Information 5*(3–4), 349–385.

Moortgat, M. (1999). Constants of grammatical reasoning. In G. Bouma, E. Hinrichs, G.-J. Kruijff, and R. T. Oehrle (Eds.), *Constraints and Resources in Natural Language Syntax and Semantics*, pp. 195–219. Stanford: CSLI.

Moot, R. (2002). *Proof Nets for Linguistic Analysis*. Ph. D. thesis, Utrecht Institute of Linguistics OTS, Utrecht University.

Moot, R. and Q. Puite (2002). Proof nets for the multimodal Lambek calculus. *Studia Logica 71*. Special issue on the occasion of Lambek's 80th birthday, edited by Wojciech Buszkowski and Michael Moortgat.

Morrill, G. (2000). Incremental processing and acceptability. *Computational linguistics 26*(3), 319–338.

Oehrle, R. T. (1994). Term-labeled categorial type systems. *Linguistics & Philosophy 17*(6), 633–678.

Pentus, M. (1993). Lambek grammars are context free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pp. 429–433. IEEE Computer Society Press.

Pentus, M. (1994). Language completeness of the Lambek calculus. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science*, pp. 487–496. IEEE Computer Society Press.

Stabler, E. (1997). Derivational minimalism. In C. Retoré (Ed.), *Logical Aspects of Computational Linguistics*, Volume 1328 of *Lecture Notes in Artificial Intelligence*, Berlin, pp. 68–95. Springer.

Stabler, E. (1999). Remnant movement and complexity. In G. Bouma, E. Hinrichs, G.-J. Kruijff, and R. T. Oehrle (Eds.), *Constraints and Resources in Natural Language Syntax and Semantics*, pp. 299–326. Stanford: CSLI.

Steedman, M. (2000a). Information structure and the syntax-phonology interface. *Linguistic Inquiry 31*(4), 649–689.

Tiede, H.-J. (2001). Lambek calculus proofs and tree automata. In M. Moortgat (Ed.), *Logical Aspects of Computational Linguistics*, Volume 2014 of *Lecture Notes in Artificial Intelligence*, pp. 251–265. Springer.

van Benthem, J. (1987). Categorial grammar and lambda calculus. In D. Skordev (Ed.), *Mathematical Logic and Its Applications*, pp. 39–60. New York: Plenum Press.

Vijay-Shanker, K. and D. Weir (1994). The equivalence of four extensions of context free grammars. *Mathematical Systems Theory 27*(6), 511–546.

Supplementary References

Ajdukiewicz, K. (1935). Die syntaktische Konnexität. *Studia Philosophica 1*, 1–27. (English translation in Storrs McCall (ed.) Polish Logic, 1920-1939. Oxford (1996), 207-231).

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

Buszkowski, W., W. Marciszewski, and J. van Benthem (Eds.) (1998). *Categorial Grammar*. Amsterdam: Benjamins.

Dowty, D., R. Wall, and S. Peters (1981). *Introduction to Montague Semantics*. Dordrecht: Reidel.

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

Girard, J.-Y., Y. Lafont, and P. Taylor (1988). *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press.

Kruijff, G.-J. and R. Oehrle (2002). *Resource sensitivity in Binding and Anaphora*. Dordrecht: Reidel.

Lambek, J. (1999). Type grammar revisited. In A. Lecomte, F. Lamarche, and G. Perrier (Eds.), *Logical Aspects of Computational Linguistics*, Volume 1582 of *Lecture Notes in Artificial Intelligence*, pp. 1–27. Springer.

Lambek, J. (2001). Type grammars as pregroups. *Grammars 4*(1), 21–39.

Montague, R. (1974). *Formal Philosophy: Selected papers of Richard Montague*. Yale University Press.

Moortgat, M. (1997). Categorial type logics. In J. van Benthem and A. ter Meulen (Eds.), *Handbook of Logic and Language*, Chapter 2, pp. 93–177. Elsevier/MIT Press.

Morrill, G. (1994). *Type Logical Grammar: Categorial Logic of Signs*. Dordrecht: Kluwer Academic Publishers.

Oehrle, R., E. Bach, and D. Wheeler (Eds.) (1988). *Categorial Grammars and Natural Language Structures*. Dordrecht: Reidel.

Restall, G. (2000). *An Introduction to Substructural Logics*. Routledge.

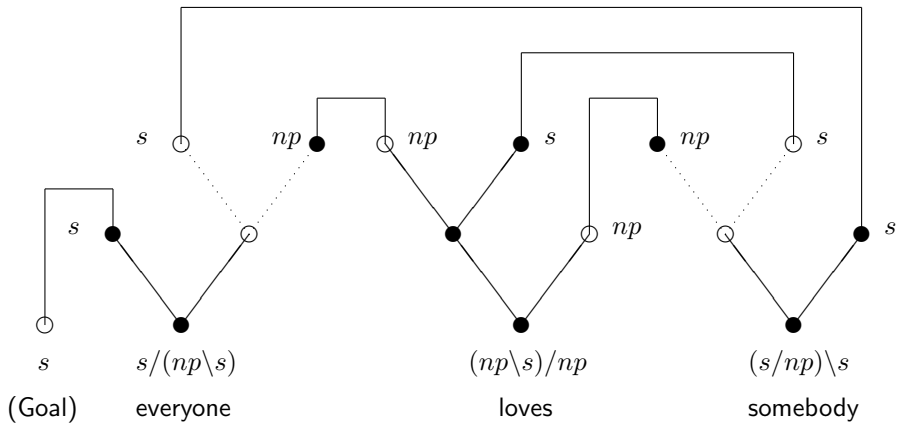
Retoré, C. and E. Stabler (Eds.) (2002). *Resource logics and minimalist grammars*. Proceedings ESSLLI'99 workshop (Special issue Language and Computation).

Steedman, M. (1996). *Surface structure and interpretation*. Linguistic Inquiry Monograph. Cambridge, MA: MIT Press.

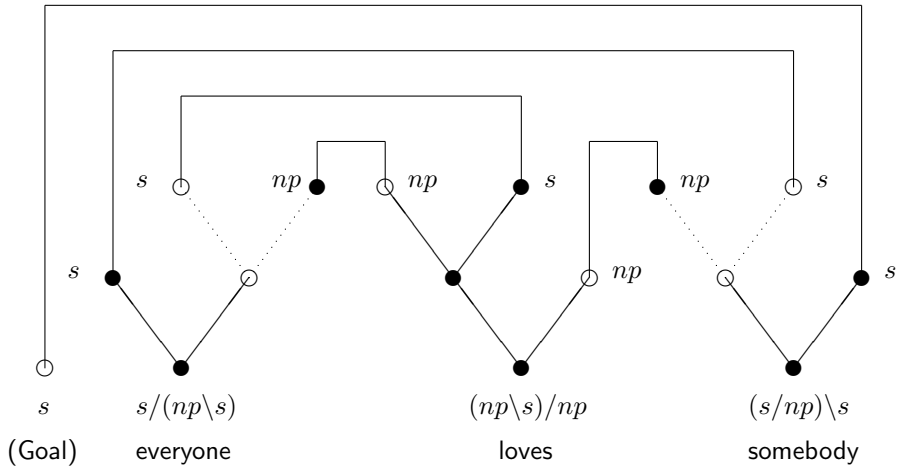
Steedman, M. (2000b). *The Syntactic Process*. Cambride, MA: MIT Press.

van Benthem, J. (1995). *Language in Action: Categories, Lambdas and Dynamic Logic*. Cambridge, Massachusetts: MIT Press.

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



Subject wide scope: $\forall\ (\lambda x\ \exists\ (\lambda y\ ((\mathbf{love}\ y)\ x)))$



Object wide scope: $\exists\ (\lambda y\ \forall\ (\lambda x\ ((\mathbf{love}\ y)\ x)))$

Figure 7: A proof net for the sentence 'everyone loves somebody'. Formula decomposition trees with polarized vertices (black: input; white: output). Solid (dotted) edges for input (output) slashes. A linking of leafs with opposite polarities is well-formed if it produces a graph which is connected, acyclic (for each removal of a dotted egde from a pair), and planar. The net is constructed in a left-to-right incremental fashion. Processing complexity is measured in terms of the number of unresolved dependencies. The subject wide-scope reading for 'everyone loves somebody' (maximum of unresolved dependencies: 3) is preferred over the object wide-scope reading (maximum of unresolved dependencies: 4). Sources: Johnson (1998), Morrill (2000).